С.Н.Лукин

Visual Basic 6.0

самоучитель

для начинающих

2001

Содержание

Глава 0. Предисловие

Какая все-таки прелесть этот Visual Basic! Впечатление такое, будто у вас в руках волшебная палочка. Легкий взмах – и программа готова: вы видите на экране собственный калькулятор или систему управления базами данных, или мультфильм…. Или вот, например: Мрачная городская улица. Появляется автомобиль. Он стремительно приближается к главному герою, спрятавшемуся с бластером в руках за рекламной тумбой. Из автомобиля доносятся выстрелы. Пули щелкают по тумбе. В последний момент – визг тормозов, и машина застывает. Теперь не медли – бей по клавишам клавиатуры, стреляй, пока крестные отцы наркомафии до тебя не добрались! Автомобиль вспыхивает, из него выскакивают “братки” и, спасаясь от твоих выстрелов, бросаются в ближайшие укрытия, откуда открывают ожесточенный огонь. Схватка продолжается… После изучения этой книжки вы безусловно будете способны создать на Visual Basic двумерную версию такой игры.

Конечно, взмах волшебной палочкой – всего лишь первое впечатление. Да и неинтересно "палкой махать" вместо того, чтобы самому программировать. Но факт остается фактом – с появлением Visual Basic программировать стало гораздо легче и приятней, чем раньше. И программы можно делать гораздо более мощные. Раньше начинающий программист и помыслить не мог о том, чтобы запрограммировать игру со сложным поведением нескольких персонажей, подобную той, что я только что описал. С появлением Visual Basic это стало возможно.

Visual Basic принадлежит к разряду новых и замечательных инструментов программирования, а именно – к средам визуальной разработки программ, к которым принадлежат также Delphi (в основе которой лежит современная версия языка Паскаль – Object Pascal), Visual C++ и некоторые другие. Отличие сред визуальной разработки от более ранних инструментов программирования состоит в том, что они вместо программиста “пишут” наиболее часто встречающиеся и скучные части программы, освобождая его таким образом для интересного и творческого труда. Иногда, если задача простая, случается, что компьютер создает всю программу полностью. Еще одна особенность – программисту, работающему в Visual Basic и других подобных средах, никуда не деться от так называемого объектного программирования, завоевавшего в последние годы программистский мир.

0.1. Кому предназначена эта книга?

Это самоучитель. То есть написана книга с расчетом на то, что, изучив ее <u>без посторонней помощи</u>, вы сможете <u>без посторонней помощи</u> составлять программы на Visual Basic и выполнять их на компьютере тоже <u>без посторонней помощи</u>. Автор приложил специальные усилия для того, чтобы изложение было понятным. Все объясняется на примерах. Рассмотрение почти каждой темы заканчивается задачами на составление программы (всего таких задач – 132). Подавляющее большинство задач снабжено ответами, так что читатель может эффективно контролировать усвоение материала.

Если вы хотите научиться программировать, но никогда в жизни не написали ни одной программы и плохо представляете, как компьютер устроен, читайте эту книгу с начала до конца – вы и программировать научитесь, и об устройстве компьютера узнаете все необходимое. Я хочу, чтобы прочтя книгу и решив все задачи, вы могли с удовлетворением сказать себе – Да, я научился программировать, я – программист!

Если вам интересно программировать **игры**, то знайте, что моя книга именно на это и ориентирована. Изучив ее, вы будете вполне готовы к созданию собственных игр типа морского боя, крестиков-ноликов, танковой битвы (где множество танков движутся по квадратному полю), стрелялок или той хотя бы, что я описал выше. Впрочем, подробнее об играх – ниже, в 0.2.

Предположим, что **вы школьник или студент,** первый год изучающий программирование на Visual Basic, и вам предстоит сдавать по нему экзамен. Вы чувствуете, что вы «на нуле» и НИЧЕГО НЕ ПОНИМАЕТЕ и даже не знаете, с чего начать. В этом случае, изучив книгу и выполнив приведенные в ней задания, **вы будете прочно знать основы Visual Basic**. Достаточно ли этого?. Ведь Visual Basic – это богатая новогодняя елка, на которой великое множество игрушек. В моей книге рассказано все, что нужно для новичка: про ствол и главные ветви елки, про то, как игрушки крепятся к веткам, рассмотрены самые интересные игрушки. Однако, где гарантия, что ваш преподаватель не предпочитает другие? Может быть, взять книгу потолще? Уверяю вас, если вы возьмете любую самую толстую книгу по Visual Basic, то и там наверняка не найдете всего, что от вас требуется. Игрушек в Visual Basic слишком много даже для трехтомника. Поэтому сначала изучите основы Visual Basic по книге вроде моей. Затем или постарайтесь найти толстую книгу, в которой рассказано конкретно про то, что вам нужно, или воспользуйтесь для этого системой помощи (Help). Сразу же читать толстую книгу затруднительно – обычно она не предназначена для начинающих, если даже и утверждает обратное. Впрочем, толстая книга может и не понадобиться – в моей книге я постарался рассмотреть все, что разумный преподаватель может спросить на первом году обучения.

Если вы опытный программист в средах визуальной разработки программ, но хотите изучить еще и Visual Basic, вам эта книга не нужна - изложение рассчитано на начинающих.

Если у вас **под рукой нет компьютера**, то дела ваши плохи. Без компьютера, по одной только книге вы сможете научиться лишь писать на бумаге программный текст. Конечно, и это немало, однако напоминает чтение натощак, когда в холодильнике пусто, рецепта приготовления роскошного бифштекса. В среде визуальной разработки программ для Windows, которой является Visual Basic, программный текст – далеко еще не все.

Книга учит не только составлять программы, но и **выполнять программы на компьютере**. Все, что вам нужно знать заранее, это где кнопка включения компьютера. Все остальное в книге объяснено, включая инструкцию по установке Visual Basic на ваш компьютер.

0.2. Задачи, игры.

При создании книги очень важно было решить – какие выбрать примеры, на основе которых строить изложение материала. Здесь действует суровое правило: плохие примеры – плохая книжка, хорошие примеры – хорошая книжка, скучные примеры –

скучная книжка, интересные примеры – интересная книжка.

Вот солидная и популярная тема для примеров – системы управления базами данных. Однако надолго интересной она может стать только для тех, кто имеет уже перед собой задачу эффективного управления большими и сложными базами данных, а таких читателей меньшинство. Поэтому базам данных посвящено ровно столько материала, сколько нужно, чтобы читатель понял, что это такое, сам мог создать базу данных, проделать с ней основные операции и затем понять толстые книжки, базам данных посвященные.

Я решил, что самой простой и интересной темой для примеров будут игры и другие занимательные задачи. Этот путь мне кажется самым веселым. А результат - ничуть не хуже, чем если бы мы занимались базами данных или чем-нибудь другим серьезным. Это не значит, что мы все время будем забавляться и играть. В программировании есть вещи, которые удобнее иллюстрировать на неигровых примерах.

Вот задачи на программирование, решенные в книге:

- Игра "Автогонки"
- Игра "Сачок", где вы должны ловить множество движущихся шариков
- Игра в «Угадай число»
- Задача: как поставить пароль на свою программу, чтобы никто, кроме вас, не мог ее запустить
- Задача: как запрограммировать сохранение в играх
- Собственный калькулятор (при желании - с вашей фотографией, музыкой, прыгающими кнопками и прочими «приколами»)
- Собственный будильник-секундомер
- Воспроизведение музыкальных произведений и вашего голоса.
- Воспроизведение видео.
- Собственный мультфильм
- Создание своих рисунков и узоров (круги на воде, звездное небо, мыльные пузыри, компакт-диск, башня, ковер и тому подобное)
- Задача: как в Интернете зайти в чужой компьютер (предварительно постучавшись)
- Задача: как в Интернете поместить на свою страничку игру
- Собственный графический редактор
- Зашифровка и расшифровка секретных файлов (объяснена простейшая шифровка и расшифровка)
- Определитель быстроты реакции вашей и ваших друзей
- Программа, определяющая, "экстрасенс ли вы"

Вот примеры задач, предлагаемые для самостоятельного решения:

- Задача: как изготовить собственный компакт-диск с вашими любимыми хитами
- Простейшая игра-стрелялка "Торпедная атака"
- Игра в крестики-нолики на бесконечном поле (пять в линию)
- Игра в морской бой
- Игра «Танковый бой» на неподвижном поле (вид сверху), где одновременно движутся и стреляют маленькие фигурки ваших и вражеских танков

Если вы энтузиаст программирования игр, то я должен вам сказать о том, какие игры будет НЕ под силу создавать начинающему программисту после изучения моей книги. Прежде всего это трехмерные (3D) игры типа Quake или Unreal. Учтите, что эти игры создают не новички, а профессиональные программисты и не в одиночку, а целыми фирмами. И используют они для этого специальные программные средства (например, Direct X), разобраться в которых можно, только съев пуд соли в геометрии и программировании. Изучение конкретного языка программирования - это всего лишь первая и не самая сложная ступенька в деле создания таких солидных продуктов, как качественная 3D-игра.

Если вы думаете, что начав изучать программирование не с Visual Basic, а с Delphi или C++, вы быстрее и легче придете к программированию более сложных игр, то глубоко ошибаетесь. Если бы вы обратились за советом, с чего начать, к профессиональному программисту, который всю жизнь писал игры на C++, то в ответ почти наверняка услышали бы: «Начни с Бэйсика, дружок!»

Так что, если вы вообразили описанную выше игру про мафию, как 3D, то ошиблись. Вот что вы сможете реально сделать. На экране – неподвижная картинка города, сколь угодно красивая, нарисованная вами или где-нибудь добытая. По городу движутся автомобили, прохожие, летят пули, вспыхивают лазерные лучи. Вы сможете создать голоса персонажей, звуки выстрелов и прочее. Однако движения персонажей будут не такими реалистичными, как в 3D-играх. Вам легко будет сделать, чтобы фигуры персонажей в застывшей позе двигались в любых нужных вам направлениях и даже при этом уменьшались в размерах (удаляясь в перспективе) или увеличивались (приближаясь). Немного труднее будет заставить их при этом передвигать ногами. Еще больше времени уйдет, чтобы запрограммировать более сложные движения, такие как повернуться боком, пригнуться, лечь, зарядить винтовку, улыбнуться, залезть в карман и тому подобное. Если герою вздумается долго идти направо, то вам трудно будет заставить город услужливо прокручиваться налево, как это происходит в играх-аркадах.

0.3. Почему Visual Basic?

Почему Visual Basic? Этот вопрос стоит перед новичками, которые не знают, с какого языка начать. Вот краткий обзор возможных вариантов.

Прежде всего, для полезного, занимательного и веселого изучения основных идей программирования существуют специальные учебные языки, рассчитанные на детей и новичков. Это «Кенгуренок», «Пылесосик», Лого. Кстати, Лого - язык достаточно богатый, чтобы программировать на нем и несложные игры с примитивной графикой. Но, к сожалению, эти языки мало распространены в России и по ним почти нет литературы. Так что вопрос о них отпадает. Поэтому перейдем к рассмотрению профессиональных взрослых языков. Сейчас наиболее известны Бэйсик, Паскаль, Си, Джава (Ява) в их самых разных версиях.

Но Ява пока применяется в основном только в Интернете. Так что остаются Бэйсик, Паскаль, Си.

Не так давно в ходу были старые, невизуальные версии сред программирования для этих языков. Теперь положение резко изменилось. Нет никакого вопроса в том, в чем вам программировать – в старой среде или в визуальной. Это все равно, что спрашивать, из чего лучше стрелять – из рогатки или из пушки. Остается выбрать конкретную среду – Visual Basic, Delphi, Visual C++ (для Бэйсика, Паскаля и Си соответственно) или какую-нибудь другую.

Си – высокопрофессиональный язык, программы, написанные на нем, имеют превосходное быстродействие, в среде программистов он распространен очень широко, но слишком сложен для восприятия новичком и с него лучше не начинать. Если вам очень уж хочется программировать на Си, рекомендую начать с Бэйсика. После него освоение Си пойдет гораздо легче.

Delphi по простоте программирования и эффективности получающихся программ занимает промежуточное положение между Visual Basic и Visual C++.

По мнению профессионалов наиболее легок и прост из упомянутых языков как раз Visual Basic. Профессиональные программисты любят его за то, что для создания на нем приличной программы требуется гораздо меньше времени, чем в других средах. Поэтому популярность у него огромная. Конечно, за все приходится платить, и у Visual Basic есть свои недостатки, но новичок почувствует их очень не скоро.

0.4. Какой вам нужен компьютер и какая версия Visual Basic?

Компьютер вам подойдет любой, на котором установлена операционная система Windows.

Любая программа, которую вы встретите в этой книге, является правильной программой в версии Visual Basic 6.0, а абсолютное большинство программ - и в более ранних версиях. Это естественно, так как основа языка едина для этих версий.

Теперь – о работе на компьютере. Каждая версия Visual Basic предлагает свой способ работы на компьютере (каждая следующая версия - все более удобный и мощный). В широком смысле этот способ называется **средой программирования** или **средой разработки программ**. Как я уже говорил, среды программирования версий Visual Basic 5.0 и Visual Basic 6.0 очень близки между собой в тех рамках, которыми я ограничиваюсь в книге. Все, что я буду говорить, относится к любой из них.

Последняя версия Visual Basic, а именно Visual Basic.NET, кардинально отличается от своих предшественниц. Она гораздо мощнее, однако, я бы сказал, менее легка для начинающих. Я рекомендую ее тем, кто на первых порах готов платить больше.

0.5. Краткое содержание с рекомендациями

Книга состоит из трех частей и двух приложений:

Часть I. Программирование без программирования.

У этой части две цели:

- Научить самым основным и элементарным приемам работы в Visual Basic.
- Показать, что в Visual Basic можно добиваться впечатляющих результатов практически безо всякого программирования.

Доказательством тому - проект "Калькулятор", который получится у вас гораздо ярче и забавнее, чем стандартный калькулятор Windows.

Часть II. Программирование на Visual Basic – первый уровень. Здесь начинается настоящее программирование. Цель этой части – провести вас от создания самых простых программ до сложных. Здесь вы учитесь программировать на Visual Basic самым удобным способом - на примерах, то есть по принципу “делай, как я”. Вы научитесь создавать небольшие программы, включающие циклы, ветвления, процедуры и использующие графику и звук. Заканчивается часть созданием двух интересных проектов и заданием на создание простейшей игры-стрелялки. Предполагается, что после выполнения этого задания у вас должно возникнуть ощущение всесильности, вы должны почувствовать, что теперь вам по плечу программа любого размера, а все, что вам может понадобиться в будущем, так это только дополнительные сведения о работе с теми или иными элементами Visual Basic.

Часть III. Программирование на Visual Basic – второй уровень. Цель этой части – снабдить вас этими самыми сведениями. Вы познакомитесь с действиями над массивами, коллекциями, строками, файлами, объектами и другими элементами Visual Basic. Вы изучите процедуры и функции с параметрами, модули, узнаете, что такое рекурсия и сортировка. Здесь же работа в Интернет, базы данных, создание собственных классов объектов.

Приложение 1. Это теоретический ликбез. Предназначен для тех, кто не знает, как устроен и работает компьютер.

Приложение 2. Это практический ликбез. Предназначен для тех, кто ни разу не садился за компьютер или садился только для игр. Здесь вы научитесь включать и выключать компьютер, работать с окнами, папками и файлами Windows, вводить в компьютер текст, то есть приобретете все нееобходимые навыки, чтобы спокойно начать работу в среде Visual Basic.

Не зная материала приложений, вы будете в Visual Basic спотыкаться на каждом шагу.

Затем в книге приводятся решения к заданиям и солидный предметный указатель.

От автора

Хочу выразить искреннюю признательность Николаю Геннадьевичу Волчёнкову, прочитавшему рукопись и высказавшему ряд важных замечаний по ее содержанию.

Также хочу поблагодарить Алексея Лукина за ценное обсуждение.

К читателю

Буду благодарен за критические замечания и буду рад услышать ваши отзывы о книге. Мой e-mail: lukin63@mail.ru.

Другие мои книги по программированию и по работе на компьютере вы можете найти на сайте www.learncomp.narod.ru.

Часть I. Программирование без программирования

Эта часть нужна для того, чтобы, прочтя ее, вы могли сказать себе: "Я освоился и могу делать в Visual Basic интересные вещи, не напрягаясь". То есть, поясню, почти без программирования. Почти, но не абсолютно. Простенький программный текст все-таки придется писать.

В 1 главе вы создадите свой первый проект на Visual Basic.

Во 2 главе вы создадите собственный калькулятор, снабдив его фотографией, музыкой и даже видео. Цель главы - распахнуть перед вами основные простые и одновременно яркие возможности Visual Basic.

Завершается часть главой 3 "Работа в среде Visual Basic", которая излагает основные приемы работы на компьютере при программировании на Visual Basic.

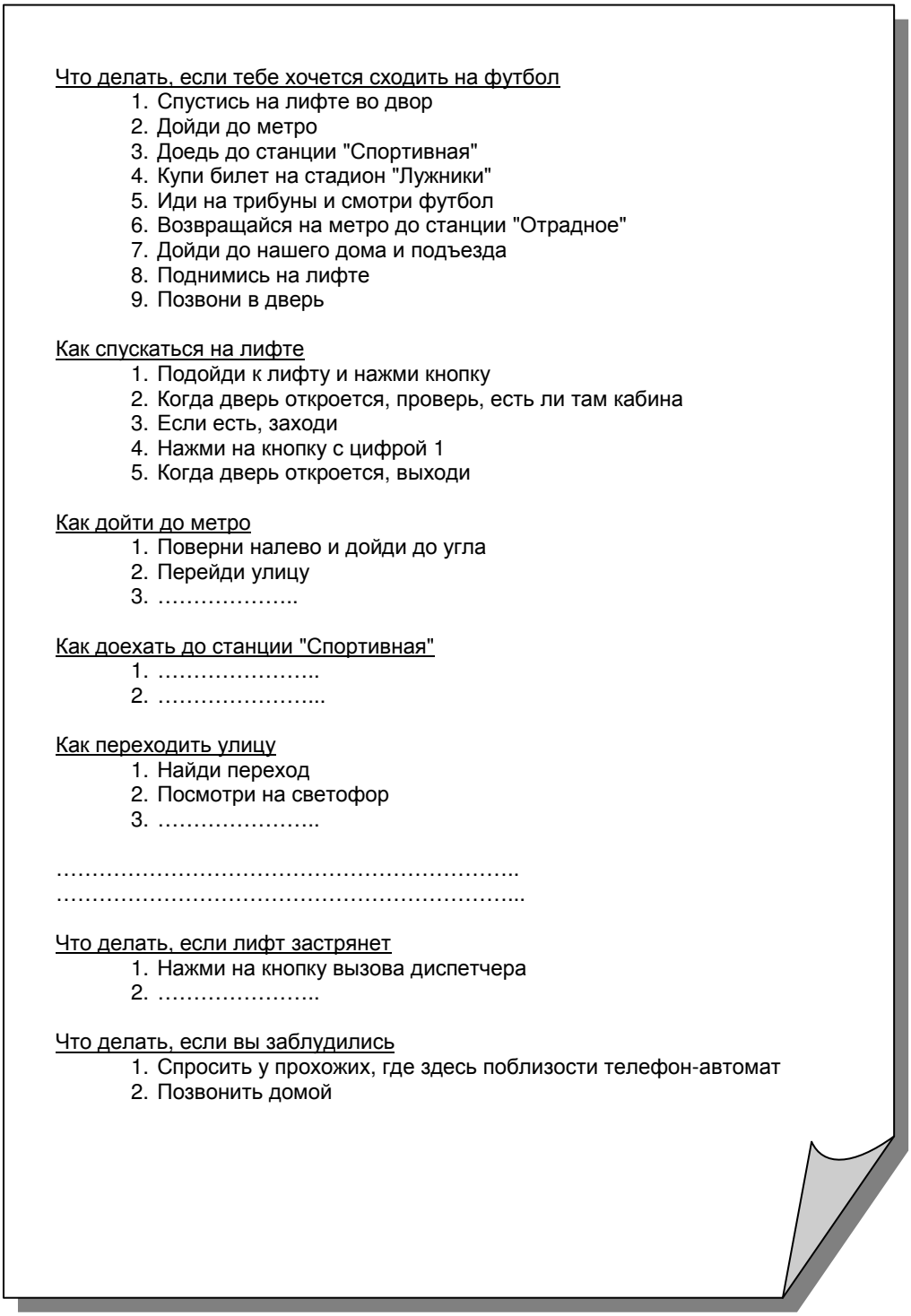
Глава 1. Первые шаги

В этой главе вы создадите и опробуете на компьютере свою самую первую программу на Visual Basic. Для этого глава и написана.

1.1. Что такое программа?

Прежде чем программировать на компьютере, надо бы узнать, что такое компьютер и программа. Подробно об этом рассказано в Приложении 1 и всех начинающих я настоятельно отправляю туда. Если же вы считаете, что переросли тот "детский" материал, то вам достаточно будет того, что сказано ниже.

Что такое программа с точки зрения Visual Basic, удобнее всего рассмотреть на аналогии. Представьте себе, что к вам, живущему в большом городе, в гости приехал ваш знакомый, никогда не выезжавший из своего поселка. Он хочет сходить на футбол, а вам идти вместе с ним некогда. Чтобы он смог добраться до стадиона и вернуться живым и здоровым, вы пишете ему на листе бумаги такую инструкцию.



Что делать, если тебе хочется сходить на футбол

1. Спустись на лифте во двор
2. Дойди до метро
3. Доедь до станции "Спортивная"
4. Купи билет на стадион "Лужники"
5. Иди на трибуны и смотри футбол
6. Возвращайся на метро до станции "Отрадное"
7. Дойди до нашего дома и подъезда
8. Поднимись на лифте
9. Позвони в дверь

Как спускаться на лифте

1. Подойди к лифту и нажми кнопку
2. Когда дверь откроется, проверь, есть ли там кабина
3. Если есть, заходи
4. Нажми на кнопку с цифрой 1
5. Когда дверь откроется, выходи

Как дойти до метро

1. Поверни налево и дойди до угла
2. Перейди улицу
3. ……..

Как доехать до станции "Спортивная"

1. ……..
2. ……...

Как переходить улицу

1. Найди переход
2. Посмотри на светофор
3. ……..

……..
……...

Что делать, если лифт застрянет

1. Нажми на кнопку вызова диспетчера
2. ……..

Что делать, если вы заблудились

1. Спросить у прохожих, где здесь поблизости телефон-автомат
2. Позвонить домой

Как видите, на листке - несколько инструкций. Они - двух типов. Одни начинаются со слов "Что делать, если …", другие - со слова "Как".

Самая верхняя главная инструкция состоит из 9 команд и предписывает строгий порядок действий для достижения цели. Инструкции, начинающиеся со слова "Как", описывают каждое из этих действий более подробно. Так, инструкция "Как дойти до метро" подробно описывает выполнение команды "Дойди до метро". Так как в этой инструкции встречается команда "Перейди улицу", которая сама нуждается в пояснении, то имеется инструкция "Как переходить улицу". И так далее.

Зачем я написал так много инструкций типа "Как"? Не проще ли было бы написать одну длинную главную инструкцию из "пары тыщ" команд, в которой бы задавались по порядку все мелкие действия от начала до конца похода, начиная с "Подойди к лифту и нажми кнопку" и кончая "Подойди к дверям нашей квартиры и позвони"? Проще, оно может быть и проще, да вот инструкция получится слишком длинной. Почему длинной? Потому что, например, переходить улицу надо будет раз восемь, и выходит, что в инструкции придется восемь раз писать одни и те же пояснения, как это делать. И еще потому не нужно писать длинную инструкцию, что человеку гораздо приятней и удобней воспринимать короткие инструкции, чем длинные.

Порядок выполнения упомянутых инструкций строго определен. Попробуйте нарушить его и вы увидите, что будет. Например, сначала попытайтесь пройти на трибуны, и только потом купить билет. Или сначала зайдите в дверь лифта и только потом проверьте, есть ли там кабина.

Однако, жизнь сложна, и в ней могут происходить *события*, которые трудно привязать к какому-то конкретному этапу выполнения инструкции. Например, вы можете заблудиться (причем, в любом месте, как по пути туда, так и по пути обратно), или ваш лифт может застрять. На этот случай пишутся инструкции, начинающиеся словами "Что делать, если …" и предписывающие, как реагировать на то или иное событие.

В программировании на Visual Basic все эти инструкции называются **процедурами**. Команды, из которых они составлены, называются **операторами**. Весь набор инструкций на листке назовем **программой**. А события так и будем называть **событиями**.

1.2. Не программа, а проект

Вообразите, что вы не программу на Visual Basic пишете, а собираете на столе игрушечную железную дорогу. В вашем распоряжении находятся **объекты**: стол, рельсы, паровозы, вагоны, светофоры, машинисты, стрелочники и т.д. Чтобы игрушка заработала, вам нужно выполнить 3 этапа:

1. Собрать ее вручную, то есть поместить на свои места и правильно соединить рельсы, вагоны и т.д.
2. Для лучшего понимания 2 этапа вообразим себе, что наша игрушка очень интеллектуальная, то есть поведение многих объектов (таких как машинисты, стрелочники, светофоры и другие) может быть сложным и осуществляться по программе. Тогда вторым этапом вашей работы будет - написать для каждого из таких объектов программу поведения. Так, в программе для машиниста одна из команд может быть такой: "Если впереди на светофоре красный свет - тормози", а в программе для светофора - такой - "Если два поезда опасно сблизились - зажигай красный свет".
3. Запустить игрушку и посмотреть, как она работает. Если произошло крушение или что-нибудь еще пошло не так, как надо, то значит одно из двух: или вы неправильно собрали игрушку или написали неправильные программы. Вы исправляете ошибки и снова запускаете игрушку. Снова смотрите, снова исправляете. И так до тех пор, пока все не заработает нормально. Этот процесс называется **отладкой**.

Современный программист, работающий в средах визуальной разработки программ, выполняет те же три этапа. На 1 этапе он вручную расставляет на экране компьютера объекты, которые будут выполнять его программу, на 2 этапе пишет программу, на 3 этапе запускает ее. Что конкретно я имею в виду под расстановкой объектов, вы узнаете в следующем разделе, а сейчас остается сказать, что из-за 1 этапа работа программиста теперь напоминает работу конструктора-проектировщика и частично из-за этого же продукт работы программиста называют теперь не программой, а **проектом**. Хотя термин этот еще не привился и чаще пользуются прежним. Я тоже, наверное, буду их путать. Правда, обещаю, что не в ущерб смыслу.

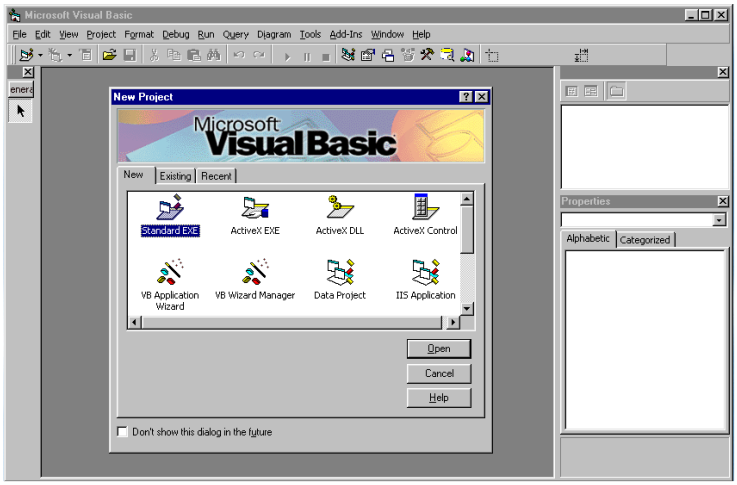
1.3. Первые шаги - за ручку

Лучший способ познакомиться с Visual Basic - поскорее создать и запустить какой-нибудь проект, пусть маленький и никому, кроме нас, не нужный, но зато с ним мы пройдем все этапы работы в Visual Basic. Этим мы сейчас и займемся. Садитесь за компьютер и приступим. Без компьютера этот раздел принесет вам мало пользы. Все, что я говорю, вы отныне должны немедленно выполнять на компьютере, чтобы проверить, правду я говорю или нет. Учиться Бэйсику без компьютера - все равно, что учиться кататься на велосипеде без велосипеда. Иногда я буду прямо говорить: "Нажмите такую-то клавишу" или "Сделайте на компьютере то-то и то-то". Но чаще я буду просто излагать материал, не повторяя по сто раз, что нужно проверять на компьютере каждое мое слово. Например, я говорю, что данная процедура по такой-то причине работает медленнее другой, и сразу же перехожу к другой теме. Стоп! Не надо спешить за мной. Сначала проверьте, так ли это на самом деле.

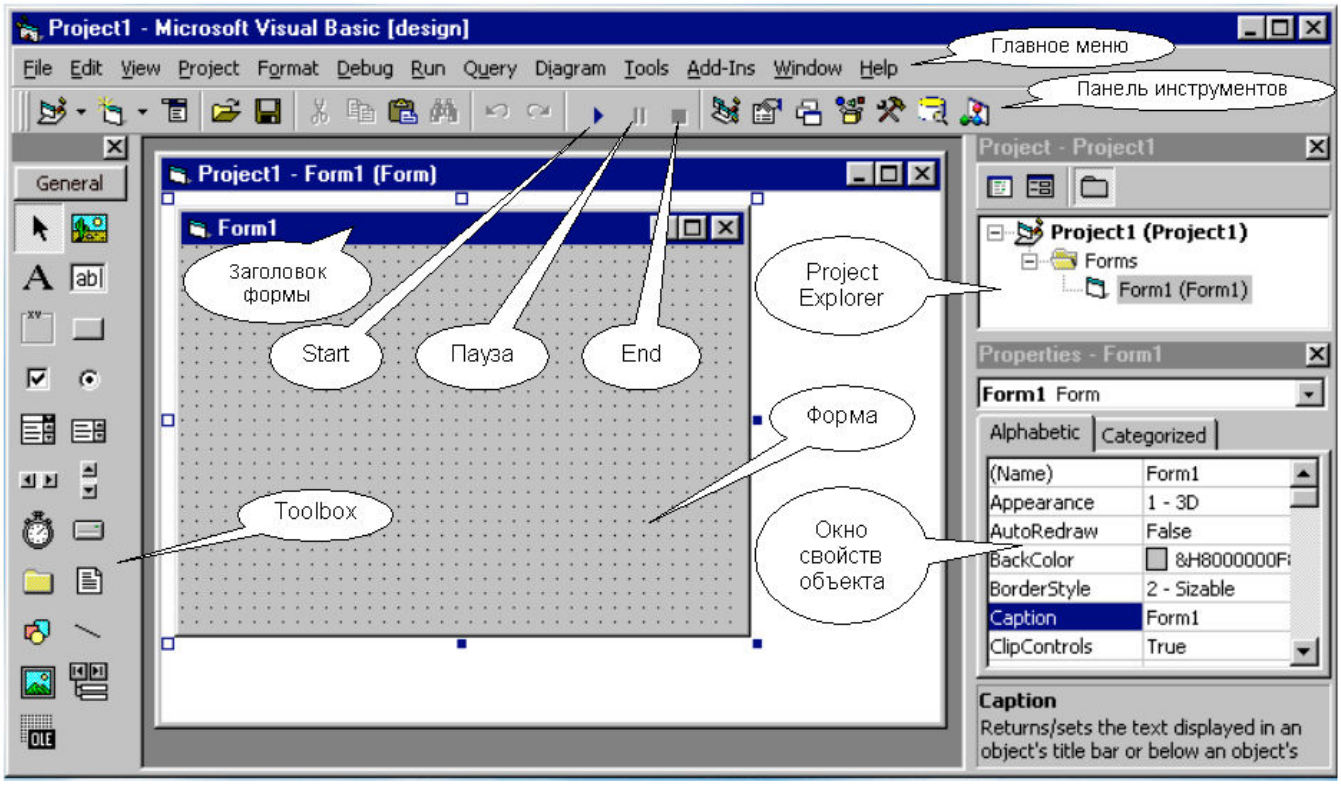
Запускаем Visual Basic

Начинается все, конечно, с того, что вы запускаете Visual Basic. Если Visual Basic у вас на компьютере не установлен, читайте 3.2, а если установлен, но вы не знаете, как его запускать, читайте 3.4. В результате запуска на экране появляется главное окно Visual Basic, в центре которого вы видите другое окно - приглашение начать создавать новый проект (см. рисунок). Если приглашение не появилось, то щелкните по пункту меню File, а затем выберите в выпавшем меню пункт Add Project - и оно появится.

(В дальнейшем вместо словесного описания щелчков мыши в меню я для краткости буду писать - **File→Add project**).



Вам предлагается выбрать тип проекта, который вы хотите создавать. Пока нам достаточно стандартного типа. Щелкните в окне приглашения в закладке New по значку Standard EXE, а затем по кнопке Open. Перед вами возникает следующая картинка, говорящая о том, что вы можете приступать к проектированию (внешний вид картинки у вас может быть немножко другой):



Что вас на картинке больше всего должно интересовать, так это серый прямоугольник в точечку, расположенный в цетре на белом фоне. Это так называемая **форма**. Что же это такое - форма? Вспомним нашу игрушечную железную дорогу. Так вот, форма - это пустой стол, тот самый стол, на котором мы эту дорогу будем собирать. Когда вы запустите проект, форма станет окном Windows.

Обратите внимание на три черных квадратика – маркера – в правой нижней части формы. Ухватившись за них мышкой, вы можете менять размеры формы. Но таскать форму по экрану вам здесь не удастся.

Начнем работу над проектом с 1 этапа. Чтобы собрать железную дорогу, нам нужны объекты: рельсы, вагоны и т.п. Чтобы создать проект, нам тоже нужны объекты - и Visual Basic нам их предоставляет. В левой части главного окна Visual Basic мы видим вертикальную серую панель. Это **Toolbox** (блок или палитра элементов управления), то есть набор наиболее популярных стандартных объектов, применяемых при создании проектов на Visual Basic. В их числе кнопка, текстовое поле, таймер, изображение и др. Чтобы познакомиться с названием какого-нибудь элемента, поместите на него мышиный курсор, не нажимая.

Если Toolbox вашего компьютера слишком узкий и на нем не помещаются все элементы, вы можете для его расширения перетащить мышкой его границу.

Замечание: Понятие "Объект" имеет в Visual Basic гораздо более широкий смысл, чем форма или элемент управления, но поскольку других объектов мы пока не знаем, я буду называть объектами именно их.

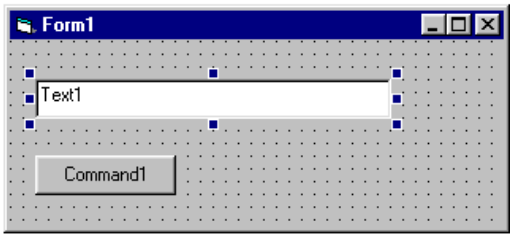
Размещаем объекты на форме

Ну что ж, начнем сборку. Возьмем **кнопку** (CommandButton) и поместим на форму. Для этого щелкнем по кнопке, а затем

проведем мышкой внутри формы небольшую "рамочку". Что значит провести "рамочку"? Это значит поместить мышиный курсор куда-нибудь внутрь формы, нажать на левую клавишу мыши и, не отпуская клавишу, "протащить" мышь немного наискосок. При этом за курсором будет тянуться "рамочка". Отпустите мышь. На месте рамочки возникнет кнопка с надписью Command1.

По краям кнопки вы увидите 8 черных квадратиков - **маркеров** изменения размеров. Если щелкнуть по форме мимо кнопки, они пропадут, если снова щелкнуть по кнопке - появятся. Только не надо пока двойных щелчков. Перетащите мышкой любой из маркеров - размеры кнопки изменятся. Можно таскать кнопку по форме, если ухватиться мышью не за маркер, а за любое место внутри кнопки. Чтобы уничтожить кнопку, щелкните по ней правой клавишей мыши и в выскочившем контекстном меню выберите опцию Delete. Проделайте все это. А теперь, если вы ее уничтожили, то снова создайте.

Аналогичным образом поместим на форму объект **текстовое поле** (TextBox). Теперь у нас на форме два объекта.



Обратите внимание на слово [design] (проектирование) в заголовке главного окна Visual Basic. Оно говорит о том, что в настоящий момент Visual Basic находится в **режиме проектирования**, во время которого мы можем собирать наш проект (1 этап) и писать для него программу (2 этап).

Пока мы с вами создаем сами не знаем что. Ведь никакой задачи мы перед собой не поставили. Тем не менее, перескочим через 2 этап (составление программы) и сразу выполним 3 этап, то есть запустим наш проект на выполнение. Для этого щелкнем мышкой по кнопке Start на **панели инструменов** (не путать с блоком элементов управления).

Исторический момент. Что мы видим? Форма немного изменилась, пропали точки, она потеряла связь с главным окном Visual Basic и ведет себя, как независимая программа. Для убедительности сверните на панель задач главное окно Visual Basic. Теперь форма одна царит на экране. Ее можно таскать по всему экрану, ухватившись за заголовок, и ее значок вы можете видеть на панели задач Windows. Она надлежащим образом реагирует на щелчки по кнопкам в ее правом верхнем углу. Если подвести острие курсора к краю или углу формы, то можно менять ее размеры. А вот размеры кнопки и текстового поля мы не можем больше менять. Зато теперь на кнопку можно нажимать мышкой. Нажимаем. Ничего не происходит. Естественно - мы ведь не написали программу, объясняющую компьютеру, что нужно делать при нажатии на кнопку.

В текстовое поле теперь можно вводить текст и как угодно там его менять. Но и это без толку. Компьютер никак не реагирует, что бы мы там ни написали. Причина та же.

Если вы при несвернутом главном окне Visual Basic случайно щелкнете по нему мышкой мимо формы, то форма пропадет из вида, так как будет загорожена главным окном. Не беда - щелчок по значку формы на панели задач Windows вернет ее на передний план.

Обратите внимание на слово [run] (бег, работа) в заголовке главного окна Visual Basic. Оно говорит о том, что в настоящий момент Visual Basic находится в **режиме работы**, то есть в режиме выполнения проекта (3 этап), во время которого мы ни собирать проект (1 этап), ни писать для него программу (2 этап) не можем.

Итак, мы видим, что без программы наш проект не стоит ни гроша. Завершим выполнение проекта кнопкой End на панели инструментов или щелкнув по крестику в правом верхнем углу формы. Visual Basic вышел из режима [run] и вернулся в режим [design]. Теперь в проекте можно что-нибудь поменять и снова его запустить. И так далее.

Пишем программу

Давайте придумаем себе задачу для нашего проекта. Пусть на экране при нажатии кнопки что-нибудь происходит. Легко сказать - пусть! Для этого нужно знать, что вообще умеет делать Visual Basic при нажатии на кнопки, а если даже мы это знаем, то все равно не знаем, как это приказать, то есть как правильно написать программу. Ну что ж. Программирование мы будем изучать постепенно и постепенно всему научимся, а сейчас я выберу что-нибудь самое простое и подскажу, как это делается. К компьютеру не подходите, пока я не скажу.

Прикажем компьютеру при нажатии кнопки поменять цвет формы на красный. Команда (оператор) для этого пишется так:

Form1**.**BackColor = vbRed

Разберемся, что здесь написано.

Form1	- это имя нашей формы (его дал Visual Basic и мы пока не будем его менять)
BackColor	- переводится "цвет фона", то есть это цвет, которым покрашена форма (пока он серый)
vbRed	- это красный цвет (Red - красный, vb - Visual Basic)

Таким образом, наш оператор можно перевести так:

Form1**.**цвет = красный

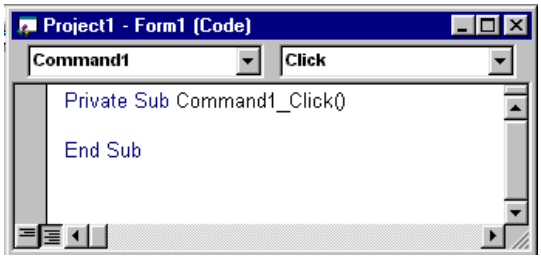
Правило такое: Слева от точки мы записываем имя объекта, справа от точки - его **свойство**, а справа от знака равенства - значение этого свойства. Точку писать обязательно. Аналогия -

Коля**.**рост = 140

Бразилия**.**климат = жаркий

Visual Basic воспримет наш оператор как приказ поменять цвет формы на красный. Теперь как нам сделать, чтобы компьютер выполнил этот оператор именно при нажатии на кнопку, а не в какой-либо другой момент? Садитесь за компьютер. Проверьте, находимся ли мы в режиме проектирования [design]. Сделайте двойной щелчок по кнопке Command1. Перед вами воз-

никнет новое окно - окно программного кода или просто **окно кода**. Чтобы мы не спутались, в заголовке окна присутствует слово (Code).



Будем называть **кодом** любой текст программы. Таким образом, в окно кода будем записывать всю программу нашего проекта (большие проекты могут использовать несколько окон кода).

Мы видим, что в окне кода уже что-то написано. Строку Private Sub Command1_Click() можно вольно перевести как "Что делать, если щелкнуть (по-английски - Click) мышкой по кнопке Command1". Слово Private переводится "локальный" и о его смысле поговорим позже. Слово Sub обозначает "процедура". Таким образом, более точно эту строку переводим так: "Локальная процедура, в которой компьютеру пишется инструкция, что делать, если человек мышкой нажмет на кнопку Command1".

Ниже оставлено свободное место и там уже мигает курсор, приглашая нас вводить туда какие нам угодно операторы. Они-то и будут выполнены при щелчке по кнопке. Еще ниже находится строка End Sub, что означает "конец процедуры".

Таким образом, мы видим перед собой не что иное, как заготовку процедуры, которая должна выполниться при щелчке по кнопке Command1.

Введите на свободное место с клавиатуры наш оператор Form1.BackColor = vbRed. Если у вас нет опыта ввода текста в компьютер, то прервитесь и изучите Приложение 2. Вы не имеете права допустить ни малейшей ошибки в строке. Даже лишний пробел может быть ошибкой.

Кстати, Visual Basic очень услужлив. Когда вы введете точку, он (чтобы вам не утомлять свои персты вводом слова BackColor) развернет перед вами список всех уместных после точки слов. Вам останется только сделать двойной щелчок на нужном слове.

Если вы случайно щелкнете мышкой мимо окна кода, оно может пропасть из вида. Не беда. Ваши действия: **View→Code**.

Если же окно кода загораживает от вас форму, а вам хочется что-нибудь в ней сделать, то ваши действия: **View→Object**.

Но вот ввод оператора закончен. Теперь окно кода должно иметь такой вид:

Перед вами готовая процедура. Проверьте, не ввели ли вы по ошибке что-нибудь выше, ниже или правее этих трех строк. Если ввели, то сотрите.

Пришла пора снова запускать проект. Предположим, вы при вводе кода в окно нигде не ошиблись. Щелкните кнопку Start. На экране опять появляется знакомый вид формы с кнопкой и текстовым полем. Щелкните по кнопке - форма стала красной. Поздравляю! Ваш первый проект заработал. Щелкните еще раз. Ничего не изменилось. Естественно. Завершим выполнение проекта кнопкой End.

Если у вас при запуске получилось не то, что нужно, прочтите чуть ниже параграф "Как реагировать на сообщения Visual Basic об ошибках".

Сохранение, создание, открытие, закрытие проекта

А теперь нам пора сохранить проект. Вообще-то, сохраниться нам надо было еще давно, до первого запуска, но не хотелось отвлекаться. О том, зачем и как сохранять и загружать проект в Visual Basic, читайте в 3.5. Если же у вас есть опыт сохранения при работе в Windows, то можете сразу сохраняться при помощи **File → Save Project**, только помните, что проект сохраняется не в один, а в несколько файлов, поэтому обязательно для каждого проекта создавайте свою папку. Visual Basic предложит вам стандартные имена для сохраняемых файлов. Пока не меняйте их.

Сохранившись, вы можете спокойно выйти из Visual Basic, щелкнув по крестику в правом верхнем углу главного окна, и можете даже выключить компьютер. Чтобы продолжить через некоторое время работу над проектом, запустите Visual Basic и загрузите (откройте) сохраненный проект при помощи **File → Open Project**.

Если вы хотите начать новый проект, то выполните **File → New Project**.

Через некоторое время у вас накопится несколько сохраненных проектов. Вот вы поработали с одним из них и хотите, не выходя из Visual Basic, поработать с другим. Пока у вас еще мало опыта, я рекомендую такой порядок действий:

- сохраните проект
- удалите его из главного окна Visual Basic (не с диска, конечно) - **File → Remove Project**. Теперь в главном окне Visual Basic и в окне Project Explorer у вас должно быть пусто. Заголовок Project Explorer должен быть таким: Project - No Open Projects.
- Откройте другой проект при помощи **File → Open Project** или создайте новый при помощи **File → New Project**.

Если вы попытаетесь открыть или создать другой проект, не сохранив старый, Visual Basic предложит вам его сохранить и после сохранения автоматически удалит из среды. Если вы по ошибке вместо **File → Open Project** или **File → New Project** выполните **File → Add project**, то старый проект не будет удален и в главном окне Visual Basic вы будете иметь вместо одного

сразу два проекта. Вы это сразу же заметите, взглянув в окно Project Explorer. Иногда иметь два проекта удобно, иногда просто необходимо, но пока вам будет не повернуться в такой тесноте. Если это все же произошло, жмите **File → Remove Project**, отвечая отказом на предложения сохраниться, пока главное окно Visual Basic не опустеет.

Часто новый проект, который мы создаем, бывает похож на старый, и удобнее не новый создавать, а старый переделывать. Мой опыт таков. Я копирую папку старого проекта. Затем открываю копию проекта и переделываю ее в новый проект. Как копировать папки, написано в 3.5.

Как реагировать на сообщения Visual Basic об ошибках

Если вы при вводе кода в окно где-то ошиблись, то Visual Basic при запуске или сделает что-нибудь не то (например, покрасит форму в черный цвет) или даст вам знать об ошибке специальным сообщением. Сообщения могут возникать как в режиме проектирования, так и в режиме работы. Сейчас имеет смысл потренироваться. Давайте будем специально допускать ошибки и смотреть, что будет:

*Form1.BackColor = vbRed**d***

Visual Basic вообще не заметит этой ошибки и покрасит форму не в тот цвет. Завершите работу проекта и исправьте ошибку.

*Private Sub Command1_Clic**t**k()*

И этой ошибки Visual Basic не заметит. Просто при нажатии на кнопку ничего красится не будет.

*Form1.Back**Colo** = vbRed*

Visual Basic не заметит этой ошибки до тех пор, пока вы в режиме работы не нажмете на кнопку. Тогда он выдаст сообщение об ошибке, пока для вас, впрочем, невразумительное, и выделит подозрительное место в программе. Вы можете нажать Help и тогда Visual Basic подробнее пояснит, что он имел в виду. Но пока смысл пояснения вряд ли до вас дойдет. Нажимайте OK и Visual Basic выделит желтым заголовок подозрительной процедуры. Завершите работу проекта и исправьте ошибку.

Foorm1.BackColor = vbRed

И этой ошибки Visual Basic не заметит до тех пор, пока вы в режиме работы не нажмете на кнопку. В окне сообщения об ошибке нажмите Debug, тогда Visual Basic выделит подозрительную строчку в программе. Завершите работу проекта и исправьте ошибку.

*Form1.BackColor = **=** vbRed*

А вот эту ошибку Visual Basic заметит еще в режиме проектирования, как только вы попытаетесь убрать курсор из ошибочной строки в другое место. Подозрительное место будет выделено. Жмите OK и исправляйте ошибку.

Кстати, когда вы убираете курсор из вводимой строки, Бэйсику кажется, что вы закончили с ней работу. Он проверяет ее на наличие грамматических ошибок и если не находит, то немного изменяет ее внешний вид, приведя к стандартному.

Visual Basic не всегда правильно находит подозрительные места. Часто программисту самому приходится превращаться в детектива. Я не буду здесь останавливаться на сложном процессе поиска ошибок, так как вы еще к этому не готовы. Достаточно будет, если вы еще раз посмотрите, так ли выглядит ваш код, как на рисунке. Нет ли чего лишнего, в том числе ниже строки End Sub? Все ли буквы английские, или есть русские? Человеку очень легко спутать английские буквы с русскими того же начертания, компьютер же не спутает никогда и придерется.

Усложняем проект

Итак, вы сохранили свой первый проект. Закройте его - **File→Remove Project**. Скопируйте папку проекта тут же, рядышком. Над копией будем продолжать работать.

Пусть при щелчке по кнопке происходит что-нибудь еще, кроме окрашивания формы. Например, изменяется размер шрифта в текстовом поле. Для этого достаточно в окне кода дописать следующий оператор:

Text1.FontSize = 16

Здесь Text1 - имя текстового поля, FontSize - размер шрифта. Наш оператор можно перевести так: Сделать размер шрифта текстового поля Text1 равным 16. Теперь содержимое окна кода таково:

```
Private Sub Command1_Click()
    Form1.BackColor = vbRed
    Text1.FontSize = 16
End Sub
```

При щелчке по кнопке Visual Basic выполняет по порядку все операторы процедуры между строками Private Sub Command1_Click() и End Sub. Запустим проект кнопкой Start. Убедимся, что оба оператора выполнились. Поскольку компьютер работает очень быстро, нам будет казаться, что оба оператора выполнились одновременно. Но это не так: сначала поменялся цвет формы, а уж затем размер шрифта.

Продолжим работу. Пусть при нажатии на кнопку кроме всего прочего в текстовом поле появляется текст "Теперь форма красная". Для этого достаточно в окне кода дописать еще один оператор:

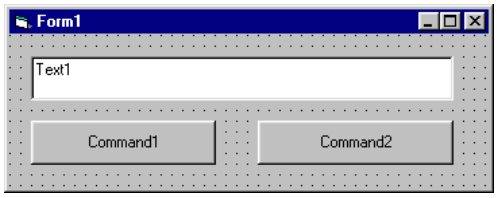
Text1.Text = "Теперь форма красная"

Здесь Text1 - имя текстового поля, Text – его свойство, а именно – текстовое содержимое. Привыкайте к почти одинаковым обозначениям разных вещей (у нас это Text1 и Text) и будьте внимательны. Наш оператор можно перевести так: Содержимое текстового поля Text1 сделать таким - "Теперь форма красная". Вид окна кода стал таким:

```
Private Sub Command1_Click()
    Form1.BackColor = vbRed
    Text1.FontSize = 16
    Text1.Text = "Теперь форма красная"
End Sub
```

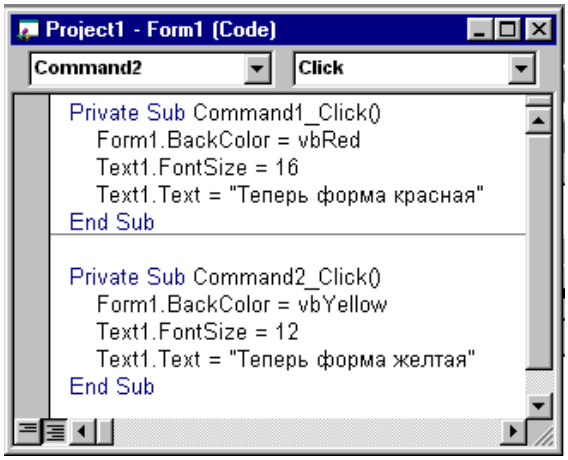
После каждого изменения в окне кода проверяйте, как работает проект.

Продолжим. Создадим на форме еще одну кнопку:



Ее имя - Command2. Пусть при щелчке по ней форма окрашивается в желтый цвет (vbYellow), размер шрифта становится равным 12, в текстовом поле появляется текст "Теперь форма желтая"

Для этого так же, как и в случае с кнопкой Command1, сделаем по кнопке Command2 двойной щелчок мышкой. Перед нами - снова окно кода, но в нем появилась заготовка другой процедуры, то есть новое приглашение - для ввода операторов, реагирующих на щелчок кнопки Command2. Введем их. Теперь содержимое окна кода таково:



Запустите проект. Пощелкайте по кнопкам.

Давайте проведем аналогию между нашей программой и программой, которую мы дали любителю футбола в 1.1.

- Та программа состояла из нескольких процедур, эта также состоит из двух процедур.
- Та программа описывала поведение человека в городе, эта описывает поведение объектов на форме Form1.
- В той программе процедура состояла из команд, записанных одна за другой и выполнявшихся в порядке записи. В этой программе - то же самое: процедура состоит из операторов, записанных один за другим и выполняющихся в порядке записи.
- В той программе встречались процедуры разных типов ("Как…..", "Что делать, если…"). В этой программе обе процедуры одного типа - "Что делать, если нажата кнопка".
- В той программе события - это "Застрял лифт", "Заблудился", в этой события - это нажатия на кнопки.

Что дальше?

Итак, проект готов и работает. Что дальше? Пока не очень понятно, как нам запрограммировать что-нибудь посложнее, например, игру с перестрелкой из предисловия. Могу пояснить. Вы растягиваете форму во весь экран и придаете ей не цвет, а фотографию или нарисованный вами рисунок города. Получается город во весь экран. Далее берете в Toolbox и расставляете по форме объекты типа Image (изображение) - это ваши будущие автомобили, прохожие, гангстеры, пули и т.п. Затем придаете каждому объекту нужную внешность. Для этого тоже разыскиваете где-то фотографии или сами рисуете. Наконец, пишете для каждого из этих объектов программу поведения, включая реакцию на нажатия клавиш и мышиные щелчки. Игра готова, можно запускать. Основная трудность здесь, конечно, в написании программы, она будет достаточно сложной и вам предстоит еще многому научиться, чтобы почувствовать себя в силах ее создать.

В Visual Basic много любопытных и полезных элементов управления. Так, вставив в форму объект типа Timer, вы сможете управлять работой проекта "по секундам". Многие элементы управления на Toolbox не показаны. Но их легко туда поместить и пользоваться ими. Так, вставив в форму объект типа Microsoft Multimedia Control 6.0, вы сможете сопровождать игру музыкой, звуковыми эффектами и видео. Следующая глава познакомит вас со многими полезными и приятными возможностями Visual Basic.

Глава 2. Проект - "Калькулятор"

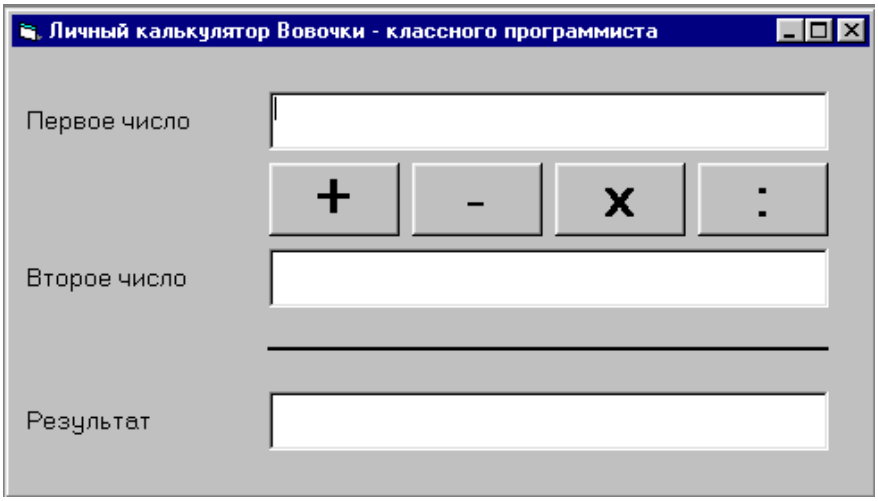
В этой главе на примере создания калькулятора я хочу познакомить вас с простыми и приятными возможностями Visual Basic. По сути, калькулятор будет нам проводником по элементарным средствам Visual Basic. Многие из них вы найдете именно в этой главе. По ходу дела нам удастся углубить и расширить наши знания Visual Basic. В то же время глава носит несколько рекламный оттенок. Перед вами распахнется скатерть-самобранка, на которой вы увидите в основном то, что повкуснее и не требует разгрызания. Однако, среди вкусного будет и полезное, и необходимое, поэтому читать главу "по диагонали" никак нельзя, без нее не будет понятно все остальное.

2.1. Задание на проект

В Глава 1 мы с вами ничего путного не создали, зато прошли с начала до конца весь путь создания проекта на Visual Basic.

Сейчас мы поставим перед собой задачу сделать реальную вещь - калькулятор. Примерно такой же, какой имеется в Windows. Если вы его забыли, то в Windows (а отнюдь не в Visual Basic), выполните такие действия: **Пуск → Программы → Стандартные → Калькулятор**. Как по вашему - много сил затратила фирма Microsoft на его создание? Калькулятор, который создадим мы, в смысле математики будет попроще, зато он будет красивым, снабжен музыкой, паролем и разными другими "приколами".

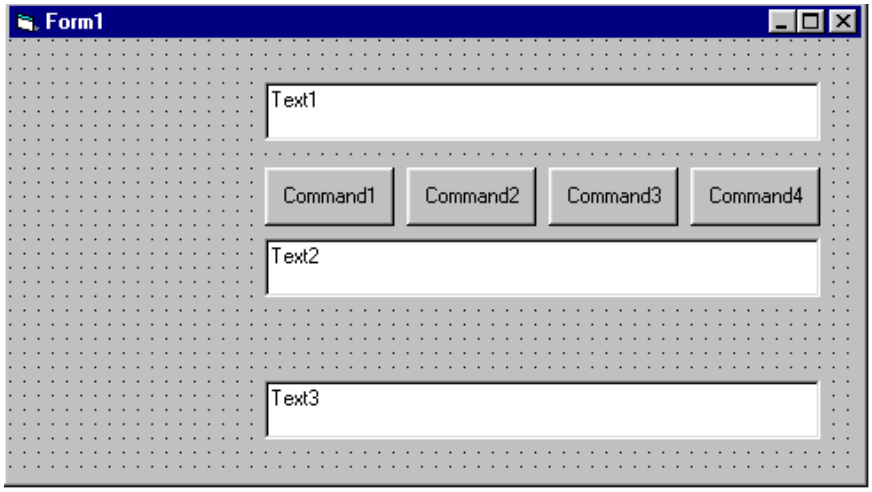
Начнем с того, что придумаем нашему калькулятору внешний вид (пока скромный):



Идея такая - вы набираете в двух верхних текстовых полях первое и второе число, затем щелкаете по одной из кнопок и в нижнем текстовом поле получаете результат.

2.2. Проектируем

Создайте новый проект и разместите на форме три текстовых поля и четыре кнопки. У вас получится вот так:



Чтобы продолжить дальше, нам нужно поближе познакомиться со свойствами форм и элементов управления.

2.3. Свойства форм и элементов управления

Чуть раньше мы уже познакомились с такими свойствами объектов, как BackColor, FontSize, Text. Количество свойств у каждого из объектов довольно большое. Многие из них мы можем менять в режиме работы [run] при помощи программы, как мы это уже делали (например, Form1.BackColor = vbRed). Оказывается, многие свойства форм и элементов управления можно менять и в режиме проектирования [design]. Делается это вручную, безо всякого программирования (для того и задумано, что вручную легче!).

Вы сейчас в режиме [design]? Хорошо. Один щелчок по кнопке Command1. При этом на кнопке появляются маркеры. Говорят, что объект **выделен**, стал **активным**. В этом случае его свойства вы можете видеть в **окне свойств объекта** (см. рисунок в 1.3). Если это окно кажется вам маловатым, попробуйте растянуть его, ухватившись за края. В окне два столбца: название свойства и его значение. Для вашего удобства свойства могут быть упорядочены по алфавиту (закладка alphabetic) или сгруппированы по категориям (закладка categorized).

Попробуем изменить значение какого-нибудь свойства кнопки Command1. Найдите в окне свойство Width (ширина), а справа - его численное значение. Введите вместо имеющегося там числа другое число, например, 200. Вы видите, что ширина кнопки изменилась. Точно такого же результата вы добились бы в режиме работы [run], выполнив оператор Command1.Width = 200. А теперь поступим наоборот - перетащим немного влево или вправо один из маркеров кнопки. Мы увидим, что соответственно изменилось и число.

Выделим щелчком какой-нибудь другой объект. Теперь в окне свойств - его свойства. Выделим форму, щелкнув по любому свободному месту ее поверхности - теперь мы видим и можем менять ее свойства. И так далее.

С названиями и смыслом разных свойств я буду знакомить вас постепенно.

2.4. Имена и надписи

У каждого объекта есть **имя** (Name). У многих есть **надпись** (Caption). Имя и надпись являются свойствами объекта. Попробуем их изменить и разобраться, зачем они нужны. В предыдущем проекте мы не заботились об именах и надписях и хорошо делали, потому что Visual Basic в случае нашей беспечности (как говорится - **по умолчанию**) сам придает значения свойствам, в том числе именам и надписям, причем, недолго думая, имя и надпись он делает одинаковыми. Проверим. Выделим форму. Заглянем в окно свойств и найдем там имя (Name) - оно в алфавите выше всех. Так и есть, имя нашей формы - Form1. Изменим его, скажем, на слово Калькулятор. Изменили. И что же? В заголовке формы ничего не изменилось. Дело, конечно, в том, что в заголовке мы видим не имя, а надпись. Имя остается скрытым (полковник Исаев), а надпись видна всем (Штирлиц). Убедимся в этом. Найдем в окне свойств свойство надпись (Caption). Ну конечно - надпись на нашей форме - тоже Form1. Изменим ее на Личный калькулятор Вовочки - классного программиста. Теперь все в порядке.

Зачем мы изменили надпись, нам понятно. А почему надо было менять имя? Ну, хотя бы потому, что оператор

Калькулятор.Width=6000

как-то понятнее, чем

Form1.Width=6000

Займемся кнопками. Выделим кнопку Command1 и дадим ей имя Кл_сложения.

<u>Внимание!</u> Имя должно состоять только из букв, цифр и знаков подчеркивания, причем начинаться имя должно с буквы. Раз в именах запрещены пробелы, я использовал вместо них знак подчеркивания.

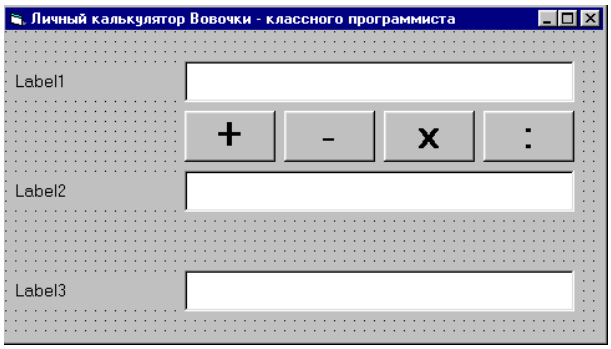
Подробнее об именах см. в 4.3.

Надпись нашей кнопки должна состоять из единственного символа **+**. Найдем **+** на клавиатуре и введем. На кнопке он получился маловат. Чтобы его увеличить, найдем в окне свойств свойство Font и щелкнем по нему. В поле значения появилась кнопочка с троеточием - это всегда приглашение к дальнейшему разговору. Щелкнем по ней - перед нами появилось так называемое **диалоговое окно**, предлагающее выбрать размер шрифта (Size), стиль (Font Style), сам шрифт (Font) и кое-что другое. Стиль может быть обычным (Regular), *курсивом* (Italic), **полужирным** (Bold) и ***полужирным курсивом*** (Bold Italic). Среди самих шрифтов (Font) попадаются любопытные. Выберите размер и прочее по вкусу. С клавишей сложения закончили.

Аналогичным образом придайте имена и надписи трем другим кнопкам. Имена: Кл_вычитания, Кл_умножения, Кл_деления. В качестве надписей для клавиш умножения и деления можете выбрать букву X и двоеточие.

Теперь займемся текстовыми полями. Придумаем и дадим им имена: Число1, Число2, Результат. А вот свойства Надпись (Caption) у текстовых полей нет. Вместо него есть свойство Text. Поскольку мы хотим, чтобы в начале работы с калькулятором в текстовых полях было пусто, сотрем в окне свойств значения свойства Text для всех трех полей.

А теперь займемся пояснительным текстом в левой части калькулятора (Первое число, Второе число, Результат). Для этого нам понадобится новый элемент управления - Label (метка), который в основном для пояснений и применяется. Найдите Label в окне Toolbox и поместите на форму три метки. Ваш проект выглядит так:



Заглянем в свойства меток. Мы видим, что Visual Basic по обыкновению дал каждой метке совпадающие имя и надпись (Label1, Label2, Label3). Имена менять мы не станем затрудняться, потому что в программе они никак участвовать не будут, а надписи, конечно, переменим на Первое число, Второе число, Результат, да и шрифт, пожалуй, увеличим.

Нам остается поместить на форму горизонтальную линию. Для этого найдем в окне Toolbox элемент управления Line (линия) и проведем линию на форме. Чтобы она была потолще, изменим в окне свойств свойство линии BorderWidth.

Итак, проектирование первой версии калькулятора закончено! Теперь калькулятор выглядит так, как задано. Можно приступать к программированию.

2.5. Программируем. Проект готов

Запустим проект. Введем в верхнее текстовое поле число 3, а в среднее введем 2. Щелкнем по кнопке сложения. Ничего не произошло. Естественно. Ведь никакой процедуры мы для кнопки не написали. Завершим работу проекта. Двойным щелчком по кнопке сложения откроем окно кода и попытаемся выдумать, что же там написать. Рассуждать будем так: 3 и 2 - не что иное, как значения свойства Text текстовых полей Число1 и Число2. По-другому, это Число1.Text и Число2.Text. Нам нужно, чтобы Результат.Text равнялся их сумме. Что если написать такой оператор:

Результат.Text = Число1.Text + Число2.Text

Сказано - сделано. Получаем:

```
Private Sub Кл_сложения_Click()
   Результат.Text = Число1.Text + Число2.Text
End Sub
```

Запускаем проект. Вводим 3 и 2. Щелкаем по кнопке сложения. Результат есть. Но не совсем тот, что мы ожидали. Вместо 5 получилось 32. В чем причина? Дело в том, что Visual Basic привык считать знак **+** по отношению к содержимому текстовых полей не знаком сложения, а знаком "соединения". Проверьте. Вместо 3 и 2 введите Коро и бочка, в результате получится Коробочка. Недаром текстовые поля называются текстовыми, а не числовыми. То, что мы назвали их Число1 и Число2, делу никак не помогло, потому что Visual Basic не обращает внимания на смысл имен, для него имена - просто бессмысленные сочетания символов.

Что делать? Надо приказать Visual Basic обращаться в данном случае с содержимым текстовых полей не как с текстом, а как с числами. Для этого достаточно записывать их не в таком виде - Число1.Text и Число2.Text, а в таком - Val(Число1.Text) и Val(Число2.Text). Здесь **Val** - сокращение от Value - величина, численное значение. Теперь наш оператор будет выглядеть так:

Результат.Text = Val(Число1.Text) + Val(Число2.Text)

Кстати, после ввода открывающей скобки Visual Basic услужливо предложит подсказку на тему о том, что должно быть в скобках. Со временем вы научитесь эту подсказку понимать.

Запускаем проект. Вводим два любых целых числа и убеждаемся, что все складывается правильно.

Аналогично программируем три остальные кнопки. Вот как будет выглядеть после этого окно кода:

```
Private Sub Кл_сложения_Click()
```

```
    Результат.Text = Val(Число1.Text) + Val(Число2.Text)
End Sub
_____________________________________________
Private Sub Кл_вычитания_Click()
    Результат.Text = Val(Число1.Text) - Val(Число2.Text)
End Sub
_____________________________________________
Private Sub Кл_умножения_Click()
    Результат.Text = Val(Число1.Text) * Val(Число2.Text)
End Sub
_____________________________________________
Private Sub Кл_деления_Click()
    Результат.Text = Val(Число1.Text) / Val(Число2.Text)
End Sub
```

Пояснения: В языках программирования умножение обозначается звездочкой *, а деление - косой чертой /.

При вводе в текстовые поля десятичных дробей вместо запятой ставьте точку. Результат же будет выводиться с запятой.

Итак, калькулятор готов!

Предостережения: Наш калькулятор пока не защищен от ввода вместо чисел всякой ерунды (например, текста КУ-КУ), от ввода слишком больших или слишком маленьких чисел, от деления на ноль. В таких случаях Visual Basic дает неправильный или неудобочитаемый результат или сообщение об ошибке. Защиту вы найдете в 5.9.

Задание 1 Создайте кнопку возведения в квадрат числа из верхнего текстового поля. Указание: Возвести в квадрат - значит умножить само на себя.

Задание 2 На нашем калькуляторе не хватает кнопки СБРОС, которая опустошала бы все три текстовых поля. Создайте ее. Указание: Для этого вам понадобятся операторы типа Число1.Text = "". В программах текстовое содержимое текстовых полей должно указываться в кавычках. В данном случае у нас внутри кавычек пусто, что и требовалось.

2.6. Кое-какие другие свойства объектов

Начнем в режиме проектирования украшать и усовершенствовать наш калькулятор. Для этого рассмотрим некоторые другие свойства объектов, из которых он сконструирован. Многие из этих свойств имеются у большинства объектов, некоторые - только у одного-двух. Прочтя материал об очередном свойстве, вам надлежит тут же проверить, как его различные значения влияют на вид и поведение формы, кнопок, текстовых полей и меток в режиме [run].

Полезный совет: У объектов очень много не объясненных мной свойств. Природная любознательность толкнет вас "поперед батьки" разобраться, что они значат, придавая им для этого наугад какие-нибудь значения. Нет проблем - ломайте голову на здоровье. Проблема же в том, что кое-кто норовит *сохранить* проект с измененными значениями неведомых свойств. А потом проект начинает себя вести как-то непонятно. Мораль - "Верни, как было!"

BackColor (цвет объекта) - знакомое свойство. Щелкнем по нему. В поле значения появилась кнопочка с черной треугольной стрелкой. Щелкнем по ней - перед нами появилось окно с двумя закладками. Выберем Palette (палитра) и понравившийся цвет.

ForeColor (цвет текста и линий, рисуемых на объекте).

Appearance (внешний вид) - 3D (трехмерный) или Flat (плоский).

BorderStyle (стиль границы). Здесь 6 вариантов, и заведуют они не только стилем границы, но и количеством кнопок на форме, и возможностью изменять размеры формы в режиме [run].

ToolTipText (всплывающая подсказка). Вы, возможно, привыкли к тому, что когда во время работы в приложениях Windows, таких как Word, вы помещаете курсор мыши на значок какого-нибудь инструмента, то рядом со значком появляется подсказка, поясняющая, зачем этот инструмент нужен.

Введите любой текст в качестве значения свойства ToolTipText. Запустите проект. Поместите курсор мыши на объект. Текст всплыл. Очень приятное свойство.

MouseIcon, **MousePointer** (значок мыши, указатель мыши). Эти свойства позволяют вам менять внешний вид мышиного курсора. Выберите кнопку и задайте для ее свойства MousePointer значение 2 (крест). Запустите проект. Поместите курсор мыши на объект. Курсор приобрел вид креста.

Если вы зададите вид мышиного курсора для объекта Форма, то он будет иметь заданный вид над всей поверхностью формы и теми объектами на форме, для которых он еще не изменен.

Всего у свойства MousePointer 16 скромных значений. MouseIcon позволяет вам задать более живописный значок для курсора. Для этого предварительно установите значение MousePointer в 99. Затем щелкните по троеточию в MouseIcon и в открывшемся окне проводника найдите значки мышиного курсора в папке Windows\Cursors или в папках Cursors и Icons, расположенных в папке Graphics, которая сама расположена в папке, куда установлен ваш Visual Basic.

Поиграли со значками мышиного курсора? А теперь верните все, как было. Не стоит без дела отвыкать от стандартного интерфейса - это рассеивает внимание. Экзотика хороша в экзотических случаях.

Следующие четыре свойства применимы только к форме.

MaxButton (кнопка максимизации - квадратик в правом верхнем углу формы). Сейчас значение этого свойства - True (Правда). Если установить его в False (Ложь), то квадратик пропадет или станет недоступен.

MinButton (кнопка минимизации - горизонтальная черточка в правом верхнем углу формы). Сейчас значение этого свойства - True (Правда). Если установить его в False (Ложь), то черточка пропадет или станет недоступна.

Moveable (можно ли двигать). Установите это свойство в False и вы увидите, что в режиме [run] форму нельзя таскать по экрану за заголовок. Хотя можно перетаскивать ее границы.

WindowState определяет, в каком виде возникает форма при запуске проекта: 0 - в нормальном, 1 - в минимизированном (то есть вы найдете ее на панели задач Windows) или 2 - максимизированном (во весь экран).

Visible (видимый). Обычно значение этого свойства - True (Правда). Если установить его в False (Ложь), то элемент управления перестанет быть виден в режиме работы. Но будет слушаться программу.

Enabled (в рабочем состоянии). Обычно значение этого свойства - True. Если установить его в False, то элемент управления виден будет, но он не будет работать и им нельзя будет пользоваться. Обычно он приобретает при этом бледно-серый оттенок.

Alignment. В зависимости от значения этого свойства текст в текстовом окне или метке будет располагаться вплотную к левому краю элемента, правому краю или по центру.

MultiLine. Если установить это свойство текстового поля в True, то в текстовое поле вы сможете вводить не одну, а много строк. А если вы измените значение его свойства **ScrollBars**, то снабдите текстовое поле одной или двумя полосами прокрутки.

2.7. Помещаем фото на калькулятор

Мы хотим, чтобы поверхность формы была покрыта какой-нибудь фотографией. Для этого необходимо, чтобы файл с этой фотографией уже хранился на жестком диске вашего компьютера. Если фото нет, то сойдет и рисунок, созданный вами в каком-нибудь графическом редакторе, например, в Paint.

Выделите форму. Найдите свойство **Picture**. Затем щелкните по троеточию. В открывшемся окне проводника доберитесь до нужного вам графического файла. Щелкните по Open - в результате фото окажется на форме. Можно использовать и другие типы графических изображений (см.9.3). Если у вас установлен Microsoft Office, то вы можете в его папке найти папку Clipart и там поискать файлы векторной графики.

Если вам не хочется занимать под фото все пространство формы, вы можете поместить на форму один из двух новых для вас элементов управления - **PictureBox** или **Image** - и аналогичным образом поместить изображение в них.

Графику можно поместить и на кнопку (предварительно установив свойство Style кнопки в Graphical).

2.8. Музыка в проекте

Если у вашего компьютера есть звуковая карта, то вы можете добавлять в ваш проект самые разнообразные музыкальные и звуковые эффекты.

Сделаем так, чтобы при запуске проекта раздавалась музыка (а если вы умеете с микрофона записывать свой голос в файл - то и ваш голос). Звуковой файл с музыкой или вашим голосом уже должен находиться на жестком диске или на компакт-диске. Вы можете прекрасно прослушивать файлы в форматах MID, WAV, а в большинстве случаев и MP3 и некоторых других. Кстати, в Интернете и на компьютерных компакт-дисках подавляющая часть музыки имеет формат MP3. Этот формат становится все более популярным, так как позволяет уместить на диск в десять раз больше музыки, чем, скажем, WAV. Если вы не знаете ни одного звукового файла, то могу подсказать, что несколько десятков их хранятся в вашем компьютере по адресу C:\Windows\Media. Пусть мы хотим воспроизвести один из них - файл Canyon.mid.

Проверьте настройки вашей звуковой карты в программе «Микшер». Для этого в среде Windows нажмите **Пуск → Программы → Стандартные → Развлечения → Регулятор громкости**. В возникшем окне снимите флажки (галочки) выключения канала и установите максимальный уровень у каналов Wave (для WAV-файлов и MP3-файлов) и MIDI (для MID-файлов и RMI-файлов). Для работы с микрофоном зайдите в «Свойства», затем в закладку «Запись», установите флажок включения канала и установите максимальный уровень у канала «Микрофон».

Чтобы воспользоваться звуковым файлом, вам нужно расположить на форме элемент управления Microsoft Multimedia Control 6.0. Но его нет в стандартном наборе на панели Toolbox. Сначала его нужно туда поместить. Для этого: **Projects → Components →** вы находите его в длинном списке и ставите против него галочку**→OK**. Он появился в Toolbox. Теперь им можно пользоваться обычным образом.

Поместим его на форму. Дадим ему имя (например, Плеер). Вы видите, что у него есть клавиши. Чтобы все нужные вам клавиши были работоспособны в режиме [run], установите соответствующие свойства в окне свойств, но проще так: щелкните по Плееру правой клавишей мыши**→ Properties → Controls →** поставьте по две галочки против каждой нужной вам клавиши.

Для того, чтобы музыка зазвучала, Visual Basic должен выполнить следующие операторы:

```
Плеер.DeviceType = "Sequencer"
Плеер.FileName = "c:\Windows\Media\Canyon.mid"
Плеер.Command = "Open"
Плеер.Command = "Play"
```

Пояснения: Первая строка выбирает тип устройства (DeviceType) внутри вашей звуковой карты, которое будет проигрывать ваш звуковой файл. Для файлов с расширением mid, rmi используется устройство Sequencer. Для файлов с расширением wav,

mp3 используется устройство WaveAudio.

Вторая строка объясняет компьютеру, откуда брать звуковой файл.

Третья строка дает команду (Command) на открытие (Open) файла. Это необходимо для дальнейшей работы с ним.

Четвертая строка дает команду на начало воспроизведения (Play) файла и вы слышите музыку.

Мы привыкли, что все, что ни происходит в проекте, происходит при нажатии кнопок. Потому что все процедуры, которые мы писали до этого, объясняли компьютеру, как реагировать на одно-единственное **событие** - щелчок мышкой по кнопке. Как сделать, чтобы вышеприведенные 4 оператора выполнялись сразу же при запуске проекта без нажатия каких бы то ни было кнопок? Существуют ли какие-либо другие события, не сводящиеся к щелчкам мыши? Да, и их много (например, нажатие клавиши на клавиатуре или факт изменения текста в текстовом поле). Но мы их раньше не замечали, так как Visual Basic на них никак не реагировал. Одно из них - загрузка формы (Form_Load) - наступает автоматически при запуске проекта (конечно, если вы этому специально не воспрепятствуете) и кроме всего прочего имеет следствием появление формы на экране. Оно-то нам и нужно.

Сделаем двойной щелчок по форме. Возникнет заготовка новой процедуры:

```
Private Sub Form_Load()

End Sub
```

Ее заголовок можно вольно перевести так: "Что делать, когда загружена форма", а для нас это фактически означает "Что делать, когда запущен проект". Естественно, в эту процедуру мы записываем наши 4 оператора. Музыка готова. Пока не запускайте проект.

Если вы хотите, чтобы ваше музыкальное сопровождение правильно работало, нужно позаботиться, чтобы звуковой файл после окончания воспроизведения своевременно закрывался командой "Close". В нашем случае других звуковых файлов не используется, поэтому закрытие файла можно отложить на самый последний момент - момент завершения работы проекта. С ним связано событие Form_Terminate. Вот ваши следующие три строки:

```
Private Sub Form_Terminate()
    Плеер.Command = "Close"
End Sub
```

Помните, что событие Form_Terminate наступает только тогда, когда мы завершаем работу проекта, щелкнув по крестику в правом верхнем углу формы, а не кнопкой End на панели инструментов.

Проверьте, как работает наша музыка, запустив проект.

Музыка в кнопках

Поставим задачу - сделать так, чтобы при нажатии на калькуляторе каждой из четырех клавиш арифметических действий раздавался какой-нибудь короткий мелодичный звук, причем для каждой клавиши свой. Таких звуков много по адресу C:\Windows\Media. Там они записаны в файлах, имеющих расширение wav. Выберем из них Chime.wav, Notify.wav, Tada.wav и Logoff.wav.

Разместим в проекте еще один элемент Microsoft Multimedia Control 6.0. Дадим ему имя Звук. Его клавиши нам не нужны, нам ни к чему управлять короткими звуками. А раз так, то сделаем объект Звук невидимым. Для этого его свойству **Visible** придадим значение False.

Устройством для воспроизведения Wav-файлов является WaveAudio. Чем раньше мы объясним это компьютеру, тем лучше. Поэтому поместим соответствующий оператор .

Звук.DeviceType = "WaveAudio"

в процедуру, которая выполняется раньше всех - в Form_Load.

В каждую из четырех процедур кнопок поместим четыре новых оператора следующего вида:

Звук.FileName = "c:\Windows\Media\.....wav"
Звук.Command = "Open"
Звук.Command = "Sound"
Звук.Command = "Close"

Пояснения: В нашем случае команда "Sound" имеет то же действие, что и команда "Play", но отличается от нее тем, что задерживает выполнение остальных операторов проекта до тех пор, пока не закончится воспроизведение звукового файла. Вы можете убедиться в этом, когда запустите готовый проект. Пока при помощи команды "Play" воспроизводится длинная мелодия Canyon.mid, вы можете спокойно пользоваться калькулятором, нажимая на кнопки. Когда же при помощи команды "Sound" воспроизводится звук Chimes.wav, весь проект ненадолго "замерзает".

Замечание: Если ваша звуковая карта достаточно хорошего качества, то вы сможете одновременно услышать и Canyon.mid и Wav-файлы.

Вот так будут выглядеть в окне кода ваши процедуры, связанные со звуком:

```
Private Sub Кл_сложения_Click()
   Звук.FileName = "c:\Windows\Media\Chimes.wav"
   Звук.Command = "Open"
   Звук.Command = "Sound"
   Звук.Command = "Close"
   Результат.Text = Val(Число1.Text) + Val(Число2.Text)
End Sub

Private Sub Кл_вычитания_Click()
   Звук.FileName = "c:\Windows\Media\Notify.wav"
   Звук.Command = "Open"
   Звук.Command = "Sound"
   Звук.Command = "Close"
```

```
    Результат.Text = Val(Число1.Text) - Val(Число2.Text)
End Sub
.....................................................
.....................................................

Private Sub Form_Load()
    Звук.DeviceType = "WaveAudio"
    Плеер.DeviceType = "Sequencer"
    Плеер.FileName = "c:\Windows\Media\Canyon.mid"
    Плеер.Command = "Open"
    Плеер.Command = "Play"
End Sub

Private Sub Form_Terminate()
    Плеер.Command = "Close"
End Sub
```

Проигрывание аудиодисков

Элемент управления Microsoft Multimedia Control 6.0 можно использовать и для проигрывания из вашего проекта самых обычных некомпьютерных аудио-компакт-дисков. Разместите этот элемент управления на форме, придайте ему имя "CDPlayer". Вставьте диск в дисковод CD-ROM вашего компьютера. Теперь вам достаточно выполнить такую цепочку операторов:

```
CDPlayer.DeviceType = "CDAudio"
CDPlayer.Command = "Open"
CDPlayer.Command = "Play"
```

Вы скажете - Я могу это делать и безо всякого Visual Basic. Это верно. Но из Visual Basic это делать интереснее. В элементе Microsoft Multimedia Control 6.0 имеются возможности тонкого управления проигрыванием, которые вы не найдете в стандартных проигрывателях.

Плеер ваших любимых хитов

Если у вас набралось на диске 5-6 любимых звуковых файлов, то вы уже можете создать проект - плеер, в котором будет соответственно 5-6 кнопок с названиями исполняемых произведений. При нажатии кнопки звучит соответствующее произведение.

Вы можете сделать так, чтобы на форме тут же появлялась и подходящая картинка. Например, если исполняется песня группы "Столбняк", то пусть на форме появляется фотография задумчивых исполнителей этой группы. Для появления картинки подойдет оператор вида

Form1.Picture = **LoadPicture**("C:\Program Files\Microsoft Office\Clipart\Popular\Agree.wmf")

Здесь LoadPicture означает Загрузить картинку. В скобках с кавычками указывается адрес картинки на вашем компьютере. Кстати, в указанной папке вы найдете несколько десятков картинок.

Вы также можете сделать, чтобы при проигрывании мелодии вы видели и текстовое описание мелодии или, скажем, биографию композитора. Для этого поместите на форму большую метку (Label) и в подходящие места программы вставляйте операторы вида

Label1.Caption = "Композитор - Гладков. Впервые исполнена в 1970 году."

При работе со звуковыми файлами возникает вопрос - когда закрывать файлы? Если перед открытием следующего файла не закрыть предыдущий, то нормальной работы не получится. Связка

Плеер.Command = "Play"
Плеер.Command = "Close"

не подойдет, так как музыка закончится, не успев начаться. Здесь подойдет такая связка:

Плеер.Command = "Close"
Плеер.FileName = "c:\Windows\Media\Canyon.mid"
Плеер.Command = "Open"
Плеер.Command = "Play"

Здесь команда "Close" выполняется самой первой и закрывает любой файл, который исполнялся или мог исполняться раньше. После этого команда "Open" спокойно открывает нужный файл.

Задание 3: Создайте только-что описанный мною *"Плеер ваших любимых хитов".*

Задание 4 "Ваш собственный музыкальный компакт-диск": Сейчас широко распространены в продаже компьютерные компакт-диски такого рода: вы вставляете его в компьютер, он инсталлирует свою программу, затем вы запускаете ее. На экране появляются красочная заставка типа "Ваши любимые песни" и список песен. Вы выбираете одну из них. Песня звучит, возникают подходящие фото и текст. В любой момент вы можете прервать песню и выбрать другую.

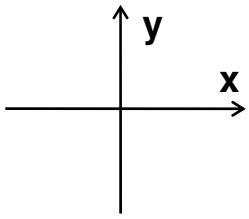
Если у вас или у вашего друга есть устройство записи на компакт-диски CD-RW, то вы вполне можете создать свой собственный компакт-диск, который будет делать то же самое. На диске будут находиться как сами файлы песен, так и инсталляционный пакет программы для их воспроизведения. Вам нужно где-то достать файлы песен, а также, перелистав эту книгу вперед, прочесть в 3.8, как переносить свою программу на другие компьютеры. В остальном вам достаточно знания предыдущего материала и советов из предыдущей задачи.

2.9. Система координат

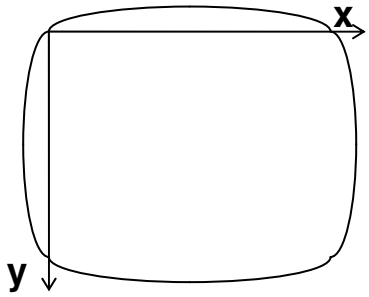
Чтобы рисовать фигуры, чтобы помещать объекты в нужное место экрана или формы, чтобы при анимации двигать объекты в нужном направлении, вы должны уметь объяснить компьютеру, где, в каком месте экрана или формы вы хотите нарисовать фигуру или поместить объект. Сделать это можно только имея представление о системе координат.

Если вы еще не знаете, что такое система координат, то все же постарайтесь изучить этот раздел, без него дальше будет трудно.

В школе вы привыкли к такой системе координат:



На экране компьютера применяется такая:



Как видите, ось y направлена вниз. Это не очень привычно. Если вас это раздражает, то Visual Basic может вам предложить исправить ситуацию. Однако, редко кто пользуется этой возможностью, поэтому и вам будет полезно привыкнуть к компьютерной системе.

Как нам управлять положением объектов на экране? Выделите любой объект на форме и загляните в его свойства **Left** и **Top**.

Значение свойства Left - это расстояние левого края объекта от левого края формы.
Значение свойства Top - это расстояние верхнего края объекта от верхнего края формы.

Таким образом, для объектов на форме действует компьютерная система координат с началом в верхнем левом углу формы. Убедитесь в этом - потаскайте любой объект по форме, следя за значениями Left и Top. Добейтесь:

- нуля в одном свойстве
- нуля в другом свойстве
- нуля в обоих свойствах
- максимума в одном свойстве
- максимума в другом свойстве
- максимума в обоих свойствах
- минуса в одном свойстве
- минуса в другом свойстве
- минуса в обоих свойствах

Единицей измерения расстояния в Visual Basic является **твип**. Это очень маленькая величина и на пространстве экрана ее нельзя однозначно выразить в долях миллиметра. Ее значение зависит от разрешающей способности видеорежима. В твипах выражаются свойства Left, Top, **Width** (ширина объекта) и **Height** (высота объекта). Потаскайте правую и нижнюю границы объекта, следя за значениями Width и Height.

В режиме проектирования мы не можем таскать форму по экрану за заголовок, да это и не нужно. Изменяя свойства Left и Top формы, мы управляем местом ее появления на экране после запуска проекта. Для формы начало компьютерной системы координат находится в левом верхнем углу экрана

В режиме работы положением и размерами объектов мы управляем просто:

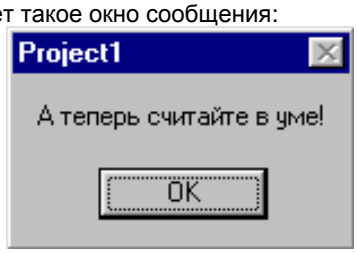
Form1.Left = 2000

Похвальное дело снабжения нашего калькулятора различными "приколами" я вывожу в задание для самостоятельной работы:

Задание 5: Пусть при нажатии на клавишу вычитания эта клавиша прыгает куда-нибудь совсем в другое место калькулятора и на ней вместо минуса появляется текст типа "Я устала вычитать" или "Не трогай меня - я нервная!". Когда это у вас получится, сделайте так, чтобы при нажатии на клавишу СБРОС клавиша вычитания скромно возвращалась на место и принимала свой прежний вид.

2.10. Вывод сообщений - MsgBox

Можно заставить компьютер в любой момент выполнения программы выводить нам какое-нибудь сообщение. Например, пусть калькулятор при завершении работы выдает такое окно сообщения:



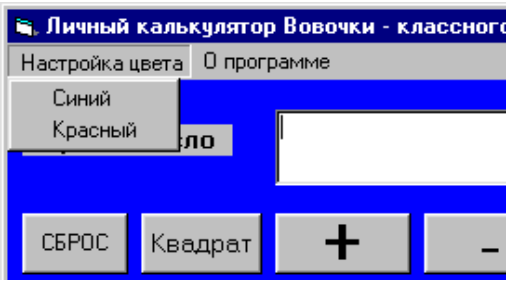
Для этого подойдет такой новый для нас оператор:

MsgBox ("А теперь считайте в уме!")

Чтобы он выполнился именно при завершении работы, его нужно поместить в процедуру Form_Terminate.
Прочитав сообщение, щелкните OK.
Более подробно об окнах сообщений написано в 5.10.
Вы не забыли, что все, мною сказанное, нужно проверять?

2.11. Меню пользователя

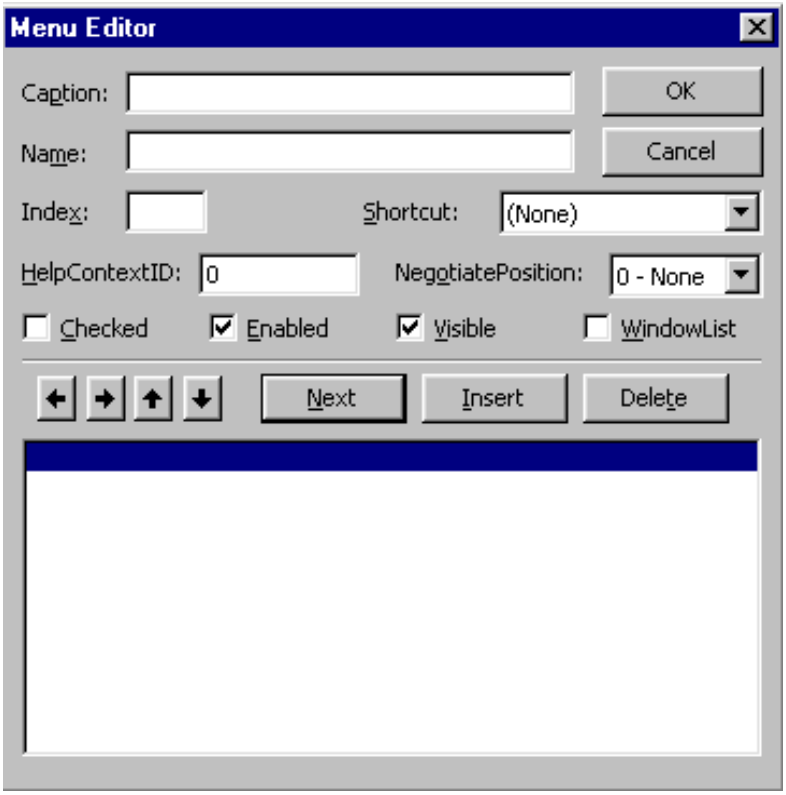
Какая же программа без своего меню! Нашему калькулятору оно, вроде бы, ни к чему, но, все равно, интересно и полезно сделать хотя бы простенькое. Пусть оно выглядит так:



Задачу поставим такую: Щелкнув по пункту "Настройка цвета", мы должны увидеть выпадающее меню из двух пунктов. Щелкнув по пункту "Синий" этого выпавшего меню, мы красим калькулятор в синий цвет, щелкнув по пункту "Красный" - в красный.

Щелкнув по пункту "О программе", мы вызываем сообщение с кратким описанием программы.
Нам предстоит создать меню, а затем заставить его работать.

Создаем меню. Чтобы его создать, достаточно в главном меню Visual Basic выбрать **Tools→Menu Editor**. Перед вами возникнет следующее диалоговое окно:



Начнем с настройки цвета. Введем в поле Caption текст "Настройка цвета". Это для того, чтобы пункт меню с этим текстом появился на форме. По мере ввода этот текст возникает и в поле, расположенном ниже.

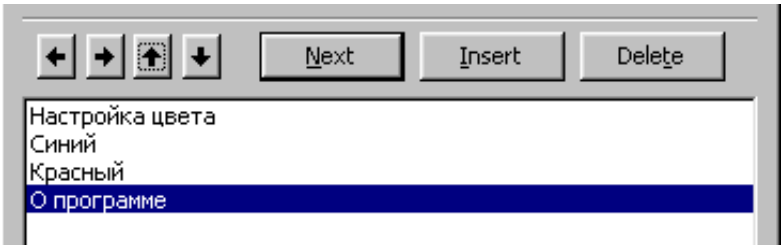
Теперь придумаем этому пункту имя, например, пункт_меню_Настройка_цвета, и введем его в поле Name. Это чтобы пункт меню мог заработать. Если хотите вызывать этот пункт не только мышкой, но и с клавиатуры, выберите что-нибудь из списка Shortcut.

Щелкните по кнопке Next и займитесь пунктом "Синий", дав ему имя пункт_меню_Синий.

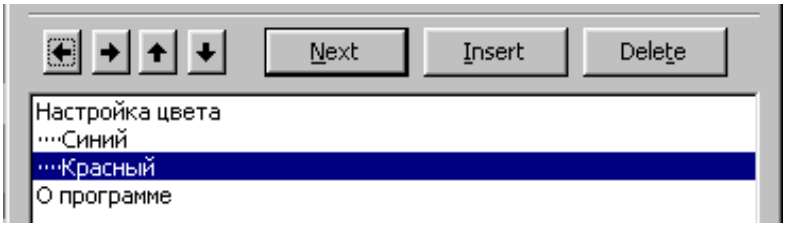
Щелкните по кнопке Next и займитесь пунктом "Красный", дав ему имя пункт_меню_Красный.

Щелкните по кнопке Next и займитесь пунктом "О программе", дав ему имя пункт_меню_О_программе.

В результате в окне возникает список из 4 пунктов:



Теперь нужно объяснить компьютеру, что пункты "Синий" и "Красный" входят внутрь пункта "Настройка цвета". Для этого выделим их по очереди и сдвинем направо кнопкой →. Получается такая картинка:



Если у вас что-то не получается, то вы все же можете добиться своего, используя все клавиши, показанные на картинке. Стрелки сдвигают выделенный пункт меню, Delete удаляет, Insert вставляет новый пункт.

Все в порядке - внешний вид меню готов. Запустите проект. Вы видите, что меню желаемого вида появилось на калькуляторе. Пощелкайте по пунктам. Естественно, ничего не происходит. Для того, чтобы происходило, нужно заставить меню работать, а для этого нужно для каждого пункта написать свою процедуру.

Заставляем меню работать. В режиме проектирования выберем пункт меню "Синий" и щелкнем по нему. В окне кода появляется заготовка процедуры:

```
Private Sub пункт_меню_Синий_Click()

End Sub
```

Вы уже наверняка догадались, что это приглашение объяснить компьютеру, что нужно делать при выборе пункта "Синий". Для этого подойдет парочка операторов:

Form1.Picture = LoadPicture()
Form1.BackColor = vbBlue

Второй оператор красит форму в синий цвет, первый убирает с формы картинку, если она там была (сравните с материалом перед заданием 3 из 2.8).

Аналогично программируем пункт "Красный". А сообщение о программе обеспечит оператор

MsgBox ("Программа создана в 2000 году")

Запустите проект и проверьте, как он работает.

Средствами Visual Basic можно создавать, преобразовывать и уничтожать меню не только в режиме проектирования, но и в режиме работы. Кроме обычных меню Visual Basic умеет создавать и так называемые контекстные меню (что это такое, рассказано в 3.5). Ни на том, ни на другом я не буду останавливаться.

2.12. Кино в проекте

Все есть у нашего калькулятора - и картинки, и звук, и меню, и прыгающие кнопки. Для полного счастья не хватает кино. Нет проблем! Причем их нет двумя способами!

Первый способ. **Project→Components→Microsoft Windows Common Controls-2 6.0**. На Toolbox появится несколько новых элементов управления. Берите из них **Animation** и размещайте на форме - это ваш экран для кино. Его имя - Animation1. Этот элемент управления позволяет прямо в работающем проекте просматривать видеофайлы в формате AVI. Если у вас на компьютере нет порядочного кино в этом формате, то несколько маленьких анимационных роликов вы все-таки найдете в папке Videos, находящейся внутри папки, куда устанавливалась ваша Visual Studio. Скорее всего они находятся по такому адресу: "C:\Program Files\Microsoft Visual Studio\Common\Graphics\Videos".

Сделайте в проекте кнопку и назовите ее, например, "Видео". Вот процедура, которая при нажатии кнопки "Видео" бесконечно воспроизводит видеоролик FILECOPY.AVI:

```
Private Sub Видео_Click()
   Animation1.Open "C:\Program Files\Microsoft Visual Studio\Common\Graphics\Videos\FILECOPY.AVI"
   Animation1.Play
End Sub
```

Здесь полная аналогия с аудиоплеером, который я рассматривал в 2.8 Первая строка процедуры открывает файл, вторая его воспроизводит.

Во время демонстрации вы можете выполнять на калькуляторе другие свои дела. Чтобы "заморозить" демонстрацию, нужно выполнить оператор Animation1.**Stop**, а чтобы совсем прекратить и убрать с экрана - Animation1.**Close**.

Если вам нужно 3 раза воспроизвести кадры видеофильма с 5 по 12, вы пишете

Animation1.Play 3, 5, 12

Второй способ. Используйте нашего старого знакомого - мастера на все руки - Microsoft Multimedia Control 6.0, который тоже позволяет просматривать видеофайлы в формате AVI. Разместите его на форме и назовите, скажем, "Кино". Вот цепочка операторов, приводящая к результату:

```
Кино.DeviceType = "AVIVideo"
Кино.FileName = "C:\Program Files\Microsoft Visual Studio\Common\Graphics\Videos\ FILECOPY.AVI"
Кино.Command = "Open"
Кино.Command = "Play"
```

Просмотр идет в отдельном окне, размер и положение которого вы можете мышкой менять в процессе просмотра, что само по себе любопытно. К тому же, вы можете пользоваться управляющими кнопками элемента управления.

2.13. Кое-что необходимое напоследок

Комментарии

Комментарии - это пояснения к тексту программы. Зачем они нужны?

Когда человек со стороны посмотрит на вашу программу, например, на эту (из калькулятора):

```
Private Sub Кл_сложения_Click()
   Звук.FileName = "c:\Windows\Media\Chimes.wav"
   Звук.Command = "Open"
   Звук.Command = "Sound"
   Звук.Command = "Close"
   Результат.Text = Val(Число1.Text) + Val(Число2.Text)
End Sub
```

он вряд ли поймет, в чем здесь смысл и для какой задачи программа написана. Если Звук, то при чем здесь сложение? Да и все остальное… Ну да ладно, это полбеды, а беда в том, что если через пару лет вам срочно понадобится самому разобраться в этой старой своей программе (так как ее выдвинули на Мобилевскую премию), а вы за это время ни разу не работали со звуком, то вы сами не сможете ничего понять, так как все забыли!

Любой профессиональный программист знает две вещи. Первое - любая старая программа через год забывается напрочь.

Второе - почти любая старая программа или ее часть через полтора года бывает вдруг позарез нужна как исходный материал для новой программы и поэтому в ней надо срочно разобраться. Наученный горьким опытом, программист снабжает многие строчки кода собственными комментариями. Получается вот что:

```
                    'Процедура, которая объясняет компьютеру, что ему делать, если мы щелкнули
                    'по клавише сложения калькулятора, а именно: проиграть короткую мелодию
                    'Chimes.wav на объекте с именем Звук, а затем сложить два числа.
Private Sub Кл_сложения_Click()
  Звук.FileName = "c:\Windows\Media\Chimes.wav"        'Указываем адрес звукового файла на диске.
  Звук.Command = "Open"                                'Перед проигрыванием файл нужно обязательно открыть.
  Звук.Command = "Sound"                               'Включить воспроизведение.
  Звук.Command = "Close"                               'После проигрывания файл нужно обязательно закрыть.
  Результат.Text = Val(Число1.Text) + Val(Число2.Text) 'Склалываем числа, причем Val преобразует текст в число
End Sub
```

Компьютеру комментарий не нужен, он его не поймет, а если начнет понимать, то ничего хорошего из этого не выйдет. Так что заглядывать в него компьютеру не нужно. И чтобы ясно показать компьютеру, куда ему не надо заглядывать, программист в каждой строке кода перед комментарием ставит одинарную кавычку. Visual Basic, выполняя очередную строку кода, просматривает ее слева направо, и как только наткнется на кавычку, правее уже не глядит.

Перенос длинного оператора на следующую строку

Иногда оператор получается такой длинный, что не умещается на экране. Это не беда - в окне кода он уместится, так как окно гораздо шире экрана, надо только его прокрутить. Но все равно неприятно, прокручивать не хочется. Оператор можно перенести на другую строку комбинацией пробела и подчеркивания. Например, вместо

```
Результат.Text = Val(Число1.Text) + Val(Число2.Text)
```

можно записать

```
Результат.Text = Val(Число1.Text)  _
               + Val(Число2.Text)
```

или

```
Результат.Text =  _
  Val(Число1.Text)  _
+ Val(Число2.Text)
```

Как видите, в конце строк стоит пара символов - пробел и за ним знак подчеркивания.

Запись нескольких операторов в одной строке

Visual Basic допускает писать в одной строке несколько операторов, разделяя их двоеточиями, вот так:

```
Звук.Command = "Open"   :   Звук.Command = "Sound"   :   Звук.Command = "Close"
```

Это приятно экономит место по вертикали экрана.

Превращаем наш калькулятор в независимую программу

В Windows вы привыкли запускать игры и другие программы двумя способами: щелкнув по значку программы на рабочем столе Windows (или в папке или в проводнике) или выбрав программу в стартовом меню на панели задач. Наш калькулятор запускается пока только из Visual Basic, что, возможно, уязвляет вашу гордость. Что ж, превратим его в независимую программу, которая будет запускаться, как и все, без запуска Visual Basic.

Для этого - **File→Make Project1.exe**. Затем в открывшемся окне выбираем папку, где будет сохранен будущий файл нашей независимой программы, и задаем ему имя (скажем, Суперкалькулятор). Затем ОК - и ваш файл Суперкалькулятор.exe готов. Если вы в качестве папки сохранения файла выберете c:\Windows\Рабочий стол, то ваш калькулятор расположится на рабочем столе Windows, а если вы ярлык этого файла (ярлыков мы не проходили) поместите в папку c:\Windows\Главное меню\Программы\Стандартные, то он будет красоваться в запускающем меню рядом со стандартным калькулятором Windows (это все равно, что сидеть с Биллом Гейтсом в одном Мерседесе).

Однако, если вы попытаетесь скопировать ваш Суперкалькулятор.exe на другой компьютер и там его запустить, то вас может ждать неудача. Здесь нужны более сложные действия, чем я описал (см. 3.8).

Как мы в дальнейшем улучшим наш калькулятор

- Предохраним его от деления на ноль и от прочих досадных ситуаций, описанных в 2.5.
- Снабдим его паролем, чтобы все, кому не лень, не могли наслаждаться его возможностями (то и другое см. в 5.9).
- Обеспечим привычный для глаз вид результатов (см. в 4.5 "Форматирование результатов").

Но для этого нужно знать переменные величины.

Глава 3. Работа в среде Visual Basic

В этой главе мы не будем программировать, мы будем учиться нажимать кнопки. Мы познакомимся с такими сторонами работы в среде Visual Basic, как инсталляция Visual Basic, сохранение проектов, работа с окнами и меню Visual Basic и т.п. Это всё необходимые моменты, но не все они необходимы вам именно сейчас. В любом случае прочитайте главу, а проверяйте лишь то, что вам покажется нужным.

Самые основы работы в среде Visual Basic уже изложены в первых двух главах на примере создания проектов. Кроме этого, значительную часть материала, относящуюся к работе в среде Visual Basic, мне показалось уместным изложить позже, в других местах книги. Вот эти места:

- 4.2 - Пошаговый режим
- 6.1 - Зацикливание
- Глава 7 - Отладка программы
- 9.1 - Объекты. Их свойства, их события, их методы
- 14.4 - Рамка (Frame)
- 19.3 - Структура проекта. Окно Project Explorer.
- Приложение 2. Работа в Windows. Ввод текста

Все остальное, что мне показалось необходимым, изложено в этой главе.

3.1. Что нужно знать и уметь перед тем, как сесть за компьютер

Я постарался написать книгу так, чтобы сесть за компьютер и программировать на Visual Basic мог даже тот, кто ни разу за компьютер не садился. Специально для такого человека я написал Приложение, в котором объясняю все элементарные и необходимые для этого вещи: как устроен компьютер, какова структура папок на диске, как работать с текстом в текстовом редакторе, как управляться с окнами Windows. Если вы чувствуете пробелы хотя бы в одной из этих областей, то ни в коем случае не садитесь за компьютер и дальше не читайте, а читайте сначала Приложение.

Начнем, конечно, с инсталляции Visual Basic на ваш компьютер.

3.2. Инсталляция Visual Basic

Вообще-то, инсталляция - дело не для новичка. Но если рядом нет никого, кто мог бы вам помочь … В общем, смело вперед. Вероятность того, что у вас получится, достаточно высока.

Чтобы работать с Visual Basic, он должен быть на вашем компьютере установлен (инсталлирован). Что такое установка программы, вы можете прочесть в 3.8.

Если вы не знаете, установлен ли Visual Basic на вашем компьютере, попробуйте запустить его из стартового меню, как это описано в 3.4. Если в стартовом меню его нет, то почти наверняка он не установлен. Есть более надежный способ проверки - посмотреть в Панель управления Windows, но для новичка это, пожалуй, опасно.

Чтобы установить Visual Basic, вам нужен компакт-диск. Во многих случаях Visual Basic продается на компакт-дисках не отдельно, а в составе пакета Microsoft Visual Studio, куда кроме Visual Basic входят средства программирования Visual C++ и Visual FoxPro.

Итак, у вас в руках компакт-диск (или пакет компакт-дисков) с надписью Visual Basic или Visual Studio. На разных компьютерах и разных операционных системах установка Visual Basic идет чуть-чуть по-разному. Однако, в подавляющем большинстве случаев она проходит именно так, как написано ниже.

Ваши действия:

1. Вставьте диск (или диск №1 из пакета) в дисковод CD-ROM. Для этого нужно сначала нажать на кнопку дисковода и подождать, когда из него выдвинется поднос для диска. Аккуратно установите диск на поднос блестящей стороной вниз. Старайтесь не дотрагиваться до блестящей поверхности пальцами. Снова нажмите на кнопку дисковода. Поднос вдвинется в дисковод, унося с собой диск. Здесь возможно одно из двух. Если автоматически запустится программа установки, то переходите к пункту 4. Если же через пару минут ничего не произошло, то читайте дальше.
2. Вам нужно добраться до файла Setup.exe на компакт-диске. Для этого зайдите в значок Мой компьютер на рабочем столе Windows и найдите в открывшемся окне значок дисковода CD-ROM. Щелкните по нему правой клавишей мыши и в открывшемся контекстном меню выберите пункт "Открыть". Перед вами откроется окно со списком папок и файлов компакт-диска.
3. Найдите в этом окне файл Setup.exe и запустите его на выполнение.
4. Начнет работу мастер инсталляции. Дальнейшие ваши действия задаются мастером. Ваше дело - отвечать на вопросы мастера и выбирать из предложенных вариантов. Здесь я прослежу установку Visual Basic из пакета Visual Studio 6.0. Установка Visual Basic с отдельного диска во многом аналогична, хоть и несколько покороче. Время от времени вам придется выполнять просьбу мастера вставить в дисковод другой диск.

5. Мастер предлагает вам общие сведения по установке. Прочтите их и нажмите кнопку Next.
6. Мастер спрашивает, согласны ли вы нести ответственность за незаконную установку Visual Studio. Если согласны, жмите I accept the agreement. Затем Next.
7. В диалоговом окне - три поля. В верхнее поле мастер предлагает ввести идентификационный номер устанавливаемого продукта. Его вам должен был сказать продавец. В другие два поля вы вводите свое имя и название своей фирмы. Можете ввести туда любую тарабарщину. Затем Next.
8. Из трех типов установки выбирайте Custom.
9. Мастер предлагает выбрать папку, в которую будут устанавливаться некоторые файлы программы. Новичкам рекомендую согласиться с его предложением. Next.
10. Жмите Continue.
11. Жмите OK.
12. Мастер предлагает вам компоненты Visual Studio для установки. Если вы не любите транжирить дисковое пространство, то поставьте флажки у следующих копонент:
    - Microsoft Visual Basic
    - Active X
    - Data Access
    - Graphics
    - Tools

    С остальных компонентов флажки снимите. Continue.
13. Мастер довольно долго копирует файлы с компакт-диска на жесткий диск, иногда задавая вам вопросы, смысла которых вы скорее всего не понимаете. Постарайтесь отвечать так, как рекомендует сам мастер.
14. После этого мастер предложит вам перезагрузиться (кнопка Windows Restart). Если у него самого не получится, то просто выключите компьютер, как положено, а затем включите.
15. После перезагрузки мастер может предложить вам установить еще несколько компонентов Visual Studio. Отказывайтесь. Вам это не скоро понадобится.
16. После окончания инсталляции можете вынуть компакт-диск.

3.3. Порядок работы над проектом в Visual Basic

(1) Загрузите Visual Basic

(2) Поработайте немного над проектом, то есть разместите на форме несколько элементов управления или запишите в окно кода несколько строк программы.

(3) Сохраните проект на жестком диске

(4) Запустите проект на выполнение (кнопкой Start или клавишей F5).

Если результаты вас удовлетворяют, перейдите к пункту 6

(5) Исправьте ошибки в проекте (подробно об этом см. в Глава 7)

Вернитесь к пункту 3

(6) Добавьте на форму еще несколько элементов управления и допишите в окно кода еще несколько строк.

Вернитесь к пункту 3

Рассмотрим подробнее некоторые пункты этого алгоритма.

3.4. Загрузка Visual Basic и выход из него

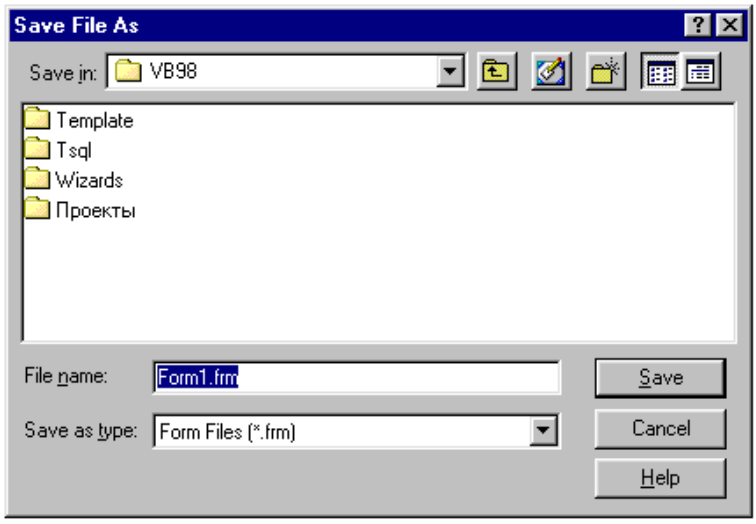
Загрузка Visual Basic ничем не отличается от загрузки любой другой инсталлированной программы. Нажмите кнопку "Пуск" на панели задач. В открывшемся стартовом меню выберите последовательно пункты **Программы → Microsoft Visual Studio → Microsoft Visual Basic**.

Выход из Visual Basic совершенно такой же, как и из большинства других приложений Windows - щелчок мышкой по крестику в правом верхнем углу главного окна Visual Basic или **File → Exit**.

3.5. Сохранение проекта на диске. Загрузка проекта с диска

Итак, вы готовы запустить новый проект на выполнение. Во время выполнения может произойти неприятная вещь - Visual Basic может зависнуть. Это означает, что изображение на экране замрет, и никакими клавишами или мышкой вы не сможете вывести Visual Basic из ступора. Придется выходить из Visual Basic "аварийно" - при помощи клавиш Ctrl-Alt-Del. Поскольку ваш проект находится пока только на экране и в оперативной памяти, то при аварийном выходе из Visual Basic он пропадет и вам придется создавать его снова. Чтобы избежать лишней работы, вы должны перед выполнением проекта записать (сохранить) его на диск.

Для этого в меню File есть опция Save. Как только вы ее выберете, перед вами появится диалоговое окно:



В списке Save in компьютер предлагает вам выбрать папку, в которую вы хотели бы записать проект. В первый раз он предлагает вам ту папку, в которую сам установлен (на рисунке это папка VB98). Я ни в коем случае не рекомендую с этим соглашаться, так как в этом случае файлы вашего проекта будут перемешаны с рабочими файлами Visual Basic. Прежде чем порекомендовать, в какую папку вам следует сохранить свой проект, я расскажу вам, как, не выходя из нашего диалогового окна, путешествовать по папкам и дискам, как создавать папки и делать другие необходимые вещи.

В большом списке папок и файлов в середине диалогового окна вы видите содержимое папки, выбранной в списке Save in. Вы видите все папки и ни одного файла, хотя их там множество. Чтобы увидеть все файлы, что совершенно не обязательно, выберите в списке Save as type вариант All Files. В противном случае вы будете видеть кроме папок только файлы форм с расширением frm, о чем говорит предлагаемый вам вариант Form Files (*.frm).

Вы можете двойным щелчком мыши войти в любую из папок в большом списке. Тогда именно она появится в списке Save in, а в большом списке появится ее содержимое. Таким образом вы можете продвигаться вглубь папок, как внутрь матрешки. А как продвигаться наружу? Для этого есть кнопка . Щелчком по ней вы выйдете из папки, показанной в списке Save in, в папку, внутрь которой она входит. Еще щелчок - еще один шаг наружу. И так далее, пока не окажетесь на рабочем столе (Desktop), на котором расположены и значок Мой компьютер и другие значки. Если хотите опять добраться до дисков, войдите в Мой компьютер.

Щелкнув по черной треугольной стрелке в раскрывающемся списке Save in, вы раскроете список папок в виде части дерева, похожего на дерево Проводника Windows. Это позволит вам быстрее путешествовать по папкам.

Внутри папки, выбранной в списке Save in, вы можете создавать новые папки. Новая папка будет создана, если вы щелкнете по кнопке . Тут же вам нужно будет ввести с клавиатуры ее имя.

В приложениях Windows очень удобно пользоваться **правой клавишей мыши**. Каждый раз, когда вы хотите что-нибудь сделать с папкой, файлом или другим объектом на экране, но не знаете, на какие кнопки для этого нажимать, щелкните по этому объекту правой клавишей мыши. Немедленно из объекта выпадет так называемое **контекстное меню**, в котором перечислены самые популярные действия над этим объектом. Вам останется только выбрать. Новички! Будьте осторожны. Не выбирайте Cut (вырезать) или Delete (удалить).

Чтобы **скопировать** файл или папку в другое место, выберите в его контекстном меню опцию Copy (копировать). Затем доберитесь до папки, в которую хотите данный файл или папку скопировать, то есть добейтесь ее присутствия в списке Save in, и щелкните правой клавишей мыши по свободному пространству внутри большого списка папок и файлов. В открывшемся контекстном меню выберите Paste (вставить). Если вы хотите, чтобы копия находилась рядом с оригиналом, в список Save in заходить не нужно.

Чтобы **переместить** файл или папку в другое место, делайте все то же, что и при копировании, только вместо Copy выберите в контекстном меню опцию Cut (вырезать).

Чтобы **переименовать** файл или папку, выберите в контекстном меню опцию Rename (переименовать).

Итак, чтобы правильно сохраниться в первый раз, создайте внутри предлагаемой вам папки VB98 папку, в которой вы будете хранить все папки своих проектов, назовите ее как-нибудь, например, "Мои проекты". Затем зайдите в нее и создайте внутри нее папку для своего первого проекта. Затем зайдите в нее. В поле File Name компьютер предлагает вам имя для файла вашей формы - Form1.frm. Хотите - оставьте его неизменным, хотите - измените. Нажмите кнопку Save. Visual Basic сохранит файл формы и предложит вам сохранить главный файл вашего проекта. Причем под именем Project1.vbp. Сохраните и его под этим или другим именем.

Таким образом, ваш проект сохраняется в нескольких файлах.

После выполнения проект обычно исправляют и дополняют и перед следующим выполнением опять сохраняют: **File → Save**. Но в этот раз диалоговое окно не появляется. Visual Basic, ни о чем вас не спрашивая, стирает с диска весь ваш проект (в том виде, как он был сохранен в последний раз) и на его место записывает с тем же именем его исправленную и дополненную версию, то есть ту, что вы видите на экране. (На усмотрение программиста оставляется решить, а лучше ли новая версия старой и не жалко ли старой версии.) Так поступают все современные программные продукты.

Загрузка проекта с диска. Предположим, вы вчера сохранили свой проект, а сегодня хотите продолжить с ним работу.

Вы включаете компьютер, запускаете Visual Basic, затем - **File → Open**. На экране появляется диалоговое окно, очень похожее на то, что появляется при сохранении проекта. И действия ваши очень похожи: вы должны отыскать в окне папку вашего сохраненного вчера проекта, для чего вам придется попутешествовать по папкам. Зайдя в папку, откройте главный файл вашего проекта. Ваш проект появляется на экране.

Если ваш новый проект делается на основе старого, то чтобы не начинать новый проект с нуля, откройте окно загрузки проекта, скопируйте папку старого проекта и делайте новый проект на основе копии старого. Скопировать папку можно и в Windows, не заходя в Visual Basic.

3.6. Окна среды Visual Basic

До сих пор я не рассматривал систематически окна и меню Visual Basic 6.0. И вы, наверное, уже столкнулись в связи с этим с некоторыми проблемами. Для дальнейшего движения вперед необходимо рассмотреть их более подробно.

Начнем с окон. Их в среде Visual Basic довольно много, ведут они себя, как и положено окнам Windows: налезают друг на друга, заслоняют друг друга, часто нужное окно вообще куда-то исчезает. В общем, нужен порядок. Если у вас нет минимального опыта управляться с окном Windows, читайте Приложение 2.

Сначала разберемся, какие окна есть вообще. Для этого посмотрите на рисунок в 1.3. Вот названия окон по-английски:

Object - это окно, имеющее у нас заголовок Project1 - Form1 (Form). Его назначение - служить вместилищем для формы. Саму форму в режиме проектирования двигать мы не можем, так хоть окно подвигаем.

Toolbox - вместилище элементов управления.

Code - окно кода. Сейчас нам нужно научиться заставлять Visual Basic выводить в окно кода заготовку процедуры для обработки любого события. Пока мы умеем только, щелкнув дважды мышкой по кнопке Command1, получить заготовку процедуры Command1_Click. Но с кнопкой связана масса событий, а не только щелчок мышью. Как написать процедуры для других событий?

Пусть в вашем проекте есть такие объекты: форма, Command1 и Text1. Все их вы найдете в списке, находящемся в левой верхней части окна кода, если щелкнете по черной стрелке, раскрывающей этот список. Пусть вам нужно написать процедуру для обработки двойного щелчка мышью (DblClick) по объекту Text1. Выбираете в упомянутом списке Text1, затем раскрываете список, находящийся справа от упомянутого, и находите в нем все события, связанные с Text1. Выбираете из них DblClick - и в окне кода тут же возникает нужная вам заготовка:

```
Private Sub Text1_DblClick()

End Sub
```

Один момент: Если вы случайно щелкните по одной из двух маленьких кнопочек в левом нижнем углу окна кода, то часть процедур (или все они) куда-то пропадет. Щелкните по соседней кнопке.

Properties - окно свойств объекта.

Project Explorer - в этом окне мы видим структуру нашего проекта. Пока от него толку нет, так как у нас структура простая - одна форма и связанный с ней код. Сложные проекты могут состоять из многих форм и других элементов, взаимосвязь которых удобно видеть в этом окне.

В верхней части окна Project Explorer можно видеть три кнопки. Левая и средняя из них удобны для быстрого переключения между формой и окном кода. Более подробно работа Project Explorer описана в 19.3.

Если какое-то из описанных выше или ниже окон не открыто, идите в пункт **View** главного меню, а затем щелкните по имени того окна, что вам нужно. Если окно открыто, но заслонено другими окнами, то идите в пункт **Window** главного меню, а затем щелкните по имени нужного окна. Со всеми этими окнами можно делать все, что допускает любое окно Windows.

Следующие окна на рисунке я не показал:

Form Layout - позволяет вручную задавать позицию формы на экране после запуска проекта.

Object Browser - делится на два вертикальных списка. В левом приведены типы (**классы**) объектов Visual Basic. Если выделить в нем какой-нибудь объект, например, TextBox, то в правом списке можно видеть свойства этого объекта, события, которые могут происходить с этим объектом, и так называемые методы (см. 9.1), ему принадлежащие. Свойств здесь, между прочим, больше, чем в окне свойств, так как здесь приведены и те свойства, которые имеют смысл только в режиме [run]. Например, свойство SelText объекта TextBox, которое выдает фрагмент текста, выделенного нами в текстовом поле. Object Browser - довольно удобный способ увидеть, что "умеет" делать любой объект и что можно делать с ним.

Более подробно работа Object Browser описана в 9.1.

Color Palette - позволяет удобно раскрашивать объекты и текст в режиме проектирования.

Многие окна имеют привычку при сближении состыковываться, "склеиваться" между собой или "прилипать" к краям главного окна Visual Basic. Сделано это для удобства программистов, но поначалу может восприниматься, как неудобство. Вы можете отменить это свойство, зайдя в **Tools → Options→ Docking** и убрав оттуда флажки.

После запуска проекта ваша форма как бы отделяется от среды и "плавает" по экрану самостоятельно. При этом, если вы

неосторожно щелкните мимо формы, то она может пропасть из виду, загороженная другими приложениями Windows, в том числе - главным окном Visual Basic. Чтобы она снова появилась, достаточно щелкнуть по ее значку на панели задач Windows.

3.7. Главное меню Visual Basic

А теперь рассмотрим все нужные вам на первых порах пункты главного меню среды. Для понимания дальнейшего отметим, что в среде можно одновременно работать с несколькими проектами. О том, зачем это нужно, написано в 19.3.

File

New Project. Удаляет из среды все открытые проекты и создает новый проект.

Open Project. Удаляет из среды все открытые проекты и открывает для работы один из проектов, сохраненных вами ранее.

Add Project. Если у вас в среде уже открыто несколько проектов, то добавляет к ним или новый или (если вы выберете закладки Existing или Recent) один из проектов, сохраненных вами ранее. Если ни одного проекта не открыто, то действует, как New Project или Open Project.

Remove Project. Удаляет из среды ваш проект, а если у вас открыто несколько, то один из них, а именно тот, что выделен в окне Project Explorer.

Save Project. Сохраняет ваш проект на диске.

Save Project As. Сохраняет на диске копию вашего проекта.

Print. Распечатывает программу или формы вашего проекта на принтере.

Print Setup. Настройка печати.

Make Project1.exe. Превращает ваш проект в исполняемый файл (см. 2.13).

Ниже этих пунктов расположен удобный список для открытия проектов, которые вы открывали последними.

Exit. Выход из Visual Basic.

Edit

Undo. Отменить последние действия.

Redo. Вернуть отмененные действия.

Cut, Copy, Paste, Delete. Обычные и привычные для вас команды перемещения, копирования, удаления слов и фрагментов текста вашей программы. Применяются не только к тексту, но и к элементам управления на форме. Как проводят эти действия с текстом, рассказано в Приложении 2. А с элементами управления вот как. Часто, когда нам нужно иметь на форме несколько совершенно одинаково настроенных элементов управления, удобнее не брать их по-одиночке в Toolbox, а разместив один на форме и настроив его нужным образом, затем скопировать его. Копировать можно двумя способами:

- Щелчок по копируемому объекту → **Copy** → щелчок по форме → **Paste**.
- Щелчок по копируемому объекту правой клавишей мыши → пункт **Copy** в выпавшем контекстном меню → щелчок по форме правой клавишей мыши → пункт **Paste** в выпавшем контекстном меню.

Аналогично используются **Cut** (вырезать для переноса в другое место) и **Delete** (уничтожить).

Find, Find Next, Replace. Команды поиска и замены в тексте вашей программы отдельных слов и фрагментов.

View

Code, Object, Object Browser, Project Explorer, PropertiesWindow, Form Layout Window, Toolbox, Color Palette - это все названия окон среды Visual Basic, о которых речь была выше. Щелчок по любому из этих пунктов делает соответствующее окно видимым.

Immediate, Locals, Watch - окна, которые нужны для отладки проекта (см. Глава 7).

Project

Add Form- добавить форму. Ваша форма в режиме проектирования - это будущее окно в режиме [run]. Если вы выберете этот пункт, то в режиме проектирования у вас будет уже две формы. Это значит, что в режиме [run] вы сможете иметь два окна. Кстати, у каждой формы - свое окно кода, значит, окон кода у вас тоже два. Вы можете добавлять сколько угодно форм. Более подробно работа работа с несколькими формами описана в 19.1.

Remove Form - удалить форму.

Components. Если вы щелкните по закладке Controls, то это пункт позволяет добавить в окно Toolbox нестандартные элементы управления. Если вы щелкните по закладке Insertable Objects, то сможете вставлять в форму окна других приложений Windows. Например, если у вас на компьютере установлен Word, то найдите в списке пункт Microsoft Word Document и поставьте галочку. В окне Toolbox у вас появится значок документа Word. Берите его как обычный инструмент и располагайте на форме. Теперь у вас на форме расположен документ Word и, не выходя из Visual Basic, вы сможете пользоваться многими возможностями Word как в режиме проектирования, так и в режиме [run]. Для этого достаточно сделать двойной щелчок мышкой по документу, размещенному вами на форме.

Format

Align, Make Same Size и другие - эти пункты имеют дело с размерами, формой и местоположением элементов управления на форме и друг относительно друга, то есть с тем, с чем вы и так прекрасно справляетесь безо всяких меню. Выделите один или несколько объектов на форме и попробуйте применить эти пункты, посмотрите, что получится. Чтобы выделить несколько объектов, щелкайте по ним при нажатой клавише Ctrl или обведите их рамочкой.

Order. Бывает, что в результате вашего проектирования формы некоторые элементы управления перекрываются другими. Для примера поместите на проект две большие кнопки так, чтобы одна частично закрывала другую. Тогда приобретает важность вопрос - какой из объектов ближе к вам, а какой дальше. Управлять этим вы можете, выбрав Bring to Front (выдвинуть на передний план) или Send to Back (задвинуть на задний план)

Lock Controls. Иногда неловким движением мыши вы можете сдвинуть или изменить размер объекта в тщательно создан-

ном вами проекте. Чтобы этого не произошло, и нужен этот пункт. Объекты примерзнут к месту. Когда вы захотите их разморозить, снова выберите этот пункт.

Debug

Этот пункт нужен для отладки проекта и о нем особый разговор (см. Глава 7).

Run, Query, Diagram

Эти пункты меню мы не будем затрагивать.

Tools

Здесь находится Menu Editor, который применяется для создания меню в проекте (см. 2.11). Но нам сейчас интересен не он, а пункт **Options**, который позволяет **настраивать среду Visual Basic**: Вот его закладки:

- Закладка *Editor* - здесь рекомендую установить все флажки, так как каждый из них обеспечивает те или иные удобства работы в окне кода. Особенно хочу подчеркнуть флажок Require Variable Declaration, который требует обязательного объявления переменных величин. Поле Tab Width регулирует отступ при нажатии клавиши Tab.
- Закладка *Editor Format* - здесь вы можете настроить шрифт, размер шрифта, цвет шрифта и цвет фона в окне кода.
- Закладка *General* - здесь вы, изменяя числа в двух полях, можете настроить расстояние между линиями сетки на форме по горизонтали и вертикали. Флажок Align Controls to Grid требует, чтобы очертания элементов управления проходили только по линиям сетки. Убрав флажок Show Grid, вы сделаете линии сетки невидимыми. Остальные флажки оставьте установленными, переключатель не трогайте.
- Закладка *Docking* - Как я уже говорил, окна среды Visual Basic имеют привычку при сближении состыковываться, "склеиваться" между собой или "прилипать" к краям главного окна Visual Basic. Вы можете установить или отменить это свойство у каждого из окон, установив или убрав флажки.
- Закладка *Environment* - Переключатель When Visual Basic starts определяет, будет ли Visual Basic при своей загрузке предлагать создать или открыть проект (Prompt for project) или без предупреждения создавать новый (Create default project). Переключатель When a program starts определяет, будет ли Visual Basic перед любым запуском проекта на выполнение автоматически сохранять несохраненный проект (Save Changes), предлагать сохранить проект (Prompt To Save Changes) или не сохранять (Don't Save Changes). Беспечным советую первое. Флажки должны быть везде установлены.
- Закладка *Advanced* - пока она нам не нужна.

Add-Ins

Здесь нам будет интересен Visual Data Manager, применяющийся для работы с базами данных. Разговор о нем - в 22.2.

Window

Мало чем отличается от аналогичного пункта в других приложениях Windows. Позволяет разными способами упорядочить сумятицу окон на экране или выдвинуть на передний план спрятавшееся окно.

Split позволяет смотреть на ваш код через два окна, а не через одно, что бывает удобно, когда кода много.

Help

Когда вы пишете оператор на Visual Basic, вы должны, во-первых, понимать, что он означает и что означает каждая его часть, а во-вторых, вы должны знать правила его записи. Visual Basic чрезвычайно богат, в него входят тысячи элементов - свойств, событий и др. Невозможно запомнить смысл и правила записи каждого элемента. Вот здесь-то и нужен Help - система помощи, которая позволит вам быстро найти информацию по любому элементу. Если, конечно, вы знаете английский.

Если вы хотите узнать подробности о каком-то служебном слове, например, BackColor, вы щелчком мыши ставите на него текстовый курсор и нажимаете на клавиатуре клавишу F1.

Contents представляет вам информацию в слегка систематизированном виде.

Index удобен тогда, когда вы заранее знаете название элемента, который вас интересует (например, FileName).

Система помощи в Visual Basic 6.0 осуществляется через так называемую библиотеку MSDN (**MSDN Library**), которая инсталлируется или с отдельных компакт-дисков или входит в состав пакета Visual Studio.

Панель инструментов

Предназначена для быстрого доступа к популярным пунктам меню. Задержите мышку на любом значке панели - и он назовет себя. В правой части панели - две пары чисел. Это свойства Left, Top, Width и Height выделенного объекта.

Кроме главной панели инструментов в Visual Basic есть еще несколько. Добраться до них можно так: **View →Toolbars**. Там вы найдете, в частности, панель Form Editor, при помощи которой вы сможете делать с элементами управления на форме многое из того, что делается в пункте Format главного меню, но быстрее. Также вам, возможно, пригодится панель Edit для работы с программным текстом в окне кода. Там же вы можете создать собственную панель инструментов или изменить стандартные.

3.8. Перенос вашего проекта на другие компьютеры

Когда ваш проект, например, Калькулятор, заработает на вашем компьютере, вам захочется, чтобы он заработал и на компьютерах ваших друзей. Но вот беда - на их компьютерах не установлен Visual Basic. Раньше для решения этой проблемы достаточно было создать исполняемый файл вашего проекта (назовем его Калькулятор.exe) и скопировать его на компьютер вашего друга. (Как создавать исполняемый файл, перечтите в 2.13.) Теперь не то. У новых версий Visual Basic гораздо более солидный и тяжелый подход к переносу программ с компьютера на компьютер. *Не копировать, а инсталлировать - вот девиз!* В причины отказа от легкой жизни я не буду вдаваться, основная такая - Visual Basic слишком зависит от Windows, а на разных компьютерах Windows разные или настроены по-разному.

Если вы когда-нибудь инсталлировали какую-нибудь игру на компьютер, то что такое инсталляция знаете. Вот вы купили

компакт-диск (или дискету) с понравившейся игрой или программой (пусть это будет Microsoft Office). Во многих случаях купленная программа просто так не запустится, нужна **установка** или по-другому **инсталляция** (от английского install - устанавливать). Говорят, что на диске находится не сама программа Microsoft Office, а **инсталляционный пакет** программы Microsoft Office, то есть набор файлов, в который в том числе входят в сжатом виде и файлы Microsoft Office. Вы находите в инсталляционном пакете программу установки (запускающий файл - **Setup.exe**) и запускаете ее. Все ваши дальнейшие действия диктуются этой программой. Фактически вам приходится только отвечать на ее вопросы. Например, вы отвечаете на вопрос, в какую папку жесткого диска вы желаете установить Microsoft Office. Программа установки разворачивает сжатые файлы Microsoft Office и копирует их на жесткий диск вашего компьютера. Кроме этого она настраивает Windows на работу с Microsoft Office. После окончания установки вы сможете запускать Microsoft Office из меню "Пуск" на панели задач Windows.

Во время установки игр вам, наверное, приходили в голову мысли о суперпрограммистах, которые мало того, что создают игру, они еще и сжимают файлы этой игры, переписывают их на компакт-диск и делают так, чтобы программа установки правильно установила ее на ваш компьютер. Сейчас я предлагаю вам стать таким "суперпрограммистом". Наша цель - через полчаса получить одну-две дискеты (а если у вас есть CD-RW - то компакт-диск) с инсталляционным пакетом, готовым к употреблению.

Первый этап - подготовка проекта

В вашей программе могут встретиться операторы типа

Плеер.FileName = "c:\Windows\Media\Canyon.mid"

или

Form1.Picture = LoadPicture("C:\Program Files\Microsoft Office\Clipart\Popular\Agree.wmf")

То есть в режиме работы ваш проект обращается к файлам, находящимся где-то далеко на диске. Вам никогда не приходило в голову, что будет, если кто-то, не подозревая, что эти файлы нужны вашей программе, сотрет их? Ничего хорошего не будет. А если вы к тому же предназначаете ваш проект для чужого компьютера, опасность и морока возрастают вдвойне.

Отсюда совет: Все такие файлы заранее скопируйте в папку вашего проекта, туда, где находятся файлы Project1.vbp и Form1.frm. По принципу "Все мое ношу с собой". Соответственно вам придется переделать и адреса в приведенных выше операторах. Пусть папка вашего проекта находится по адресу "c:\Проекты\Калькулятор". Тогда эти два оператора будут выглядеть так:

Плеер.FileName = " c:\Проекты\Калькулятор\Canyon.mid"

Form1.Picture = LoadPicture("c:\Проекты\Калькулятор\Agree.wmf")

Однако тут возникает еще одна проблема. Папку вашего проекта вы когда-нибудь можете захотеть перенести в другое место диска, от чего эти адреса сразу станут неправильными. Да и на чужом компьютере ваш проект наверняка попадет в папку с совсем другим адресом. В Visual Basic есть средство справиться с этой проблемой. Итак, вы скопировали файлы Canyon.mid и Agree.wmf в папку вашего проекта. Теперь вам достаточно приведенные выше операторы переписать в следующем виде:

Плеер.FileName = **App.Path** & "\Canyon.mid"

Form1.Picture = LoadPicture (**App.Path** & "\Agree.wmf")

Имеется некий объект App, назначение которого - в любой момент выполнения проекта кое-что знать о нем, в частности, его свойство Path как раз имеет значение адреса выполняемого проекта "c:\Проекты\Калькулятор".

Картинки, которые вы загрузили в объекты на этапе проектирования, не нуждаются во всех этих усилиях, так как они уже неявно сохранены в одном из файлов в папке проекта.

Если вы хотите, чтобы исполняемый файл вашего проекта имел собственный значок в Windows, придайте в режиме проектирования значение свойству **Icon** вашей формы.

Рекомендую с самого начала разработки проекта сохранять его не под предлагаемым именем Project1.vbp, а под уникальным именем, например, Калькулятор.vbp.

Второй этап - компиляция проекта в исполняемый файл

Как это сделать, вкратце описано в 2.13. Напоминаю ваши действия: **File→Make Project1.exe →** на экране возникает окно Make Project → укажите имя исполняемого файла (Калькулятор.exe) и если хотите, выберите папку, куда он будет записан. Если не станете выбирать, он по умолчанию будет записан в папку проекта (что и рекомендую) → жмите OK → файл готов.

Если исполняемый файл работает плохо, вам придется компилировать сначала. При этом в окне Make Project зайдите в **Options→ Compile** и проверьте, чтобы был установлен флажок Compile to Native Code. Затем зайдите в Advanced Optimizatons и убедитесь, что ни один флажок не поставлен.

Третий этап - создание инсталляционного пакета

Выйдите теперь из Visual Basic в Windows. **Инсталляционный** (или **дистрибутивный**) **пакет** создается при помощи мастера Package & Deployment Wizard, который должен был установиться на ваш компьютер при установке Visual Basic. Добираются до него через стартовое меню Windows. Там он находится недалеко от Visual Basic.

Мастером называется программа, позволяющая пользователю делать сложные вещи с минимальной затратой умственных сил. Обычно работа мастера состоит из нескольких шагов, на каждом из которых мастер задает пользователю вопросы. Переход к следующему шагу осуществляется кнопкой Next. Вернуться назад вы можете кнопкой Back.

Запустите мастер Package & Deployment Wizard. В появившемся окне мастера кнопкой Browse найдите файл вашего проекта Калькулятор.vbp и нажмите кнопку Package. Через несколько секунд работы в качестве следующего шага работы мастера появляется окно Package Type.

- *Package Type*. Некоторые шаги начинающие программисты могут без большого вреда пропустить. Такие шаги я объяснять не буду. Выделите верхнюю из двух строк. Теперь нажмите кнопку Next. Следующий шаг - окно Package Folder.
- *Package Folder*. Мастер спрашивает, в какую папку на вашем диске записывать весь дистрибутивный пакет. Рекомендую согласиться с предложением мастера, тогда он создаст внутри папки вашего проекта папку Package и запишет все в нее. Если вам это не нравится, жмите кнопку New Folder. Если нравится, нажмите кнопку Next. Мастер спросит, согласны ли вы на создание папки Package. Отвечайте Yes. Мастер перейдет к следующему шагу:
- *Included Files*. Здесь мастер показывает вам для вашего сведения список несжатых пока файлов, включаемых в инсталляционный пакет. Вы должны позаботиться о том, чтобы добавить в этот список загружаемые во время работы звуковые и другие файлы, речь о которых шла на первом этапе. Для этого нажмите кнопку Add. Поскольку в возникшем окне вам видны не все нужные файлы, настройте фильтр Files of type этого окна в значение All Files. Найдите теперь нужный файл и нажмите Open. Файл окажется в списке. Проделайте то же самое с остальными нужными файлами. Next.
- *Cab Options*. Большинство файлов дистрибутивного пакета будут сжаты внутри так называемых CAB-файлов. Эти файлы могут быть довольно большими, поэтому мастер спрашивает вас, что делать - много маленьких (Multiple cabs) или один большой (Single cab). Если вы хотите создать дистрибутивный пакет на дискетах, то установите переключатель в положение Multiple cabs и выберите размер дискеты. Тогда CAB-файлы будут создаваться такого размера, чтобы умещаться на дискете. Если вас не волнует размер CAB-файла, то выберите Single cab. Next.
- *Installation Title*. Мастер просит ввести заголовок устанавливаемой программы, который для вящей солидности будет крупным шрифтом отображаться на заднем плане в процессе инсталляции. Next.
- *Start Menu Items*. Настройка стартового меню Windows - в каком его месте и под каким заголовком будет фигурировать ваша программа. Попробуйте разобраться сами. Или просто нажмите Next.
- *Install Locations*. Next.
- *Shared Files*. Next.
- *Finished!*. Finish.
- Вслед за этим покажется рапорт мастера о проделанной работе. Можете закрыть его, не читая.
- Можете закрыть окно мастера. *Дистрибутивный пакет создан*.

Находясь в Windows, загляните в папку проекта. Вы обнаружите в ней папку Package. Загляните в нее. Там вы увидите один или несколько CAB-файлов, в которые ужался и ваш проект, и некоторые неведомые вам файлы, необходимые для инсталляции и нормальной работы вашего проекта на чужом компьютере. Кроме CAB-файлов вы увидите файлы Setup.exe и SETUP.LST. Если вы хотите посмотреть на инсталляционные файлы в несжатом виде, загляните в папку Support, находящуюся в той же папке Package. Папка Support не входит в состав инсталляционного пакета. *Инсталляционным пакетом являются все файлы, находящиеся в папке Package за исключением файлов из папки Support*! На дискеты или на жесткий диск чужого компьютера нужно копировать только инсталляционный пакет файлов.

Четвертый этап - инсталляция

Инсталляцию я уже вкратце описал в этом параграфе. Сначала в качестве эксперимента проведите инсталляцию на своем компьютере. Запустите Setup.exe.

Первое, что после запуска Setup.exe предлагает программа инсталляции, это выйти из всех запущенных программ. Сделайте это и нажмите OK.

В следующем окне под заголовком Directory вам предлагается адрес папки, в которую будет устанавливаться ваша программа. Если он вам не нравится, нажмите кнопку Change Directory и выберите нужную папку или впишите адрес в текстовое поле. Когда вы будете довольны адресом под заголовком Directory, щелкните над ним по большой квадратной кнопке.

Теперь вам предлагают выбрать или самому назвать программную группу для вашей программы в стартовом меню Windows. Послушайтесь и нажмите Continue.

Программа установки устанавливает файлы на жесткий диск и рапортует об успешном окончании работы. OK.

Теперь можете запускать свою программу из стартового меню.

Вам будет полезно заглянуть в папку, куда установилась ваша программа, и посмотреть, что там есть.

Чтобы установить вашу программу на компьютер вашего друга, перепишите дистрибутивный пакет на его компьютер и запустите Setup.exe. Или можете, не переписывая, просто вставить первую дискету с дистрибутивным пакетом (ту, на которой Setup.exe) в его дисковод и запустить установку. Что делать дальше, я уже описал.

Часть II. Программирование на Visual Basic – первый уровень

"Вы мне это прекратите!" - сказал Камноедов тем, кто рвался подняться на лифте. И мы прекращаем. Мы будем подниматься пешком.

Вариации на темы Стругацких

Кончилась наша сладкая жизнь. Вернее, в ней наметился большой перерыв. Действительно, калькулятор достался нам без особого напряжения. Работы мало - удовольствия много. Есть ли еще возможности почти без программирования быстренько получать что-нибудь "эдакое"? Есть, и их много. В популярных журналах, посвященных программированию, вы найдете немало программок из двух-трех строчек кода, скопировав которые в свой компьютер, вы получите на экране любопытные вещи. Только вот беда - хоть этих строчек всего две-три, понять их нет никакой возможности. Кто вы в таком случае - программист или переписчик иероглифов? И это тот гранит науки, та скала, которую не объедешь. Придется грызть.

Первые пять из девяти глав этой части - как раз те пять камней, которые нужно сгрызть, пять этажей, которые нужно преодолеть, чтобы подняться на первый в вашей жизни уровень программирования. Только грызите потихоньку, не надо пытаться откусить сразу большой кусок - можно поломать зубы.

Не все, ох не все пройдут сквозь огонь и воду этих пяти глав! Но с теми, кто пройдет, можно идти в разведку. Если вы с честью выйдете из схватки, наградой вам будут остальные четыре главы: графика, рисунки, узоры, движение, мультики, проект "Собственный будильник" и первая в вашей жизни собственная игра "Гонки".

К тому же - кто сказал, что первые пять глав скучные?! Ветвления, циклы, диалог с компьютером, измеритель шустрости, определитель - экстрасенс ли вы, пароль на калькулятор, движение объектов по экрану, поиск минимумов и максимумов - по-моему, это жутко интересно!

Глава 4. Переменные величины

Чтобы сделать в Visual Basic что-нибудь более существенное, чем примитивный калькулятор или примитивный плеер, нам нужно резко увеличить объем знаний о Visual Basic. Первая ступенька на лестнице набирания знания - переменные величины. Удовольствие больше для Холмса, чем для Брюса Ли. Но что может сделать Брюс Ли с Холмсом?

4.1. Переменные величины. Оператор присваивания

Понятие переменной величины вам известно из школьной математики. Пусть несколько лет назад ваш рост равнялся 130 см. Обозначим этот факт так: r=130. Теперь он равен 160 см, то есть r=160. Получается, что величина r изменилась.Поэтому она называется переменной величиной. Числа 130 и 160 называются **значениями** переменной величины r.

Любой язык программирования умеет обращаться с переменными величинами. Без них он был бы очень слаб и смог бы извлечь из компьютера только возможности калькулятора. Точно так же алгебра без переменной величины превратилась бы в арифметику. Однако, преимущества применения переменных величин нам откроются позже, а пока наша задача - к ним привыкнуть.

Что же мы можем делать с переменными величинами, программируя на Visual Basic? Прежде всего, мы можем задавать компьютеру значение той или иной переменной величины. Это мы можем сделать при помощи оператора, который называется **оператором присваивания**. Так, если мы хотим сказать, что a имеет значение 6, то должны просто записать **a=6**. Запись **a=6** и называется оператором присваивания. Говорят, что величине a присваивается значение 6. С момента выполнения оператора a=6 компьютер будет помнить, что a равно шести.

В старых версиях Бэйсика оператор присваивания для понятности записывали так:

Let a = 6

что означало

"Пусть a = 6".

Мы широко пользовались оператором присваивания в части I. Например, в операторе

Form1.BackColor = vbRed

мы присваивали свойству "цвет" нашей формы значение "красный". Пока, чтобы не рассеивалось внимание, мы в операторе присваивания не часто будем касаться объектов, их свойств и значений. Ограничимся переменными, имеющими численные значения.

После выполнения следующего фрагмента программы

```
a=2*3+4
b=a
y=a+b+1
```

компьютер будет знать, что *a* равно 10, *b* равно 10, *y* равно 21. Итак, при помощи оператора присваивания вы можете и вычислять тоже. Мы видим, что справа от знака равенства в операторе присваивания можно писать не только числа, но и переменные величины, и **выражения**. Выражения в Visual Basic могут быть разных типов, но об этом мы будем говорить позже. Пока под выражением будем понимать **арифметическое выражение**, то есть такое, к которому вы привыкли в школьной математике. Здесь это были *2*3+4*, *a*, *a+b+1*. Любое арифметическое выражение имеет численное значение.

Не надо пока проверять это на компьютере. Пока только читайте.

Теперь чуть-чуть о свойствах объектов. Выражение может включать в себя не только переменные величины, но и свойства объектов, имеющие подходящее значение. Например,

Y = a + b + Form1.Width.

В левой части оператора присваивания также может стоять не только переменная, но и свойство объекта. Например,

Form1.Width = a + b + y

В последнем случае действие этого оператора вы сразу увидите на экране.

Еще пара примеров:

ФРАГМЕНТ ПРОГРАММЫ	ЧТО ЗАПОМНИТ КОМПЬЮТЕР
v = -2+10 : h = 10*v : s = v+h+0.01	v - 8 h - 80 s - 88.01
t = 0 : n = 2*t+40 : z = -n	t - 0 n - 40 z - -40

Задание 6: Определите устно, какое значение будет присвоено переменной t после выполнения фрагмента:

k=1+2 : s=2*k : t=6-s ?

Необходимо помнить, что если слева от знака равенства стоит переменная величина, то Visual Basic выполняет оператор

присваивания "в уме". Имеется в виду, что результат его выполнения не отображается на экране, а только запоминается. Как же увидеть значение переменной величины? Создайте в проекте текстовое поле и дополните программу последнего примера:

```
t = 0  :  n = 2*t+40  :  z = -n  :  Text1.Text=z
```

Теперь значение величины z, равное -40, вы будете видеть в текстовом поле.

Однако, для учебных целей и для отладки программ удобней пользоваться оператором **Debug.Print**. Так оператор

Debug.Print t, n, z

покажет нам значения всех трех переменных величин. При этом не нужно создавать никаких текстовых полей, результат появляется в услужливо возникающем окне Immediate.

А вот теперь садитесь за компьютер. Создайте новый проект. Ничего реального мы из него делать не будем, он будет служить только для упражнений с переменными величинами. Разместите в нем кнопку, а в окно кода запишите такую программу:

```
Private Sub Command1_Click()
    a = 2 * 3 + 4
    b = a
    y = a + b + 1
    Debug.Print a, b, y, b + y
End Sub
```

Запустите проект. Щелкните по кнопке Command1 - в окне Immediate должны появиться 4 числа:

```
10        10        21        31
```

Вы видите окно Immediate? Если нет, то - **View→Immediate Window**.

Какая польза от переменных величин?

Самая крохотная и простая польза в том, что с их помощью удобно решать несложные вычислительные задачи. Например, даны стороны прямоугольника: a=27018, b=3954. Вычислить его площадь и периметр (напомню, что периметр - это сумма длин сторон прямоугольника).

Создайте проект с кнопкой. Задачу решает следующая программа:

```
Private Sub Command1_Click()
    a = 27018
    b = 3954
    S = a * b                'Площадь
    p = 2 * a + 2 * b        'Периметр
    Debug.Print S, p
End Sub
```

После запуска и щелчка по кнопке вы увидите в окне Immediate следующие два числа:

```
106829172    61944
```

Первое из них - площадь, второе - периметр. Без применения переменных величин операторы получились бы более громоздкими:

```
    S = 27018 * 3954              'Площадь
    p = 2 * 27018 + 2 * 3954      'Периметр
```

что особенно заметно в больших программах. Есть и масса других преимуществ, которые вы почувствуете в следующих главах.

Продолжаем о переменных величинах

Нужно иметь в виду, что в операторе присваивания слева от знака равенства не может стоять число или выражение. Можно писать c=34, но нельзя писать 34=c. Можно писать z=f-v+990 , но нельзя писать f-v+990=z. Правило это принято вот почему. Оператор присваивания устроен так, что сначала смотрит или вычисляет, какое значение имеет правая часть, а затем присваивает это значение тому, что стоит в левой части. То, что справа от знака равенства, присваивается тому, что слева от знака равенства, а не наоборот. Нет смысла присваивать значение числу или выражению.

Обратите внимание еще на один важный момент. Когда школьник видит выражение (например, d+2d), он не обязательно его вычисляет. Он может его преобразовать или, скажем, упростить (получив 3d). Компьютер же, видя выражение, сначала его, может быть, и упростит, но затем обязательно вычислит. А для этого он должен знать численные значения входящих в него величин (в нашем случае это величина d). Таким образом, вычисляя правую часть оператора присваивания (например, y=a+b+1), компьютер должен обязательно заранее знать, чему равны переменные, из которых эта правая часть состоит (в нашем случае это a и b). Ответственность за это знание лежит полностью на программисте. Пусть забывчивый программист записал такой фрагмент: . . . a=10 : y=a+b+1 . . . , нигде в программе не придав b никакого значения. Естественно, при вычислении выражения a+b+1 компьютер не будет знать, чему равно b. В такой ситуации разные языки программирования поступают по-разному. Некоторые "злые" языки просто отказываются вычислять выражения, "вредный" Турбо-Паскаль может подставить вместо b какую-нибудь ерунду (и молчок, между прочим), Visual Basic же добрый, он подставляет вместо b нуль.

Проверим. Заставим Visual Basic в нашем последнем проекте про прямоугольник не выполнять оператор a = 27018. Для этого его можно просто стереть, но удобнее просто поставить перед ним кавычку, тогда Visual Basic подумает, что это комментарий и выполнять не будет:

```
'a = 27018
```

Вот результат:

```
0        7908
```

Действительно, все выполнилось так, как если бы a было равно нулю.

4.2. Объявление переменных величин

Когда на бал к графине N приезжает герцог M, то слуга объявляет на весь зал: "Герцог M!", отчего все гости сразу узнают, что перед ними герцог, а не какой-нибудь там баронишко, и обращаются к нему соответственно. Все хорошо, все прилично. Если бы слуга проморгал и забыл объявить герцога, то в большинстве случаев тоже ничего страшного бы не произошло. Однако иногда все же могли бы приключиться досадные недоразумения. Например, к герцогу могли бы обратиться так: "Эй, человек, подай-ка лимонад!", а это уже скандал!

Бал - это проект в режиме работы. Гости - это переменные. Слуга - это вы. До сих пор вы никого не объявляли и тем не менее все шло как по маслу. Это потому что Visual Basic умный, он по глазам определяет, кто герцог, а кто нет. Однако не на каждом балу глаза говорят правду. Прежде чем убедиться в этом, изучим еще один полезный и приятный элемент Visual Basic:

InputBox

Как мы можем задать компьютеру какую-нибудь величину? Оператором присваивания - раз. Введя ее в текстовое поле, как мы делали в калькуляторе - два. Есть еще один удобный и приятный способ - **InputBox**.

В вашем проекте о прямоугольнике вы можете хоть сто раз нажимать на кнопку - результаты все время будут одинаковые. Это потому что исходные данные a=27018 и b=3954 никто не меняет. Скучно. Хорошо бы компьютер при нажатии на кнопку каждый раз спрашивал нас, чему равны стороны прямоугольника. А мы бы ему отвечали. А уж потом он вычислял.

Для этого нам нужно слегка изменить программу:

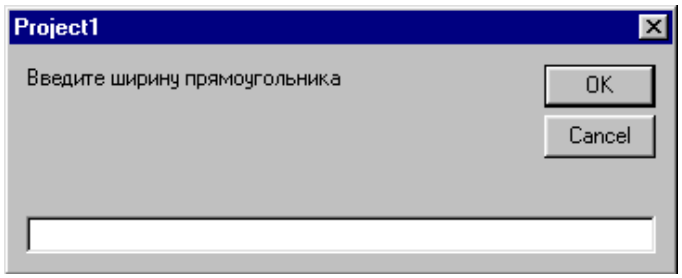
```
Private Sub Command1_Click()
    a = InputBox("Введите ширину прямоугольника")
    b = InputBox("Введите высоту прямоугольника")
    S = a * b
    p = 2 * a + 2 * b
    Debug.Print S, p
End Sub
```

Как видите, заменены первые две строки: a=27018 и b=3954.

Конструкция

a = InputBox("Введите ширину прямоугольника")

есть оператор присваивания и означает она приказ компьютеру вывести сначала на экран вот такое окно:



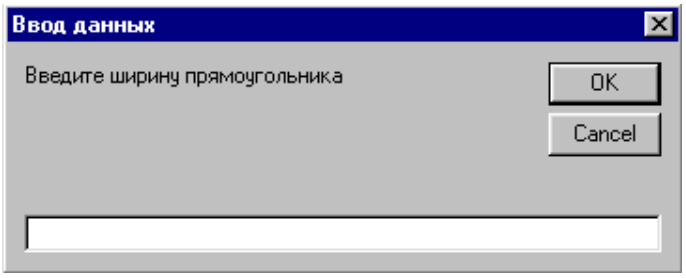
после чего человек вводит в белое текстовое поле этого окна любое число и нажимает OK. После этого компьютер присваивает переменной **a** введенное значение и продолжает работу, то есть переходит к выполнению следующего оператора.

Запустите проект и посмотрите, как он работает.

Если вам не нравится строка Project1 в заголовке окна, то вы можете задать свою строку, так дополнив оператор:

a = InputBox("Введите ширину прямоугольника", "Ввод данных")

Результат:



Почему плохо не объявлять переменные величины

Посмотрим, где Visual Basic дает сбой, если не объявлять переменные величины. Рассмотрим простую задачу: Даны три стороны треугольника. Вычислить его периметр.

Создаем новый проект с кнопкой для задачи о треугольнике. По аналогии с процедурой о прямоугольнике легко пишем процедуру для треугольника:

неправильная процедура:

```
Private Sub Command1_Click()
    a = InputBox("Введите первую сторону треугольника")
    b = InputBox("Введите вторую сторону треугольника")
```

```
    c = InputBox("Введите третью сторону треугольника")
    p = a + b + c
    Debug.Print p
End Sub
```

Запускаем проект, вводим стороны - 2, 3, 4 и какой результат мы видим? Вместо 9 мы видим 234! Почему? Здесь тот же случай, что и с калькулятором, там было то же самое, пока мы не применили Val. Почему? *Первое*: Visual Basic считает содержимое текстового поля окна InputBox (точно так же, как и содержимое обычного текстового поля) текстом, а не числом, если даже оно состоит из цифр. А раз так, то и переменные a, b, c, берущие свое значение из текстового поля, он тоже считает текстовыми, а не числовыми. Не пей из копытца - козленочком станешь! *Второе:* Visual Basic знает, что по правилам языка знак **+** имеет несколько смыслов в зависимости от того, к чему он применяется. В частности, при работе с текстом это не приказ складывать, а приказ выстраивать в цепочку. А раз так, то и результат получился тот, что мы видели.

Наши необъявленные переменные, как голодные дворняги, готовы есть все, что им дают. Введите вместо трех чисел три таких текста: *Ко*, *ро*, *бочка*. Результат - *Коробочка*.

Почему же тогда в примере о прямоугольнике все прошло нормально? Потому что там было еще умножение. Visual Basic знает, что по правилам языка знак умножения * не имеет смысла при работе с текстом. Значит (догадывается умный и услужливый Visual Basic) мой господин - человек не хочет, чтобы переменные a, b, c были текстовыми. Значит быть им числовыми! Проверьте: подсуньте компьютеру вместо чисел коробочку - Visual Basic запротестует.

В примере о прямоугольнике Visual Basic догадался правильно, а в примере о треугольнике ума не хватило. Ну что, теперь везде при сложении применять Val? Не хочется, программы будут получаться громоздкими. Да и разве все дело только в сложении? Как узнаешь, где в следующий раз соломки подстелить?

Так вот, чтобы не перенапрягать умственные способности Visual Basic и предохранить себя от неприятностей, нужно все переменные величины объявлять!

Как объявлять переменные величины

Добавим в окно кода нашего проекта, на самый верх, над заголовком процедуры четыре строки. Получится:

правильная процедура:

```
Dim a As Integer
Dim b As Integer
Dim c As Integer
Dim p As Integer
Private Sub Command1_Click()
    a = InputBox("Введите первую сторону треугольника")
    b = InputBox("Введите вторую сторону треугольника")
    c = InputBox("Введите третью сторону треугольника")
    p = a + b + c
    Debug.Print  a, b, c,  p
End Sub
```

Объявление

Dim	**a**	**As**	**Integer**

переводится так:

Переменная величина	**a**	**как**	**целое число**

то есть вы обязуете Visual Basic обращаться с переменной величиной a как с целым числом и ничем иным. Говорят, что "переменная a имеет **тип** Integer". Аналогично объявлены и переменные b, c, p. Объявления будем пока писать выше процедур. Подробнее о местах, где можно делать объявления, написано в 17.2.

Запустите проект. Проверьте его работу.

Вместо четырех строк

```
Dim a As Integer
Dim b As Integer
Dim c As Integer
Dim p As Integer
```

можно для краткости написать одну:

```
Dim a As Integer, b As Integer, c As Integer, p As Integer
```

Но если вы захотите написать еще короче:

```
Dim a, b, c, p As Integer
```

то Visual Basic неправильно вас поймет.

Типы данных

Много ли прав на нашем балу у барона Integer? Посмотрим. Попробуем ввести дробное значение первой стороны треугольника - *2,3*. Именно так - с запятой, а не точкой. Посмотрим результат. Visual Basic считает a целым числом 2. А теперь - *2,6*. Visual Basic считает a целым числом 3. Все правильно. Наш барон высокомерно отказывается быть дробным числом и поэтому любое число, которое мы ему предлагаем, округляет до целого.

Предположим, мы этим недовольны. Мы желаем, чтобы все числа могли быть дробными. Пожалуйста! Для этого достаточно всех гостей объявить князьями. Вот так:

```
Dim a As Single
Dim b As Single
Dim c As Single
Dim p As Single
Private Sub Command1_Click()
    a = InputBox("Введите первую сторону треугольника")
    b = InputBox("Введите вторую сторону треугольника")
    c = InputBox("Введите третью сторону треугольника")
    p = a + b + c
    Debug.Print  a, b, c,  p
End Sub
```

Объявление

Dim	**a**	**As**	**Single**

переводится так:

Переменная величина	**а**	**как**	**десятичная дробь обычной точности**

то есть вы обязуете Visual Basic обращаться с переменной величиной а как с десятичной дробью и ничем иным. Десятичная дробь - это любое число, которое может иметь целую и дробную часть (например, 27,3908), в частности это и целое число. Об обычной точности попозже. Говорят, что "переменная а имеет **тип Single** ".

Законный вопрос: зачем нам нужен тип Integer, когда тип Single "гораздо лучше"? Ответ: Не во всем и не всегда он лучше, иначе не стали бы его создавать вообще. Подробнее об этом - в 4.5.

На бал к Visual Basic вхожи не только бароны и князья, то есть типов данных в Visual Basic довольно много и каждый полезен по-своему. Рассматривать новые типы я буду по мере надобности. А сейчас отмечу главное - в солидном проекте не объявлять переменные неприлично! Поэтому переменные величины нужно объявлять всегда. А чтобы по нашей человеческой рассеянности какая-нибудь переменная не проскочила на бал необъявленной, поручим компьютеру не пускать таких на бал, то есть прикажем Visual Basic выдавать соответствующее сообщение при нашей попытке запустить проект с необъявленными переменными. Для этого - **Tools→Options→Editor→Require Variable Declaration**. Отныне в окне кода любого создаваемого вами проекта будет появляться строка Option Explicit, что означает требование обязательно объявлять переменные этого проекта. В общем, "*Посторонним В!…*" Тем не менее, свобода есть свобода, и вы всегда можете с песней "*Мама - Анархия!*" в каком-нибудь проекте стереть строку Option Explicit.

Я сам в примерах этой книги поначалу все, что можно, буду объявлять, но затем для экономии "места и смысла" перестану это делать. Не берите с меня пример!

Вот еще причина, по которой удобно использовать Option Explicit. Как по вашему, что напечатал у меня следующий фрагмент:

```
x = 2
Debug.Print   x
```

Если вы думаете, что 2, то ошибаетесь. Ничего не напечатал. Потому что в первой строке присутствует латинская буква "икс", а во второй строке я нечаянно написал русскую букву "хэ". Я был в недоумении. Не помогло и объявление переменной "икс" (не помогло бы, между прочим, и объявление "хэ"). Visual Basic не нашел в программе никаких ошибок. И только применение Option Explicit дало намек на истинную причину ошибки.

Вот какие типы данных вы найдете в этой книжке:

- Integer, Single -4.2
- Long, Double, Currency -4.5
- String -4.7
- Date -11.1
- Boolean -11.3
- Перечислимый -11.3
- Массивы -Глава 13
- Variant -14.1
- Пользовательский -14.2
- Коллекции -14.3
- Объекты -14.3, Глава 20

Переменные величины и память

Полностью механизм работы переменных величин не понять, если не узнать, как они хранятся в оперативной памяти компьютера.

Оперативнвя память нужна компьютеру для того, чтобы хранить во время выполнения программы саму эту программу и данные, с которыми программа работает.

Как устроена оперативная память (или просто память)? Представьте себе тетрадный листок в клеточку. В каждую клетку вы имеете право записать карандашом какую-нибудь букву или цифру или знак + или вообще любой символ, который можно найти на клавиатуре. А можете и стереть ластиком и записать другой символ. Много ли букв можно записать на листе? Ровно столько, сколько на нем клеток.

Оперативная память компьютера устроена аналогично такому листу. Только клеточек в ней гораздо больше. Каждая клеточка называется **байтом**. Для запоминания слова КОШКА понадобится 5 байтов. Подробнее о работе памяти см. в Приложении 1.

Значения переменных величин во время выполнения программы компьютер хранит в оперативной памяти (потому что диски для этого слишком медлительны). Для этого он предварительно под каждую переменную отводит в памяти место. При этом компьютер "рассуждает" так: Раз в программе упомянута переменная величина, значит она в каждый момент времени будет иметь какое-то значение, которое, хочешь не хочешь, надо помнить. Лучше, чтобы не спутаться, отвести в памяти определен-

ное место для запоминания текущего значения каждой переменной величины. Будем называть место, отведенное в памяти под данную переменную, **ячейкой**. Представьте себе город, в котором имеются разные по размерам комнаты и каждому жильцу отведена ровно одна комната. Это тот самый случай.

Теперь о размере комнат. В городе размер комнат меряют в квадратных метрах, в памяти размер ячейки выражают в байтах. Герцогу нужна более вместительная комната, чем барону. Переменной типа Single нужна более вместительная ячейка, чем переменной типа Integer. Так оно и есть на самом деле: под переменную типа Integer в памяти отводится ячейка размером 2 байта, а под переменную типа Single - 4 байта. Только не думайте, что переменная типа Single не может быть длиннее 4 цифр, компьютер для записи чисел в ячейки использует более компактную систему, чем для записи символов.

Что делает оператор присваивания с памятью

Я только что рассказывал о работе оператора присваивания, используя такие выражения, как "компьютер знает", "компьютер помнит". Но нам необходимо более строгое понимание работы этого оператора, понимание "ближе к железу".

Рассмотрим пример программы:

```
Dim a As Integer
Dim b As Integer
Dim y As Integer
Private Sub Command1_Click()
    a = 10   :    b = 6    :    y = a + b + 1
    Debug.Print   y + 200
End Sub
```

В программе встречаются три переменные, поэтому все они объявлены. Компьютер отведет для них в памяти три двухбайтовые ячейки. Вот как будет работать оператор присваивания:

Выполняя оператор присваивания (например, y=a+b+1), компьютер сначала смотрит на его правую часть (a+b+1). Если в ней встречаются переменные (в нашем случае это a и b), то компьютер перед вычислением ищет их значения в отведенных под них ячейках памяти (и находит там 10 и 6, так как их туда записали операторы *a=10* и b = 6), подставляет эти значения в правую часть и вычисляет ее. Затем вычисленное значение (17) компьютер записывает в ячейку памяти, отведенную под переменную, поставленную в левой части (y).

Таким образом, когда мы говорим "Компьютер запомнил, что a равно 2", мы подразумеваем "Компьютер записал в ячейку памяти, предназначенную для a, число 2".

А теперь рассмотрим, как будут заполняться информацией ячейки a,b,y в процессе выполнения нашей программы. В самом начале выполнения программы в них находятся нули. Первым выполняется оператор a=10. Согласно только что приведенному определению оператора присваивания в ячейку a будет записано число 10. Затем выполняется оператор b= 6 и в ячейке b появляется шестерка. Затем выполняется оператор y= a+b+1. Компьютер смотрит, что находится в ячейках a и b, видит там 10 и 6, подставляет их в выражение a+b+1, получает 17 и записывает в ячейку y. Наконец выполняется оператор Debug.Print y + 200. Компьютер заглядывает в ячейку y, видит там 17, вычисляет 17+200 и выводит 217 в окно Immediate.

Схематически этот процесс можно изобразить так:

ПОРЯДОК ИСПОЛНЕНИЯ ОПЕРАТОРОВ	ЧТО НАХОДИТСЯ В ЯЧЕЙКАХ ПАМЯТИ			ЧТО ВИДИМ В ОКНЕ Immediate
	Ячейка для a	Ячейка для b	Ячейка для y	
a= 10	10	0	0	
b= 6	10	6	0	
y= a+b+1	10	6	17	
Debug.Print y + 200	10	6	17	217

Теперь мы можем также уточнить работу оператора Debug.Print :

Если в операторе Debug.Print встречаются выражения с переменными величинами, то Visual Basic находит в памяти значения этих величин, подставляет их в выражения, вычисляет выражения и результат выводит на экран. В память же ничего не записывается.

Задание 7: Ответьте устно, что произойдет, если поменять местами операторы b= 6 и y= a+b+1?

Режим прерывания. Пошаговый режим выполнения программы

Компьютер выполняет программу со страшной скоростью. Не успели мы нажать кнопку - а результаты уже получены. Пока все идет хорошо, это нас восхищает. Когда же результаты не те, что надо (а ошибок в программе мы не видим), это начинает раздражать. Хотелось бы, чтобы компьютер работал помедленнее и хорошо бы как-нибудь подсмотреть во время работы значения переменных в памяти, это помогло бы найти ошибку. Режим прерывания для этого и предназначен.

Запустите предыдущий проект на выполнение не кнопкой Start, как вы привыкли, а клавишей **F8** на клавиатуре. Это горячая клавиша для **Debug→Step Into**. Проект начнет выполняться как обычно. Нажмите, как водится, кнопку Command1. И тут вы почувствуете разницу. Вместо того, чтобы полностью выполниться и показать результат, проект остановится на самой первой строке процедуры, а именно, на Private Sub Command1_Click(), в знак чего эта строка окажется подсвечена желтым цветом. Итак, только-только начав выполняться, наш проект замерз до тех пор, пока следующее нажатие **F8** его не разморозит. "Мгновение остановилось".

Интересно, чему во время остановки равны значения переменных в памяти компьютера? Для того, чтобы узнать это, достаточно поместить мышиный курсор на обозначение переменной в тексте процедуры в окне кода. Как и положено, "на табло пока одни нули".

Еще раз **F8**. Ничего не происходит, только полоса подсветки прыгает на следующую строку. В ней находится первый исполняемый оператор вашей процедуры – a = 10.

F8. Visual Basic выполняет a = 10, а следующий оператор подсвечивается. Проверьте, чему сейчас равны a, b, y в памяти.

F8. Visual Basic выполняет b = 6, а следующий оператор подсвечивается. Проверьте, чему сейчас равны a, b, y в памяти.

Итак, правило простое – при нажатии на **F8** Visual Basic выполняет очередной оператор программы и подсвечивает тот оператор, которому предстоит быть выполненным.

Обратите внимание, что сменился режим Visual Basic. Мы знаем пока два режима: режим проектирования [design] и режим работы [run]. Теперь в заголовке главного окна Visual Basic вы видите слово **[break]**. Это означает, что Visual Basic сейчас находится в **режиме прерывания**. Можно сказать, что режим прерывания - это режим работы с остановками. Вы можете указать Бэйсику, на каких именно операторах останавливаться. О том, как это сделать, написано в Глава 7. Когда же вы нажимаете на **F8**, вы приказываете Бэйсику останавливаться на каждом операторе. Такая разновидность режима прерывания называется **пошаговым режимом**.

F8. Visual Basic выполняет y= a+b+1, а следующий оператор подсвечивается. Проверьте, чему сейчас равны a, b, y в памяти.

F8. Visual Basic выполняет Debug.Print y + 200, а следующий оператор подсвечивается. В окне Immediate возникает число 217.

F8. Подсветка уходит, так как процедура выполнена. Можно снова жать на кнопку Command1.

Замечание об отладке в пошаговом режиме: При отладке в пошаговом режиме вам часто нужно на экране видеть одновременно окно кода, окно Immediate и саму форму. Но при нажатии на F8 то одно, то другое часто пропадает из вида, будучи закрыто другими окнами. Чтобы этого не происходило, еще до начала отладки расположите окно кода и окно Immediate так, чтобы они не перекрывали друг друга, а главное окно Visual Basic уменьшите до части экрана. Затем после запуска отладки форму перетащите на освободившуюся часть экрана. Тогда окно Visual Basic не будет перекрывать форму и проблема исчезнет.

Итак, компьютер сделал все то, что сделал бы при нажатии Start, но только в медленном темпе.

В любой момент пошагового выполнения программы вы можете вместо **F8** нажать Start, которая по такому случаю переименовывается в Continue, и программа продолжит выполняться в обычном режиме.

Оператор присваивания меняет значения переменных величин

Пока я не рассматривал программы, в которых переменные меняют свою величину. Теперь настало время такую программу рассмотреть:

```
Dim k As Integer
Private Sub Command1_Click()
   k = 10: Debug.Print k:        k = 25: Debug.Print k:        k = 4: Debug.Print k
End Sub
```

Запишем схематически процесс изменения информации в ячейке k:

ПОРЯДОК ИСПОЛНЕНИЯ ОПЕРАТОРОВ	ЧТО НАХОДИТСЯ В ЯЧЕЙКЕ ПАМЯТИ k	ЧТО ВИДИМ В ОКНЕ Immediate
k=10	10	
Debug.Print k	10	10
k=25	25	
Debug.Print k	25	25
k=4	4	
Debug.Print k	4	4

Как видите, в процессе работы программы содержимое ячейки k меняется. Так, при выполнении оператора k=25 там вместо значения 10 появляется 25. А куда же девается десятка? Она стирается, то есть компьютер забывает ее безвозвратно. Здесь действует общий принцип работы всех компьютеров:

Если в какое-нибудь место памяти или диска записывается новая информация, то старая информация, записанная там раньше, автоматически стирается, даже если она кому-то и нужна.

Раз теперь вместо 10 в ячейке k находится 25, то оператор Debug.Print k печатает уже 25. (Слова "печатает" и "Print" устарели, они остались от тех времен, когда мониторов еще не было, а вместо них были принтеры. Когда я по старинке говорю, что информация печатается, я подразумеваю, что она появляется на мониторе в том или ином окне.) Следующий оператор k=4 запишет на место 25 четверку, а Debug.Print k ее напечатает. Проверьте.

А что напечатает следующая программа?

```
Dim f As Integer
Private Sub Command1_Click()
   f = 30:    f = f + 4:     Debug.Print f
End Sub
```

Оператор f=30 запишет в ячейку f число 30. А что сделает странный оператор f=f+4 ? Не думайте, что это уравнение или что-нибудь в этом роде. Это оператор присваивания, а значит и выполнится он согласно определению оператора присваивания, то есть сначала вычислит правую часть f+4 , подставив туда вместо f его значение, взятое из ячейки, и получит 34.

Затем число 34 будет записано в ячейку, отведенную под переменную, обозначенную в левой части, то есть опять в ячейку f. При этом старое значение 30 будет стерто. Debug.Print f напечатает 34.

Таким образом, оператор f=f+4 просто увеличивает число в ячейке f на четверку или, другими словами, увеличивает f на 4.

Зачем это нужно? Узнаете позже. Очень даже нужно.

Задание 8:

Определите без компьютера, что будет напечатано при выполнении следующих фрагментов программ:

- a=100 : a=10*a+1 : Debug.Print a
- a=100 : a= -a : Debug.Print a
- a=10 : b=25 : a=b-a : b=a-b : Debug.Print a, b

4.3. Еще об именах

Как правильно дать имя объекту, мы говорили в 2.4. Переменные же величины мы привыкли обозначать буквами (a, s, d ...). Большинство языков программирования, в том числе и Visual Basic, позволяет обозначать переменные именами. Вот два равносильных фрагмента программы:

a=3;
b=4-a;
Debug.Print a, b+50
	Summa=3;
	ROBBY=4-Summa;
	Debug.Print Summa, ROBBY+50

В том и другом случае будут напечатаны числа 3 и 51. Очевидно, компьютеру все равно, как мы обозначаем переменные величины или объекты, в смысл имен он не вдумывается и не удивляется, что переменная Summa никакой суммой не является, а просто числом 3.

Многие авторы не рекомендуют использовать в именах русские буквы, говорят, что это опасно. Лично я в связи с этим практически ни разу с большими проблемами не столкнулся. Мой совет: если у вас с английским все ОК, то избегайте русских букв, в противном случае ничего страшного, если вы будете все именовать по-русски.

Примеры правильной записи имен:

a
polnaja_Summmma
tri_plus_dva
s25
a1b88qqQQQQQQQQQQQQ
oshibka
Это_не_имя

Примеры неправильной записи имен:

polnaja summa - содержится символ (пробел), не являющийся буквой, цифрой или знаком подчеркивания
Это правильное имя- содержится символ (пробел), не являющийся буквой, цифрой или знаком подчеркиван
2as - начинается с цифры
Domby&Son - содержится символ & , не являющийся буквой, цифрой или знаком подчеркивания

Visual Basic игнорирует в именах разницу между строчными и прописными буквами. Так, для него **Summa** и **sUmmA** - одно и то же имя. И он присматривает за тем, чтобы в окне кода они были написаны одинаково (но не за всеми русскими именами он так присматривает).

4.4. Математика. Запись арифметических выражений

Простые арифметические вычисления лучше делать на калькуляторе, чем на компьютере, а вот сложные - наоборот. В этом разделе вы научитесь правильно вводить в компьютер сложные формулы. Если вы - школьник не самых старших классов, то не все, что здесь написано, будет вам понятно. Не огорчайтесь, при дальнейшем чтении непонятные вещи вам не понадобятся.

Действия арифметики обозначаются в Visual Basic следующим образом:

ДЕЙСТВИЕ	РЕЗУЛЬТАТ	СМЫСЛ
2 **+** 3	5	плюс
4 **-** 1	3	минус
2 ***** 3	6	умножить
10 / 2	5	разделить
17 \ 5	3	целочисленное деление (17 делится на 5, получается 3, в остатке 2)
17 **Mod** 5	2	остаток от целочисленного деления
2 **^** 3	8	2^3 (два в кубе) - возведение в степень

На уроках математики мы привыкли писать ab+cd , подразумевая: a умножить на b плюс c умножить на d. В Visual Basic это выражение мы обязаны писать так: a*b+c*d. Иначе компьютер подумает, что нужно к переменной, имеющей имя ab, прибавить переменную, имеющую имя cd. Во избежание двусмысленности знак умножения положено писать всегда, в том числе и перед скобками. Например, a * (b+c).

Скобки. Ввиду того, что с клавиатуры всю информацию приходится вводить символ за символом в одну строку, ввод двухэтажных выражений, таких как

$$\frac{a+1}{b+1}$$

очень затруднен. Поэтому для обозначения деления и выбрана косая черта. Это выражение на Visual Basic положено записывать так: (a+1)/(b+1). Если бы мы не поставили скобок, то выражение получилось бы таким a+1/b+1, а это неправильно, так как компьютер, как и мы, всегда перед сложением и вычитанием выполняет умножение и деление, поэтому в последнем случае он бы сначала разделил 1 на b , а затем к результату прибавил a и 1.

Вопрос: когда в выражениях можно ставить скобки? Ответ: всегда, когда у вас возникают сомнения в правильной очередности действий. Лишняя пара скобок не помешает. Пример: записать на Visual Basic выражение:

$$\frac{1+\frac{a}{2+ab}}{3+a}\times b$$

Его можно было бы записать так:

(1 + a / (2+a*b)) / (3+a) * b .

Разберитесь в этой записи. К сожалению, в выражениях разрешается писать только круглые скобки. Квадратные и фигурные запрещены. От этого сложные выражения с большим количеством скобок на глаз воспринимаются с трудом, так как трудно для конкретной скобки увидеть ее законную пару. В этом случае я могу посоветовать идти "от малого к большому", то есть сначала заметить самые малые из взятых в скобки фрагменты выражения (у нас это 3+a и 2+a*b). После этого будет уже легче заметить более крупные фрагменты, такие как 1 + a / (2+a*b) , и т.д.

Разобрались? Эта запись меня совсем не удовлетворяет, так как мы не знаем, что Visual Basic будет делать раньше - делить (1 + a / (2+a*b)) на (3+a) или умножать (3+a) на b. А от этого зависит результат. Добавим для верности пару скобок:

((1 + a / (2+a*b)) / (3+a)) * b

Теперь все в порядке.

Запись десятичных дробей. Путаница с точками и запятыми. Почти во всех языках программирования и уж, конечно, в Visual Basic, в десятичных дробях принято вместо запятой ставить точку. Пример: 62.8 - шестьдесят две целых восемь десятых. Однако, если помните, при вводе дробей в InputBox Visual Basic требовал запятую, да и результаты он выводит с запятой. В чем дело? Visual Basic, являясь приложением Windows, частично воспринимает от нее привычку пользоваться в России запятой. Особой проблемы тут нет. Используйте метод "научного тыка" - пользуйтесь точкой, а если Visual Basic жалуется или начинает делать что-то не то, тогда меняйте ее на запятую.

Математические функции. Кроме нескольких действий арифметики Visual Basic может выполнять и другие математические действия, например, извлечение квадратного корня. На компьютере нет клавиши со значком $\sqrt{\ }$, поэтому в Visual Basic имеется специальная функция - Sqr. Например, корень из 25 обозначается так - Sqr(25), корень из a+b так - Sqr(a+b). Здесь Sqr - сокращение от английского выражения Square root - квадратный корень. То, из чего нужно извлечь корень, записывается в скобках.

Приведу неполный список математических функций Visual Basic:

ДЕЙСТВИЕ	РЕЗУЛЬТАТ	СМЫСЛ
Sqr (25)	5	корень квадратный
Round (5.82716)	6	округление до целых
Round (5.82716, 3)	5.827	округление до трех знаков после точки
Abs (-20)	20	абсолютная величина (модуль) числа
Fix (3.98)	3	целая часть числа (дробная часть отбрасывается)
Fix (-3.98)	-3	
Int (3.98)	3	наибольшее целое число, не превышающее выражение в скобках
Int (-3.98)	-4	
Rnd	0.73088	случайное число из диапазона (0 - 1)

Кроме этого, имеются функции Sin, Cos, Tan *(что означает тангенс)*, Atn *(что означает арктангенс)*, Exp, Log и др.
Работа со случайными величинами описана в 5.4, 9.9.

Примеры:

Выражение	**(2+1)^2**	при вычислении даст **9**
Выражение	**1+ (2+8)^3**	при вычислении даст **1001**
Выражение	**1+Abs(5-8)**	при вычислении даст **4**
Выражение	**2^4+Sqr(35+1)**	при вычислении даст **22**
Выражение	**Sqr (8+Int(41.5))**	при вычислении даст **7**
Выражение	**21 \ (Round (2.54+1))**	при вычислении даст **5**

Задание 9: Определите без компьютера, что напечатает данный фрагмент программы:

```
a = (2^2+1) * (20- (2^2)^2) - 11 :  b=11 \ (a-4)  :   Debug.Print  a^2 + b - 1
```

4.5. Типы данных и точность вычислений

Если вы считаете, что компьютер все вычисления выполняет абсолютно точно, то вы ошибаетесь. Компьютер всего лишь очень точен. Очень, а не абсолютно. В этом вы скоро убедитесь.

Integer и Long - целые числа

Создайте проект с кнопкой и введите такую программу:

```
Dim a As Integer
Dim b As Integer
Private Sub Command1_Click()
    a = 100
    b = 1
    Debug.Print a, b, a + b
End Sub
```

Работает она нормально. Посмотрим, насколько большие числа способна воспринимать наша программа. Заменим b=1 на b=40000. Visual Basic выдает сообщение об ошибке "Overflow", что означает "Переполнение ячейки". В чем дело?

Как вы уже знаете, в Visual Basic принято правило, что если человек объявил переменную, как Integer, то он разрешает ей принимать значения только целых чисел. Число типа Integer занимает в памяти два байта. Значит, под переменные a и b компьютер отводит в памяти ячейки по два байта каждая. Два байта - это маленький объем памяти и уместиться в него может лишь небольшое целое число, а именно - число в диапазоне от -32768 до 32767.

Для того, чтобы переменная имела право принимать значения больших целых чисел, она должна быть объявлена не как Integer, а как **Long** (Длинное Целое). Под переменную типа Long компьютер отводит в памяти 4 байта и поэтому она может принимать значения в диапазоне от -2147483648 до 2147483647.

Зачем нужен Integer, если есть Long? Ну, хотя бы для того, чтобы экономить память, она же не резиновая.

Задание 10: Население Москвы равняется a=9000000 жителей. Население Васюков равняется b=1000 жителей. Вся Москва переехала в Васюки. Сколько там стало жителей? Используйте переменные величины.

Single и Double - десятичные дроби

Создайте проект с кнопкой и введите такую программу:

```
Dim a As Single
Dim b As Single
Private Sub Command1_Click()
    a = 100.78656954325 :   b = 40000.1234567895 :      Debug.Print a, b, a + b
End Sub
```

Запустите проект. Вот результат:

```
100,7866    40000,13      40100,91
```

Как видите, Visual Basic обрезал наши длинные числа до 7 значащих цифр. Сделал он это потому, что 4 байта, отведенные под ячейку памяти для переменной типа Single, не в состоянии вместить больше. Если мы хотим иметь большую точность, то объявляем наши переменные имеющими другой тип - **Double** - десятичная дробь двойной точности. Под переменную типа Double компьютер отводит в памяти 8 байтов и поэтому она может быть гораздо длиннее.

```
Dim a As Double
Dim b As Double
Private Sub Command1_Click()
    a = 100.78656954325 :   b = 40000.1234567895 :   Debug.Print a, b, a + b
End Sub
```

Запустите проект. Вот результат:

```
100,78656954325        40000,1234567895        40100,9100263327
```

Здесь максимум - 15 значащих цифр. Если вам интересно задать в программе еще более длинное значение b, например, b = 40000.12345678957453457, то у вас ничего не получится, Visual Basic обрежет его прямо в программе, зная, что с такими длинными числами работать он все равно не умеет.

Зачем нужен Single, если есть Double? Ну хотя бы для экономии памяти.

Целые числа или десятичные дроби? Числовой тип Currency

Вы спросите: зачем использовать типы целых чисел Integer и Long, если типы десятичных чисел Single и Double обеспечивают нам работу и с целыми и с дробными числами? Кроме экономии памяти здесь есть еще проблема абсоютной точности вычислений. Дело в том, что при использовании типов десятичных дробей вполне мыслима ситуация, когда дважды два будет не точно 4, а, скажем, 4.00000000000381. Связано это с особенностями представления десятичных дробей в компьютерах. В большинстве реальных задач такая маленькая погрешность несущественна, однако существуют задачи, где точность нужна абсоютная. При использовании же типов целых чисел Visual Basic присматривает за тем, чтобы все числа и результаты были абсолютно точными целыми числами. Конечно, при делении в этом случае приходится округлять.

Если вам нужна абсолютная точность при работе с числами, а величина чисел превышает два миллиарда, тип Long будет слишком узок для вас. Воспользуйтесь типом Currency. Этот тип хоть и имеет дробную часть в размере 4 десятичных знаков после запятой, действия над числами выполняет абсолютно точно. Диапазон представляемых им величин весьма велик - примерно от -900 000 000 000 000 до 900 000 000 000 000.

Совет: Если вам необходимо, чтобы переменная была целым числом и никогда дробным, объявляйте ее целым типом, в противном случае - десятичным.

Не очень устаревшие способы объявления переменных

"Я советую всем нарочно написать на бумаге Испания, то и выйдет Китай"
Гоголь

Вы еще не столкнулись со странной привычкой Visual Basic ставить после чисел в окне кода какие-то значки? Например, вы не объявляли переменную **a** и собираетесь написать a = 123456789012345, получается же a = 123456789012345#. Ничего страшного, не обращайте внимания, это Visual Basic сообщает вам, что с его скромной точки зрения переменная a имеет тип Double.

У шести типов есть свои значки, по которым их можно узнать (тип String еще не проходили):

Integer	Long	Single	Double	String	Currency
%	&	!	#	$	@

Дальше. Если вам лень писать Dim b As Double, вам достаточно в окне кода, там, где переменная b встречается в первый раз, написать b# вместо b, например, b# = 5 вместо b=5. Visual Basic будет считать, что вы нормально объявили переменную. Впрочем, этот способ объявления мне не нравится.

Есть еще один способ объявления, совсем страшный. Написав **DefInt I - M**, вы тем самым объявите компьютеру, что переменнные, имена которых начинаются с букв I, J, K, L, M, N, обязаны иметь тип Integer. Ну и для других типов аналогично: **DefLng**, **DefSng**, **DefDbl** и так далее.

Форматирование результата

Взгляните на такую программу:

```
Dim a As Double
Dim b As Double
Dim y As Double
Private Sub Command1_Click()
   a = 2457642345034.78 :    b = 0.00000000037645 :    y = a / b :    Debug.Print   y
End Sub
```

Прикиньте без компьютера, каков будет результат. Ясно, что очень большое число, но какое - неясно. Запустите программу. Вот результат:

```
6,52846950467467E+21
```

Что это значит? Это значит, что Visual Basic, видя, что вы работаете с такими гигантскими числами, подумал, что вы большой ученый, и представил вам результат не в нормальном, как для простых смертных, а в так называемом **экспоненциальном** или **научном формате** (виде). Но не так страшен черт. Оказывается, это получается шесть целых и сколько-то там дробных и все это умножено на 10 в 21-й степени. Конструкция E+21 и означает умножение на 10^{21}. По-другому, вам нужно передвинуть запятую на 21 позицию направо - и получится нормальное число, то есть 6528469504674670000000. Кстати, о точности. Обратите внимание, что поделено было неточно. Точно получилась бы бесконечная периодическая дробь, а где ее хранить, бесконечную?

Если бы вместо E+21 было E-21, это означало бы умножение на 10^{-21}, то есть деление на 10^{21}, то есть передвижение запятой на 21 позицию налево, то есть очень маленькое число.

Если после этих объяснений вы все еще не полюбили экспоненциальный формат, вы можете приказать компьютеру, чтобы он вас им не утомлял, а показывал результаты по-человечески. Для этого в операторе Debug.Print нужно вместо **y** написать **Format(y, "0.0000")**. Получится Debug.Print Format(y, "0.0000"). Конструкция в кавычках состоит из нулей и точки и означает, что вы желаете видеть число в обычном виде, дробную часть числа состоящей ровно из 4 цифр, а остальные пусть Visual Basic вам не показывает. Целую же часть числа Visual Basic покажет вам полностью в любом случае, какова бы она ни была. Вот тот же результат в новом формате:

```
6528469504674670000000,0000
```

Вот вам и первое улучшение для калькулятора, ведь он тоже норовит показывать длинные результаты в экспоненциаль-

ном формате. Вместо

Результат.Text = *Val(Число1.Text) / Val(Число2.Text)*

можете написать

Результат.Text = **Format(***Val(Число1.Text) / Val(Число2.Text)***, "0.00000000000000000000")**

Только имейте в виду - если ваш результат будет такой маленький, что 20 цифр, указанных мной после точки, его "не почувствуют", то ничего, кроме нулей, вы в результате и не увидите, а вот экспоненциальный формат покажет вам настоящий, хоть и непривычный для чтения, результат.

Еще о пользе переменных

Значения переменных величин не обязаны, подобно надписям и значениям текстовых полей, отображаться "на медлительной поверхности проекта". Они спрятаны глубоко в сверхбыстрой оперативной памяти компьютера, там над ними удобно и быстро проводить вычисления и разнообразные логические преобразования. Фактически вся мыслительная работа компьютера проводится над переменными величинами. И лишь иногда, когда человеку понадобится, они показываются "на поверхности" в виде содержимого текстовых полей или как-нибудь еще.

Создавая калькулятор, мы не ведали ни о каких переменных, поэтому вместо изящного

Рез = Чис1 + Чис2

писали громоздкое

Результат.Text = Val(Число1.Text) + Val(Число2.Text)

Вообще, попытки использовать для вычислений вместо переменных текстовые поля напоминает попытку неуклюжих королей-тугодумов (текстовых полей) договориться между собой. После безуспешных попыток они вызывают своих шустрых министров иностранных дел (переменные величины), которые в два счета договариваются и отдают готовый договор на подпись королям (имеется в виду, что результат вычислений показывается в текстовом поле).

С учетом сказанного попробуем улучшить программу калькулятора:

```
Dim Чис1 As Double          'Переменная, содержащая число из текстового поля Число1
Dim Чис2 As Double          'Переменная, содержащая число из текстового поля Число2
Dim Рез As Double           'Переменная-результат, предназначенный для текстового поля Результат
_____________________________________________________________________________
Private Sub Кл_сложения_Click()
   Чис1 = Число1.Text       'Значения исходных данных переходят в переменные из текстовых полей
   Чис2 = Число2.Text
   Рез = Чис1 + Чис2        'Обработка переменных для получения результата
   Результат.Text = Рез     'Значение результата переходит из переменной в текстовое поле
End Sub
```

Эта схема, когда информация из текстовых полей (или других средств задания исходных данных) передается в переменные, затем обрабатывается, а затем из переменных передается обратно - в текстовые поля - весьма разумна и я рекомендую ей пользоваться.

Три совета

Дорогой читатель! Если вы сейчас сидите за компьютером, то вот вам три моих совета, по своей силе приближающихся к непререкаемым приказам:

1. Программы, которые вы видите в книге, вам необходимо вводить в компьютер и выполнять их, даже если они кажутся вам почти понятными, и даже если я явно этого не требую. В ряде случаев вы получите неожиданные результаты, из чего сделаете вывод, что программы эти вы поняли не до конца.

2. В каждой из этих программ экспериментируйте, то есть разными способами изменяйте в них то, что я как раз в этот момент объясняю. Например, если я объясняю оператор **For i=1 To 5***, пробуйте* **For i=1 To 10.**

3. Выполняйте и сверяйте с ответом все задания. Это, конечно, главный совет из трех. Учтите, что сверенная с ответом правильно работающая программа – ваша победа, сверенная с ответом неправильно работающая программа – временное поражение, отказ от сверки - разгром.

Если вы пожалеете времени и пренебрежете этими советами, то через несколько страниц можете обнаружить трудности в понимании материала и вскоре не сможете правильно составить большинство программ.

4.6. Порядок создания простого вычислительного проекта

Все, что здесь сказано, полезно не только для вычислительных, но и для всех других проектов.

Задача:

Даны размеры спичечной коробки. Вычислить площадь основания коробки, ее объем и полную площадь поверхности.

Порядок создания проекта:

1. Программист сам должен знать решение задачи. Ведь программа - это инструкция по ее решению. Нельзя давать инструкцию, не зная, как решать.

В нашем случае программист должен знать формулы для вычисления всего, что нужно:

1. площадь основания = ширина x толщину
2. объем = площадь основания x высоту
3. периметр основания = две ширины + две толщины
4. площадь боковой поверхности = периметр основания x высоту
5. полная площадь поверхности = две площади основания + площадь боковой поверхности

Как видите, я для стройности вычислений ввел периметр основания и площадь боковой поверхности.

2. Нужно придумать имена переменным. Имя переменной должно говорить о ее смысле. Если смыслом является ширина коробки, то не ленитесь и не называйте ее a, потому что через полгода, разбираясь в своей полузабытой программе, вы будете долго тереть лоб и думать – Что, черт возьми, я обозначил через a? Называйте ее Ширина (если вы не знаете английского) или, к примеру, Width (если знаете). Так делают все профессиональные программисты (а они, как известно, терпеть не могут трудиться зря, значит, зачем-то это им нужно).

Удовлетворимся такими именами:

Ширина	-	ширина
Толщина	-	толщина
Высота	-	высота
S_основ	-	площадь основания
V	-	объем
Периметр	-	периметр основания
S_бок	-	площадь боковой поверхности
S_полн	-	полная площадь поверхности

3. Нужно определить, какого типа будут переменные. Поскольку нам заранее неизвестно, будут ли исходные данные целыми, объявляем все переменные - Double. Первые строки программы будут такими:

```
        'Задача вычисления площади основания, объема и полной площади поверхности
        'спичечной коробки по известным ее размерам
                'Объявляем переменные величины
    Dim Ширина As Double       'ширина
    Dim Толщина As Double      'толщина
    Dim Высота As Double       'высота
    Dim S_основ As Double      'площадь основания
    Dim V As Double            'объем
    Dim Периметр As Double     'периметр основания
    Dim S_бок   As Double      'площадь боковой поверхности
    Dim S_полн As Double       'полная площадь поверхности
```

4. Перед вычислениями нужно задать исходные данные решения задачи. Для этого нужно решить, каким способом пользователь будет задавать размеры коробки - при помощи текстовых полей, функции InputBox (4.2) или как-то по-другому. Выберем InputBox.

Вот следующие строки программы:

```
    Private Sub Command1_Click()
                'Ввод исходных данных
        Ширина = InputBox("Введите ширину коробки")
        Толщина = InputBox("Введите толщину коробки")
        Высота = InputBox("Введите высоту коробки")
```

5. Теперь нужно задать компьютеру действия, которые он должен проделать с исходными данными, чтобы получить результат.

```
                'Вычисление результатов
        S_основ = Ширина * Толщина
        V = S_основ * Высота
        Периметр = 2 * Ширина + 2 * Толщина
        S_бок = Периметр * Высота
        S_полн = 2 * S_основ + S_бок
```

6. После получения результатов их нужно показать человеку. Действительно, все операторы присваивания компьютер выполняет "в уме". После их выполнения в ячейках памяти будут находиться числовые результаты решения задачи. Чтобы их увидеть, человек может использовать текстовые поля, как это мы только что сделали, улучшая калькулятор. В 4.10 описывается, как выводить информацию в текстовое поле. Однако, я использую новый для нас оператор **Print**, который в нашем случае будет печатать результат прямо на поверхности формы (в отличие от Debug.Print, который печатает в окне Immediate):

```
            'Отображение результатов
        Print "Площадь основания ="; S_основ
        Print "Объем ="; V
        Print "Полная площадь поверхности ="; S_полн
    End Sub
```

Обратите внимание, что здесь в первом операторе Print - два элемента печати, разделенные точкой с запятой: текстовое пояснение "Площадь основания =" и собственно переменная, значение которой мы хотим увидеть - S_основ. То же самое и в других операторах Print. То же самое можно делать и в Debug.Print. Полностью правила записи и возможности этих операторов вы найдете в 4.8.

Запустите проект и убедитесь в его работоспособности. Вы спросите - зачем было так долго трудиться для решения такой простой задачи? Действительно, для простых исходных данных эту задачу быстрее решить в уме. Однако, соблюдение приведенного мной порядка составления программы облегчит вам в дальнейшем программирование реальных задач для компьютера. Этот порядок - начинаем со ввода исходных данных, затем преобразовываем их в переменные величины, затем, обрабатывая переменные величины, получаем результат, затем преобразовываем результат из переменных величин в нечто видимое - общепринят.

Замечание. Попробуйте в режиме работы потаскать форму по экрану, нечаянно затащив часть ее напечатанных результатов за пределы экрана. Вы видите, что эта часть результатов стерлась. Чтобы такого не происходило, в режиме проектирования установите свойству **AutoRedraw** формы значение True.

Задания 11-13:

Написать с использованием переменных программы для решения следующих задач:

11) Автомобиль 3 часа ехал со скоростью 80 км/час и 2 часа со скоростью 90 км/час. Вычислить среднюю скорость автомобиля (она равна суммарному пути, деленному на суммарное время). Значения переменным задать операторами присваивания, результат напечатать оператором Debug.Print с пояснениями.
12) В самом углу прямоугольного двора стоит прямоугольный дом. Подсчитать площадь дома, свободную площадь двора и длину забора. Примечание: в углу, где дом, забора, естественно, нет. Размеры дома и двора вводим при помощи InputBox, результаты отображаем в текстовых полях. Все числа целые.
13) Ввести из текстового поля радиус окружности. Вычислить длину окружности и площадь круга. Результаты с пояснениями печатать с 5 знаками после десятичной точки с помощью оператора Print.

4.7. Строковые переменные

Строковые переменные очень важны. Без них, например, невозможен разговор с компьютером. Да и вообще, значительная часть информации, с которой работает компьютер, текстовая (то есть строковая).

Создайте двухкнопочный проект с такой программой:

```
Dim a As Integer
Dim b As String
_____________________________
Private Sub Command1_Click()
    a = 98
    Debug.Print a
End Sub
_____________________________
Private Sub Command2_Click()
    b = "Привет всем!"
    Debug.Print b
End Sub
```

Сравним две процедуры.

Объявление Dim a As Integer говорит о том, что переменная a обязана иметь числовое значение, и поэтому в первой процедуре оператор a=98 записывает в ячейку a число 98.

Объявление **Dim b As String** говорит о том, что переменная b обязана иметь строковое (текстовое) значение, то есть ее значением будет не число, а произвольная цепочка символов, например, Привет всем! или рпН2Н(*fD6:u . Поэтому во второй процедуре оператор b ="Привет всем!" записывает в ячейку b строку Привет всем! . Оператор Debug.Print b, поскольку он обязан всегда выводить на экран содержимое ячейки b, выведет на экран текст Привет всем!

Обратите внимание, что в программе текст должен браться в двойные кавычки, а в памяти он хранится без кавычек и на экран выводится без кавычек.

Информация в ячейке памяти под строковую переменную может в процессе выполнения программы меняться точно так же, как и в ячейке для числовой переменной. Например, при выполнении фрагмента

```
a="Минуточку!"  :  Debug.Print a  :  a="Здравствуйте!"  :  a="До свидания!"  :  Debug.Print a
```

в ячейке a будут по очереди появляться строки

Минуточку! *Здравствуйте!* *До свидания!*

а на экран будут выведены строки:

```
Минуточку!
До свидания!
```

Еще пример:

```
a="Цикл"  :  Debug.Print a  :  a=a+a :  Debug.Print a+a+a +"Конец цикла"
```

Поскольку знак **+** по отношению к строкам есть знак соединения, то *a+a* будет равняться *"ЦиклЦикл"* и это же - новое значение *a*. Поэтому здесь на экран будут выведены строки:

```
Цикл
```

```
ЦиклЦиклЦиклЦиклЦиклЦиклКонец цикла
```

Выражения *а+а* и *а+а+а +"Конец цикла"* в последнем примере являются ни чем иным, как **строковыми выражениями**, по той простой причине, что в них производятся действия, результатом которых является строка.

Строковую переменную можно задавать не только оператором присваивания, но и функцией InputBox. При вводе ставить кавычки тоже не надо. Пример:

```
Dim a As String
Private Sub Command1_Click()
    a = InputBox("Введите какое-нибудь слово")
    Debug.Print "Вы ввели слово "; a
End Sub
```

Обратите внимание на пробел перед второй кавычкой в операторе Debug.Print. Он нужен для того, чтобы слова при выводе не сливались.

Прежде, чем идти дальше, нужно, конечно, разобраться с хитросплетениями запятых, точек с запятыми, кавычек и пробелов в операторе Print:

4.8. Как выводить информацию оператором Print

Будем пробовать, вы тоже пробуйте:

Фрагмент программы	Результат на экране
Print	*Печатается пустая строка*
Print 1	1
Print 1; 66	1 66
Print 1; -2; 3.14	1 -2 3.14
Print 1, -2, 3.14	1 -2 3.14

Ага, значит, оператор распечатывает список **элементов**, разделенных точками с запятой или запятыми. Причем, если перед элементом стоит точка с запятой, он печатается почти вплотную к предыдущему, а если запятая, то подальше, выравниваясь по столбцам. Проверим фрагмент из трех операторов:

Print 12345, 4, 67824, 240	12345	4	67824	240
Print 345, -94, 67, 240456	345	-94	67	240456
Print 45, 45678, 67, 0	45	45678	67	0

Еще:

Print 45; 45678, 67; 0	45 45678	67 0	

Что такое элемент? Мы видели, что это может быть число. А еще что? Выражение. Проверим:

Print 1+2	3	
a = 88 : Print a	88	
a = 88 : Print a+1	89	
a = 88 : Print a+1, a -1	89	87

В последнем операторе - два элемента.

До сих пор мы печатали только числовые переменные и выражения. Но элемент может быть и строкой:

Print "Кошка"	Кошка	
Print "Кошка", "Собака"	Кошка	Собака

Как видите, строку мы обязаны брать в двойные кавычки.

Print "Кошка"; "Собака"	КошкаСобака

Как видите, если между строками стоит точка с запятой, они печатается вплотную друг к другу. Числа же в этом случае все-таки разделяются пробелами.

Как все же отодвинуть собаку от кошки на пару пробелов? Очень просто - поставьте эти два пробела внутри "собачьих" кавычек перед буквой *С*:

Print "Кошка"; " Собака"	Кошка Собака

А можно внутри "кошачьих" после буквы *а*.

Visual Basic не обращает внимания, что написано внутри кавычек, и ничего там не вычисляет. Он просто копирует это на экран, включая пробелы, цифры, знаки арифметических действий и любую ерунду:

Print "№??:?№()*):%*%;*_?::"	№??:?№()*):%*%;*_?::
Print "3+2"	3+2
Print "3+2="; 3+2	3+2=5

Элемент может быть строковой переменной или выражением:

a = "Кошка" : Print a	Кошка
a = "Кошка" : Print a+a	КошкаКошка

Итак, элемент оператора Print может быть числом, числовой переменной и числовым выражением, а также строкой, строковой переменной и строковым выражением. Есть и другие типы, о них мы пока не говорим.

Вообще, в будущем, объясняя какой-нибудь новый оператор, я часто буду для простоты ограничиваться коротенькими примерами его записи, например, Print 66 **или** Print "Кошка"**. Вы должны знать, что почти везде на месте числа может стоять числовая переменная или арифметическое выражение, а на месте строки - строковая переменная**

или строковое выражение. И вообще, вместо константы данного типа может стоять переменная или выражение этого типа.

Обычно в операторе Print используют вперемешку строковые и числовые данные. Пусть вес поросенка хранится в памяти, в переменной Ves. Тогда распечатать его можно таким оператором:

Print "Вес поросенка =" ; Ves ; "килограммов"	Вес поросенка = 35 килограммов

Здесь вы видите три элемента, разделенные точками с запятой.

Пусть название месяца года хранится в памяти, в переменной Mes. Пусть вы хотите напечатать, что именно этот месяц у вас отпускной. Распечатать это можно таким оператором:

Print "Месяц " ; Mes ; " для меня отпускной."	Месяц май для меня отпускной.

Обратили внимание на пробелы внутри кавычек? Если бы их не было, было бы вот что:

Print "Месяц" ; Mes ; "для меня отпускной."	Месяцмайдля меня отпускной.

Мы привыкли, что каждый следующий оператор Print печатает с новой строки. Если в конце оператора Print поставить запятую или точку с запятой, то следующий оператор Print будет продолжать печатать в той же строке, а если не поставить - то начнет со следующей. Проверим фрагмент из четырех операторов:

Print 1; 66, Print 2; Print 3 Print 4;	1 66 2 3 4

Проверьте этот пример, запустив его один раз, затем еще раз. Объясните увиденное.

Все, что здесь было сказано, относится и к оператору Debug.Print.

То, *что* печатать, указывается в самом операторе Print, а вот *как* печатать и с какого места, зависит от свойств объекта, на котором ведется печать (в нашем случае объект - форма):

Form1.**CurrentX** = 1000 Form1.**CurrentY** = 3000 Print 99	*Число 99 будет напечатано на 1000 твипов правее и на 3000 твипов ниже левого верхнего угла формы.*
Form1.CurrentX = 1000 Print 99 Form1.CurrentX = 1000 Print 44	*Числа 99 и 44 будут напечатаны в одном месте. Получится мазня.*
Form1.**FontSize** = 20	*Размер шрифта*
Form1.**FontName** = "Arial"	*Название (начертание) шрифта*
Form1.**FontBold** = True	*Полужирный шрифт (True - назначить, False - отменить)*
Form1.**FontItalic** = True	*Курсив (наклонный шрифт)*
Form1.**ForeColor** = vbBlue	*Цвет шрифта*
Form1.**FontUnderline** = True	*Подчеркнутый шрифт*
Form1.**FontStrikethru** = True	*Перечеркнутый шрифт*

Если вы установите в окне свойств формы свойство **FontTransparent** равным False, то через пространство между буквами не будет просвечивать поверхность формы.

У оператора Print есть еще кое-какие возможности, но, пожалуй, хватит.

4.9. Диалог с компьютером

Напишем программу, которая осуществляла бы такой диалог человека с компьютером:

КОМПЬЮТЕР ВЫВОДИТ НА ЭКРАН:	Здравствуй, я компьютер, а тебя как зовут?
ЧЕЛОВЕК ВВОДИТ С КЛАВИАТУРЫ:	*Коля*
КОМПЬЮТЕР ВЫВОДИТ НА ЭКРАН:	Очень приятно, Коля. Сколько тебе лет?
ЧЕЛОВЕК ВВОДИТ С КЛАВИАТУРЫ:	*16*
КОМПЬЮТЕР ВЫВОДИТ НА ЭКРАН:	Ого! Целых 16 лет! Ты уже совсем взрослый!

Пусть человек вводит свои реплики при помощи InputBox, а компьютер печатает свои с помощью Print. Для хранения в памяти имени человека выдумаем переменную imya, а для возраста - vozrast.

Вот программа:

```
Dim imya As String
Dim vozrast As Integer
Private Sub Command1_Click()
    Print "Здравствуй, я компьютер, а тебя как зовут?"
    imya = InputBox("Жду ответа")
    Print "Очень приятно, "; imya; ". Сколько тебе лет?"
    vozrast = InputBox("Жду ответа")
    Print "Ого! Целых"; vozrast; "лет! Ты уже совсем взрослый!"
End Sub
```

Вам понятно, зачем в операторе *Print "Очень приятно, "; imya; ". Сколько тебе лет?"* внутри кавычек нужны запятая и точка? Если нет, то попробуйте их убрать и посмотрите на результат.

Диалог будет отличаться только той информацией, которую вводит человек. Так, в другой раз по этой же программе будет осуществлен следующий диалог:

КОМПЬЮТЕР:	Здравствуй, я компьютер, а тебя как зовут?
ЧЕЛОВЕК :	*Фантомас!*
КОМПЬЮТЕР:	Очень приятно, Фантомас! . Сколько тебе лет?
ЧЕЛОВЕК:	*100*
КОМПЬЮТЕР:	Ого! Целых 100 лет! Ты уже совсем взрослый!

Не подумайте, что эта программа очень умная. Она совершенно не анализирует, какую информацию человек ввел с клавиатуры. Поэтому с ней возможен и такой диалог:

КОМПЬЮТЕР:	Здравствуй, я компьютер, а тебя как зовут?
ЧЕЛОВЕК:	*Сгинь с моих глаз!*
КОМПЬЮТЕР:	Очень приятно, Сгинь с моих глаз!. Сколько тебе лет?
ЧЕЛОВЕК:	*-2*
КОМПЬЮТЕР:	Ого! Целых -2 лет! Ты уже совсем взрослый!

4.10. Как выводить информацию в текстовое поле

Конечно, оператором присваивания:

Фрагмент программы	Результат в текстовом поле
Text1.Text = 2001	2001
Text1.Text = 3+2	5
Text1.Text = "Кошка"	Кошка
a = "Кошка" : Text1.Text = a	Кошка

Трудность в том, что в операторе присваивания правая часть не может состоять из нескольких элементов, как в операторе Print, это один-единственный элемент, который может быть числом, строкой, числовой или строковой переменной, числовым или строковым выражением. Правая часть вычисляется и становится содержимым текстового поля.

Text1.Text = "Кошка" + " Собака"	Кошка Собака

Что же делать, если мы хотим вывести в текстовое поле сразу несколько элементов, как в примере об отпускном месяце из 4.8? Там все три элемента строковые и разделены точками с запятой. Здесь мы должны выстроить их в один элемент. Сделаем же это знаком +. У нас получится одно выражение:

Text1.Text = "Месяц " + Mes + " для меня отпускной."	Месяц май для меня отпускной.

А вот с примером оттуда же о поросенке немного посложнее. Попробуем проделать то же самое:

Text1.Text = "Вес поросенка =" + Ves + "килограммов"	*Ошибка "Type mismatch" - Несовпадение типов*

Дело вот в чем. Visual Basic присматривает за тем, чтобы программист не складывал "бочки и селедки". То есть, если складываешь, то уж складывай одни числа, или уж одни строки (как в задаче об отпускном месяце), а строки с числами складывать никак нельзя. Ведь *"Вес поросенка ="* и *"килограммов"* это строки, а *Ves* - это числовая переменная.

Но и здесь есть, конечно, выход. Помните, как в 2.5 при помощи Val мы приказали компьютеру считать текст числом (другими словами - преобразовали текст в число)? Совершенно аналогично здесь мы прикажем компьютеру считать число текстом (другими словами - преобразуем число в текст). И сделает это функция **Str**:

Text1.Text = "Вес поросенка =" + Str(Ves) + " килограммов"	Вес поросенка = 35 килограммов

В 11.3 и !!!! показан более простой способ примирить строковые и числовые данные.

Задание 14: Напишите программу для следующей задачи: Компьютер запрашивает названия двух планет, радиусы их орбит (в миллионах километров) и скорости движения по орбите (в миллионах километров в сутки). После этого он вычисляет продолжительность года на планетах и выдает результат в таком виде: Продолжительность года на планете Земля – 365 суток, а на планете Эоэлла – 12 суток. Результат - в двух вариантах: печать на форме оператором Print и вывод в текстовое поле.

Указание для тех, кто не знает физики и геометрии: Год равен времени одного оборота по орбите, а оно равно длине орбиты, деленной на скорость движения по орбите. Длина орбиты равна 2□R, где R - радиус орбиты.

4.11. Оглядимся вокруг

Ну вот, с переменными величинами мы разобрались. Узнали что-то. А научились ли мы в результате этого знания делать что-нибудь новенькое и интересное? Вроде, нет. Ничего особенно приятного, за исключением, может быть, диалога с компьютером. Нда-а-а… Зачем же все мучения?

В общем, так: Материал этой главы - патрон без пистолета, ненужная сама по себе вещь. А вот материал следующих глав - это пистолеты самых разных марок, которые без этого патрона не могли бы стрелять.

Глава 5. Разветвляющиеся программы

Выбор (ветвление) в Visual Basic осуществляют в основном три оператора:

- Однострочный If
- Многострочный If
- Оператор выбора Select Case

Их применению в программировании на Visual Basic и посвящена эта глава.

5.1. Что такое выбор (ветвление)

У начинающего программиста интерес должен вызывать такой вопрос: как компьютер думает, как он принимает решения, как он выбирает, какое действие из нескольких возможных нужно выполнить в данный момент? Ведь пока компьютер выполнял все, что ему приказывали, не рассуждая. Попробую ответить.

Возьмем игру в воздушный бой. Предположим, самолет на экране летит на автопилоте. Это значит, он должен самостоятельно поддерживать высоту полета между 3000 м и 3200 м. Для этого ему достаточно периодически определять высоту, и если она меньше 3000, лететь вверх, а если больше 3200 - вниз. Естественно, этот выбор делает программа для игры в воздушный бой, сам по себе компьютер ничего выбрать не может. В программе заранее пишутся процедуры для полета вверх и для полета вниз, а выбор между ними делает специальная команда (оператор) выбора, имеющаяся в каждом языке программирования. Вот алгоритм выбора между полетом вверх и вниз:

1. Определи высоту.
2. Если высота < 3000, то выполняй процедуру ВВЕРХ.
3. Если высота > 3200, то выполняй процедуру ВНИЗ.
4. Продолжай полет.

Здесь команды 2,3 - команды выбора. В общем случае команда выбора содержит условие, от которого зависит, будет ли выполняться какая-нибудь команда или группа команд. Это условие может быть самым разным: нажата или нет любая клавиша, нажата или нет конкретная клавиша, был ли щелчок мышью над таким-то объектом, больше ли одно число другого, правда ли, что с клавиатуры введено такое-то слово и т.д. В нашем случае условие - это *высота < 3000* или *высота > 3200*.

Напишем для примера примитивный алгоритм, позволяющий имитировать вежливое общение компьютера с человеком при включении компьютера:

1. Покажи на мониторе текст "Здравствуйте, я - компьютер, а вас как зовут?"
2. Жди ответа с клавиатуры.
3. Если на клавиатуре человек набрал "Петя" или "Вася", то покажи на мониторе текст "Рад встретиться со старым другом!", иначе покажи на мониторе текст "Рад познакомиться!"
4. Покажи на мониторе текст "Чем сегодня будем заниматься - программировать или играть?"
5. Жди ответа с клавиатуры.
6. Если . . . . . .
   . . . . . .

Выбор называют ветвлением по аналогии с разветвляющимся деревом (когда мы забираемся на дерево, мы время от времени делаем выбор, по какой из нескольких веток забираться дальше).

Идею разветвления в программе я изложил. Как видите, команды ветвления довольно просты. Как же с помощью таких простых команд запрограммировать сложное поведение компьютера? Ответ вы найдете в материале этой главы. Добавлю только, что вся мыслительная деятельность во всех программах (в том числе и той, что выиграла в шахматы у Каспарова) осуществляется при помощи сложного набора таких вот простых команд ветвления (выбора).

5.2. Условный оператор If или как компьютер делает выбор

Теперь посмотрим, как писать разветвляющиеся программы на Visual Basic.

Выучим сначала три английских слова:

If	читается "иф"	переводится "если"
Then	читается "зэн"	переводится "то"
Else	читается "элз"	переводится "иначе"

Теперь приведем пример записи нового для вас оператора:

If a=28 Then Print f Else k=44

Переводится он так:

ЕСЛИ a=28 ТО печатай f ИНАЧЕ присвой переменной k значение 44.

Другими словами, мы предлагаем компьютеру сначала подумать, правда ли, что a=28 , и если правда, то выполнить оператор Print f , в противном случае выполнить оператор k=44 . Таким образом, мы с вами впервые написали оператор, при выполнении которого компьютер не просто выполняет, что приказано, а сначала думает и делает выбор (пока одного из двух).

Мы видим, что оператор If включает в себя другие операторы, которые выполняются или не выполняются в зависимости от какого-то условия. Тем не менее, вся эта запись считается одним оператором If. Чтобы привыкнуть к оператору If, рассмотрим пару задач.

Задача 1. Компьютер должен перемножить два числа - 167 и 121. Если их произведение превышает 2000, то компьютер должен напечатать текст ПРОИЗВЕДЕНИЕ БОЛЬШОЕ, иначе текст ПРОИЗВЕДЕНИЕ МАЛЕНЬКОЕ. После этого компьютер в любом случае должен напечатать само произведение.

Программа:

```
Dim a As Integer
Dim b As Integer
Dim y As Integer
Private Sub Form_Load()
    a = 167
    b = 121
    y = a * b
    If y > 20000 Then Debug.Print "ПРОИЗВЕДЕНИЕ  БОЛЬШОЕ" _
                 Else Debug.Print "ПРОИЗВЕДЕНИЕ МАЛЕНЬКОЕ"
    Debug.Print y
End Sub
```

Пояснение: В процедуре 5 операторов, последний – Debug.Print y. Поскольку эти 5 операторов выполняются по порядку, то он выполнится обязательно.

Обязательно выполните эту программу в пошаговом режиме. Обратите внимание, что подсветка после *If y > 20000 Then* перескакивает на *Debug.Print "ПРОИЗВЕДЕНИЕ БОЛЬШОЕ"* , а затем на *Debug.Print y*.

Теперь замените в программе *a = 167* на *a = 1* и снова выполните программу в пошаговом режиме. Обратите внимание, что теперь подсветка после *If y > 20000 Then* перескакивает на *Debug.Print "ПРОИЗВЕДЕНИЕ МАЛЕНЬКОЕ"* , а затем уже на *Debug.Print y*.

Задача 2. В компьютер вводятся два произвольных положительных числа - длины сторон двух кубиков. Компьютер должен подсчитать объем одного кубика - большего по размеру .

Обозначим a1 - сторону одного кубика, a2 - сторону другого, bol - сторону большего кубика, V - объем кубика. Приведем три варианта программы:

ВАРИАНТ 1	Dim a1 As Double Dim a2 As Double Private Sub Command1_Click() a1 = InputBox("Введите сторону одного кубика") a2 = InputBox("Введите сторону другого кубика") If a1 > a2 Then Debug.Print a1 * a1 * a1 Else Debug.Print a2 * a2 * a2 End Sub
ВАРИАНТ 2	Dim a1 As Double Dim a2 As Double Dim V As Double Private Sub Command1_Click() a1 = InputBox("Введите сторону одного кубика") a2 = InputBox("Введите сторону другого кубика") If a1 > a2 Then V = a1 * a1 * a1 Else V = a2 * a2 * a2 Debug.Print V End Sub
ВАРИАНТ 3	Dim a1 As Double Dim a2 As Double Dim bol As Double Private Sub Command1_Click() a1 = InputBox("Введите сторону одного кубика") a2 = InputBox("Введите сторону другого кубика") If a1 > a2 Then bol = a1 Else bol = a2 Debug.Print bol * bol * bol End Sub

Каждый из вариантов должен быть вами понят. Если возникают трудности, то используйте пошаговый режим и следите за значениями переменных. Для каждого варианта пошаговый режим используйте два раза - когда первый кубик больше и когда второй кубик больше. Как видите, одна задача может решаться разными программами.

Итак, если паровая машина избавила человека от тяжелого физического труда, то оператор if избавил человека от тяжелого умственного труда, в нашем случае - от необходимости решать, какое из двух чисел больше другого.

Оператор If можно записывать и без части Else. Например, If s<t Then w=a+1. Это означает, что если s<t, то нужно выполнить оператор w=a+1, в противном случае ничего не делать, а просто перейти к следующему оператору.

Для примера рассмотрим простейшую задачу: В компьютер вводится слово. Компьютер должен просто распечатать его. Однако, если введенным словом будет "колхозник", то компьютер должен напечатать вместо него слово "фермер".

Вот как будет выглядеть наша программа-"цензор":

```
Dim Slovo As String
Private Sub Command1_Click()
    Slovo = InputBox("Введите слово")
    If Slovo = "колхозник" Then Slovo = "фермер"
    Debug.Print Slovo
End Sub
```

До сих пор мы после Then и после Else писали только по одному оператору. А если нужно больше?

Задача: Если a не равно 4, выполнить операторы b=3 и Print b, а в противном случае - операторы b=0, a=b+5 и c=0.

Вот оператор, решающий эту задачу:

If a <> 4 **Then** b=3 : Print b **Else** b=0 : a=b+5 : c=0

Как видите, после Then и Else можно писать по нескольку операторов, разделенных двоеточиями.

5.3. Правила записи однострочного оператора If

Любой оператор Visual Basic нужно записывать по определенным грамматическим правилам, в противном случак Visual Basic выдает сообщение об ошибке. У каждого человеческого языка есть своя грамматика, включающая в себя правила, по которым должны выстраиваться в цепочку слова и другие элементы языка, чтобы получилось правильное предложение. Совокупность этих правил образует часть грамматики, называемую **синтаксисом**. В языках программирования тоже есть предложения. Такими предложениями здесь являются операторы. Очевидно, у языков программирования тоже должен быть свой синтаксис, который описывает правила, по которым записываются операторы языка и из операторов составляется программа. После того, как человек запускает программу на выполнение, любая порядочная среда программирования прежде, чем действительно выполнять ее, сначала проверит, нет ли в ней синтаксических ошибок, и если есть, то программу выполнять не будет, а выдаст сообщение, указывающее человеку, в чем ошибка. А Visual Basic проверяет программу еще на стадии ввода кода.

У Visual Basic есть две формы оператора If: **однострочная** и **многострочная**. Пока мы пользовались только однострочным If и поэтому приведем правило записи только для него. Приведем это правило в виде **синтаксической схемы**:

If *условие* **Then** *операторы* **Else** *операторы*

Как понимать эту схему? Ее следует понимать, как образец, шаблон записи оператора, указывающий порядок, в котором оператор записывается из отдельных слов. Слова, которые в схеме я записал жирными буквами, при записи оператора просто копируются. Вместо слов, которые в схеме записаны курсивом, нужно при записи оператора подставить то, что они означают. Поясним, что обозначают эти слова.

операторы	любой оператор Visual Basic или группа операторов, разделенных двоеточиями
условие	пока под условием будем понимать два арифметических или строковых *выражения*, соединенных *знаком сравнения*
знак сравнения	знаков сравнения шесть: **>** больше **>=** больше или равно **=** равно **<** меньше **<=** меньше или равно **<>** не равно

Пример: **If** 5*a+4 <= a*b **Then** Print b **Else** a=b+5

Здесь

Print b - один оператор,
a=b+5 - другой оператор,
5*a+4 <= a*b - условие,
5*a+4 - одно выражение,
a*b - другое выражение,
<= - знак сравнения.

В 5.2 вы уже видели, что однострочный оператор If можно записывать в краткой форме. Вот синтаксическая схема для этой формы:

If *условие* **Then** *операторы*

Таким образом, это уже вторая синтаксическая схема, касающаяся одного оператора. Удобно же весь синтаксис оператора иметь перед глазами в одной схеме. Соединим две схемы в одну. Вот эта схема:

Синтаксическая схема однострочного оператора If:

If *условие* **Then** *операторы* **[Else** *операторы* **]**

Квадратные скобки здесь означают, что их содержимое можно писать, а можно и не писать в операторе.

Полезное замечание: Вычисляя выражения, стоящие в условии оператора If, Visual Basic не записывает их значения в память. Например, после выполнения фрагмента - *b=6 : If b+1>0 Then s=20* **- в ячейке** *b* **будет храниться** *6*, **а не** *7*. **То же относится и к выражениям из оператора Print. Например:** *b=6 : Print b+1* **. И здесь тоже в ячейке** *b* **останется храниться** *6*, **а не** *7*. **И вообще, информация в ячейках памяти не меняется при вычислении выражений без присвоения.**

Примеры работы оператора If:

ФРАГМЕНТ ПРОГРАММЫ	ЧТО НА ЭКРАНЕ
a=10 **If** a>2 **Then** Print "!!!" **Else** Print "!"	!!!
a=4 **If** a>5 **Then** a=a+10 **Else** a=a-1 Print a	3
s=6 **If** s-8<>0 **Then** s=s+10 Print s	16
s=6 **If** s<0 **Then** s=s+10 s=s+1 Print s	7

Задания 15-17:

Определить без компьютера, что будет напечатано при выполнении следующих фрагментов программ:

15. k=20: k=k+10: **If** k+10<>30 **Then** k=8 **Else** k=k-1
Print k

16. k=20: k=k+10: **If** k+10 = 30 **Then** k=8 **Else** k=k-1
Print k

17. p=1: **If** p>0 **Then** p=p+5
If p>6 **Then** p=p+1
Print p

Задания 18-20:

18. В компьютер вводятся два числа. Если первое больше второго, то напечатать их сумму, иначе - произведение. После этого компьютер должен напечатать текст ЗАДАЧА РЕШЕНА.

19. В компьютер вводятся длины трех отрезков. Компьютер должен ответить на вопрос, правда ли, что первый отрезок достаточно мал, чтобы образовать с другими двумя отрезками треугольник.
Указание: Для этого его длина должна быть меньше суммы длин двух других отрезков. Замечание: Пока не думайте о том, что слишком длинными могут быть второй или третий отрезки. Об этом – задание 26 из 5.7.

20. Дракон каждый год отращивает по три головы, но после того, как ему исполнится 100 лет - только по две. Сколько голов и глаз у дракона, которому N лет?

Если в выражения, входящие в условие оператора If, включить свойства объектов, то вы можете заставить компьютер работать с ними. Например, компьютеру нужно определить, правда ли, что форма красная. Делает это такая программа:

```
Private Sub Command1_Click()
    If Form1.BackColor = vbRed Then MsgBox ("Правда") Else MsgBox ("Неправда")
End Sub
```

Задание 21: Если кнопка на форме слишком высоко, пусть при нажатии на нее она будет пониже на 200 твипов.

5.4. Случайные величины

Без случайных величин компьютер всегда бы, как робот, на одинаковые действия человека реагировал одинаково. Но тогда невозможны игры.

Запустите такую программу:

```
Private Sub Command1_Click()
    p = Rnd
    Debug.Print p
End Sub
```

Вот результат: 0,7055475. Это случайное число из диапазона от 0 до 1. Вырабатывает это число функция **Rnd**.

Щелкнем несколько раз по кнопке. Получим серию случайных чисел:

```
0,7055475
0,533424
0,5795186
0,2895625
0,301948
```

Завершим работу программы и снова запустим. Получаем ту же серию:

```
0,7055475
0,533424
0,5795186
0,2895625
0,301948
```

Выходит, что числа хоть и случайные, но после каждого запуска одинаковые. Не очень-то, получается, случайные. Как сделать их разными от запуска к запуску? Добавим оператор **Randomize**:

```
Private Sub Command1_Click()
    Randomize
    p = Rnd
    Debug.Print p
End Sub
```

Теперь числа и случайные и разные от запуска к запуску.

Как получить случайное число из диапазона от 0 до 20? Так - p = 20 * Rnd. А из диапазона от 6 до 7? Так - p = 6 + Rnd. А из диапазона от 200 до 210? Так - p = 200 + 10 * Rnd.

Как получить случайное *целое* число из диапазона от 200 до 210? Так - p = Int(200 + 11 * Rnd). Подумайте, почему я написал 11, а не 10. Если не можете додуматься, запустите такой проект:

```
Private Sub Command1_Click()
  t = 200 + 11 * Rnd
  p = Int(t)
  Debug.Print t, p
End Sub
```

Щелкайте по кнопке, наблюдая за значениями t и p, до тех пор, пока не поймете, в чем тут дело.

Задание 22: *"Ловля кузнечика или измеритель шустрости".* Создайте проект с большой формой и одной маленькой кнопкой. При нажатии на кнопку она должна прыгать в случайное место формы.

Щелкая по кнопке, старайтесь щелкать как можно чаще. Можете засечь, сколько раз вам удалось щелкнуть за 1 минуту. (В дальнейшем вы сможете научить компьютер, чтобы он сам засекал время и сам подсчитывал количество нажатий. Кстати, попробуйте опередить книгу и сами организуйте подсчет. В этом вам поможет оператор вида k=k+1.) Указание: Чтобы кнопка прыгнула в случайное место формы, вам достаточно задать случайные значения двум свойствам кнопки - Left и Top. При этом вы должны добиться, чтобы кнопка не "упрыгивала" с формы. Подсказка: Вам могла бы понадобиться такая, например, случайная величина - Form1.Width * Rnd. Только имейте в виду, что размеры формы больше размеров ее рабочего пространства на размеры заголовка и бордюров. Поэтому указанную формулу надо немного подкорректировать в сторону уменьшения.

Задание 23: *"Угадай число или экстрасенс ли вы".* Это ваша первая простейшая игра с компьютером. Компьютер загадывает число - 0 или 1. Ваше дело - отгадать. А дело компьютера - сказать "Угадал" или "Не угадал". Некоторые экстрасенсы утверждают, что благодаря сверхчувственному контакту с компьютером они могут из 100 раз угадать 80.

Программа готова? Настройтесь на контакт! Пуск!

Указание: Здесь вам нужно получить целое число из диапазона от 0 до 1. Получается оно по той же методе, что и целое число из диапазона от 200 до 210.

5.5. Многострочный If

Вспомним недавнюю задачу: Если a не равно 4, выполнить операторы b=3 и Print b, а в противном случае - операторы b=0, a=b+5 и c=0. Вот однострочный оператор If, решающий эту задачу:

```
If a <> 4  Then  b=3 : Print b  Else  b=0 : a=b+5 : c=0
```

Однако, часто количество операторов после Then и Else бывает гораздо большим, да и сами эти операторы бывают гораздо более сложными и длинными. В этом случае строка становится неудобочитаемой, да и вообще не умещается на экране. Для таких случаев создан **многострочный** (или **блочный**) оператор **If**. Вот как решается наша задача с его помощью:

```
If a <> 4  Then
      b=3
      Print b
Else
      b=0
      a=b+5
      c=0
End If
```

Конструкция End If означает просто, что в этом месте оператор If заканчивается.

Часть Else может и отсутствовать. Например,

```
If a <> 4   Then
        b=3
        Print b
End If
```

Самое замечательное в блочном If то, что здесь можно одно за другим проверять несколько условий. Проиллюстрирую на примерах.

Задача: В компьютер вводится целое число a.

- Если a<0, то компьютер должен сказать "Число отрицательно".
- Если a=0, то компьютер должен сказать "Вы ввели нуль".
- Если a>100, то компьютер должен сказать "Число большое".
- В остальных случаях компьютер ничего не должен говорить, а только вычислить и напечатать его квадрат.

В любом случае после всего этого компьютер должен сказать "До свидания".

Вот программа:

```
Private Sub Command1_Click()
    a = InputBox("Введите число")
    If a < 0 Then
        MsgBox ("Число отрицательно")
    ElseIf a = 0 Then
        MsgBox ("Вы ввели нуль")
    ElseIf a > 100 Then
        MsgBox ("Число большое")
    Else
        Print a ^ 2
    End If
    MsgBox ("До свидания!")
End Sub
```

ElseIf переводят так - "иначе если". Получается вот что: *Если a < 0, то ……. иначе если a = 0, то …… иначе если a > 100, то ……*

Блочный If выполняется так: Сначала проверяется первое условие (a < 0). Если оно не выполняется, то Visual Basic переходит к проверке второго условия(a = 0) и так далее. Наткнувшись наконец на условие, которое выполняется, Visual Basic выполняет операторы, стоящие после его Then и на этом заканчивает работу, даже если ниже есть условия, которые тоже выполняются. Если не выполняется ни одно из условий, Visual Basic выполняет операторы, стоящие за Else.

Вот синтаксис многострочного оператора If:

```
If условие Then
        операторы
        операторы
        ……————
 [ ElseIf условие Then
        операторы
        операторы
        ……————  ]
……————————————————————
[ Else
        операторы
        операторы
        ……————  ]
End If
```

Частей ElseIf может быть сколько угодно или совсем не быть. Часть Else, если она есть, то одна и стоит последней.

Имейте в виду, что у вас нет права, экономя место по вертикали экрана, объединять строки многострочного оператора If, например, так:

~~*If условие Then операторы ElseIf операторы*~~

и я вам не советую (а часто вы и не сможете) переносить слова Then, ElseIf и другие со своего законного места, например, так:

~~*If условие*~~
~~*Then операторы*~~
~~*ElseIf операторы*~~

Задание 24: Компьютер спрашивает пользователя, как его зовут, а затем приветствует его в соответствии с именем: Колю - "Привет", Васю - "Здорово", Джона - "Hi", а остальных - "Здравствуйте". Для Васи, кроме этого, он красит форму в зеленый цвет.

Задание 25: Видоизменить диалог с компьютером, начатый в 4.9. Пусть компьютер, узнав возраст человека, дальнейшую беседу ведет по двум вариантам. Если возраст больше 17, то компьютер должен задать вопрос: "В каком институте ты учишься?" и получив ответ, глубокомысленно заметить "Хороший институт". Если же возраст меньше или равен 17, то соот-

ветственно - "В какой школе ты учишься?" и "Неплохая школа". После этого, каков бы ни был вариант, компьютер должен попрощаться: "До следующей встречи!".

5.6. Ступенчатая запись программы

Ступенчатая запись программы - это как правила хорошего тона - можно и не соблюдать, но посмотрят косо.

Возьмем бессмысленную программу и посмотрим, как она записана. Конкретнее - обратим внимание на то, на что не обращает внимания компьютер, а именно на отступы от левого края листа в записи каждой строки.

```
Private Sub Command1_Click()
    a = InputBox("Введите число")
    If a > 4   Then
        b=3
        Print b
    Else
        b=0
        a=b+5
        c=0
    End If
    b=5
    MsgBox ("До свидания!")
End Sub
```

Строки начала и конца процедуры записаны без отступа. Мы видим, что процедура состоит из четырх операторов: a=, If, b= и MsgBox. Все они выполняются по порядку, один за другим, поэтому каждый из них записан с одинаковым отступом. Если оператор сложный, то есть включает в себя другие операторы (мы знаем пока один такой оператор - If), то составляющие его операторы записываются еще правее. Так, у нас операторы b=0, a=b+5 и c=0 входят в состав оператора If и должны выполняться по порядку один за другим, поэтому их отступ слева одинаков и больше, чем у If.

Сделано все это для удобства чтения программы, для того, чтобы глаз мог сразу же уловить структуру программы, а именно, из каких частей состоит как сама программа, так и каждый из элементов, ее составляющих. Впрочем, вам с первого взгляда может показаться, что такая запись, наоборот, неудобна для чтения. Однако, заметьте, что она принята во всем мире и глаза профессиональных программистов привыкли именно к ней. Настолько привыкли, что программа, записанная без соблюдения ступенчатого стиля, вызывает раздражение.

Конечно, допустимы и некоторые отклонения от ступенчатого стиля. Например, как я уже говорил, несколько коротких похожих операторов вполне можно записать в одну строку:

a=0: b=0: c=0: f=4

Этим мы экономим дефицитное место по вертикали экрана или листа бумаги.

5.7. Вложенные операторы If. Логические операции и выражения

Умный компьютер решает сложные логические задачи. Для этого нужны сложные средства языка. Рассмотрим их.

Вложенные операторы If

Согласно синтаксической схеме оператора If, после Then и Else может стоять любой оператор Visual Basic, в том числе и If.

Решим задачу: В компьютер вводится число (пусть для конкретности это будет дальность какого-нибудь выстрела). Если оно находится в интервале от 28 до 30, то напечатать текст ПОПАЛ, иначе - НЕ ПОПАЛ.

Сначала составим алгоритм: Введи число. Если оно меньше 28, то печатай НЕ ПОПАЛ, в противном случае надо еще подумать. А о чем же думать? А вот о чем: Если число меньше 30, то печатай ПОПАЛ, иначе печатай НЕ ПОПАЛ.

А теперь по составленному алгоритму напишем программу:

```
Private Sub Command1_Click()
    a = InputBox("Введите дальность выстрела")
    If a < 28 Then
        MsgBox ("НЕ ПОПАЛ")
    Else
        If a < 30 Then MsgBox ("ПОПАЛ") Else MsgBox ("НЕ ПОПАЛ")
    End If
End Sub
```

Здесь оператор If a < 30 входит в состав оператора If a < 28. Говорят, что он вложен в него.

Эту же программу можно записать и без вложенного If:

```
Private Sub Command1_Click()
    a = InputBox("Введите дальность выстрела")
    If a < 28 Then
        MsgBox ("НЕ ПОПАЛ")
    ElseIf a < 30 Then
        MsgBox ("ПОПАЛ")
```

```
        Else
            MsgBox ("НЕ ПОПАЛ")
        End If
    End Sub
```

Задание 26: Усложним задание 19 из 5.3: В компьютер вводятся длины трех отрезков. Компьютер должен ответить на вопрос, правда ли, что эти отрезки могут образовать треугольник.

Указание: Для этого каждый отрезок должен быть меньше суммы длин двух других отрезков.

Логические операции

Применение вложенных If создает довольно громоздкую, трудную для понимания программу. Поэтому в Visual Basic есть возможность записывать многие программы короче и понятнее, используя вместо вложенных друг в друга If только один If. Для этого используются так называемые **логические операции**. Что это такое, разберем на примерах.

Задача "Разборчивая принцесса". В прихожей у принцессы - длинная очередь женихов. Принцессе нравятся только голубоглазые маленького роста. Устав принимать женихов и отбирать из них подходящих, принцесса вместо себя поставила компьютер, написав для него программу, которая говорит ВЫ МНЕ ПОДОЙДЕТЕ тем, у кого цвет глаз голубой и рост меньше 140 см. Остальным программа говорит ДО СВИДАНИЯ.

Вот эта программа:

```
Dim Tsvet As String              'Цвет
Dim Rost As Integer:             'Рост
Private Sub Command1_Click()
    Tsvet = InputBox("Каков цвет ваших глаз?")
    Rost = InputBox("Введите ваш рост в сантиметрах")
    If  Tsvet = "Голубой" And Rost < 140   Then Print  "ВЫ МНЕ ПОДОЙДЕТЕ"    Else Print "ДО СВИДАНИЯ"
End Sub
```

Мы видим, что условие в операторе If уже не такое простое, как мы описывали раньше, а сложное, то есть состоящее из двух условий, соединенных знаком логической операции **And** (переводится "и"). Весь оператор If можно прочесть так - если цвет глаз голубой **И** рост меньше 140 сантиметров, то печатай ВЫ МНЕ ПОДОЙДЕТЕ, иначе печатай ДО СВИДАНИЯ.

Знак логической операции *And*, поставленный между двумя условиями, говорит о том, что должны выполняться сразу оба эти условия.

Поэтому наш оператор If ответит ДО СВИДАНИЯ и высоким голубоглазым, и высоким неголубоглазым, и маленьким неголубоглазым. И лишь маленьким голубоглазым он ответит ВЫ МНЕ ПОДОЙДЕТЕ. В общем, And - строгий знак.

Программа для задачи ПОПАЛ - НЕ ПОПАЛ при использовании логических операций значительно упростится:

```
Private Sub Command1_Click()
    a = InputBox("Введите дальность выстрела")
    If a > 28 And a < 30 Then MsgBox ("ПОПАЛ") Else MsgBox ("НЕ ПОПАЛ")
End Sub
```

Задача "Неразборчивая принцесса". Неразборчивой принцессе нравятся все маленькие независимо от цвета глаз и все голубоглазые независимо от роста. Программа неразборчивой принцессы будет отличаться от программы разборчивой одним единственным знаком логической операции:

```
If   Tsvet ="Голубой"   Or   Rost<140                'Если цвет голубой  ИЛИ  рост<140
```

Знак логической операции Or переводится "или".

Поставленный между двумя условиями, знак *Or* говорит о том, что достаточно, если будет выполняться хотя бы одно из них.

Поэтому теперь оператор If ответит ВЫ МНЕ ПОДОЙДЕТЕ и высоким голубоглазым и маленьким голубоглазым и маленьким неголубоглазым. И лишь высоким неголубоглазым он ответит ДО СВИДАНИЯ.

Знаками And и Or можно объединять сколько угодно условий. Например:

```
If   a>2 Or x=b Or c<>1   Then k=99   Else k=33
```

Здесь выполнится оператор k=99, если верно хотя бы одно из трех условий, и лишь когда все три неверны, будет выполняться оператор k=33.

Кроме логических операций And и Or применяется еще логическая операция **Not** (переводится "НЕ"). Запись If Not a>b Then... переводится так - ЕСЛИ НЕПРАВДА, ЧТО a больше b, ТО.... Вот фрагмент:

```
a=2:    b=3:    If   Not a>b   Then k=1   Else k=0
```

Здесь выполнится оператор k=1, так как неправда, что 2>3.

Логические выражения

Выражения

```
a>b
```

a > 28 And a < 30
Tsvet ="Голубой" Or Rost<140
a>2 Or x=b Or c<>1

имеют ту общую черту, что про каждое из них можно сказать, верно оно или нет в каждый момент времени. Такие выражения называются **логическими выражениями**. Если логическое выражение верно, то говорят, что оно имеет значение **True** (Правда). Если логическое выражение неверно, то говорят, что оно имеет значение **False** (Ложь). Любое логическое выражение может стоять в качестве условия в операторе If.

Логические выражения могут быть сложными - содержать одновременно операции And, Or, Not. Например, такое:

a>2 And Not *b>3 Or s>8*

Чтобы его понять, нужно знать порядок действий. В арифметике сначала выполняется умножение, потом сложение. В логических выражениях сначала выполняется Not, затем *And*, затем *Or*. Для облегчения понимания не возбраняется расставлять скобки:

*(a>2 And (*Not *b>3)) Or s>8*

Это выражение равносильно предыдущему.

Скобки можно расставлять и чтобы изменить порядок действий:

a>2 And Not *(b>3 Or s>8)*

Примеры:

ФРАГМЕНТ	РЕЗУЛЬТАТ
a=8: b=6: **If** a>b **And** b>7 **Then** k=1 **Else** k=0	k=0
a=8: b=6: **If** a>b **Or** b>7 **Then** k=1 **Else** k=0	k=1
a=8: b=6: **If** a<b **Or** b>7 **Then** k=1 **Else** k=0	k=0
a=8: b=6: **If** **Not** a=8 **Then** k=1 **Else** k=0	k=0

Задание 27: "Замысловатая принцесса". Определите, кто нравится принцессе, по фрагменту из ее программы:

```
If Tsvet ="Черный" And (Rost<180 Or Rost>184) Then Print "ВЫ МНЕ ПОДОЙДЕТЕ" Else Print "ДО СВИДАНИЯ"
```

Задание 28: Усложним нашу задачу про ПОПАЛ - НЕ ПОПАЛ: Человек вводит в компьютер число. Если оно находится в интервале от 28 до 30, то нужно напечатать текст ПОПАЛ, если оно больше или равно 30 - то ПЕРЕЛЕТ, если оно находится на отрезке от 0 до 28, то НЕДОЛЕТ, если число меньше нуля - НЕ БЕЙ ПО СВОИМ.

Задание 29:

Человек вводит с клавиатуры строку, смысл которой - приветствие при встрече. Компьютер тоже должен ответить приветствием. Отвечать нужно в соответствии со следующей таблицей:

ПРИВЕТСТВИЕ ЧЕЛОВЕКА	ОТВЕТ КОМПЬЮТЕРА
Привет	Привет
Здравствуйте	Здравствуйте
Добрый день	Салют
Приветик	Салют
Салют	Салют
Здравия желаю	Вольно
Любое другое приветствие	Я вас не понимаю

5.8. Оператор варианта Select Case

У авторов языков программирования есть похвальное стремление сделать язык попроще, попонятнее. Они с ужасом взирают на многочисленные "иначе если" и логические операции и стараются, где можно, от них избавиться. Возьмем, например, такую задачу: Компьютер спрашивает школьника, какую он получил отметку по физике, и реагирует на нее подходящим текстом. Вот программа без нововведений, использующая If:

```
Dim Otmetka As Integer
Private Sub Command1_Click()
    Otmetka = InputBox("Какую отметку ты получил по физике?")
    If Otmetka = 1 Or Otmetka = 2 Then
        Print "Кошмар!"
    ElseIf Otmetka = 3 Then
        Print "Неважно"
    ElseIf Otmetka = 4 Then
        Print "Неплохо"
    ElseIf Otmetka = 5 Then
        Print "Молодец!"
    Else
        Print "Таких отметок не бывает"
    End If
```

```
End Sub
```

Здесь может вызвать раздражение слишком часто встречающееся имя Otmetka, а также то, что и в такой простой задаче не обошлось без логических операций. Хорошо бы программу можно было писать попроще, например, так (по-русски):

```
Выбери вариант отметки
    Вариант    1, 2
        печатай "Кошмар!"
    Вариант    3
        печатай "Неважно"
    Вариант    4
        печатай "Неплохо"
    Вариант    5
        печатай "Молодец!"
    Вариант    остальное
        печатай "Таких отметок не бывает"
Конец выбора
```

И такой оператор варианта был придуман и назван **Select Case**, что и означает в переводе ВЫБЕРИ ВАРИАНТ. Я просто-напросто переписываю русский вариант программы по-английски:

```
Dim Otmetka As Integer
Private Sub Command1_Click()
    Otmetka = InputBox("Какую отметку ты получил по физике?")
    Select Case Otmetka
        Case 1, 2
            Print "Кошмар!"
        Case 3
            Print "Неважно"
        Case 4
            Print "Неплохо"
        Case 5
            Print "Молодец!"
        Case Else
            Print "Таких отметок не бывает"
    End Select
End Sub
```

Логика работы Select Case абсолютно такая же, как и у блочного If. В процессе исполнения оператора компьютер сравнивает значение переменной Otmetka по очереди со всеми значениями, перечисленными в вариантах. Наткнувшись на совпадающее значение, он выполняет операторы, стоящие в этом варианте. На этом исполнение оператора Select Case завершается. Если же совпадающего значения так и не нашлось, то выполняются операторы, стоящие в варианте Else (в нашей программе он полезен на тот случай, если ученик болен манией величия и вводит число 6).

Оператор Select Case предоставляет более широкие возможности, чем в только что рассмотренном примере. Проиллюстрируем их на другом примере:

```
Private Sub Command1_Click()
    a = 3
    Select Case a * a + 1
        Case 8, 4 * a, 26
            k = 0
            Print k
            Print a
        Case 7, 10, 84 To 90
            k = 1
            Print k
        Case Is < 0, 2, 4, 12 To 18 + a, 44, 68, Is > 100 + a
            k = 3
    End Select
End Sub
```

Эта программа напечатает 1. Здесь мы видим несколько новых для нас элементов:

Во-первых, после слов Select Case стоит не переменная, а выражение, поэтому с перечисленными в вариантах значениями будет сравниваться число 10, полученное как 3*3+1. В качестве значений вариантов тоже могут быть выражения, как, например, у нас - 4 * a.

Во-вторых, здесь у нас в двух вариантах не по одному, а по нескольку выполняющихся операторов.

В-третьих - конструкция 84 To 90. Она обозначает то же, что и 84,85,86,87,88,89,90, и служит в нашем случае для сокращения записи.

В-четвертых - конструкция **Is < 0**. Слово Is служит заменителем выражения a*a+1 и тоже используется для сокращения. Таким образом, Select Case не только сравнивает значения на равенство, но и проверяет неравенства.

В-пятых, здесь отсутствует вариант Else. Это значит, что если бы в нашей программе мы вместо a=3 написали a=0, то оператор Select Case, не найдя совпадения, не выбрал бы ни один из своих вариантов и, не найдя также Else, завершил бы свою работу, так ничего и не сделав.

Чем платим за удобство Select Case по сравнению с If? Что может If такого, чего не может Select Case? Самое главное - условия в If могут быть совершенно произвольны, а в Select Case мы привязаны к a*a+1.

Вот синтаксис оператора Select Case:

```
Select Case проверяемое выражение
        [Case значение, значение……
                [операторы
                 операторы
                ……..]]
        [Case значение, значение……
                [операторы
                 операторы
                ……..]]
        ……
        [Case Else
                [операторы
                 операторы
                ……..]]
End Select
```

Здесь ***значение*** - это:
- *выражение*
- *выражение* **To** *выражение*
- **Is** *знак сравнения* *выражение*

Выражения могут быть не только числовые, но и строковые. Пример:

```
Private Sub Command1_Click()
    a = "Дом"
    Select Case a
        Case "Домик"
            k = 3
            Print a
        Case "Дом"
            k = 0
            Print k
    End Select
End Sub
```

Здесь будет напечатан 0.

Задание 30: Ученик вводит с клавиатуры букву русского алфавита. Компьютер должен сказать, какой звук обозначает это буква - гласный, согласный звонкий, согласный глухой или какой-нибудь другой (можно и НЕ ЗНАЮ).

Задание 31: Если у вас есть микрофон и вы можете записывать свой голос в файл, то попробуйте усовершенствовать ваш диалог с компьютером из 4.9. Пусть компьютер подает вам реплики голосом. Для этого вам заранее придется записать на диск некоторое количество звуковых файлов с репликами компьютера и при помощи оператора Select Case выбирать между ними в зависимости от реплики человека с клавиатуры. Вы сразу же заметите, что компьютер поглупел, но зато стал изъясняться вслух.

5.9. Улучшаем калькулятор.

Наш калькулятор не предохранен от попыток выполнения арифметических действий над текстом и от деления на ноль. Предохраним его от этого, а также научимся ставить пароль на проект.

Проверка ввода чисел в текстовое поле

Давайте запустим наш калькулятор в том виде, который он получил в 4.5

```
Dim Чис1 As Double          'Переменная, содержащая число из текстового поля Число1
Dim Чис2 As Double          'Переменная, содержащая число из текстового поля Число2
Dim Рез As Double           'Переменная-результат, предназначенный для текстового поля Результат
_____________________________________________________________________
Private Sub Кл_сложения_Click()
   Чис1 = Число1.Text       'Значения исходных данных переходят в переменные из текстовых полей
   Чис2 = Число2.Text
   Рез = Чис1 + Чис2        'Обработка переменных для получения результата
   Результат.Text = Рез     'Значение результата переходит из переменной в текстовое поле
End Sub
..............................
```

Попробуем ввести в верхнее текстовое поле вместо числа какую-нибудь ерунду, например, *КУ-КУ*. Щелкнем по кнопке сложения. Visual Basic выдает сообщение об ошибке "Type mismatch" - несовпадение типов числовой переменной Чис1 и вводимой текстовой информации. Все верно.

Теперь подумаем - хорошо ли это, что Visual Basic реагирует на ваши неправильные действия собственными сообщениями об ошибке? Для вас, которые уже что-то умеют в программировании и делают калькулятор исключительно для собственного удовольствия, может быть и хорошо. Ну а представьте, что вы решили похвастаться калькулятором перед своей подругой, которая знакома с программированием только понаслышке, и она, работая на нем, допустила такую же ошибку? Что она подумает, увидев неожиданно возникшее окно с непонятными английскими словами? Вы думаете, она поймет, что это сообщение об ошибке? Скорее всего она подумает, что это сломался компьютер. И на какие кнопки, скажите на милость, ей нажимать? "Ну и гадость твой калькулятор!" - скажет она наконец и будет сто раз права.

Та же проблема стоит и перед всеми профессиональными программистами, которые создают свои продукты для использования людьми, не обязанными ничего знать о программировании. Программист обязан сделать так, чтобы на все неправильные действия пользователя реагировал не Visual Basic своими непонятными сообщениями, а сама программа, причем понятно, по-русски, и тут же давала возможность исправиться.

Для этого в Visual Basic должны быть средства. И они там есть. Что нам нужно сейчас? Нам нужно средство, которое определяло бы, число ли записано в текстовом окне или же все, что угодно, но только не число. Это функция **IsNumeric**. Ее аргумент может быть любого типа, в том числе и строкой. Функция анализирует строку и если видит, что она состоит из цифр, знака минус и запятой таким образом, что получается число (и больше ничего в строке нет), то функция получает значение **True**, что означает "Правда", в противном случае - **False**, что означает "Ложь". Проверим:

Оператор Debug.Print IsNumeric("КУ-КУ") печатает False.

Оператор Debug.Print IsNumeric("-67,3") печатает True.

Раз так, то функцию IsNumeric можно включать в логические выражения и использовать в качестве условия оператора If. Например,

```
If IsNumeric("-67,3") Then MsgBox ("Это число") Else MsgBox ("Это не число")
```

что означает: "Если -67,3 - число, то …" Поэтому же совсем не обязательно писать If IsNumeric("-67,3") = True Then ….

Алгоритм работы калькулятора с проверкой звучит примерно так: Если в текстовом поле *Число1* введено число и в текстовом поле *Число2* введено число, то делай все, что положено, иначе выдавай сообщение "Вводите только числа". Процедура сложения с учетом этого алгоритма будет выглядеть так:

```
Private Sub Кл_сложения_Click()
    If IsNumeric(Число1) And IsNumeric(Число2) Then
        Чис1 = Число1.Text
        Чис2 = Число2.Text
        Результат.Text = Чис1 + Чис2
    Else
        MsgBox ("Вводите только числа")
    End If
End Sub
```

Здесь я немного сэкономил на переменной Рез. Поскольку мы ее никак не обрабатываем, я обошелся без нее. Процедуры для остальных действий пишутся аналогично.

Запрет деления на ноль

Вторая ошибка, на которую реагирует Visual Basic, это деление на ноль. От нее мы избавимся, если запретим компьютеру делить на ноль, записав вместо оператора

```
Результат.Text = Чис1 / Чис2
```

такой:

```
If Чис2 <> 0 Then Результат.Text = Чис1 / Чис2 Else MsgBox ("На ноль делить нельзя")
```

Вот как будет выглядеть теперь процедура деления:

```
Private Sub Кл_деления_Click()
    If IsNumeric(Число1) And IsNumeric(Число2) Then
        Чис1 = Число1.Text
        Чис2 = Число2.Text
        If Чис2 <> 0 Then Результат.Text = Чис1 / Чис2 Else MsgBox ("На ноль делить нельзя")
    Else
        MsgBox ("Вводите только числа")
    End If
End Sub
```

Как видите, здесь в состав многострочного If входит однострочный.

Ставим пароль на калькулятор

Ваш калькулятор стал достаточно надежен и удобен. Теперь его не стыдно показать друзьям. Ну а защищаться от врагов будем паролем.

Наша задача - сделать так, чтобы при попытке запустить калькулятор на экране появлялось приглашение ввести пароль, известный только вам. При попытке ввода неправильного пароля, программа должна заканчивать свою работу.

Поскольку приглашение на ввод пароля должно появляться раньше всего остального, то программируем его в процедуре Form_Load, в самом начале. Предварительно объявим строковую переменную для хранения пароля и выдумаем сам пароль, например, "калям".

```
Dim Parol As String          'Переменная-пароль
```

```
Private Sub Form_Load()
    Parol = InputBox("Введите пароль")
    If Parol <> "калям" Then MsgBox ("Пароль неверный!"): End
End Sub
```

Новый для вас оператор **End** делает всего одну вещь - вызывает завершение программы.

Запустите проект и проверьте, как он работает.

То же самое можно было бы запрограммировать короче, без использования переменной:

```
Private Sub Form_Load()
    If InputBox("Введите пароль") <> "калям" Then MsgBox ("Пароль неверный!"): End
End Sub
```

Вообще, переменные нужны, если нужна неоднократная обработка какой-то информации: складывать что-то, затем сравнивать это что-то с чем-то другим и т.д. В нашем же случае пароль нужен всего один раз, для сравнения, так что можно обойтись и без переменной.

Вы скажете: Кто угодно перед запуском моей программы посмотрит в ее текст и сразу же увидит пароль. Совершенно верно. Чтобы текст программы не был виден, преобразуйте проект в исполнимый файл, как мы это делали в 2.13.

Сделаем так, чтобы в случае ввода пароля в текстовое поле там появлялись не буквы пароля, а звездочки. В этом случае никто из-за вашей спины не сможет пароль подсмотреть. Для этого достаточно до ввода пароля выполнить строку

Text1.**PasswordChar** = "*"

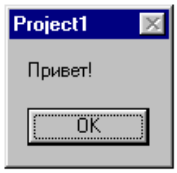
5.10. Функция MsgBox

В 2.10 я вкратце описал, как пользоваться **оператором MsgBox**. Здесь я более подробно разберу действие **функции MsgBox**. Функция отличается от оператора тем, что она встречается в выражениях и в правой части оператора присваивания, и еще тем, что подобно переменной имеет значение. Например, возьмем оператор присваивания:

y = MsgBox("Привет!")

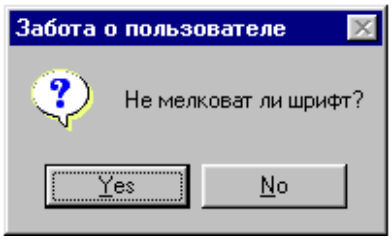
При выполнении этого оператора мы увидим на экране точно такое же окно сообщения, как при выполнении оператора

MsgBox("Привет!")



Но вдобавок к этому, после того, как мы нажмем на кнопку OK, переменной y будет присвоено некое значение. Что это за значение и зачем оно нужно, мы сейчас выясним.

Задача: В текстовом поле Visual Basic для нашей пользы печатает, сколько будет дважды два. После чего, заботясь о нашем зрении, он выводит следующее окно сообщения:



При нажатии на Yes шрифт увеличивается, при нажатии на No ничего не происходит.

Вот программа:

```
Private Sub Command1_Click()
   Text1.Text = 2 * 2
   y = MsgBox ("Не мелковат ли шрифт?",   vbQuestion + vbYesNo,   "Забота о пользователе")
   If  y = vbYes  Then  Text1.FontSize = 20
End Sub
```

Пояснения: Здесь полужирным шрифтом я выделил новые для вас элементы. Константа **vbQuestion** означает приказ изменить внешний вид окна сообщения, конкретнее - изобразить в нем картинку со знаком вопроса. Константа **vbYesNo** означает приказ поместить в окне сообщения кнопки Yes и No. Знаком плюс эти константы соединены, так как имеют некие численные значения, о которых вам, впрочем, не обязательно задумываться. После нажатия на кнопку Yes функция MsgBox приобретает значение vbYes. После нажатия на кнопку No функция MsgBox приобретает значение vbNo. Поэтому фрагмент

If y = vbYes …

можно перевести так: "Если была нажата кнопка Yes …"

Вот константы для задания внешнего вида окна сообщения: **VbCritical, vbQuestion, vbExclamation, vbInformation**.

Вот константы для задания кнопок в окне сообщения: **VbOKOnly, vbOKCancel, vbAbortRetryIgnore, vbYesNoCancel, vbYesNo, vbRetryCancel**.

Вот константы для задания значения функции MsgBox после нажатия на кнопку: **vbOK, vbCancel, vbAbort, vbRetry,**

vbIgnore, vbYes, vbNo.

Поэкспериментируйте со внешним видом окна сообщения. Поведение окна сообщения определяется еще кое-какими константами, но мы на них не будем останавливаться.

Названия кнопок сами по себе не играют никакой роли, их действие полностью определяется кодом, который вы напишете в процедуре. Однако, в среде пользователей и программистов уже утвердились некоторые привычки, которые вам небесполезно знать:

OK - просто принять сообщение к сведению, *Cancel* - отменить намечавшееся действие, *Abort* - прекратить неудавшуюся попытку (например, когда ваша процедура занимается распечаткой документа на принтере и приходит сообщение "Принтер не готов"), *Retry* - повторить неудавшуюся попытку, *Ignore* - проигнорировать предупреждение и продолжать, как ни в чем не бывало, *Yes-No* - ответить да или нет на вопрос, содержащийся в сообщении.

Глава 6. Циклические программы

Идею цикла я поясню на примере движения объектов по экрану. Возьмем игру в воздушный бой. Самолетик по экрану должен двигаться. Но в списках операторов большинства языков программирования, используемых профессиональными программистами для создания игр, нет команды движения. Здесь нужно задаться вопросом - а что такое движение? Рассмотрим иллюзию движения, возникающую на экране кинотеатра. Если вы держали в руках кинопленку фильма, изображающего, скажем, движение автомобиля, то должны были обратить внимание, что она состоит из множества неподвижных слайдов-кадров, на каждом следующем из которых автомобиль находится чуть-чуть в другом месте, чем на предыдущем. Показывая эти кадры один за другим с большой скоростью, мы создаем иллюзию движения автомобиля. Точно так же поступают с созданием иллюзии движения на экране компьютера. Запишем алгоритм движения самолетика по экрану слева направо:

1. Зададим в уме компьютера позицию самолетика в левой части экрана.
2. Нарисуем самолетик.
3. Сотрем его.
4. Изменим в уме компьютера позицию самолетика на миллиметр правее.
5. Перейдем к команде 2.

Каждая из приведенных команд алгоритма легко программируется на большинстве языков. Любой компьютер, выполнив очередную команду, автоматически переходит к выполнению следующей. Так, выполнив команду 2, компьютер всегда перейдет к выполнению команды 3, независимо от того, какую команду он выполнял перед командой 2. Однако, если мы захотим, то можем заставить компьютер изменить этот порядок, что мы и сделали в команде 5. Из-за нее компьютер выполняет команды в таком порядке: 1-2-3-4-5-2-3-4-5-2-3-…. Таким образом, многократно выполняется последовательность команд 2-3-4-5. Такая многократно выполняемая последовательность называется **циклом**. Можно сказать, что цикл - это одно из средств заставить компьютер долго работать при помощи короткой программы.

В коротком промежутке времени после выполнения команды 2 самолетик будет появляться на экране и на команде 3 исчезать, но этого достаточно, чтобы человеческий глаз его заметил. Благодаря циклу самолетик будет мелькать каждый раз в новом месте, а поскольку смена “кадров” будет очень быстрой, нам будет казаться, что происходит плавное движение самолетика.

Не нужно думать, что циклы применяются лишь при движении. Они пронизывают все программирование.

6.1. Оператор перехода GoTo. Цикл. Метки

Посмотрим, как осуществить цикл в Visual Basic. Предположим, мы хотим, чтобы компьютер бесконечно повторял выполнение следующего фрагмента:

```
Debug.Print "Это ";
Debug.Print "тело ";
Debug.Print "цикла";
Debug.Print "      ";
```

в результате чего в окне Immediate мы бы увидели:

```
Это тело цикла     Это тело цикла     Это тело цикла     Это тело цикла  . . . .
```

Если бы операторы Visual Basic можно было писать по-русски, то для достижения нашей цели было бы естественно воспользоваться такой конструкцией:

```
метка m1:    Debug.Print  "Это " ;
             Debug.Print  "тело " ;
             Debug.Print  "цикла" ;
             Debug.Print  "      " ;
             иди к оператору, помеченному меткой m1
```

Здесь мы видим новый для нас "оператор" ИДИ, который выполняется после Debug.Print " " и единственная работа которого заключается в том, чтобы заставить компьютер перескочить к выполнению оператора Debug.Print "Это " , помеченного меткой m1.

А вот как этот фрагмент выглядит реально на Visual Basic:

```
Private Sub Command1_Click()
m1:   Debug.Print "Это ";
      Debug.Print "тело ";
      Debug.Print "цикла";
      Debug.Print "      ";
      GoTo m1
End Sub
```

Здесь GoTo - оператор безусловного перехода, переводится "иди к", m1 - метка.

Метка - это произвольное имя или произвольное не слишком большое целое положительное число. Оператор GoTo можно

писать в любых местах процедуры и метку можно ставить перед любым оператором процедуры, заставляя компьютер таким образом перескакивать внутри процедуры откуда угодно куда угодно. Правда, в сложных процедурах и внутри сложных операторов эта свобода перескакивания существенно ограничивается, так что я не советую вам врываться снаружи внутрь вложенных операторов, а вот изнутри наружу - пожалуйста. Метка должна заканчиваться двоеточием (хотя, в случае метки-числа это не обязательно).

А терерь запустите эту программу в пошаговом режиме. Посмотрите, как заполняется окно Immediate. Чтобы оно заполнялось быстрее, нажмите клавишу F8 и не отпускайте.

Зацикливание. *Если вы уже запустили эту программу обычным образом (не в пошаговом режиме), то через некоторое время перед вами должен встать жизненно важный вопрос – как же ее остановить? Вы обнаружите, что кнопки не отзываются на нажатие мыши, и вообще, программа никак не реагирует ни на мышку, ни на клавиши клавиатуры. Любопытно, что так поступает любая нормальная программа, выполняя операторы кода. Разница в том, что в нормальной программе исключены ситуации, когда код выполняется бесконечно или на протяжении слишком долгого времени. Если же вы допустили ошибку и в программе выполняется бесконечный цикл, то возникает как раз такая ситуация. Вы вечно будете смотреть на экран, по которому бесконечно бегут непонятные числа или слова или рисуются графические объекты, а возможно и ничего не происходит, экран пустой – все зависит от характера ошибки.*

Для прерывания работы программы, в том числе и зациклившейся, существует комбинация клавиш Ctrl-Break. Имеется в виду, что, удерживая нажатой клавишу Ctrl, вы должны щелкнуть по клавише Break. Программа прерывает свою работу, но не заканчивает. Visual Basic переходит в режим прерывания. Оператор программы, на котором она была прервана, выделяется полосой желтого цвета. Если вы снова запустите программу, она продолжит работу с прерванного места. Кстати, продолжить можно и в пошаговом режиме. Чтобы начать сначала, необходимо завершить работу программы обычным образом.

Группа операторов, выполняющихся многократно, называется **телом цикла**. У нас это все операторы, начиная с Debug.Print "Это " и кончая GoTo m1.

Пример программы:

```
     k = 6
     a = 100
     GoTo 8
     a = a + k
     k = 2 * k
     Print a
8:   a = a + 1
     k = k + 10
     Print k, a
```

Эта программа напечатает 16 101. Операторы выполняются в такой последовательности:

```
k=6
a=100
GoTo 8
a=a+1
k=k+10
Print k, a
```

А операторы a=a+k, k=2*k, Print a выполнены не будут вообще, несмотря на то, что написаны. Цикла здесь нет.

Задание 32: Определить без компьютера, что будет печатать программа:

```
        n = 10
        k = 0
        Debug.Print "Считаем зайцев"
met5:   Debug.Print n;
        n = n + k
        GoTo m1
        n = n + 1
m1:     Debug.Print "зайцев"
        k = k + 1
        GoTo met5
        Debug.Print "Посчитали зайцев"
```

Не можете определить - посмотрите в пошаговом режиме.

Задача: Бесконечно печатать 200 205 210 215 220 225 ...

Программа:

```
Private Sub Command1_Click()
        n = 200
m1:     Debug.Print n
        n = n + 5
        GoTo m1
End Sub
```

Задания 33-35:

Написать программы для выполнения следующих заданий:

33. Бесконечно печатать букву А: AAAAAAAAAAA.....

34. Бесконечно печатать 10000 9999 9998 9997 9996 .....

35. Бесконечно печатать 100 50 25 12.5.... с 8 десятичными знаками.

Движение объектов по экрану

Вам уже приходилось заставлять кнопки прыгать по экрану. Попробуем добиться плавного движения объекта. Создайте проект с большой формой и добавьте в него маленький элемент управления Image (изображение). Поместите его в левой части формы. Придайте ему картинку (свойство Picture). Лучше всего, пока вы еще не умеете работать с изображениями, в качестве картинки взять один из файлов значков, находящихся по адресу c:\Program Files*папка,посвященная Visual Basic*\Graphics\Icons.

Напишем программу, которая двигала бы изображение плавно направо:

```
Dim x As Double
Private Sub Command1_Click()
        x = Image1.Left        'Компьютер узнает, откуда начинать движение
m1:     x = x + 1              'Компьютер увеличивает в уме горизонтальную координату на 1
        Image1.Left = x        'Изображение встает на место, указанное гор. координатой
        GoTo m1
End Sub
```

Если движение получилось слишком медленным, то прибавьте шаг - x = x + 2. Если слишком быстрым, то уменьшите - x = x + 0.3.

Не удивляйтесь, что вам не пришлось в цикле рисовать и стирать объект. Когда дело касается объекта, заботы о перерисовке берет на себя Visual Basic. А вот когда вы сами будете рисовать различные фигуры на форме и попытаетесь их двигать, тогда вам придется их и рисовать и стирать.

Задание 36: Заставьте изображение двигаться налево, вниз, вверх.

Пока мы никак не можем влиять на полученное движение. Только можем останавливать компьютер с клавиатуры. Как с помощью мышки или клавиатуры влиять на движение во время движения? Как хотя бы запрограммировать остановку в нужном месте? Об этом - в 6.2 и в 11.4.

6.2. Выход из цикла с помощью If

Итак, как нам запрограммировать завершение работы цикла? Для этого нужно применить оператор GoTo внутри оператора If.

Задача: При помощи цикла напечатать:

Начало счета 3 5 7 9 11 13 15 ….. 329 331 333 Конец счета

Для удобства отладки изменим условие задачи - напечатать:

Начало счета 3 5 7 9 Конец счета

Вы уже достаточно опытны в программировании, чтобы догадаться, что программа для измененного условия будет копией программы для исходного, за исключением одного числа.

Вот 4 варианта программы. Первый – самый простой, а остальные нам понадобятся в дальнейшем. Все 4 варианта делают одно и то же, они очень похожи, но чем-то и отличаются. Вот в этом отличии вам и надо разобраться, иначе не будет понятен дальнейший материал.

Создайте проект с 4 кнопками и выполните в пошаговом режиме все 4 варианта (не обращая пока внимания на непонятные слова в заголовках таблиц, эти слова понадобятся чуть позже):

1 ВАРИАНТ **(Do …. Loop While)**	2 ВАРИАНТ **(Do …. Loop Until)**
Private Sub Command1_Click() Debug.Print "Начало счета"; f = 3 m: Debug.Print f; f = f + 2 If f <= 9 Then GoTo m Debug.Print "Конец счета" End Sub	Private Sub Command2_Click() Debug.Print "Начало счета"; f = 3 m1: Debug.Print f; f = f + 2 If f > 9 Then GoTo m2 Else GoTo m1 m2: Debug.Print "Конец счета" End Sub

Вот в каком порядке выполняются операторы программы первого варианта:

Debug.Print "Начало счета" f=3 Debug.Print f **{печатается 3}** f=f+2 **{f становится равным 5}** If f<=9 Then GoTo m Debug.Print f **{печ. 5}** f=f+2 **{f = 7}** If f<=9 Then GoTo m Debug.Print f **{печ. 7}** f=f+2 **{f = 9}** If f<=9 Then GoTo m Debug.Print f **{печ. 9}** f=f+2 **{f = 11}** If f<=9 Then ~~GoTo m~~ Debug.Print " Конец счета"

Здесь оператор GoTo m выполняется три раза. На четвертый раз условие f<=9 оказывается ложным и поэтому выполняет-

ся не GoTo m, а следующий за If оператор Debug.Print "Конец счета", то есть программа выходит из цикла и завершает свою работу.

3 ВАРИАНТ **(Do While …. Loop)**	4 ВАРИАНТ **(Do Until …. Loop)**
Private Sub Command3_Click() Debug.Print "Начало счета"; f = 3 m1: If f <= 9 Then GoTo m3 Else GoTo m2 m3: Debug.Print f; f = f + 2 GoTo m1 m2: Debug.Print "Конец счета" End Sub	Private Sub Command4_Click() Debug.Print "Начало счета"; f = 3 m1: If f > 9 Then GoTo m2 Else GoTo m3 m3: Debug.Print f; f = f + 2 GoTo m1 m2: Debug.Print "Конец счета" End Sub

Задача: Напечатать пары чисел - 1 1001 2 1002 3 1003 …….. 100 1100.
Программа:

```
Private Sub Command1_Click()
    f = 1
m:  s = f + 1000
    Debug.Print f; s
    f = f + 1
    If f <= 100 Then GoTo m
End Sub
```

Задания 37-41:

37. А. Напечатать 1 2 3 4 . . . 99 100

Б. Напечатать 100 99 . . . 3 2 1.

В. Напечатать 1 2 3 4 . . . 99 100 99 . . . 3 2 1.

38. "*Таблицы Брадиса" или "Таблицы логарифмов" (в те времена, когда не было калькуляторов, были такие напечатанные в виде брошюрок таблицы для школьников и студентов, по которым они могли быстро посмотреть численные значения квадратов, логарифмов и других математических функций).* Вычислить и напечатать с 6 десятичными знаками квадраты чисел 0.000 0.001 0.002 0.003 . . . 0.999 1.000.

39. Для x=2700, 900, 300, 100 . . . и т.д. вычислять и печатать y=x/4 + 20 и z=2y+0.23 до тех пор, пока yz не станет меньше 1/x.

40. Пусть движущееся изображение, описанное в 6.1, через некоторое время остановится.

41. Изображение, пройдя немного слева направо, поворачивает вниз и, пройдя немного, через некоторое время останавливается

6.3. Операторы цикла Do

Циклы настолько широко применяются в программах, что у программистов давным-давно появилась потребность написать специальный оператор цикла, не использующий оператор GoTo. Последний неудобен хотя бы тем, что у программистов, пишущих большие программы, много времени и внимания уходит на поиск взглядом меток в тексте программы. К тому же GoTo нарушает стройную идеологию так называемого "структурного программирования", когда порядок действий задается не скачками из одной части программы в другую, а цепочкой вложенных друг в друга операторов. В общем, нынче широко использовать GoTo так же неприлично, как не объявлять переменные.

Операторы цикла в Visual Basic делятся на 2 вида: **Do** и **For**. Операторы вида Do встречаются в 5 вариантах:

0 ВАРИАНТ	1 ВАРИАНТ	2 ВАРИАНТ	3 ВАРИАНТ	4 ВАРИАНТ
Do …. Loop	**Do …. Loop While**	**Do …. Loop Until**	**Do While …. Loop**	**Do Until …. Loop**

Операторы вида For встречаются в 2 вариантах. О них - в следующем разделе.

Оператор Do …. Loop

Попытаемся составить с использованием 0 варианта оператора Do программу решения задачи о печати чисел 3 5 7 9 из предыдущего параграфа. Для того, чтобы точно определить работу этого варианта оператора Do, приведем ее параллельно с измененным 1 вариантом программы решения этой задачи из того же параграфа. Объяснением любого оператора в правом столбце является оператор, стоящий в той же строчке в левом столбце.

ИЗМЕНЕННЫЙ 1 ВАРИАНТ	0 ВАРИАНТ ОПЕРАТОРА Do
Private Sub Command1_Click()	Private Sub Command0_Click()
Debug.Print "Начало счета";	Debug.Print "Начало счета";
f = 3	f = 3
m:	**Do**

Debug.Print f;	Debug.Print f;
f = f + 2	f = f + 2
GoTo m	**Loop**
End Sub	End Sub

Do можно перевести, как "Делай", а понимать следует просто как метку.

Loop можно перевести, как "Петля" или "Возврат назад", а понимать следует так: "Возвращайся к метке Do ".

Порядок работы обеих программ совершенно одинаков, так что можно считать слово Do заменой метки m, а слово Loop считать заменой оператора GoTo m. Обе программы бесконечно печатают 3 5 7 9 11 ….. Прерывается цикл только с клавиатуры. Толку в 0 варианте оператора Do мы видим мало (пока).

Синтаксис оператора **Do …. Loop:**

Do
операторы
операторы
…….
Loop

Строки операторов между Do и Loop называются **телом цикла**.

Оператор Do …. Loop While

Добавьте в ваш проект еще 4 кнопки и выполните в пошаговом режиме программы с вариантами оператора Do 1 - 4, которые я привел ниже. Вы увидите, что все 4 варианта делают одно и то же и они очень похожи. Вопрос о том, зачем нужно целых 4 похожих варианта, рассмотрим чуть позже. Уверяю, они все нужны.

Составим с использованием 1 варианта оператора Do программу решения задачи о печати чисел 3 5 7 9 из предыдущего параграфа. Для того, чтобы точно определить работу этого варианта оператора Do, приведем ее параллельно с 1 вариантом программы решения этой задачи из того же параграфа. Объяснением любого оператора в правом столбце является оператор, стоящий в той же строчке в левом столбце.

1 ВАРИАНТ	1 ВАРИАНТ ОПЕРАТОРА Do
Private Sub Command1_Click()	Private Sub Command5_Click()
Debug.Print "Начало счета";	Debug.Print "Начало счета";
f = 3	f = 3
m:	**Do**
Debug.Print f;	Debug.Print f;
f = f + 2	f = f + 2
If f <= 9 **Then GoTo m**	**Loop While** f <= 9
Debug.Print "Конец счета"	Debug.Print "Конец счета"
End Sub	End Sub

While переводится "Пока".

Значит, Loop While f <= 9 понимать следует так: "Возвращайся к метке Do, пока f <= 9".

Порядок работы обеих программ совершенно одинаков, так что можно считать слово Do заменой метки m, а конструкцию Loop While f <= 9 считать заменой оператора If f <= 9 Then GoTo m.

Синтаксис оператора **Do …. Loop While:**

Do
операторы
операторы
…….
Loop While *условие продолжения работы цикла*

Оператор Do …. Loop Until

2 ВАРИАНТ	2 ВАРИАНТ ОПЕРАТОРА Do
Private Sub Command2_Click()	Private Sub Command6_Click()
Debug.Print "Начало счета";	Debug.Print "Начало счета";
f = 3	f = 3
m1:	**Do**
Debug.Print f;	Debug.Print f;
f = f + 2	f = f + 2

If f > 9 **Then GoTo m2 Else GoTo m1**	**Loop Until** f > 9
m2: Debug.Print "Конец счета"	Debug.Print "Конец счета"
End Sub	End Sub

Until переводится "До тех пор, пока".
Значит, Loop Until f > 9 понимать следует так: "Возвращайся к метке Do до тех пор, пока не выполнится условие f > 9".

Синтаксис оператора **Do …. Loop Until:**

Do
операторы
операторы
…….
Loop Until *условие завершения работы цикла*

Оператор Do While …. Loop

3 ВАРИАНТ	3 ВАРИАНТ ОПЕРАТОРА Do
Private Sub Command3_Click()	Private Sub Command7_Click()
Debug.Print "Начало счета";	Debug.Print "Начало счета";
f = 3	f = 3
m1: **If** f <= 9 **Then GoTo m3 Else GoTo m2**	**Do While** f <= 9
m3: Debug.Print f;	Debug.Print f;
f = f + 2	f = f + 2
GoTo m1	**Loop**
m2: Debug.Print "Конец счета"	Debug.Print "Конец счета"
End Sub	End Sub

Do While f <= 9 понимать следует так: "Пока f <= 9 выполняй нижестоящие операторы вплоть до Loop".

Синтаксис оператора **Do While …. Loop:**

Do While *условие продолжения работы цикла*
операторы
операторы
…….
Loop

Оператор Do Until …. Loop

4 ВАРИАНТ	4 ВАРИАНТ ОПЕРАТОРА Do
Private Sub Command4_Click()	Private Sub Command8_Click()
Debug.Print "Начало счета";	Debug.Print "Начало счета";
f = 3	f = 3
m1: **If** f > 9 **Then GoTo m2 Else GoTo m3**	**Do Until** f > 9
m3: Debug.Print f;	Debug.Print f;
f = f + 2	f = f + 2
GoTo m1	**Loop**
m2: Debug.Print "Конец счета"	Debug.Print "Конец счета"
End Sub	End Sub

Do Until f > 9 понимать следует так: "Выполняй нижестоящие операторы вплоть до Loop, до тех пор, пока не выполнится условие f > 9".

Синтаксис оператора **Do Until …. Loop:**

Do Until *условие завершения работы цикла*
операторы
операторы
…….
Loop

Типичная ошибка начинающих – небрежное обращение со знаками сравнения. Многие не видят большой разницы в том, как записать – While f<=9 или While f<9 , а затем, «недополучив» результат, удивляются, почему. И здесь лучшим средством для понимания является отладочный режим. Попробуйте ошибочный вариант программы с While f<9 выполнить в пошаговом режиме.

Задача: Компьютер предлагает человеку ввести слово, после чего распечатывает это слово, снабдив его восклицательным знаком. Затем снова предлагает ввести слово и так до тех пор, пока человек не введет слово "Хватит". Распечатав его с восклицательным знаком, компьютер отвечает "Хватит так хватит" и заканчивает работу.

Придумаем строковую переменную, в которую человек будет с клавиатуры вводить слово. Назовем ее Slovo. Выберем подходящий вариант оператора Do, это будет 2-й, и пишем программу:

```
Dim Slovo As String
Private Sub Command1_Click()
    Do
        Slovo = InputBox("Введите слово")
        Debug.Print Slovo; "!"
    Loop Until Slovo = "Хватит"
    Debug.Print "Хватит так хватит"
End Sub
```

Задание 42: Усложним эту задачу. Пусть компьютер перед распечаткой каждого слова ставит его порядковый номер.

Задание 43-44: Выполнить с применением оператора Do задания 38 и 41 из предыдущего раздела.

Задание 45: Если камень бросить горизонтально со 100-метровой башни со скоростью v=20м/с, то его расстояние от башни по горизонтали s будет выражаться формулой s=vt, где t – время полета камня в секундах. Высота над землей h будет выражаться формулой $h=100 - 9.81t^2/2$. Вычислять и печатать t, s и h для t = 0, 0.2, 0.4, 0.6 и так далее до тех пор, пока камень не упадет на землю.

Разница между вариантами операторов Do

Разницы две:

- *Между While и Until.* Здесь соображения удобства. Что вам удобнее, указывать компьютеру, когда цикл нужно продолжать (f <= 9) или когда его нужно заканчивать (f > 9)?
- *В том, куда поставить условие - после Do или после Loop.* В первом случае можно придумать такое условие, когда тело цикла не выполнится ни разу. Например,

```
f = 3
Do Until f > 0
    Debug.Print f;
    f = f + 2
Loop
```

Во втором случае, каково бы ни было условие, тело цикла хотя бы раз, да выполнится.

Часто эти отличия для начинающих малосущественны, поэтому пока выбирайте оператор по вкусу.

Оператор Exit Do

Оператор Exit Do нужен для того, чтобы выходить из цикла не в начале тела цикла, не в конце, а в середине. Добавим его в тело цикла одного из вариантов программы:

```
Private Sub Command2_Click()
    Debug.Print "Начало счета";
    f = 3
    Do
      Debug.Print f;
      Exit Do
      f = f + 2
    Loop While f <= 9
    Debug.Print "Конец счета"
End Sub
```

Вот результат работы этой программы:

```
Начало счета  3  Конец счета
```

Толк от Exit Do будет тогда, когда его поместят внутрь оператора ветвления:

```
Private Sub Command1_Click()
    Debug.Print "Начало счета";
    f = 3
    Do
      Debug.Print f;
      If f >= 9 Then Exit Do
      f = f + 2
```

```
    Loop
    Debug.Print "Конец счета"
End Sub
```

Вот результат работы этой программы:

```
Начало счета  3  5  7  9  Конец счета
```

Устаревший оператор цикла

Имеется оператор цикла **While ..... Wend**, доставшийся от старых версий Бэйсика. На тот случай, если вам придется запускать чьи-то старые программы, вот вам его синтаксис:

```
While условие продолжения работы цикла
    операторы
    операторы
    ……————
Wend
```

А смысл - тот же, что и у Do While .... Loop

6.4. Оператор цикла For

Я говорил в 6.3, что имеются две разновидности оператора For. Здесь я расмотрю только одну. Вторая будет разобрана в 14.3.

Мы знаем, что выполняя программу печати чисел 3 5 7 9, оператор Do выполнил цикл 4 раза. Обычно, когда мы пишем операторы Do, нам совсем не интересно знать, сколько раз они выполнят цикл. Тем не менее, существует много задач, для решения которых цикл нужно выполнить именно определенное количество раз. В этом случае удобно использовать оператор цикла For.

Задача: 200 раз напечатать слово ФУТБОЛ.

Попробуем сначала решить задачу при помощи оператора GoTo. Начнем с такого фрагмента:

```
metka:  Print "ФУТБОЛ"
        GoTo metka
```

Но здесь цикл будет повторяться бесконечно, а нам нужно только 200 раз. Мы уже видели, что для выхода из цикла оператор GoTo нужно включить в состав оператора If. Кроме этого нужна переменная, меняющая свое значение от одного выполнения цикла к следующему. Придумаем этой величине какое-нибудь имя, скажем i. Проще всего задачу решает такая процедура:

```
Private Sub Command1_Click()
        i = 1
metka:  Debug.Print "ФУТБОЛ"
        i = i + 1                       'увеличение i на 1
        If i <= 200 Then GoTo metka
End Sub
```

Здесь i вначале равно 1, но к каждому следующему выполнению цикла оно увеличивается на 1. В первый раз выполняя оператор If, компьютер проверяет условие 2<=200 и найдя его истинным, выполняет оператор GoTo metka. Во второй раз проверяется условие 3<=200 и т.д. В 199-й раз компьютер проверяет условие 200<=200 и найдя его истинным, выполняет оператор GoTo metka. В 200-й раз компьютер проверяет условие 201<=200 и найдя его ложным, выходит из цикла.

В нашем фрагменте "полезную" работу выполняет только одна строка из четырех - Print "ФУТБОЛ" . Остальные три строки заняты тем, что обеспечивают выполнение "полезной" строки ровно 200 раз. Нам пришлось организовать специальную переменную i, значение которой в каждый момент выполнения программы говорит о том, в какой раз выполняется цикл. Переменная с таким свойством называется **счетчиком циклов**.

А теперь перепишем программу так, чтобы логика ее выполнения более менее соответствовала логике выполнения программы с оператором For, которую я привожу параллельно и поясняю немедленно.

Private Sub Command1_Click()	Private Sub Command2_Click()
i = 1	**For** i = 1 **To** 200
m1: Debug.Print "ФУТБОЛ"	Debug.Print "ФУТБОЛ"
i = i + 1: If i <= 200 Then GoTo m1	**Next** i 'увеличение i на 1 и возврат
End Sub	End Sub

Прогоните правый вариант программы в пошаговом режиме, заменив для удобства число 200 числом 5 и следя за значением переменной i.

Слово **For** переводится "для". Слово **To** переводится "до". Конструкция **For i=1 To 200** понимается так: *Для i, изменяющегося от 1 до 200, выполняй операторы, стоящие ниже вплоть до слова* **Next**. Слово Next говорит о том, что надо увеличивать i на 1 и возвращаться к началу цикла. При первом выполнении цикла i будет равно 1, при втором - 2, и так далее. При последнем - 200. Переменная i называется **переменной цикла**,

В данном конкретном случае сами по себе значения i не важны, тот же результат мы бы получили и с оператором

For i = 501 To 700

Задача: Распечатать пары чисел - 101 1000 102 990 103 980 104 970 ……. 109 920 110 910
Здесь значения переменной цикла важны. Вот программа:

```
Private Sub Command1_Click()
    For a = 101 To 110
        Debug.Print a;
        Debug.Print 1000 - 10 * (a - 101)
    Next a
End Sub
```

До сих пор переменная цикла менялась с шагом 1. Шаг можно задавать любой и тогда оператор For очень удобно использовать вместо Do - программы получаются короче. Напишем, например, программу для вычисления таблиц Брадиса (задача 38 - напомню, что нужно печатать квадраты чисел от 0 до 1 с шагом 0.001):

```
Private Sub Command1_Click()
    For a = 0 To 1.00001 Step 0.001
        Debug.Print Format(a, "0.000"), Format(a * a, "0.000000")
    Next a
End Sub
```

Здесь **Step 0.001** означает "Шаг 0.001". Для более аккуратной печати я использовал форматирование. Почему вместо 1 я написал 1.00001, объяснено в ответе к задаче 38.

Если бы мы хотели посчитать таблицы Брадиса в обратном порядке - от 1 до 0, то оператор цикла нужно было бы записать так:

For a = 1 **To** 0 **Step -0.001**

Синтаксис оператора For:

For *переменная* **=** *выражение1* **To** *выражение2* **[Step** *выражение3* **]**
операторы
операторы
……..
Next [*переменная* **]**

Переменная здесь должна иметь числовой тип. После Next вы можете ее не писать, но в программах, где много For и Next, я рекомендую это делать для легкости чтения программы, чтобы вам удобней было разобраться, для какого For этот Next.
Пример записи: For j=a+b To 2*s Step k*10
Если шаг не указан, он считается равным 1, то есть переменная на каждом выполнении цикла увеличивается на 1. Если же мы хотим уменьшать ее на 1, нам придется явно указать Step -1.

Работа оператора For при положительном (или нулевом) шаге:

Прежде всего вычисляется выражение1, и переменной цикла (пусть это будет i) присваивается его значение. Затем вычисляется выражение2 и сравнивается с i. Если i > выражения2, то оператор For завершает свою работу, так ничего и не сделав. В противном случае выполняются операторы, стоящие между строками For и Next. После их выполнения значение i увеличивается на значение выражения3 (или при его отсутствии на 1) и снова сравнивается с выражением2. Если i > выражения2, то оператор For завершает свою работу, иначе снова выполняются операторы, снова i увеличивается и т.д.

Работа оператора For при отрицательном шаге:

Прежде всего вычисляется выражение1, и переменной цикла (пусть это будет i) присваивается его значение. Затем вычисляется выражение2 и сравнивается с i. Если i < выражения2, то оператор For завершает свою работу, так ничего и не сделав. В противном случае выполняются операторы, стоящие между строками For и Next. После их выполнения значение i уменьшается на значение модуля выражения3 и снова сравнивается с выражением2. Если i < выражения2, то оператор For завершает свою работу, иначе снова выполняются операторы, снова i уменьшается и т.д.

Оператор Exit For

Оператор Exit For - еще один способ выхода из цикла For. Применяется он совершенно аналогично оператору Exit Do, описанному в 6.3, поэтому отсылаю вас туда.

Задание 46: Напечатать с помощью оператора For:

Прямой счет: -5 -4 -3 -2 -1 0 1 2 3 4 5 Обратный счет: 5 4 3 2 1 0 -1 -2 -3 -4 -5 Конец счета

6.5. Оглядимся вокруг

Итак, мы изучили три способа заставить компьютер многократно выполнять часть кода внутри процедуры:

- Операторы Do …. Loop
- Операторы For
- Устаревший оператор Goto

Существуют способы заставить многократно выполняться процедуру целиком, среди них:

- Использование таймера (см. 11.2)
- Рекурсия (см. 17.5)

Глава 7. Отладка программы

Мы уже добрались до программ, требующих для своей отладки более совершенных средств, чем простой пошаговый режим. Поговорим о них.

7.1. Типы ошибок. Сообщения об ошибках.

Сначала перечтите, пожалуйста, материал о сообщениях об ошибках в 1.3.

Ошибки в программах делятся на те, которые Visual Basic замечает в процессе ввода программы, на те, что замечает в процессе выполнения программы, и на те, что не замечает и в принципе заметить не может.

К первым относятся все синтаксические погрешности. Например, если вы вместо строки Do Until a>0 введете Do Antil a>0 и попытаетесь перейти на следующую строку, Visual Basic выдаст сообщение об ошибке

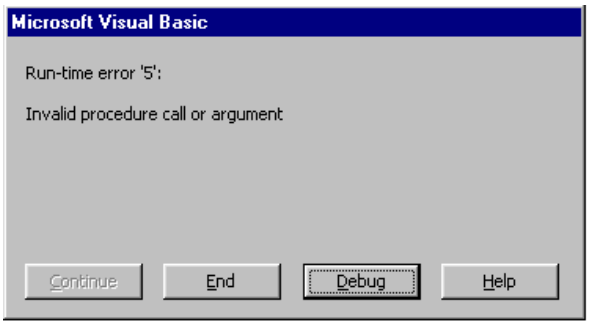
Expected While or Until or end of statement

что означает

Ждал While или Until или конца строки

Visual Basic недостаточно умен, чтобы точно описать вам вашу ошибку, но он хотя бы говорит вам, что ему от вас нужно.

На стадии выполнения Visual Basic замечает такие ошибки, как Sqr(-25), то есть квадратный корень из -25. На нее он среагирует таким сообщением:



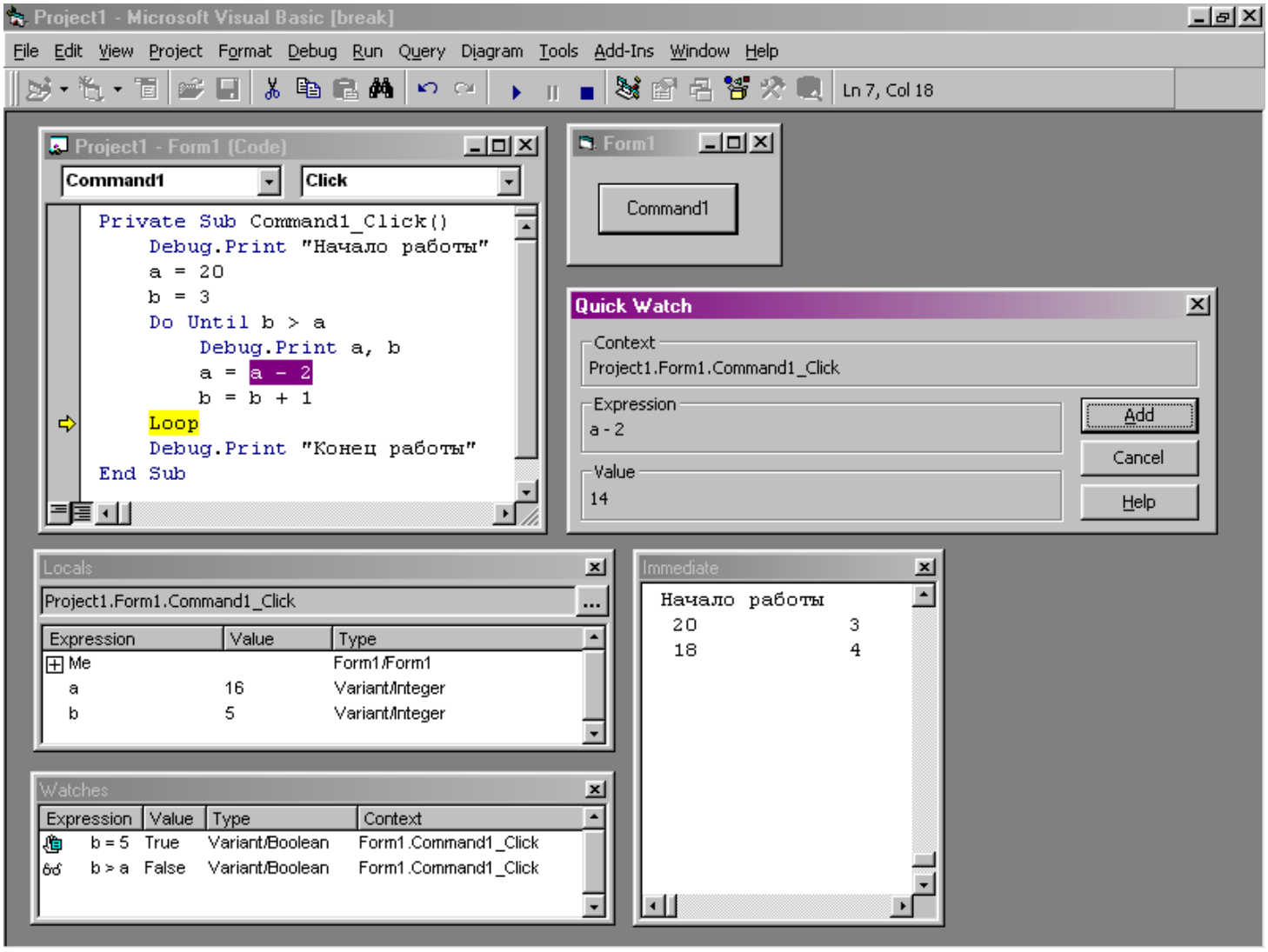
что означает Неправильный вызов процедуры или неправильный параметр процедуры. В нашем случае это, конечно, недопустимое значение параметра (-25). Чтобы узнать, на какое место программы Visual Basic грешит, нажмите кнопку Debug. Visual Basic переходит в режим прерывания и подсвечивает ошибочную по его мнению строку.

К ошибкам, которые Visual Basic не обнаруживает, относятся смысловые ошибки. Так, если вы, желая увеличить число a на 1, вместо a=a+1 пишете a=a+2, то этого не заметит ни один язык в мире.

7.2. Отладка программы. Окна отладки. Режимы отладки.

В 4.2 я изложил пошаговый способ выполнения программы. Его вполне достаточно для отладки простейших программ. В более сложных случаях можно применять другие способы, с которыми мы сейчас познакомимся. Кстати, зачастую они и более удобны. Итак рассмотрим окна Locals, Watches, Quick Watch, Call Stack и новые способы использования окна Immediate.

На рисунке в окне кода вы видите некую процедуру Command1_Click в процессе отладки:



Выполнившись с начала до конца, эта процедура печатает в окне Immediate такую информацию:

```
Начало работы
 20        3
 18        4
 16        5
 14        6
 12        7
 10        8
Конец работы
```

Вообразим, что нас такая печать привела в недоумение, мы по своей бестолковости ожидали, что будет напечатано

```
Начало работы
 20        3
 18        4
Конец работы
```

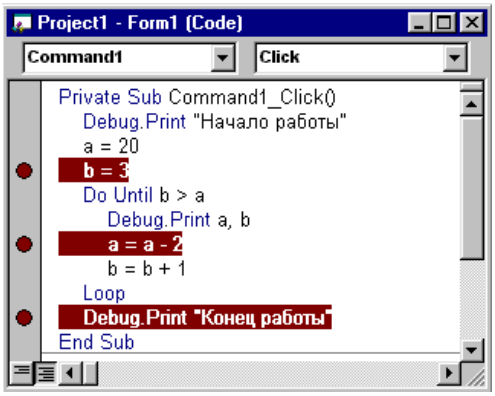
Чтобы разобраться в случившемся, мы и затеяли отладку.

Окно Locals. Прежде всего при помощи **View→Locals Window** и **View→Immediate Window** мы вызвали на экран окна Locals и Immediate. Назначение окна **Locals** – показывать в режиме прерывания значения локальных переменных выполняемой процедуры. После этого клавишей F8 запустим пошаговый режим выполнения проекта, как это объяснялось в 4.2. После нескольких нажатий на F8 окна Locals и Immediate примут тот вид, что вы видите на рисунке. Value означает "значение". Правда, мы и без окна Locals в любой момент можем узнать значения a и b, просто поместив на них мышиный курсор. Однако применение окна Locals избавляет нас и от этих мизерных трудов. К тому же вы в окне можете видеть информацию о типе переменных. Щелкните по плюсику слева от Me. Форма расскажет вам о текущих значениях своих свойств.

Если вы, выполняя таким образом процедуру, перескочите на другую процедуру, то теперь в окне Locals будут видны именно ее локальные переменные. Если вам другие процедуры не интересны, то работайте в пошаговом режиме не клавишей F8, а комбинацией клавиш Shift-F8. При этом все другие процедуры будут проскакиваться мгновенно.

Точки прерывания (Breakpoints). Когда программа большая или циклов много, жать на F8 приходится слишком часто. Но ведь и нужды-то в этом нет. Обычно вам интересно останавливаться только на некоторых строках программы, мимо осталь-

ных можно и пролететь. Visual Basic позволяет вам задать такие строки - **точки прерывания** (Breakpoints). Щелкните по вертикальной серой полосе в левой части окна кода против строки, на которой хотите прерывать выполнение программы. На полосе появится черная точка и вся строка будет выделена черным (смотри рисунок).



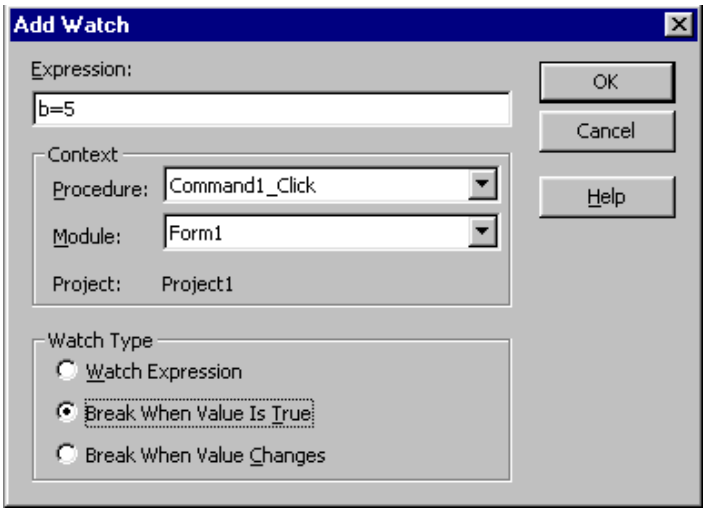
Щелкните так же против всех нужных вам строк. Запускайте проект обычным образом (кнопка Start или клавиша F5). Дойдя до первой же точки прерывания, проект перейдет в режим прерывания. Продолжайте его работу клавишей F5. Проект будет останавливаться только на точках прерывания. Убираются точки прерывания так же, как и ставятся - щелчком мыши.

"Беги до курсора" - Run To Cursor. Это еще один способ остановки в нужной вам строке. Поставьте текстовый курсор в нужную строку и нажмите Ctrl-F8. Программа будет выполняться до тех пор, пока не наткнется на строку с курсором. А теперь поставьте курсор в другую строку и снова нажмите Ctrl-F8. Программа продолжит работу с того места, где остановилась, и будет выполняться до тех пор, пока не наткнется на строку с курсором. И так далее.

В процессе выполнения программы в режиме прерывания вы можете достаточно свободно переключаться между разными способами прерывания. Ставьте и убирайте точки прерывания, переставляйте курсор, нажимайте то F8, то Shift-F8, то Ctrl-F8, то F5. Visual Basic будет вас слушаться.

Описанные в этом разделе возможности отладки и некоторые другие приведены в меню **Debug** (отладка).

Окно Watches. Вообразим, что окно Locals не заставило нас поумнеть. Нам все кажется, что цикл должен был прерваться уже при значении b=5. К тому же, мы устали жать на F8. Хорошо бы программа исполнялась в своем обычном сверхбыстром режиме [run], но в тот момент, когда b станет равным 5, перешла в режим прерывания и остановилась, чтобы мы успели посмотреть, что там к чему в этот момент. Итак, нам нужен совсем другой способ остановки - не на заданном операторе, а при выполнении заданного условия. Для этого мы используем окно **Watches**. Вызовем его так: **Debug→Add Watch**. Возникнет диалоговое окно Add Watch, в которое вы вводите логическое выражение b=5, которое по нашим расчетам поначалу равно False, но затем в какой-то момент выполнения вроде бы станет равным True. Вот этот самый момент мы и хотим, чтобы Visual Basic поймал и остановил программу. Для этого из трех возможных положений переключателя Watch Type выбираем среднее. Если бы мы хотели просто присматривать за выражением, то выбрали бы верхнее. Нижнее положение приказывает прерваться при любом *изменении* значения выражения.



Нажмите OK. А теперь давайте еще присмотрим за выражением b > a. Добавим его в окно Watches аналогично.

Запустите проект на выполнение в обычном режиме. После нажатия на кнопку Command1 процедура будет выполняться до тех пор, пока b не станет равным 5, после чего Visual Basic перейдет в режим прерывания и остановится. Этот момент вы и видите на первом из трех рисунков этого раздела. Все описанные мной окна содержат информацию именно на этот момент.

Окно Quick Watch. Кроме этих окон вы видите на рисунке еще одно окно - Quick Watch. Оно понадобилось вот для чего. Иногда в процессе работы в режиме прерывания вам может захотеться узнать, чему равно в настоящий момент значение того или иного выражения из процедуры, например a - 2. Вы можете включить это выражение в окно Watches, но быстрее поступить так: Выделите нужное выражение в окне кода, как это сделано на рисунке, затем **Debug→Quick Watch**. Окно появляется.

Окно Call Stack. Когда в программе содержится много взаимодействующих процедур, вам при отладке может быть совершенно необходимо разобраться в последовательности их выполнения. В этом случае вам поможет окно Call Stack, которое и покажет вам эту последовательность. Вызывается оно так: **View→Call Stack**.

Окно Immediate может применяться для следующих полезных вещей, о которых я раньше не говорил. Вы можете в режиме прерывания ввести в это окно текст и наблюдать следующие результаты:

Текст	Смысл
?a	В окне печатается значение переменной a в данный момент
?backcolor	В окне печатается численное значение цвета фона
?text1.Text	В окне печатается содержимое текстового окна
a=44	Приказ изменить в данный момент по нашему желанию значение переменной a. Таким образом мы можем вмешиваться в ход выполнения программы, чего раньше не делали. Более того, в режиме прерывания вы можете в окне кода на ходу менять операторы программы, так что в начале своего выполнения программа может иметь один облик, а в конце - совсем другой.
form1.BackColor =255	Приказ изменить цвета фона. Результат вы сразу же обнаружите, посмотрев на форму.
Command1_Click	Приказ выполнить процедуру
for i=1 to 10 **:** ?i **:** next	Приказ выполнить оператор. В окне будут напечатаны числа от 1 до 10.

Глава 8. Типичные маленькие программы

Каждому программисту известны такие слова, как счетчик, сумматор, вложенные циклы и другие подобные понятия, составляющие элементарную технику программирования. Без них не обходится ни одна реальная программа, и если мы хотим идти дальше, то нам без них не обойтись. В этой главе я не буду вводить новых операторов, а покажу, как программировать типичные задачи, в том числе и те, что используют упомянутые понятия.

8.1. Вычислительная циклическая программа

Задача: Во дворце 40 залов. Известны длина, ширина и высота каждого зала. Вычислить площадь пола и объем каждого зала.

Сначала напишем фрагмент для одного зала:

```
Dlina = InputBox ("Введите длину")                    'Начало фрагмента
Shirina = InputBox ("Введите ширину")
Visota = InputBox ("Введите высоту")
S = Dlina * Shirina              'Площадь пола
V = S * Visota                   'Объем
Debug.Print "Площадь пола="; S, "Объем зала="; V     'Конец фрагмента
```

Для решения задачи этот фрагмент нужно выполнить 40 раз, для чего вполне естественно вложить его в оператор For:

```
Private Sub Command1_Click()
    For i = 1 To 40
        Dlina = InputBox("Введите длину")                    'Начало фрагмента
        Shirina = InputBox("Введите ширину")
        Visota = InputBox("Введите высоту")
        S = Dlina * Shirina              'Площадь пола
        V = S * Visota                   'Объем
        Debug.Print "Площадь пола="; S, "Объем зала="; V     'Конец фрагмента
    Next i
End Sub
```

Полужирным шрифтом я выделил новые по сравнению с предыдущим фрагментом строки.

Чтобы программа подходила для любого числа залов, нужно вместо

```
For i = 1 To 40
```

написать

```
N = InputBox("Сколько залов во дворце?")
For i = 1 To N
```

Задание 47: Даны стороны N кубиков. Вычислить объем каждого.

8.2. Роль ошибок

Пусть во дворце три зала размерами 20*15*4, 30*20*5 и 10*5*3. В этом случае, выполняя программу предыдущего параграфа, мы вводим N=3 и оператор For выполняет цикл три раза.

Мы знаем, что по ошибочной программе компьютер выдает ошибочные результаты. Например, если в нашей программе мы вместо V=S*visota напишем V=S+visota, то результаты будут такими:

```
Площадь пола=300 Объем зала=304
Площадь пола=600 Объем зала=605
Площадь пола=50 Объем зала=53
```

Если же случайно вместо For i=1 To N написать For i=2 To N то результаты будут такими:

```
Площадь пола=300 Объем зала=1200
Площадь пола=600 Объем зала=3000
```

На этом программа закончит работу и не спросит размеров третьего зала. Вам не кажется странным, что она посчитала 1 и 2 залы, а не 2 и 3? Если кажется, то учтите, что пользователь ничего не знает об ошибке в программе, а компьютер не говорит ему, размеры какого по счету зала ему нужно вводить.

Задания 48-49:

Определите без компьютера, что напечатает компьютер, если

48) строку For i=1 To N поместить на три строки ниже, а именно - перед строкой $S = Dlina * Shirina$

49) поменять местами строки Debug.Print и Next

Если задания не получаются, введите программы в компьютер и используйте пошаговый режим.

8.3. Счетчики

Счетчик - это переменная величина, в которой вы что-нибудь подсчитываете. Для чего нужны счетчики? Ну хотя бы для того, чтобы подсчитать количество жизней главного персонажа в компьютерной игре.

Задача 1**:** В компьютер с клавиатуры вводятся числа. Компьютер после ввода каждого числа должен печатать, сколько среди них уже введено положительных.

Фрагмент, решающий задачу:

```
    c=0                 'Обнуляем счетчик
m:  a = InputBox("Введите очередное число")
    If  a>0  Then  c=c+1
    Debug.Print  "Из них положительных -" ,  c
    GoTo m
```

Пояснения**:** В 6.4 мы придумали переменную i, которую назвали счетчиком циклов. Здесь мы тоже придумали переменную c. Она у нас выполняет роль счетчика положительных чисел. Сердце счетчика - оператор c=c+1. Именно он в нужный момент увеличивает счетчик на 1. Но и без If a>0 Then тоже никак нельзя. Если бы его не было, то c подсчитывал бы все числа без разбору, то есть был бы обыкновенным счетчиком циклов. В нашем же фрагменте увеличение c на 1 выполняется не всегда, а лишь при положительном a.

Обязательно прокрутите программу в пошаговом режиме.

В сложных программах не забывайте обнулять счетчик перед входом в цикл, а не то он начнет считать вам не с нуля, а бог знает с чего. Как бы вам понравилось, если бы таксист в начале поездки не обнулил счетчик?

В нашем фрагменте значения счетчика печатаются при каждом выполнении цикла. Изменим задачу.

Задача 2**:** В компьютер вводится ровно 200 чисел. Компьютер должен подсчитать и один раз напечатать, сколько среди них положительных.

Программа**:**

```
Private Sub Command2_Click()
    c = 0                 'Обнуляем счетчик
    For i = 1 To 200
        a = InputBox("Введите очередное число")
        If a > 0 Then c = c + 1
    Next i
    Debug.Print "Из них положительных -"; c
End Sub
```

Пояснения**:** Путь рассуждений здесь тот же, что и в первой задаче. В результате применения оператора For тело цикла выполняется ровно 200 раз, благодаря чему счетчик c накапливает нужное значение. Оператор Debug.Print выполняется только один раз и печатает последнее накопленное значение, потому что в ячейке c будет находиться именно оно..

Задание 50: Что будет, если

1) Вместо *c=0* написать *c=10.*
2) Вместо *c=c+1* написать *c=c+2*.
3) Строки *Next* и *Debug.Print* поменять местами.
4) Строки *c=0* и *For* поменять местами.
5) Строки *For* и *InputBox* поменять местами.

А в следующей программе мы используем уже два счетчика. Изменим задачу.

Задача 3**:** В компьютер вводится ровно 200 чисел. Компьютер должен подсчитать и один раз напечатать, сколько среди них положительных чисел и сколько нулей.

Программа**:**

```
Private Sub Command3_Click()
    c_полож = 0              'Обнуляем счетчик положительных чисел
    c_нулей = 0              'Обнуляем счетчик нулей
    For i = 1 To 200
        a = InputBox("Введите очередное число")
        If a > 0 Then c_полож = c_полож + 1
        If a = 0 Then c_нулей = c_нулей + 1
    Next i
    Debug.Print "Из них положительных -"; c_полож, "Нулей -"; c_нулей
End Sub
```

Как узнать, насколько Лев Толстой любил слово "добро"? Для этого достаточно, используя с минимальными изменениями нижеприведенную программу, ввести в компьютер слово за словом его произведения.

Задача 4**:** В компьютер один за другим вводятся произвольные символы. Ввод заканчивается символом "/". Подсчитать, ка-

кой процент от общего числа введенных символов составляют символ "**W**" и символ "**:**" по отдельности.

Здесь мы организуем три счетчика одновременно: cW - для подсчета букв W, cDv - для подсчета двоеточий, а также i - счетчик общего числа введенных символов, кроме "/".

Программа**:**

```
Private Sub Command4_Click()
    Dim i As Integer, cW As Integer, cDv As Integer
    Dim procent_W As Integer, procent_Dv As Integer
    Dim simvol As String

    i = 0: cW = 0: cDv = 0                  'Обнуляем все три счетчика
    Do
        simvol = InputBox("Введи символ")
        If simvol <> "/" Then i = i + 1     'Если это не /, то "посчитай" его
        Select Case simvol
            Case "W"                        'Если это W, то
                cW = cW + 1                     'увеличь счетчик символов W
            Case ":"                        'Если это  : , то
                cDv = cDv + 1                   'увеличь счетчик символов  :
            Case "/"                        'Если это /, то
                Exit Do                         'завершай работу цикла
        End Select
    Loop
    procent_W = Round(100 * cW / i)         'Вычисляй процент символов W
    procent_Dv = Round(100 * cDv / i)       'Вычисляй процент символов :
    Debug.Print   procent_W,  procent_Dv
End Sub
```

Задание 51: В компьютер вводится N чисел. Подсчитать из них количество положительных, отрицательных и тех, что превышают число 10.

Задание 52: В компьютер вводятся пары целых чисел. Подсчитать, сколько среди них пар, дающих в сумме число 13. Подсчет закончить после ввода пары нулей.

8.4. Сумматоры

Сумматор - это переменная величина, в которой вы подсчитываете сумму чего-либо. Для чего нужны сумматоры? Ну хотя бы для того, чтобы подсчитать общее количество золота, которое вы нашли в нескольких кладах в компьютерной игре.

Если вы поняли идею счетчика, то понять идею сумматора вам будет нетрудно. Посмотрим, как будет работать следующий фрагмент:

```
      s=0                          'Обнуляем сумматор. Это не менее важно, чем обнулить счетчик
m:    a = InputBox("Введите очередное число")
      s=s+a                 'Увеличиваем сумматор
      Debug.Print "Сумма="; s
      GoTo m
```

В ячейке s накапливается сумма вводимых чисел a, поэтому назовем эту ячейку **сумматором**. Отличие сумматора от счетчика в том, что счетчик увеличивается на **1** оператором c=c+1, а сумматор - на суммируемое число оператором s=s+a.

Задача: В компьютер вводится N чисел. Вычислить и один раз напечатать их сумму.

Программа:

```
Private Sub Command2_Click()
    N = InputBox("Сколько чисел будем складывать?")
    s = 0
    For i = 1 To N
        a = InputBox("Введите очередное число")
        s = s + a
    Next i
    Debug.Print "Сумма равна"; s
End Sub
```

Задание 53: Пусть N=2, a=5 и 3. Тогда по этой программе Visual Basic напечатает 8. Что он напечатает, если:

1) Вместо *s=0* написать *s=10.*

2) Вместо *s=s+a* написать *s=s+a+1*.

3) Строки *Next* и *Debug.Print* поменять местами.

4) Строки *s=0* и *For* поменять местами.

5) Строки *For* и *InputBox* поменять местами.

6) Строки *s=s+a* и *Next* поменять местами.
7) Вместо *For i=1 To N* написать *For i=2 To N.*

Задания 54-56: Написать программы для следующих задач:

54) Во дворце 40 залов. Известны длина и ширина каждого зала. Вычислить площадь пола всего дворца.
55) Вычислить средний балл учеников вашего класса по физике.
56) Вычислить произведение N произвольных чисел. *Подсказка*: Несмотря на то, что произведение - не сумма, эта программа будет отличаться от программы суммирования всего двумя существенными символами (какими?), а структура обеих программ совершенно одинакова.

8.5. Вложение циклов в разветвления и наоборот

Реальная процедура на Visual Basic может представлять собой сложную мозаику из циклических и разветвляющихся частей, вложенных друг в друга. Мы уже видели в 6.3, как в оператор цикла были вложены операторы ветвления. В свою очередь в них могут быть вложены операторы цикла, и так до бесконечности.

Для тренировки определите, что напечатает следующий фрагмент:

```
Private Sub Command1_Click()
    For i = 1 To 5
        a = 9
        If i * i = a Then
            For k = 5 To 8
                Debug.Print k;
            Next k
        Else
            Debug.Print 1997
        End If
    Next i
End Sub
```

Здесь внутрь *For i = 1 To 5* вложен *If i * i = a* , а внутрь него вложен *For k = 5 To 8.*

Ответ:

```
1997
1997
5  6  7  8  1997
1997
```

8.6. Вложенные циклы

Вложенные циклы или цикл внутри цикла - весьма распространенная конструкция при программировании.

Поставим себе задачу - напечатать таблицу умножения. В следующем виде:

1*1=	1	1*2=	2	1*3=	3	1*4=	4	1*5=	5	1*6=	6	1*7=	7	1*8=	8	1*9=	9	1*10=	10
2*1=	2	2*2=	4	2*3=	6	2*4=	8	2*5=	10	2*6=	12	2*7=	14	2*8=	16	2*9=	18	2*10=	20
3*1=	3	3*2=	6	3*3=	9	3*4=	12	3*5=	15	3*6=	18	3*7=	21	3*8=	24	3*9=	27	3*10=	30
4*1=	4	4*2=	8	4*3=	12	4*4=	16	4*5=	20	4*6=	24	4*7=	28	4*8=	32	4*9=	36	4*10=	40
5*1=	5	5*2=	10	5*3=	15	5*4=	20	5*5=	25	5*6=	30	5*7=	35	5*8=	40	5*9=	45	5*10=	50
6*1=	6	6*2=	12	6*3=	18	6*4=	24	6*5=	30	6*6=	36	6*7=	42	6*8=	48	6*9=	54	6*10=	60
7*1=	7	7*2=	14	7*3=	21	7*4=	28	7*5=	35	7*6=	42	7*7=	49	7*8=	56	7*9=	63	7*10=	70
8*1=	8	8*2=	16	8*3=	24	8*4=	32	8*5=	40	8*6=	48	8*7=	56	8*8=	64	8*9=	72	8*10=	80
9*1=	9	9*2=	18	9*3=	27	9*4=	36	9*5=	45	9*6=	54	9*7=	63	9*8=	72	9*9=	81	9*10=	90
10*1=	10	10*2=	20	10*3=	30	10*4=	40	10*5=	50	10*6=	60	10*7=	70	10*8=	80	10*9=	90	10*10=	100

Начнем с малого - пусть нужно напечатать

1*1=1

Вот фрагмент программы:

Фрагмент 1

```
a=1
b=1
proizv = a * b
Print a,  "*"  ,b,  "="  ,proizv
```

Здесь в операторе Print 5 элементов:

* сомножитель a,
* символ знака умножения "*",
* сомножитель b,
* символ "=",

* значение произведения proizv

Усложним задачу. Попробуем заставить компьютер напечатать первую строку таблицы:

1*1= 1 1*2= 2 1*3= 3 1*4= 4 1*5= 5 1*6= 6 1*7= 7 1*8= 8 1*9= 9 1*10= 10

Замечаем, что здесь нам нужно решить 9 элементарных задач на вычисление произведения, первую из которых решает фрагмент 1. Все они очень похожи и различаются лишь значением второго сомножителя. Таким образом, для решения каждой из 9 задач подошел бы наш фрагмент 1, если бы в нем в операторе b=1 вместо единицы стояла нужная цифра. В данном случае идеально подходит оператор For:

Фрагмент 2

```
a = 1
For b = 1 To 10
    proizv = a * b
    Print a; "*"; b; "="; proizv;
Next b
```

Прокрутите программу в пошаговом режиме.

Следующая ступень усложнения - последняя - напечатать не одну строку таблицы, а девять. Для этого фрагмент 2 должен быть выполнен 9 раз, каждый раз - с новым значением a. Чтобы этого достичь, “обнимем” фрагмент 2 оператором For точно так же, как мы обнимали фрагмент 1.

Фрагмент 3

```
For a = 1 To 10
    For b = 1 To 10
        proizv = a * b
        Print a; "*"; b; "="; proizv;
    Next b
Next a
```

Прокрутите программу в пошаговом режиме, для удобства заменив 9 на 4.

Печатается все, что надо, но в одну строчку. Добавим в нужное место пустой Print, чтобы после окончания очередной строки печать начиналась с новой:

Фрагмент 4

```
For a = 1 To 10
    For b = 1 To 10
        proizv = a * b
        Print a; "*"; b; "="; proizv;
    Next b
    Print
Next a
```

Прокрутите программу в пошаговом режиме. Прочувствуйте то, что пустой Print мы поместили именно в нужное место, строчкой выше или ниже он все бы испортил.

Печатает фрагмент 4 плохо. Акуратных столбцов не получается. Мы не будем добиваться идеальной картинки. Просто поменяем точку с запятой в конце оператора Print на запятую, чтобы результаты были выровнены по столбцам. Столбцы расположены на расстоянии 14 символов друг от друга. Поменяем шрифт (свойство Font) формы на Courier, потому что у этого шрифта в отличие от многих других символы имеют одинаковую ширину.

В целом программа иллюстрирует идею вложенных циклов, когда один, внутренний, цикл вложен внутрь другого, внешнего. У нас тело внешнего цикла выполняется 10 раз, а тело внутреннего - 100 раз, так как на каждое выполнение внешнего цикла он выполняется 10 раз.

Задание 57:

1) Распечатать все возможные сочетания из двух цифр - первая цифра может быть любой от 3 до 8, вторая - любой от 0 до 7. Например, 36, 44, 80.
2) Распечатать все возможные сочетания из четырех цифр, каждая из которых может принимать значения 1,2,3. Например, 2123, 3312, 1111.
3) Подсчитать количество таких сочетаний.
4) Подсчитать из них количество неубывающих сочетаний, то есть таких, где каждая следующая цифра не меньше предыдущей - 1123, 1223, 2222 и т.п., но не 3322. Распечатать все такие сочетания.

8.7. Поиск максимального из чисел

Задача программисту: Найти максимальное из вводимых в компьютер чисел.
Задача рыбаку: Принести домой самую большую из выловленных рыб.
Решение рыбака: Рыбак приготовил для самой большой рыбы пустое ведро. Первую пойманную рыбу рыбак не глядя бро-

сает в это ведро. Каждую следующую рыбу он сравнивает с той, что в ведре. Если она больше, то он бросает ее в ведро, а ту, что была там раньше, выпускает в реку.

Решение программиста: Программист приготовил для самого большого числа ячейку и придумал ей название, скажем, max. Первое число программист не глядя вводит в эту ячейку. Каждое следующее число (назовем его chislo) он сравнивает с max. Если оно больше, то он присваивает переменной max значение этого числа.

Напишем программу для определения максимального из 10 вводимых чисел:

```
Private Sub Command1_Click()
    Max = InputBox("Введите число")          'первую рыбу - в ведро
    For i = 2 To 10                          'ловим остальных рыб:
        chislo = InputBox("Введите число")   'поймали очередную рыбу
        If chislo > Max Then Max = chislo    'и если она больше той, что в ведре, бросаем ее в ведро
    Next i
    Debug.Print Max                          'несем самую большую рыбу домой
End Sub
```

Задание 58: Найти из N чисел минимальное. Каким по порядку было введено минимальное число? *Указание*: для номера минимального числа тоже нужно отвести специальную ячейку.

Задание 59: У вас есть данные о росте ваших одноклассников. Правда ли, что рост самого высокого отличается от роста самого низкого больше, чем на 40 см.?

Глава 9. Графика

Графикой у компьютерщиков принято называть все то, что связано с рисунками и изображениями.

В этой главе вы изучите все необходимые графические средства Visual Basic. Применяя их в дальнейших главах, вы научитесь создавать все те замечательные программы и игры, которые я перечислил в предисловии, и почувствуете, что нет такой игры на неподвижном поле, которую бы вы не могли запрограммировать, будь то танковая битва, морской бой, гонки, крестики-нолики и т.д. и т.п. Правда, некоторые игры (типа лабиринтов) требуют для своего создания еще и отличного логического мышления, но к Visual Basic это уже не относится.

Прежде чем рисовать, познакомимся с некоторыми новыми понятиями.

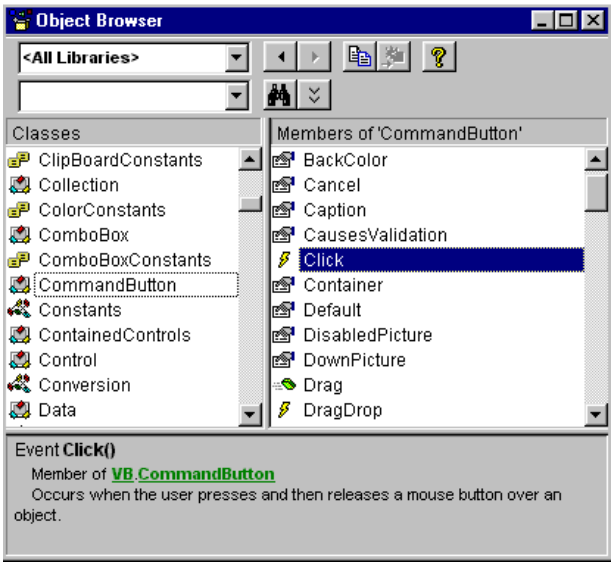
9.1. Объекты. Их свойства, их события, их методы

То, что у каждого объекта есть свойства, мы уже усвоили. Так, у формы Form1 есть свойство BackColor (цвет фона), а у метки Label8 есть свойство Caption (надпись). У кнопки Command2 тоже есть свойство Caption, но оно у ней свое. Принадлежность свойства указывается через точку - Label8**.**Caption, Command2**.**Caption, Text1**.**Width.

Что вы можете делать со свойствами объектов? Вы можете их устанавливать и менять вручную в режиме проектирования. Вы можете запрограммировать их автоматическое изменение в режиме работы.

Что касается событий, я как-то не акцентировал, кому они принадлежат. Создавалось впечатление, что они, вроде, сами по себе, не относятся ни к какому объекту. Нет же. Посмотрите на процедуры, написанные нами ранее. Кроме процедуры Command1_Click, мы встретились с такими процедурами, как Form_Load и Form_Terminate. Вы видите, что в заголовке процедуры справа от знака подчеркивания мы указываем событие, а слева - объект, к которому это событие относится. Таким образом, события принадлежат объектам и принадлежность события указывается через знак подчеркивания в заголовке процедуры.

Событий в Visual Basic огромное количество и многие из них мы изучим. Чтобы посмотреть, какие события могут случиться с таким-то объектом, нужно запустить так называемый **Object Browser** ("перелистыватель объектов" - его найдете в меню View):



В окне Object Browser в левом списке вы кроме всего прочего найдете названия всех элементов управления, находящихся в данный момент в Toolbox. Щелкните по интересующему вас объекту (я щелкнул по CommandButton) - в правом списке вы увидите свойства, события и так называемые методы данного объекта. Свойства помечены значком в виде руки, что-то нажимающей на каком-то пульте (вы можете видеть на картинке свойство Caption). События помечены значком в виде зигзага молнии, методы помечены непонятным значком, по-моему это летящий кирпич. Щелкните по интересующему вас элементу правого списка (я щелкнул по событию Click) - в нижней части окна Object Browser появится краткое описание этого элемента.

Кстати, в Object Browser вы найдете много свойств, которых нет в знакомом вам окне свойств, потому что существуют свойства, которые нельзя задавать в режиме проектирования.

Что вы можете делать с событиями? Вы можете программировать в соответствующих процедурах реакцию компьютера на них, а в режиме работы вы можете многие из них вызывать мышкой или с клавиатуры.

Кроме свойств и событий у объектов есть так называемые методы. **Метод** - это кое-что из того, что умеет делать данный объект. Вспомним пример с игрушечной железной дорогой из 1.2. Там были объекты: паровозы, светофоры, стрелочники и т.п. У паровоза могут быть такие методы: ехать, гудеть, тормозить … У светофора - зажигать красный, зажигать зеленый … У стрелочника - переводить стрелку, петь песни … Обратите внимание, что все эти умения вам не надо было программировать, они заложены в конструкцию объектов еще на заводе. Вы могли только регулировать их: с какой скоростью ехать, когда зажигать зеленый и т.д.

Для знакомства с методами Visual Basic выберем метод **SetFocus**. Чтобы его понять, сначала нужно узнать, что такое **фо-**

кус. Создайте проект с двумя кнопками и двумя текстовыми полями. Запустите его. Щелкните по одной кнопке, по другой, по одному полю, по другому. Ничего, конечно, не происходит, но если последний раз вы щелкнули по кнопке, то она имеет несколько другой, чем раньше, "нажатый" вид. А если последний раз вы щелкнули по текстовому полю, то именно в нем мигает курсор. Таким образом, можно сказать, что из множества объектов на форме один какой-то выделяется, то есть *находится в фокусе*. Этим объектом оказывается как раз тот объект, которым вы пользовались последним. Говорят, что *объект обладает фокусом*.

У объектов, способных иметь фокус, имеется два события: **GotFocus**, которое возникает в момент приобретения объектом фокуса, и **LostFocus**, которое возникает в момент потери фокуса.

Запишите в окно кода такие процедуры:

```
Private Sub Text1_GotFocus()
    Debug.Print "Text1 получил фокус"
End Sub

Private Sub Text2_GotFocus()
    Debug.Print "Text2 получил фокус"
End Sub

Private Sub Text1_LostFocus()
    Debug.Print "Text1 потерял фокус"
End Sub

Private Sub Text2_LostFocus()
    Debug.Print "Text2 потерял фокус"
End Sub
```

Пощелкайте по текстовым полям. Понаблюдайте, что появляется при этом в окне Immediate и в какой последовательности.

Фокус можно переводить на объект и программным путем. Дополним наш проект двумя процедурами:

```
Private Sub Command1_Click()
    Text1.SetFocus
End Sub

Private Sub Command2_Click()
    Text2.SetFocus
End Sub
```

Запустите проект. Щелкните по кнопке Command1. Фокус, вопреки ожиданиям, переместился не на кнопку Command1, а в поле Text1. Произошло это благодаря оператору **Text1.SetFocus**, который можно перевести так - "Установи фокус в объект Text1". Щелкните по кнопке Command2. Фокус переместился в поле Text2. Метод **SetFocus** принадлежит объекту и отделяется от своего хозяина точкой. Дело его простое - переводить фокус на своего хозяина.

Не все объекты обладают методом SetFocus. Например, у метки его нет. Зачем Бэйсику нужен фокус, мы узнаем позже.

Мы раньше уже познакомились с одним методом - это Print. Объект, обладающий этим методом, печатает информацию на своей поверхности. Поместите в проект элемент управления PictureBox (с именем Picture1) и выполните оператор Picture1**.**Print "Hello!". Текст будет напечатан не на форме, а на самом объекте.

Если хозяина метода мы не указываем, то есть пишем просто Print, то *по умолчанию* считается, что хозяином является форма. То же относится и к свойствам. Например, оператор Width = 5000 устанавливает ширину именно формы, а не чего-нибудь другого.

Всё ли умеют делать методы, которые мы видим в Object Browser? Нет, далеко не всё, и даже мало что. Гораздо более богатые результаты мы достигаем, меняя свойства объектов или программируя реакцию на события, как все время делали раньше. Но вот когда мы научимся создавать собственные объекты, тогда мы будем писать для них методы сами, и уж тогда от нас самих будет зависеть, насколько богаты они будут.

9.2. Три способа рисовать

В Visual Basic есть три основных способа использовать графику:

- Не входя в Visual Basic, в каком-нибудь графическом редакторе вы можете нарисовать то, что вам нужно, или найти на диске готовый файл с подходящим рисунком. Затем, войдя в Visual Basic, придать этот рисунок форме или другому объекту, как это объяснялось в Глава 2
- В Toolbox имеется два элемента управления: **Line** (**Линия**), имеющий вид отрезка прямой различного цвета, толщины и стиля, и **Shape** (**Фигура**), принимающий вид прямоугольников, окружностей и эллипсов тоже различного цвета, толщины и стиля. Если их на этапе проектирования поместить на форму и придать им нужные размеры и прочее, то можно получить, в общем, любую картинку по принципу "Точка, точка, огуречик, вот и вышел человечек".
- Форма, элемент управления PictureBox и некоторые другие объекты обладают методами, работа которых заключается в том, чтобы рисовать на поверхности своего хозяина точки, отрезки, прямоугольники, окружности, эллипсы, дуги и сектора, в общем, все то, что достигалось предыдущим способом. Эти методы будем называть **графическими**. Основное отличие от предыдущего способа в том, что здесь все надо программировать, а там все делалось вручную. У каждого из этих способов есть и другие преимущества и недостатки, о чем позже.

Есть и другие, более продвинутые, сложные и громоздкие способы, но ими пользуются только тогда, когда цели нельзя достигнуть указанными тремя. Рассмотрим по очереди все три способа.

9.3. Первый способ - Загрузка в Visual Basic готовых изображений

Рассмотрим первый способ использования графики в Visual Basic.

Придать объекту картинку можно, установив его свойство Picture:

- в режиме проектирования - вручную,
- в режиме работы - использовав функцию LoadPicture. Например,

Form1.Picture = LoadPicture("C:\TEMP\Rockies.bmp")

Здесь в скобках в кавычках пишем адрес файла с картинкой на диске. Вкратце этот процесс описан в Глава 2

Если у вас в проекте несколько объектов, имеющих свойство Picture, то картинку можно мгновенно "скопировать" с одного объекта на другой так:

Image4.Picture = Form1.Picture

Если вы хотите *удалить картинку с объекта*:

- в режиме проектирования вы должны буквально стереть значение свойства Picture из окна свойств.
- в режиме работы вы должны написать оператор такого вида:

Form1.Picture = LoadPicture("")

Типы графических файлов

Visual Basic 6.0 поддерживает (воспринимает и работает с ними) графические файлы следующих типов:

Расширение файла	Краткое описание
.BMP, .DIB	Фотографии и любые другие растровые изображения.
.ICO	Значки (**пиктограммы**, **иконки**) - очень маленькие
.CUR	Значки курсоров - очень маленькие
.WMF, .EMF	Метафайлы Windows - векторные изображения
.GIF	Растровые изображения, применяются в Интернете
.JPG, .JPEG	Растровые изображения (часто пониженного качества, зато экономные), применяются в Интернете

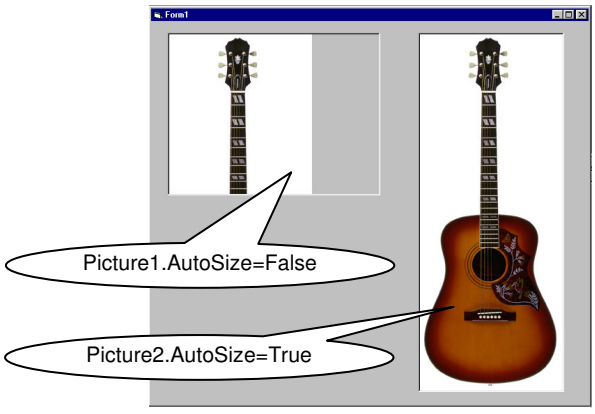
Где взять все эти файлы? Многие из этих типов вы найдете в папке Graphics, находящейся в папке, посвященной Visual Basic. Многие - в папке Clipart из папки Microsoft Office. В папке Windows вы найдете большую картинку Облака.bmp. Красивые картинки типа JPG находятся по адресу C:\Program Files\Plus!\Themes. Если вы умеете в Windows запускать поиск файлов по расширениям, то поищите, что-нибудь обязательно найдете. Если у вас есть любимая фотография или картинка в журнале, попросите своего знакомого, у которого есть сканер, отсканировать ее и записать на вашу дискету. Дело пяти минут. На дискете при обычном сканировании уместятся 3-4 фотографии, а в формате JPG - 30-40. Ну а если у вас есть Интернет, то тут и говорить не о чем.

Регулировка размеров изображений

Рассмотрим, как размещаются и умещаются ли картинки в объектах нашего проекта.

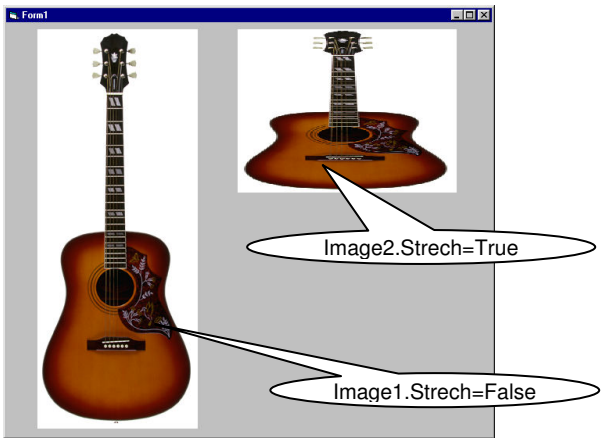
Сначала испытаем форму. Загрузим в нее любой растровый файл и попробуем изменять размеры формы. Мы видим, что и в режиме проектирования и в режиме работы эти изменения никак не влияют на саму картинку - если форму совсем уменьшить, то мы будем видеть лишь часть картинки, если слишком увеличить - справа и снизу формы останутся пустые места. Сама картинка не увеличится и не уменьшится. А вот если загрузить векторную картинку, то она будет автоматически растягиваться и сжиматься по размерам формы.

А теперь испытаем элемент управления PictureBox. Он специально предназначен для размещения картинок. Проверьте и убедитесь, что по отношению к ним он ведет себя так же, как форма. Зачем он тогда нужен? Для многого. У него есть, например, свойство **AutoSize**, которое, если установить его в True, заставляет PictureBox в режиме работы подстраивать свои размеры под размеры картинки. (А что происходит в режиме проектирования, вряд ли нас должно интересовать, правда?) Получается вот что:



В этом случае даже векторная графика перестает быть послушной и заставляет PictureBox (в режиме работы, а не проектирования!) подстраиваться под свои размеры.

А теперь испытаем элемент управления Image. Он тоже специально предназначен для размещения картинок. Проверьте, установлено ли его свойство **Strech** в False, и убедитесь, что по отношению ко всем видам графики объект Image в режиме работы подстраивает свои размеры под размеры картинки. В чем же тогда его отличие? Вот в чем. Установите свойство Strech в True, это заставит уже картинку в режиме работы подстраивать свои размеры под размеры Image, а не наоборот. Любую картинку, в том числе и растровую. Получается вот что:



Выходит, что свойство Strech - переключатель того, что подо что "прогнется" - Image под картинку или картинка под Image.

А теперь самостоятельно испытайте кнопку - CommandButton. Но перед испытанием обязательно придайтете ее свойству **Style** значение **Graphical**, иначе никаких картинок на кнопке не будет видно. Кроме свойства Picture у кнопки есть еще, например, такое свойство - **DownPicture**. Оно определяет другую картинку, а именно ту, которую мы видим, нажав кнопку и не отпуская ее.

Задание 60: Вы профессиональный продавец автомобилей. Вы приезжаете к покупателю, достаете портативный компьютер, на экране - несколько десятков кнопок, на каждой - маленькая фотография одного из продаваемых автомобилей. Покупатель говорит: "Вот этот покажите, пожалуйста". Вы нажимаете кнопку и на экране возникает та же фотография, но увеличенная.

Помощь: Если вы собираетесь в качестве кнопок использовать элементы управления CommandButton, то уменьшенные фото вам придется предварительно сделать в каком-нибудь графическом редакторе и сохранить их в отдельные файлы, постаравшись, чтобы эти маленькие картинки на экране были примерно одинаковых размеров. Но можно и избежать такой потери времени. Ведь кнопками могут служить объекты Image! Потому что у объекта Image тоже есть событие Click! Создайте на форме несколько маленьких Image и один большой и в маленькие впишите фото. По щелчку мыши по маленькому Image большой Image пусть копирует в себя его картинку.

Необязательное усложнение для тех, кто не боится системы координат: Если у вас все получилось, то вы уже обратили внимание на одну проблему. Проблема в том, что размеры и форма исходных фото разные: одни продолговатые, другие квадратные (а если у вас одинаковые, то сделайте разными, а то неинтересно). Бог с ними, с кнопками (хотя с ними та же самая проблема), нам хочется, чтобы хоть большие-то картинки располагались на экране симметрично как по горизонтали, так и по вертикали, и имели максимально возможный размер. Вот этой цели я и хочу, чтобы вы достигли. Для этого вам придется использовать оператор ветвления, а также свойства, задающие размер и местоположение объектов.

Проверьте, чтобы форма была распахнута на весь экран. Как это сделать, я объяснял в 2.6.

После щелчка мыши по кнопке компьютер должен сделать следующее:

- Настроить большой Image, чтобы он подстраивался под размеры картинки, установив сами знаете какое свойство.
- Пусть большой Image копирует в себя картинку из маленького. На форме появится более-менее большое фото неискаженных пропорций, но не по центру. Все это у вас уже давно готово. Задача - увеличить фото еще больше, и чтобы оно было по центру.
- Поделить ширину формы на ее высоту, чтобы узнать ее "продолговатость". (Это надо бы пораньше, да ладно.)
- Поделить ширину Image на его высоту, чтобы узнать "продолговатость" картинки.
- Если продолговатость картинки больше, чем продолговатость формы, то в идеале при максимальном увеличении картинка должна почти упереться левым и правым краем в края формы, а сверху и снизу должно остаться одинаковое пространство. Для этого нужно выполнить несколько операторов присваивания, увеличивающие размер и изменяющие местоположение Image, да так, чтобы продолговатость Image равнялась продолговатости картинки, а затем заставить картинку принять размеры Image.
- Если продолговатость картинки меньше, чем продолговатость формы, то в идеале при максимальном увеличении картинка упрется верхним и нижним краем в края формы, а слева и справа должно остаться одинаковое пространство. Здесь тоже нужно выполнить несколько аналогичных операторов.

Таким образом, при щелчке по кнопке на форме возникает фото и сразу же за этим вместо него - оно же, но увеличенное и по центру. Если вам не нравится такое мелькание, поэкспериментируйте в программе со свойством Visible объекта Image. Гарантирую прекрасные результаты.

Другие полезные сведения

Если вы загружаете картинку в объект в режиме проектирования, то Visual Basic сохраняет ее в одном из файлов, из которых состоит ваш проект. Это резко увеличивает размеры вашего проекта на диске и время загрузки проекта, зато теперь вы можете безнаказанно стереть исходный графический файл с диска. Если вы загружаете картинку в объект в режиме работы, то все наоборот.

Преимущество формы и PictureBox в том, что вы при помощи их методов можете печатать информацию прямо на их картинке и рисовать по ней (например, пририсовать кому-нибудь очки).

9.4. Второй способ - Объекты Line и Shape

Рассмотрим второй способ использования графики в Visual Basic.

Возьмем в Toolbox объект **Line** (Линия) и проведем наискосок мышкой по форме. На форме разместится отрезок прямой. Мы можем его перемещать за маркеры или ухватившись острием мышиного курсора за середину. Поэкспериментируйте со следующими свойствами линии:

Свойство	Смысл
BorderWidth	Толщина линии
BorderColor	Цвет линии
BorderStyle	Стиль линии (сплошная, штриховая и т.п.). Предварительно сделайте толщину = 1.
X1, Y1, X2, Y2	Координаты крайних точек отрезка

Возьмем в Toolbox объект **Shape** (Фигура) и проведем наискосок мышкой по форме. На форме разместится прямоугольник. Он может превратиться в квадрат, овал (эллипс), круг, может закруглить углы, все это в зависимости от значения его свойства **Shape** (да-да, одинаковые названия у объекта и его свойства). Поэкспериментируйте со следующими свойствами фигуры:

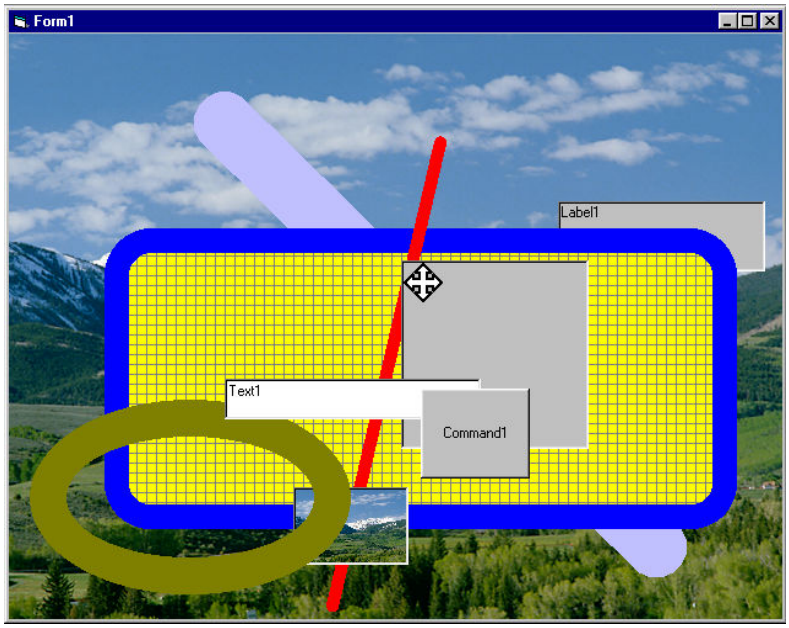
Свойство	Смысл
BorderWidth	Толщина линии
BorderColor	Цвет линии
BorderStyle	Стиль линии (сплошная, штриховая и т.п.). Предварительно сделайте толщину = 1.

Сейчас пространство внутри фигуры прозрачное. В этом легко убедиться, если на форму загружена какая-нибудь картинка. Предположим, вы хотите, чтобы пространство внутри фигуры было не прозрачным, а залито краской или заполнено узором. Для этого давайте разберемся в устройстве фигуры. Удобно представлять, что фигура - это оконная рама со вставленными в нее двумя прозрачными стеклами. Одно стекло - подальше от наших глаз, другое поближе. Прозрачностью дальнего стекла управляет свойство **BackStyle**, а цветом - свойство **BackColor**. На дальнем стекле узоров не бывает. Прозрачностью и узором ближнего стекла управляет свойство **FillStyle**, а цветом - свойство **FillColor**. Поэкспериментируйте.

Метку тоже можно сделать прозрачной, как и фигуру, если ее свойство BackStyle задать равным Transparent.

9.5. Взаимное перекрытие объектов. Метод ZOrder

На рисунке вы видите несколько объектов, размещенных на форме и частично перекрывающих друг друга. Это линии, фигуры, Image (маленькое фото), PictureBox (рамка с маленьким крестиком в углу), текстовое поле, кнопка, метка. По какому принципу они перекрывают друг друга? Почему текстовое поле перекрывает линию и фигуру, а не наоборот? Рассмотрим эту механику.



Все объекты расположены в трех слоях и мы не можем менять местами слои и перемещать объект из одного слоя в другой.

Самый дальний от наших глаз слой - это сама форма со своей картинкой и с тем текстом и изображениями, которые мы можем на ней получить при помощи ее методов.

В среднем слое помещаются объекты Line, Shape, Image, метка.

В ближнем к нам слое помещаются все неграфические объекты и PictureBox.

Мы можем перемещать объекты в пределах своего слоя ближе или дальше от глаз:

- В режиме проектирования - выделить интересующий нас объект →**Format**→**Order**→ далее нужно выбрать одну из двух команд: **Bring to Front** (перенести на передний план) или **Send to Back** (отослать на задний план).
- В режиме работы - использовать метод **ZOrder**. Так, чтобы перенести на передний план текстовое поле, мы пишем оператор:

  Text1.ZOrder (0)

  а чтобы отослать его на задний план:

  Text1.ZOrder (1)

9.6. Цвет в Visual Basic

Как до сих пор мы придавали цвет какому-нибудь элементу Visual Basic:

- В режиме проектирования мы устанавливали соответствующее свойство в окне свойств. Для этого в закладке Palette выбирали один из нескольких десятков цветов.
- В режиме работы пользовались операторами типа Form1.BackColor = vbRed. Имеется всего 8 цветов, представленных этим способом:

vbBlack	Черный
vbRed	Красный
vbGreen	Зеленый
vbYellow	Желтый
vbBlue	Синий
vbMagenta	Фиолетовый (неточный перевод)
vbCyan	Голубой (неточный перевод)
vbWhite	Белый

Но в Visual Basic существует 16 миллионов цветов с лишним (точнее - 16777216)! Мы должны научиться ими управлять. Есть несколько способов. Так, цвет можно указывать просто числом от 0 до 16777215. Например,

Form1.BackColor = 12456743

Недостаток этого способа - по числу трудно угадать, что за цвет.

Мне нравится такой способ: Вспомним, что любую краску можно получить, смешав в определенной пропорции красную (**R**ed), зеленую (**G**reen) и синюю (**B**lue) краски. В Visual Basic каждой краски в смесь можно положить от 0 до 255 единиц. Смешивает краски специальная функция **RGB** (название - по первым буквам цветов). Пусть мы хотим покрасить форму краской, в которую мы положили и смешали 100 единиц красной, 200 единиц зеленой и 50 единиц синей краски. Для этого пишем такой оператор:

Form1.BackColor = RGB(100, 200, 50)

Чем меньше каждой краски мы положим, тем темнее будет цвет, чем больше - тем светлее:

RGB(70, 90, 88)	Темный цвет (потому что числа маленькие)
RGB(210, 250, 202)	Светлый цвет (потому что числа большие)
RGB(0, 0, 0)	Черный цвет

RGB(255, 255, 255)	Белый цвет

Если каждой краски положить поровну, получится серый цвет:

RGB(90, 90, 90)	Темно-серый цвет
RGB(220, 220, 220)	Светло-серый цвет

RGB(255, 255, 0)	Желтый цвет

Результатом работы функции RGB является число, обозначающее цвет. Кстати, для любознательных. Посчитайте-ка, сколько всего цветов можно получить, перебрав все возможные в Visual Basic сочетания красного, зеленого и синего.

Задание цвета в режиме проектирования

Хорошо. В режиме работы мы научились задавать все цвета. А в режиме проектирования? Мы видим, что цвета в окне свойств закодированы строкой из каких-то непонятных значков, например:

&H0080C0FF&

Эту строку вы можете вручную изменять, ставя на место одних значков подходящие другие, и таким образом задавать любой из 16 миллионов цветов.

Попробуем разобраться, что означают значки этого кода:

&H00	**80**	**C0**	**FF**	**&**
Пока не обращайте внимания и не пытайтесь менять	Сколько в цвете **синей** краски	Сколько в цвете **зеленой** краски	Сколько в цвете **красной** краски	Пока не обращайте внимания и не пытайтесь менять

Количество каждой краски закодировано в так называемой шестнадцатиричной системе счисления. К сожалению, я не могу из-за ограниченного объема книги дать вам здесь прочное понятие о системах счисления, попытаюсь только дать способ разобраться в кодировке.

В отличие от привычной нам десятичной системы счисления, где 10 цифр (0, 1, 2, … , 9), в шестнадцатиричной системе 16 цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Не очень удобно, что старшие цифры обозначены буквами, но другие значки еще неудобнее.

Если человек, привыкший кодировать все числа в 16-й системе, скажет, что он видит *7* предметов, то обычный человек, глядя на них, тоже скажет, что их 7. Если тот скажет, что он видит *A* предметов, то обычный человек, глядя на них, скажет, что их 10.Если тот скажет, что он видит *F* предметов, то этот скажет, что их 15.

Когда обычный человек видит наше привычное десятичное число 47, то он говорит, что надо взять 4 раза по десять и еще прибавить 7.

Когда 16-й человек видит 16-е число 29, то он учит обычного человека, что надо взять 2 раза по шестнадцать и еще прибавить 9. Получится 41. Итак, 16-е число 29 означает 10-е число 41.

Когда 16-й человек видит 16-е число AE, то он учит обычного человека, что надо взять A раз (то есть 10 раз) по шестнадцать и еще прибавить E (то есть 14). Получится 174. Итак, 16-е число AE означает 10-е число 174.

Максимальное двузначное число в 16-й системе - FF. Убедитесь, что оно равно 255 в 10-й системе. Получается, что для задания количества краски в цвете Visual Basic достаточно двузначного 16-го числа. Такая кодировка и применяется на самом деле.

Теперь вы можете сознательно менять 16-е цифры в окне свойств и наблюдать результат.

А сейчас я хочу вам подсказать, как задавать количество красной, синей и зеленой краски в том самом числе, которое от 0 до 16777215. Для знатоков систем счисления достаточно знать, что переведя это число из 10-й системы в 16-ю, вы получите число из шести 16-х цифр, полностью определяющее цвет так, как я это только что описал. Для остальных скажу, что вам достаточно задавать это число таким выражением:

Form1.BackColor = **B** * 256 * 256 + **G** * 256 + **R**

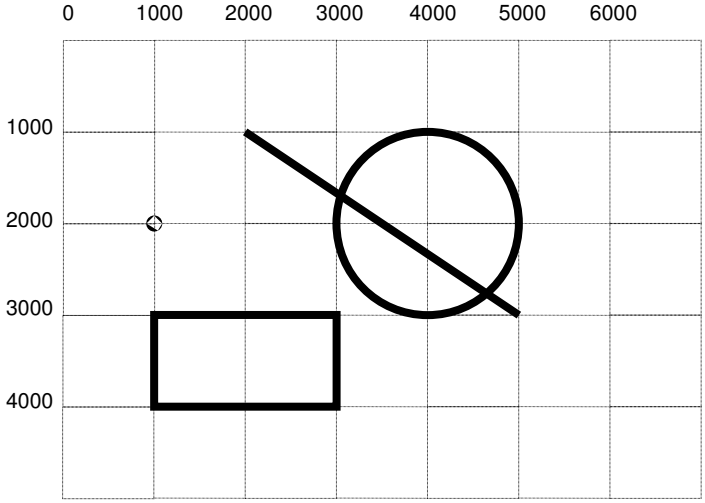
Здесь B, G, R - числовые переменные, имеющие тот же смысл, что и числа в функции RGB. А теперь, знатоки, скажите, не кажется ли вам, что перед вами 256-я система счисления?

9.7. 3 способ - Рисуем при помощи графических методов

Рассмотрим третийй способ использования графики в Visual Basic.

Методами для рисования различных геометрических фигур обладают два объекта: форма и PictureBox. Кстати, те же, что обладают и методом Print, который с полным основанием тоже называют графическим.

Напишем программу, которая рисует на форме точку, прямоугольник, окружность и отрезок прямой в тех местах, где это показано на рисунке:



Вот программа:

```
Private Sub Command1_Click()
    PSet (1000, 2000)                    'точка
    Line (2000, 1000)-(5000, 3000)       'отрезок прямой
    Line (3000, 3000)-(1000, 4000) , , B 'прямоугольник
    Circle (4000, 2000) , 1000           'окружность
End Sub
```

Пояснения: Числа на картинке обозначают горизонтальную и вертикальную координаты на форме (в твипах).

Вы видите, что методы записаны без указания объекта, которому они принадлежат. В этом случае считается, что *по умолчанию* они принадлежат форме. Если бы вы записали Picture1.Line, то рисование происходило бы на поверхности PictureBox.

Точка рисуется методом **PSet**. Два числа в скобках - координаты точки на форме, первое число - горизонтальная координата, второе число - вертикальная.

Будем называть величины, указанные в методе, **параметрами** метода.

Отрезок прямой рисуется методом **Line**. Мы знаем, что отрезок прямой можно построить, если известно положение двух его крайних точек. Они-то и задаются в обращении к методу. Первая пара параметров - координаты одной точки (любой из двух), вторая пара - другой.

Если дана пара точек, то между ними можно провести не только отрезок прямой, но и **прямоугольник**. Для этого достаточно в методе **Line** указать букву **B** после двух запятых.

Окружность можно построить, если известно положение центра и радиус. Окружность рисуется методом **Circle**, первые два параметра которого - координаты центра, третий - радиус.

Создайте новый проект и проверьте программу.

Кроме метода Print, в Visual Basic есть следующие графические методы (о тех из них, что нам не знакомы, поговорим чуть позже):

Метод	Смысл
PSet	Точка
Line	Линия или прямоугольник
Circle	Круг, эллипс, дуга, сектор
Cls	Очищает поверхность объекта от нарисованного и напечатанного
Point	Это функция, она сообщает цвет любой точки на объекте
PaintPicture	Копирует с одного объекта на другой прямоугольный кусок изображения

Настройка внешнего вида рисуемых фигур

Фигуры, изображенные методами вашей программы, нарисованы тонкими черными сплошными линиями без заливки внутренних областей. Это скучно. Вы должны уметь управлять толщиной, цветом и другими атрибутами фигур. Для этого есть три способа, которые вы можете применять по одиночке или вперемешку:

- В режиме проектирования настроить соответствующие свойства объекта. Эти свойства воздействуют сразу на *все* методы.
- В режиме работы в нужные моменты изменять эти свойства (этот способ бьет предыдущий, то есть в случае конфликта пересиливает его настройки).
- В самом методе можно задавать многие атрибуты рисуемой фигуры (этот способ бьет остальные).

Поговорим об этом подробнее.

Свойства объектов, влияющие на графические методы

Поэкспериментируйте со следующими свойствами (сначала в режиме проектирования, а затем в режиме работы):

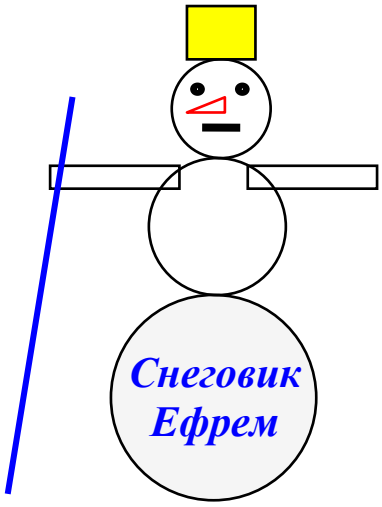
Свойство	Смысл
DrawWidth	Толщина линии

ForeColor	Цвет линии
DrawStyle	Стиль линии (сплошная, штриховая и т.п.). Предварительно сделайте толщину = 1.
FillStyle	Стиль (узор) заливки и будет ли заливка.
FillColor	Цвет заливки
AutoRedraw	Определяет, будут ли автоматически восстанавливаться графика и напечатанный текст, случайно стертые из-за того, что объект скрылся из вида.
DrawMode	Способ наложения краски. По умолчанию = 13 (Copy Pen), когда краска плотно накладывается и предыдущая картинка через нее не просвечивает. При других значениях новая краска меняет свой цвет или сложно взаимодействует со старой и иногда получается любопытный и полезный результат.

Пример:

```
PSet (1000, 2000)                'точка - тонкая, черная
Circle (4000, 2000), 1000        'окружность - тонкая, черная

DrawWidth = 20                   'меняем толщину линий
ForeColor = RGB(230, 250, 100)   'меняем цвет линий

PSet (1000, 5000)                'точка - толстая, цветная
Circle (4000, 5000), 1000        'окружность - толстая, цветная
```

Задание 61: В режиме работы нарисуйте снеговика и сделайте на нем надпись шрифтом Times:



А теперь рассмотрим подробнее графические методы.

Метод Pset

До этого я использовал методы, указывая минимальное число параметров. Теперь я хочу показать все важные параметры, которые можно указывать в методах.

Метод	Результат
PSet (1000, 2000)	Рисуется **точка** с координатами x=1000, y=2000. Цвет точки определяется свойством ForeColor
PSet (1000, 2000)**, vbRed**	Рисуется **красная** точка

Параметры метода могут быть выражениями, например:

PSet (x+200 , y) , RGB(5+a, 90, 80) + 100

Вообще, в дальнейшем я буду для простоты при описании процедур, функций, методов в качестве значения их параметров указывать числа, но вы должны иметь в виду, что почти всегда там, где допустимы числа, допустимы и выражения.

Синтаксис метода PSet:

PSet (*x* , *y*) , *цвет*

Здесь все параметры, включая цвет - числовые выражения.

В этой и следующих синтаксических схемах я по возможности воздержусь от квадратных скобок, указывающих на необязательность параметра, иначе схемы будут ими перенасыщены. Также я не буду указывать объект - владелец метода. Также не буду указывать и отложу немного объяснение параметра Step.

Метод Line

Вот объяснение на примерах:

Метод	Результат
Line (2000, 1000)-(5000, 3000)	**Отрезок** прямой между точкой с координатами (2000, 1000) и точкой с коорди-

	натами (5000, 3000).
Line (2000, 1000)-(5000, 3000) **, vbRed**	Отрезок **красного** цвета
Line (2000, 1000)-(5000, 3000) **, vbRed , B**	**Прямоугольник** красного цвета
Line (2000, 1000)-(5000, 3000) **, vbRed , BF**	Прямоугольник красного цвета, **залитый** этим же цветом
Line (2000, 1000)-(5000, 3000) **, , B**	Прямоугольник. Цвет его определяется свойством ForeColor, так как в операторе там, где должен быть указан цвет, стоит пустота

Нет смысла и запрещено писать одну букву F вместо B или BF.

Заливку можно сделать двумя способами:

- Поставив букву F, тогда заливка будет тем же цветом, что и линия.
- Не ставя букву F, тогда заливка определяется свойствами FillStyle и FillColor.

Обратите внимание на две стоящие рядом запятые в последнем примере. Это вполне понятный стиль Visual Basic - если в списке необязательных параметров какой-то параметр пропускается, то запятые нужно указывать все равно, а то будет непонятно, какой по порядку параметр вы указали правее.

Синтаксис метода Line:

Line (*x1*, *y1*) **-** (*x2*, *y2*) **,** *цвет* **,** **B** | **BF**

Здесь все параметры, включая цвет - числовые выражения.

Вертикальная черта | означает "или". Имеется в виду, что в этом месте оператора вы можете поставить или **B** или **BF**.

Метод Circle

Вот объяснение на примерах:

Метод	Результат
Circle (4000, 2000) **,** 1000	**Окружность** с центром в точке с координатами (4000, 2000) и радиусом 1000
Circle (4000, 2000) **,** 1000 **, vbRed**	**Красная** окружность
Circle (4000, 2000) **,** 1000 **, , 1 , 3**	**Дуга окружности**, начинающаяся от угла в 1 радиан* и кончающаяся углом в 3 радиана. Угол отмеряется от направления на восток против часовой стрелки
Circle (4000, 2000) **,** 1000 **, , -1 , -3**	**Сектор круга**, начинающийся от угла в 1 радиан и кончающийся углом в 3 радиана. Угол отмеряется от направления на восток против часовой стрелки
Circle (4000, 2000) **,** 1000 **, , , , 2**	**Эллипс** (эллипс - это почти овал) с центром в точке с координатами (4000, 2000). Получен из окружности радиусом 1000 **горизонтальным сжатием в 2 раза**
Circle (4000, 2000) **,** 1000 **, , , , 1/3**	**Эллипс** с центром в точке с координатами (4000, 2000). Получен из окружности радиусом 1000 **вертикальным сжатием в 3 раза**
Circle (4000, 2000) **,** 1000 **, , 1 , 3 , 2**	**Дуга эллипса**
Circle (4000, 2000) **,** 1000 **, , -1 , -3 , 2**	**Сектор эллипса**

Синтаксис метода Circle:

Circle (*x_центра* **,** *y_центра* **)** **,** *радиус* **,** *цвет* **,** *начальный_угол* **,** *конечный_угол* **,** *сжатие*

Здесь все параметры, включая цвет - числовые выражения.

CurrentX, CurrentY, Step

В процессе рисования Visual Basic постоянно меняет свойства формы **CurrentX**, **CurrentY**, которые равняются координатам последней нарисованной точки. Для метода Line это та из двух точек, координаты которой указаны в правых скобках, а если дело касается метода Circle - то это координаты центра окружности. Для метода PSet все ясно без объяснений. Метод Print тоже меняет эти координаты, устанавливая их в то место, где должен появиться следующий напечатанный символ. Вы можете отслеживать эти свойства при помощи оператора Debug.Print CurrentX, CurrentY.

Теперь поговорим о Step.

Задача: Нарисовать три точки: одну с координатами (500, 1000), а две другие правее и выше. Расстояние соседних точек друг от друга равно 179 твипов по горизонтали и 40 твипов по вертикали.

Фрагмент программы, решающий дело:

```
PSet (500, 1000)
PSet (679,   960)
PSet (858,   920)
```

Здесь нам пришлось вычислять координаты. Есть другой способ, который иногда бывает удобнее:

```
PSet       (500, 1000)
PSet Step(179,   -40)
PSet Step(179,   -40)
```

* Для тех, кто не знаком с радианами, поясню, что в одном радиане 180/□ градусов, где □=3.14

Перед любыми скобками с указанием координат вы имеете право писать слово **Step**. Тогда числа в этих скобках перестают быть абсолютными координатами, а становятся смещением по горизонтали и вертикали от точки, координаты которой определяются свойствами CurrentX, CurrentY. Положительное смещение по горизонтали - направо, по вертикали - вниз.

Попробуйте без компьютера нарисовать на бумажке, что нарисует такой фрагмент:

```
Circle (2000, 2000), 1000
Line Step(900, 0)-Step(-900, -900)
```

Метод Cls

Он просто стирает все нарисованное и напечатанное. Вставьте его в программу и проверьте в пошаговом режиме:

```
Circle (2000, 2000), 1000
Print 12345
Cls
Line Step(900, 0)-Step(-900, -900)
```

Метод Point

Задача: Вы загрузили в форму фотографию морского пляжа и хотите узнать, какого цвета зонтик вот у этой дамы слева.

Решение: Сначала вам нужно узнать координаты хоть какой-нибудь точки на зонтике. Я думаю, вы сами догадаетесь, как это сделать. (Совершенно верно, здесь вам поможет маленькая окружность, координаты которой вы подбираете так, чтобы попасть в зонтик). Пусть окружность Circle (1000, 9000), 100 оказалась прямо на зонтике. Теперь достаточно выполнить процедуру:

```
Private Sub Command1_Click()
   Debug.Print  Point (1000, 9000)
End Sub
```

Цвет вы получите в виде числа, например, 12089756. Да-да, это то самое число от 0 до 16777215. Совершенно неудобоваримое. Для того, чтобы определить, сколько в нем красной, синей и зеленой краски, вам придется провести некоторые арифметические подсчеты, идею которых вам должно подсказать выражение из 9.6. Это и будет ваше

задание 62: Определить цвет заданной точки на форме и выдать одно из трех сообщений:

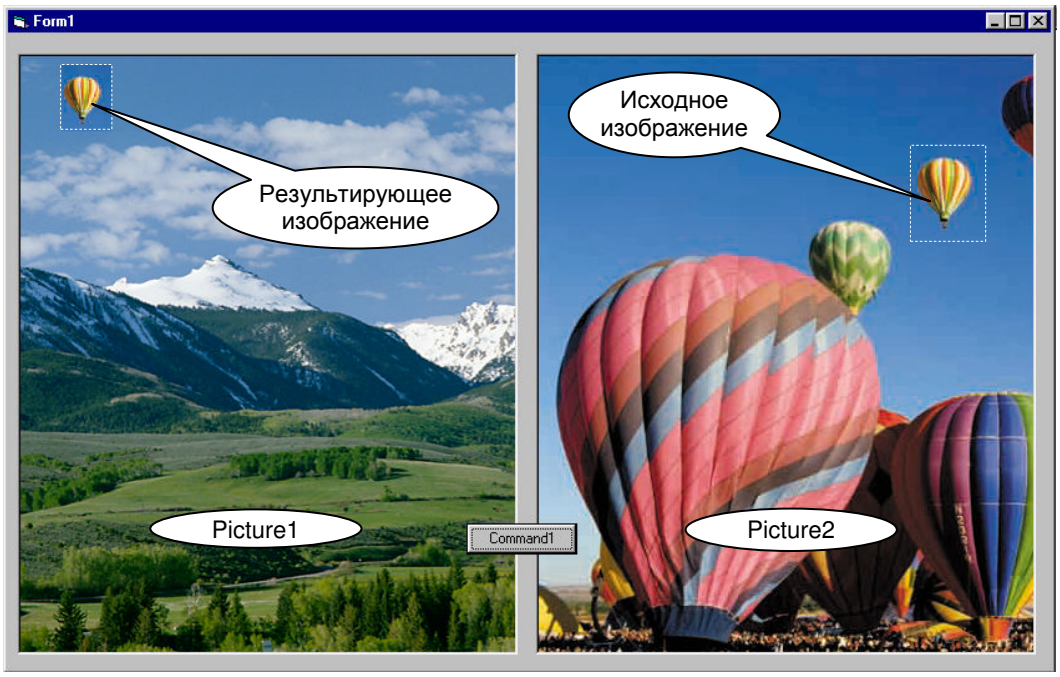
- В этом цвете красной краски больше, чем двух остальных.
- В этом цвете зеленой краски больше, чем двух остальных.
- В этом цвете синей краски больше, чем двух остальных.

Распознав цвет точки на форме, вы сделали первый шаг к решению великой и не решенной до сих пор человечеством задачи распознавания зрительных образов. Пожалуй, вы уже сейчас в силах написать программу, которая в большинстве случаев правильно отличит фотографию песчаной пустыни от фотографии океана. Но знаете ли вы, что не родился еще гений, способный написать программу, надежно отличающую хотя бы фото собаки от фото кошки? Потому что здесь дело не столько в цвете, сколько в форме. А это уже гораздо сложнее.

Решение задачи распознавания образов - ключ к осуществлению величайшей и самой дерзкой мечты ученых - созданию искусственного интеллекта, электронного разума, равного человеческому или превосходящего его.

Метод PaintPicture

Задача: В объекты Picture1 и Picture2 загружены картинки. Взять прямоугольный кусок из картинки в Picture2 и вставить, немного сжав, в определенное место картинки в Picture1.



Решение: Дело решает одна строка:

Picture1 **. PaintPicture** Picture2.Picture, 500, 100, 600, 800, 4500, 1000, 900, 1200

Словесное описание этой строки: В объект Picture1 вставить фрагмент из Picture2.Picture, причем левый верхний угол исходного изображения имеет координаты (4500, 1000), его ширина - 900, высота - 1200. Левый верхний угол результирующего изображения имеет координаты (500, 100), его ширина - 600, высота - 800.

Объектом может служить форма и PictureBox.

9.8. Используем в рисовании переменные величины

Если вы нарисовали снеговика, то наверное согласитесь, что для этого вам пришлось основательно потрудиться, хотя сам рисунок получился не слишком богатый, в нем всего-то порядка десяти элементов.

Как заставить Visual Basic короткой программой рисовать множество элементов, сплетая их в красивые узоры? Ответ: применять циклы, используя в обращениях к графическим методам вместо чисел переменные величины и арифметические выражения.

Задача: Нарисовать горизонтальный ряд окружностей радиусом 100 на расстоянии 1000 от верхнего края экрана и с такими горизонтальными координатами 500, 800, 1100, 1400, **...** , 2900.

Как видим, центры соседних окружностей отстоят друг от друга на 300. Вот примитивный фрагмент, решающий эту задачу:

```
Private Sub Command1_Click()
    Circle (500, 1000), 100
    Circle (800, 1000), 100
    Circle (1100, 1000), 100
    Circle (1400, 1000), 100
    Circle (1700, 1000), 100
    Circle (2000, 1000), 100
    Circle (2300, 1000), 100
    Circle (2600, 1000), 100
    Circle (2900, 1000), 100
End Sub
```

При вводе этой программы вас будет раздражать необходимость вводить много раз почти одно и то же. Воспользуйтесь копированием, которое объяснено в Приложении2.

Мы видим, что здесь Visual Basic 9 раз выполняет один и тот же метод, причем при каждом следующем обращении первый параметр вырастает на 300.

Придумаем для первого параметра переменную величину, например, x. Немного изменим программу:

```
Private Sub Command2_Click()
    x = 500
    Circle (x, 1000), 100:   x = x + 300
    Circle (x, 1000), 100:   x = x + 300
    Circle (x, 1000), 100:   x = x + 300
    Circle (x, 1000), 100:   x = x + 300
    Circle (x, 1000), 100:   x = x + 300
    Circle (x, 1000), 100:   x = x + 300
    Circle (x, 1000), 100:   x = x + 300
    Circle (x, 1000), 100:   x = x + 300
```

```
    Circle (x, 1000), 100:    x = x + 300
End Sub
```

Здесь последний оператор x = x + 300 я написал только для красоты, от него нет никакой пользы, хотя и вреда тоже особого нет.

Эта программа рисует абсолютно то же самое, что и предыдущая, но она проще нее, так как не пришлось самим вычислять координаты.

Что мы видим? Мы видим, что программа состоит из нескольких одинаковых фрагментов. Это прямое приглашение применить цикл:

```
Dim x As Long
Private Sub Command3_Click()
    x = 500
    Do Until x > 2900
        Circle (x, 1000), 100
        x = x + 300
    Loop
End Sub
```

Эта программа тоже рисует абсолютно то же самое, что и две предыдущие, но она короче. Здесь я перестраховался и объявил переменную x, как целую. Иначе при многократном прибавлении 300 могло бы оказаться, что результат равен не 2900, а, скажем, 2900.0000067 (Такой же случай я рассматривал в 6.4). А это значит, что последняя окружность не была бы нарисована. Можно было бы перестраховаться по-другому: вместо Do Until x > 2900 написать Do Until x > 2901.

Задание 63:. Попробуйте уменьшить расстояние между центрами окружностей, не изменяя их радиуса, нарисовав их плотнее, чтобы они пересекались, еще плотнее, пока они не образуют *“трубу”*.

Задание 64:. Удлините трубу налево и направо до краев формы.

Задание 65:. Увеличьте толщину трубы.

Заставим окружности вести себя посложнее. Например, расположим их не по горизонтали, а по диагонали формы в направлении от левого верхнего угла в правый нижний. Для этого организуем еще одну переменную - вертикальную координату y - и заставим ее тоже изменяться одновременно с x.

```
Private Sub Command4_Click()
    x = 500
    y = 200
    Do Until x > 2900
        Circle (x, y), 100
        x = x + 300
        y = y + 200
    Loop
End Sub
```

Если мы захотим менять радиус, то организуем переменную R.

Задание 66: *"Очередь трассирующими"*. Нарисуйте ряд точек по направлению из левого нижнего угла в правый верхний.

Задание 67: *“Круги на воде или радиоволны”*. Нарисуйте пару десятков концентрических окружностей, то есть окружностей разного радиуса, но имеющих общий центр.

Задание 68: *“Компакт-диск”* и "*Летающая тарелка*". Если радиус самого маленького “круга на воде” будет порядка 500, а самого большого - во весь экран, и если радиусы соседних окружностей будут различаться на 10-30 твипов, то на экране вы увидите привлекательный “компакт-диск”. Сделайте его золотым (Yellow). Если получилось, то сделайте ему внутренний и наружный ободки другого цвета. А теперь "положите" диск, то есть нарисуйте его не из окружностей, а из эллипсов, сжатых по вертикали. Получится "летающая тарелка".

Задание 69: Не трогая x, а меняя только y и R, вы получите *"коническую башню"*.

Задание 70: Меняя все три параметра, вы получите трубу, уходящую в бесконечность.

Задание 71: Разлинуйте экран в линейку.

Задание 72: А теперь в клетку.

Задание 73: А теперь в косую линейку.

Задание 74: Начертите ряд квадратов.

Чтобы получить интересные и сложные рисунки, нужно использовать богатые возможности Visual Basic: вложенные циклы, ветвление внутри цикла и т.д., например:

Задание 75: Нарисуйте шахматную доску. Помощь: Здесь основные трудности возникнут при раскраске клеток в шахматном порядке. У Волчёнкова [См. список литературы] я встретил такую идею, касающуюся того, какие клетки закрашивать, а какие нет: Те клетки, у которых сумма номеров строки и столбца четная - закрашивать, остальные - нет.

Задание 76: *"Ковер"* или *"Кольчуга"*. В задании 62 вы рисовали горизонтальный ряд пересекающихся окружностей. Теперь нарисуйте один под другим много таких рядов.

Указания: Здесь вам понадобятся вложенные циклы. Если центры соседних окружностей отстоят друг от друга на одинаковое расстояние как по горизонтали, так и по вертикали, и если удачно подобраны остальные числа, то у вас получится красивый ковер во весь экран с аккуратными краями.

Задание 77: Пусть у этого ковра будет вырезан левый нижний угол.

Задание 78: … и вдобавок вырезан квадрат посередине.

9.9. Использование случайных величин при рисовании

Со случайными числами и функцией Rnd мы с вами познакомились в 5.4. Попробуем нарисовать "звездное небо". Для этого достаточно покрасить форму в черный или синий цвет и в случайных местах формы нарисовать некоторое количество разноцветных точек (скажем, 1000). Точка ставится методом PSet. Как сделать координаты и цвет точки случайными? Тот же Rnd. Ваша форма имеет размеры Width×Height, количество цветов равно 16777216, поэтому обращение к методу рисования одной точки случайного цвета будет выглядеть так:

PSet (Width * Rnd, Height * Rnd), 16777216 * Rnd

Этот оператор нужно просто выполнить 1000 раз:

```
For i = 1 To 1000
    PSet (Width * Rnd, Height * Rnd), 16777216 * Rnd
Next
```

Результат будет ярче, если точки будут иметь случайную толщину (1 или 2).

Имейте в виду, что сколько бы раз вы не запускали программу с указанным фрагментом, картина созвездий на экране будет абсолютно одинакова. Если вам нужно, чтобы от запуска к запуску набор значений случайной величины менялся (а значит и созвездия), употребите разик до использования функции Rnd процедуру Randomize.

Задание 79: *"Звезды в окне"*. Звездное небо в пределах прямоугольника.

Задание 80: *"Дождь в луже"*. Заполните форму окружностями или эллипсами радиуса 200 в случайных местах.

Задание 81: *"Мыльные пузыри"*. То же самое случайных радиусов и цветов.

Задание 82: *"Сноп света в глаза"*. То есть пучок лучей, выходящих из одной точки. Реализуется множеством случайных разноцветных отрезков прямых, причем одна точка всех отрезков не случайна, а находится в центре формы.

Задание 83: *"Стог сена"*. Множество случайных разноцветных отрезков прямых преимущественно желтоватых оттенков, причем одна точка любого отрезка находится в случайной точке левой трети стога, другая - в случайной точке правой. Размер стога - 6000 на 6000. Используйте функцию RGB со случайными аргументами.

Задание 84: *"Атака абстракциониста"*. На экране бесконечно рисуется большое количество случайных разноцветных залитых прямоугольников или эллипсов. Совет: Если ваш компьютер быстрый, то прямоугольники или эллипсы будут сменять друг друга с огромной скоростью, что может вам не понравиться. Чтобы замедлить работу компьютера, обычно используют таймер. Но поскольку вы с ним не знакомы, вставьте для замедления внутрь цикла оператор, который, ничего не изменяя на экране, будет выполняться достаточно долго. Обычно для этих целей используют "*пустой цикл*":

For j = 1 To 1000000 : Next

Пока компьютер досчитает до миллиона, пройдет некая значительная доля секунды.

Задание 85: *"Летающие тарелки в космосе"*. Они получатся, если по нажатии одной кнопки вы будете добавлять на форму очередную порцию звезд, а по нажатии другой - очередную летающую тарелку из задачи 68, но случайного размера и в случайном месте, что нелегко. Подсказка без пояснений: При рисовании тарелки не используйте Rnd внутри цикла, все случайные значения присвойте переменным выше цикла.

Глава 10. Процедуры

До сих пор мы с вами имели дело только с процедурами, задающими реакцию компьютера на те или иные события. Если вспомнить пример программы из 1.1, то это процедуры типа "Что делать, если …". Если вы забыли, что это такое, обязательно перечитайте 1.1, так как настала пора познакомить вас с другими процедурами - процедурами типа "Как". Будем называть их процедурами пользователя. В этой главе я на одном примере проведу вас от процедур пользователя к процедурам с параметрами.

10.1. Зачем нужны процедуры пользователя

Задача: Вы записали на диск десяток фотографий, снятых во время каникул, и решили сделать фотоальбом. Поместили на форму десяток кнопок с названиями фотографий и один PictureBox. При нажатии на кнопку нужное фото для всеобщего обозрения загружается в PictureBox, а также печатается дата снимка.

Задача легкая, мы ее уже решали в 9.3. Вот программа для 4 фотографий:

```
Private Sub Command1_Click()
    Picture1.Picture = LoadPicture("c:\temp\Rockies.bmp")
    Picture1.Print , "21.07.2001"
End Sub

Private Sub Command2_Click()
    Picture1.Picture = LoadPicture("c:\temp\Porthole.bmp")
    Picture1.Print , "28.07.2001"
End Sub

Private Sub Command3_Click()
    Picture1.Picture = LoadPicture("c:\temp\Balloons.bmp")
    Picture1.Print , "12.08.2001"
End Sub

Private Sub Command4_Click()
    Picture1.Picture = LoadPicture("c:\temp\Guitar.bmp")
    Picture1.Print , "20.07.2001"
End Sub
```

Хочу предупредить, что эта программа, постепенно усложняясь, пройдет через всю главу, поэтому разберитесь в ней как следует, иначе важный материал главы будет вам непонятен.

Усложним задачу: Некоторые из фотографий сняли вы, некоторые - ваш друг. Вы хотите, чтобы в знак этого на ваших фото в левом верхнем углу появлялся один значок, а на фото вашего друга другой. Ваш любимый значок такой:

а у вашего друга такой:

Пусть они и появляются.

Давайте значки будем рисовать при помощи методов. Ваша программа станет такой:

```
Private Sub Command1_Click()
   Picture1.Picture = LoadPicture("c:\temp\Rockies.bmp")
      'Это ваше фото, рисуем значок:
   Picture1.Line (100, 100)-(300, 300), vbBlue, B
   Picture1.Line (100, 100)-(300, 300), vbBlue
   Picture1.Line (100, 300)-(300, 100), vbBlue
   Picture1.Print , "21.07.2001"
End Sub

Private Sub Command2_Click()
   Picture1.Picture = LoadPicture("c:\temp\Porthole.bmp")
      'Это ваше фото, рисуем значок:
   Picture1.Line (100, 100)-(300, 300), vbBlue, B
   Picture1.Line (100, 100)-(300, 300), vbBlue
   Picture1.Line (100, 300)-(300, 100), vbBlue
   Picture1.Print , "28.07.2001"
End Sub

Private Sub Command3_Click()
   Picture1.Picture = LoadPicture("c:\temp\Balloons.bmp")
```

```
        'Это фото вашего друга, рисуем значок:
    Picture1.Circle (200, 200), 20
    Picture1.Circle (200, 200), 70
    Picture1.Circle (200, 200), 120
    Picture1.Circle (200, 200), 170
    Picture1.Print , "12.08.2001"
End Sub

Private Sub Command4_Click()
    Picture1.Picture = LoadPicture("c:\temp\Guitar.bmp")
        'Это фото вашего друга, рисуем значок:
    Picture1.Circle (200, 200), 20
    Picture1.Circle (200, 200), 70
    Picture1.Circle (200, 200), 120
    Picture1.Circle (200, 200), 170
    Picture1.Print , "20.07.2001"
End Sub
```

Программа работает нормально, но невооруженным глазом виден ее существенный недостаток - резко возросший объем. Недостаток этот особенно будет заметен, когда число фото вырастет до десятка. Это тем более обидно, что в программе есть два повторяющихся фрагмента. Это фрагмент

```
        'Это ваше фото, рисуем значок:
    Picture1.Line (100, 100)-(300, 300), vbBlue, B
    Picture1.Line (100, 100)-(300, 300), vbBlue
    Picture1.Line (100, 300)-(300, 100), vbBlue
```

и фрагмент

```
        'Это фото вашего друга, рисуем значок:
    Picture1.Circle (200, 200), 20
    Picture1.Circle (200, 200), 70
    Picture1.Circle (200, 200), 120
    Picture1.Circle (200, 200), 170
```

которые в нашей программе встретились по два раза, а когда число фото вырастет, то встретятся многократно.

В этом случае программисты всего мира поступают так. Они придумывают повторяющимся фрагментам *имена*, например,

Рисуем_мой_значок

и

Рисуем_значок_друга

Затем они вписывают в программу специальную процедуру для каждого фрагмента с придуманным только-что именем, после чего имеют право во всей программе вместо фрагмента писать его имя. Чтобы вам было понятно, посмотрите на получившуюся программу:

```
Private Sub Рисуем_мой_значок()          'Это одна специальная процедура
    Picture1.Line (100, 100)-(300, 300), vbBlue, B
    Picture1.Line (100, 100)-(300, 300), vbBlue
    Picture1.Line (100, 300)-(300, 100), vbBlue
End Sub

Private Sub Рисуем_значок_друга()        'Это другая специальная процедура
    Picture1.Circle (200, 200), 20
    Picture1.Circle (200, 200), 70
    Picture1.Circle (200, 200), 120
    Picture1.Circle (200, 200), 170
End Sub
                                         'Это новый текст программы:
Private Sub Command1_Click()
    Picture1.Picture = LoadPicture("c:\temp\Rockies.bmp")
    Рисуем_мой_значок
    Picture1.Print , "21.07.2001"
End Sub

Private Sub Command2_Click()
    Picture1.Picture = LoadPicture("c:\temp\Porthole.bmp")
    Рисуем_мой_значок
    Picture1.Print , "28.07.2001"
End Sub

Private Sub Command3_Click()
    Picture1.Picture = LoadPicture("c:\temp\Balloons.bmp")
    Рисуем_значок_друга
    Picture1.Print , "12.08.2001"
End Sub

Private Sub Command4_Click()
    Picture1.Picture = LoadPicture("c:\temp\Guitar.bmp")
    Рисуем_значок_друга
```

```
    Picture1.Print , "20.07.2001"
End Sub
```

Пояснение того же самого другими словами: У нас добавилось две процедуры. Каждая из этих процедур представляет собой цепочку операторов, из которых состоит упомянутый фрагмент. Сверху цепочки вы пишете заголовок процедуры (Private Sub …), а снизу - конечную строчку процедуры (End Sub). Все операторы, из которых состоит процедура, кроме заголовка и конечной строки, будем называть **телом процедуры**. Как только эти процедуры написаны, компьютер "узнает", что такое Рисуем_мой_значок и Рисуем_значок_друга. Поэтому в остальных процедурах слова Рисуем_мой_значок и Рисуем_значок_друга используются, как настоящие операторы, и выполняются, как настоящие операторы. Суть их выполнения в том, что когда Visual Basic во время выполнения программы натыкается на оператор Рисуем_мой_значок, он ищет в программе процедуру с именем Рисуем_мой_значок и выполняет тело этой процедуры, после чего возвращается к выполнению программы. Этот процесс называется **вызовом** процедуры или **обращением** к процедуре.

Обязательно выполните эту программу в пошаговом режиме! Обязательно обратите внимание, что после выполнения тела вызываемой процедуры (Рисуем_мой_значок) компьютер возвращается в вызывающую процедуру (Command1_Click) к выполнению оператора, следующего за оператором Рисуем_мой_значок (в нашем случае это оператор Print).

Две дописанные нами процедуры называются **процедурами пользователя**, в отличие от привычных нам **процедур обработки событий**. Их коренное отличие от последних в том, что вызываются они не наступлением каких-то событий, а упоминанием их имени в других процедурах. Процедуры пользователя и являются "процедурами типа "Как"".

Вы можете сами посчитать, насколько новая программа будет короче старой при десяти фотографиях.

Есть еще один способ обратиться к процедуре. Вместо оператора

Рисуем_мой_значок

можно написать оператор

Call Рисуем_мой_значок

Смысл их совершенно одинаков. Вторым способом часто пользовались раньше. С английского слово "Call" переводится "Вызов".

Взаимодействие процедур в программе

Вызываемая процедура сама в процессе своей работы может вызвать какую-нибудь другую процедуру. И так далее. Потренируемся:

Определите без компьютера, что напечатает программа:

```
Private Sub Command1_Click()
   Print 1; :   A :   Print 2; :   B :   Print 3;
End Sub

Private Sub A()
   Print 4;
End Sub

Private Sub B()
   Print 5; :   C :   Print 6;
End Sub

Private Sub C()
   Print 7;
End Sub
```

Работать программа начинает по щелчку по кнопке Command1. Вряд ли вам с непривычки удасться дать правильный ответ. Тогда непременно программу - в компьютер и пошаговый режим. Желтая полоска будет скакать по программе туда-сюда. Перед каждым нажатием на F8 вы обязаны предсказать, куда она прыгнет! Не сможете - нет смысла читать книгу дальше.

Ответ:

```
1 4 2 5 7 6 3
```

Начинающим программистам не хочется писать процедуры пользователя, как не хочется им писать длинные имена и соблюдать отступы от левого края окна кода. "Ну и лопухи же эти профессиональные программисты, что осложняют себе жизнь этой морокой!" - думают они - "Наши программы отлично работают и безо всего этого". Верно, работают. Потому что программы коротенькие. Когда они станут длинными и напоминающими винегрет, все такие подрастающие программисты дружно зарыдают: "Мамочка, почему ты нас в детстве не научила слушаться взрослых?!"

Запомните еще одно хорошее правило: Размеры любой процедуры не должны превышать одного экрана монитора. Если превышают, то даже если в ней нет повторяющихся фрагментов, все равно разбейте ее по смыслу на два-три фрагмента и каждый сделайте процедурой. Ваша программа будет гораздо легче читаться.

Задание 86: Дополните ваш фотоальбом процедурой пользователя, которая перед показом очередной фотографии воспроизводит один и тот же музыкальный звук, например, "c:\Windows\Media\Chimes.wav".

10.2. Операторы Stop, End и Exit Sub

До сих пор мы писали процедуры, которые выполняли свою работу до конца и заканчивали ее только на операторе End

Sub, не раньше. Существуют ли операторы, которые подобно операторам выхода из цикла Exit Do и Exit For заставляют компьютер покинуть процедуру, не доходя до ее конца? Такие операторы существуют.

Оператор **End** заставляет Visual Basic завершить работу не только процедуры, а всего проекта, не доходя до конечного End Sub. Пример: программа

```
Private Sub Command1_Click()
    Print 1;:    Print 2;:    End:    Print 3;
End Sub
```

напечатает 1 2. Правда, заметить это вы успеете только в пошаговом режиме, так как End завершает режим работы и проект мгновенно переходит в режим проектирования.

Ненавистник пошагового режима мог бы мечтать: "Хорошо бы существовал специальный оператор паузы, чтобы наткнувшись на него, компьютер приостанавливал выполнение программы, а мы могли бы спокойно посмотреть на результаты и подумать". Такой оператор есть, это **Stop**. Наткнувшись на него, компьютер переходит в режим прерывания и делает паузу до тех пор, пока вы снова не щелкните на кнопке Start. Тогда он продолжает работу с того места, где остановился. Вот вариант программы, при котором вы успеваете разглядеть результат:

```
Private Sub Command1_Click()
    Print 1;:    Print 2;:    Stop:    End:    Print 3;
End Sub
```

Еще пример: программа

```
Private Sub Command1_Click()
    Print 2;:    A:    Print 3;:    End:    Print 4;
End Sub

Private Sub A()
    Print 6;:    End:    Print 7;
End Sub
```

напечатает 2 6.

Оператор **Exit Sub** не такой решительный, как End. Он не выбрасывает Visual Basic из режима работы, а просто заставляет компьютер выйти из процедуры, в которой он выполнился. Если он выполнился в вызываемой процедуре, то Visual Basic возвращается в процедуру, ее вызвавшую. Если он выполнился в процедуре обработки события, то Visual Basic просто завершает работу этой процедуры.

Пример: Заменим в предыдущей программе оба End на Exit Sub:

```
Private Sub Command1_Click()
    Print 2;:    A:    Print 3;:    Exit Sub:    Print 4;
End Sub
Private Sub A()
    Print 6;:    Exit Sub:    Print 7;
End Sub
```

Эта программа напечатает 2 6 3.

Задание 87: Вот вам программа с процедурами. Вам нужно, не запуская ее, записать на бумажке весь разговор, который ведут герои “Трех мушкетеров”.

```
Private Sub Command1_Click()
  Print "Я, король Франции, спрашиваю вас - кто вы такие? Вот ты - кто такой?"
  ATOS
  Print "А ты, толстяк, кто такой?"
  PORTOS
  Print "А ты что отмалчиваешься, усатый?"
  DARTANIAN
  Print "Анна! Иди-ка сюда!!!"
  Exit Sub
  Print "Аудиенция закончена, прощайте!"
End Sub

Private Sub ATOS()
  Print "Я - Атос"
End Sub

Private Sub ARAMIS()
  Print "Это так же верно, как то, что я -Арамис!"
End Sub

Private Sub PORTOS()
  Print "А я Портос! Я правильно говорю, Арамис?"
  ARAMIS
  Print "Он не врет, ваше величество! Я Портос, а он Арамис."
End Sub

Private Sub DARTANIAN()
  Print "А я все думаю, ваше величество - куда девались подвески королевы?"
  Exit Sub
```

```
    Print "Интересно, что ответит король?"
    PORTOS
End Sub
```

Сверьте с ответом. Если не сходится, запустите программу в пошаговом режиме.

Теперь вы достаточно знаете о процедурах, чтобы они стали для вас удобными кирпичиками для постройки программ. Более мощным средством являются процедуры с параметрами, о которых вы сейчас узнаете.

Задание 88: В программе для задания 60 из 9.3 о продавце автомобилей есть два повторяющихся фрагмента. Сам бог велел сделать их процедурами.

Задание 89: Аналогичную вещь можно проделать для калькулятора из 5.9

10.3. Переменные вместо чисел

Наша цель - процедуры с параметрами. Данный раздел - подготовка к взятию этой крепости.

Вернемся к задаче из 10.1 о фотоальбоме со значками. Вспомним, что мы написали процедуру пользователя для рисования значка. Вот она:

```
Private Sub Рисуем_мой_значок()
    Picture1.Line (100, 100)-(300, 300), vbBlue, B
    Picture1.Line (100, 100)-(300, 300), vbBlue
    Picture1.Line (100, 300)-(300, 100), vbBlue
End Sub
```

Значок рисуется в левом верхнем углу фото с отступом в 100 твипов как от левого края фото, так и от верхнего. Предположим, вам разонравился такой отступ и вы решили сделать его поменьше, скажем, 50. Вам нужно в программе в 6 местах поменять число 100 на 50. Вот то-то и неудобно, что в 6, а не в одном. Слишком много труда. В нашей программе это, конечно, пустяк, а вот в больших и сложных программах одна и та же величина может встречаться сотни раз, и чтобы ее изменить, придется вносить сотни исправлений.

Посмотрим, как нам помогут переменные величины. Придумаем переменную величину с именем Otstup. Теперь напишем вариант той же процедуры, но с использованием переменной величины:

```
Dim Otstup As Integer
Private Sub Рисуем_мой_значок()
    Otstup = 100
    Picture1.Line (Otstup, Otstup)-(300, 300), vbBlue, B
    Picture1.Line (Otstup, Otstup)-(300, 300), vbBlue
    Picture1.Line (Otstup, 300)-(300, Otstup), vbBlue
End Sub
```

Теперь для того, чтобы изменить отступ, достаточно заменить число только в одном месте.

Вторая причина, по которой мы используем переменные, та, что с ними программа становится понятнее, так как имена переменным мы придумываем, исходя из их смысла.

Будем дальше улучшать нашу процедуру. Теперь вы легко можете управлять отступом, но вот беда - правый нижний угол значка остается всегда в одном и том же месте и поэтому размер значка уменьшается с увеличением отступа, а при отступе = 300 значок вообще превращается в точку. Вам хочется так же легко и удобно управлять размером значка, как и отступом. Вы замечаете, что размер квадратного значка равен разнице координат в скобках операторов Line. Вы придумываете переменную с именем Razmer и переписываете программу:

```
Dim Otstup As Integer
Dim Razmer As Integer

Private Sub Рисуем_мой_значок()
    Otstup = 100
    Razmer = 200
    Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), vbBlue, B
    Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), vbBlue
    Picture1.Line (Otstup, Otstup + Razmer)-(Otstup + Razmer, Otstup), vbBlue
End Sub
```

Последнее, что вам хочется, это управлять цветом. Нет проблем:

```
Dim Otstup As Integer
Dim Razmer As Integer
Dim Tsvet As Long

Private Sub Рисуем_мой_значок()
    Otstup = 100
    Razmer = 200
    Tsvet = vbBlue
    Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), Tsvet, B
    Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), Tsvet
    Picture1.Line (Otstup, Otstup + Razmer)-(Otstup + Razmer, Otstup), Tsvet
```

```
End Sub
```

Заметьте, что Tsvet я объявил, как Long, а то 16 миллионов в Integer не уместятся.

Задание 90: Помогите вашему другу менять при помощи переменных отступ, размер и цвет его значка.

10.4. Константы

Константами называются те конкретные значения величин, которые мы видим в программе. Например, во фрагменте

```
a = 1 + 0.25
b = "Амазонка"
Debug.Print "Волга", 10
Form1.BackColor = 15767511
If a > 3 Then Cls
```

константы это 1 0.25 "Амазонка" "Волга" 10 15767511 3.

Если какая-нибудь константа встречается в программе несколько раз, то удобно, как я уже говорил в предыдущем разделе, придумать ей имя и обозначать этим именем.

Вернемся к примеру из предыдущего раздела. Там мы объявили переменную Otstup и везде в процедуре стали писать вместо константы 100 имя этой переменной. Программист имеет право вместо оператора объявления

Dim Otstup As Integer

написать другой оператор объявления:

Const Otstup **=** 100

Тогда процедура изменит свой вид:

```
Const Otstup = 100
Private Sub Рисуем_мой_значок()
   Picture1.Line (Otstup, Otstup)-(300, 300), vbBlue, B
   Picture1.Line (Otstup, Otstup)-(300, 300), vbBlue
   Picture1.Line (Otstup, 300)-(300, Otstup), vbBlue
End Sub
```

В чем разница? В первом случае величина Otstup - переменная величина. Во втором случае величина Otstup объявлена **константой**, а это значит, что ей **запрещено менять свое значение** 100, присвоенное ей при объявлении. При попытке изменить значение константы Visual Basic выдаст сообщение об ошибке. Например, ошибкой завершится выполнение такой программы:

```
Const a = 5
Private Sub Command3_Click()
   a = a + 2
End Sub
```

Хорошо это или плохо? Это плохо там, где величина по смыслу задачи должна менять свое значение, и хорошо там, где не должна. Например, если в программе

```
Const Пи = 3.14
Private Sub Command4_Click()
   r = 50
   Длина_окружн = 2 * Пи * r
   Площадь_круга = Пи * r ^ 2
End Sub
```

мы по небрежности где-нибудь напишем оператор типа Пи = 2.87, то Visual Basic выдаст сообщение об ошибке и мы будем этому рады, так как число Пи – очень известная константа, равная 3.14, и никто не имеет права менять ее значение.

Таким образом, при помощи объявления констант мы повышаем надежность программирования.

Обратите внимание, что термином "константа" мы обозначаем два близких понятия: с одной стороны число 100, строку "Волга" и т.п., а с другой стороны имена Otstup, Пи и т.п., обозначающие эти величины. Не думаю, что в будущем эта путаница принесет нам какой-нибудь вред.

Кроме перечисленных выше констант существует еще большое число так называемых **внутренних констант** Visual Basic. Объявлять их не надо, ими можно сразу пользоваться, если знать их имена и смысл. Например, такими константами являются названия цветов - vbRed, vbBlack и т.п. Каждая из таких констант имеет конкретное численное значение (например, vbRed=255), но названия запоминаются легче, чем числа. Имена, смысл и значение внутренних констант вы найдете в Object Browser.

10.5. Процедуры с параметрами

Процедуры с параметрами - мощный инструмент программирования и применяется программистами очень широко.

Вообразите, что вы решили фотографии разного с вашей точки зрения качества помечать значками разного отступа, размера и цвета, чтобы по значку легко было догадаться о качестве. Вот ваши предпочтения:

Фото неважное	Otstup = 100	Razmer = 200	Tsvet = vbBlue
Фото хорошее	Otstup = 200	Razmer = 400	Tsvet = RGB(100,250,150)

Но как это сделать? Ведь в процедуре Рисуем_мой_значок каждой из этих переменных присвоено одно единственное значение, поэтому, какую бы фотографию ни взять, значок всегда будет одинаковый. Можно было бы написать отдельную процедуру для каждого значка, но делать этого не хочется, потому что процедуры эти будут почти одинаковы. А тогда где экономия?

Выход есть. Заметим, что присваивать значения переменным не обязательно внутри процедуры Рисуем_мой_значок. Вот вариант программы, который будет работать:

```
Dim Otstup As Integer
Dim Razmer As Integer
Dim Tsvet As Long

Private Sub Рисуем_мой_значок()
   Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), Tsvet, B
   Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), Tsvet
   Picture1.Line (Otstup, Otstup + Razmer)-(Otstup + Razmer, Otstup), Tsvet
End Sub

Private Sub Command1_Click()
   Picture1.Picture = LoadPicture("c:\temp\Rockies.bmp")
                 'Фото неважное
   Otstup = 100
   Razmer = 200
   Tsvet = vbBlue
   Рисуем_мой_значок
   Picture1.Print , "21.07.2001"
End Sub

Private Sub Command2_Click()
   Picture1.Picture = LoadPicture("c:\temp\Porthole.bmp")
                 'Фото хорошее
   Otstup = 200
   Razmer = 400
   Tsvet = RGB(100, 250, 150)
   Рисуем_мой_значок
   Picture1.Print , "28.07.2001"
End Sub
```

Как видите, в процедуре Рисуем_мой_значок значения переменным не присваиваются, а присваиваются в процедурах, которые ее вызывают, причем присваиваются непосредственно перед вызовом. Убедитесь в пошаговом режиме, что все работает нормально.

Нормально-то нормально, да вот опять наша программа начала разбухать - в каждой вызывающей процедуре добавилось по три лишних строчки. За красивую жизнь приходится платить. Но и тут создатели языков программирования идут навстречу программистам. Имеется возможность вместо

```
Otstup = 100
Razmer = 200
Tsvet = vbBlue
Рисуем_мой_значок
```

писать

```
Рисуем_мой_значок   100,   200,   vbBlue
```

Все так и делают. Значения, идущие после имени процедуры через запятую, называются **параметрами процедуры**. Но откуда компьютер знает, что, например, Otstup = 100 и Razmer = 200, а не наоборот - Otstup = 200 и Razmer = 100? Компьютеру это объясняют в заголовке процедуры Рисуем_мой_значок, который теперь пишется по-другому:

Private Sub Рисуем_мой_значок (Otstup As Integer, Razmer As Integer, Tsvet As Long)

Как видите, здесь задаются и порядок параметров и их тип.

Вот как компактно теперь выглядит наша программа. Это и есть ее окончательный вид, к которому мы стремились на протяжении этой главы:

```
Private Sub  Рисуем_мой_значок  (Otstup As Integer,   Razmer As Integer,   Tsvet As Long)
     Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), Tsvet, B
     Picture1.Line (Otstup, Otstup)-(Otstup + Razmer, Otstup + Razmer), Tsvet
     Picture1.Line (Otstup, Otstup + Razmer)-(Otstup + Razmer, Otstup), Tsvet
End Sub

Private Sub Command1_Click()
     Picture1.Picture = LoadPicture("c:\temp\Rockies.bmp")
     Рисуем_мой_значок   100,   200,   vbBlue
     Picture1.Print , "21.07.2001"
End Sub

Private Sub Command2_Click()
     Picture1.Picture = LoadPicture("c:\temp\Porthole.bmp")
     Рисуем_мой_значок   200,   400,   RGB(100, 250, 150)
     Picture1.Print , "28.07.2001"
End Sub
```

Обратите внимание, что мы убрали за ненадобностью фрагмент объявлений:

```
Dim Otstup As Integer
Dim Razmer As Integer
```

```
Dim Tsvet As Long
```

так как эти переменные уже объявлены в заголовке процедуры.

Наткнувшись в процессе выполнения программы на обращение к процедуре (Рисуем_мой_значок 100, 200, vbBlue), Visual Basic присваивает параметрам, приведенным в заголовке процедуры (Otstup, Razmer, Tsvet), указанные значения, а затем выполняет тело процедуры.

В качестве значений параметров в обращениях к процедурам можно писать не только константы, но и переменные, и выражения. Например, вместо

```
Рисуем_мой_значок 100, 200, vbBlue
```

можно было написать

```
a=100
Рисуем_мой_значок a, 2 * a, vbBlue
```

Типы параметров

Параметры могут иметь не только числовой, но и строковый и многие другие типы. Пример:

```
Private Sub Печатаем_3_раза(Что_нибудь As String)
    Print Что_нибудь
    Print Что_нибудь
    Print Что_нибудь
End Sub

Private Sub Command1_Click()
    Печатаем_3_раза    "Кто там?  -  Это почтальон Печкин!"
    Печатаем_3_раза    "Дядя Федор"
End Sub
```

Здесь вы видите процедуру пользователя Печатаем_3_раза и ее параметр - строковую переменную с именем Что_нибудь. При нажатии на кнопку Command1 программа начинает работать и печатает следующий текст:

```
Кто там?  -  Это почтальон Печкин!
Кто там?  -  Это почтальон Печкин!
Кто там?  -  Это почтальон Печкин!
Дядя Федор
Дядя Федор
Дядя Федор
```

Вот другой пример. Для начала отметим, что величины, которые указываются в скобках *функций* (например, в Round (5.82716, 3) или LoadPicture("c:\temp\Rockies.bmp")), тоже называются **параметрами**, хотя их называют также **аргументами**. Здесь в функции LoadPicture адрес файла взят в кавычки, значит этот параметр является строкой. Можно написать такую программу, которая по нажатии на кнопку Command1 показывает сразу два фото:

```
Private Sub Показываем_два_фото  (Фото_1 As String,   Фото_2 As String)
    Picture1.Picture = LoadPicture (Фото_1)
    Picture2.Picture = LoadPicture (Фото_2)
End Sub

Private Sub Command1_Click()
    Показываем_два_фото    "c:\temp\Rockies.bmp" ,    "c:\temp\Porthole.bmp"
End Sub
```

Теперь мы понимаем, зачем в конце заголовка процедуры ставится пара скобок (). Это для параметров, буде они объявятся.

Дальше о процедурах вы прочтете в Глава 17.

Задание 91: В задании 86 из 10.1 вы написали процедуру пользователя, которая перед показом очередной фотографии воспроизводит один и тот же музыкальный звук. Пусть теперь перед каждой фотографией будет свой звук. Для этого напишите процедуру с параметром.

Задание 92: В задании 90 вы написали процедуру пользователя, которая рисовала значок вашего друга. Перепишите ее, сделав процедурой с тремя параметрами: отступ, размер и цвет значка.

Задание 93: Среди графических методов Visual Basic нет методов "крестик" и "треугольник". Вы можете возместить этот недостаток, написав две соответствующие процедуры с тремя параметрами: координата_x, координата_y, размер.

Задание 94: Представьте себе куб, собранный из 16777216 кубиков - по числу цветов в Visual Basic. Его высота - 256 кубиков, ширина и толщина - тоже по 256 кубиков. Каждый кубик покрашен в свой цвет. Цвета не повторяются. Систему раскраски придумать легко. Например, слева направо растет от 0 до 255 красная составляющая в цвете кубиков, сверху вниз - зеленая, от нас вдаль - синяя. Так что самый левый верхний ближний кубик получается абсолютно черным, а самый правый нижний дальний кубик - абсолютно белым. Сразу все кубики видеть мы, конечно, не можем, но мы можем делать срез куба в любом месте параллельно любой из его граней, в результате чего на срезе будем видеть квадрат, состоящий из 256*256 разноцветных квадратиков. Вот эту задачу среза я бы и хотел вам предложить. Программа предлагает пользователю выбрать один из трех основных цветов (это удобно сделать через меню) и его насыщенность (число от 0 до 255). Этим определяется место среза. Затем программа чертит на форме этот разноцветный срез. Конечно, квадратики получатся очень маленькими, но это ничего.

Указание: Используйте процедуру с двумя параметрами: выбранный пользователем цвет (один из трех) и его насыщенность.

Кстати, догадайтесь, из каких цветов составлена главная диагональ куба, проведенная между двумя упомянутыми мной кубиками.

Глава 11. Работа с таймером, временем, датами

Мы с вами пока не умеем управлять временем. А это нам необходимо для работы с анимацией, а также для решения полезных и интересных задач, приведенных в этой главе. Собственно, все игры с одновременным движением персонажей делаются с помощью таймера. Венец главы - создание вашего первого реального проекта "Будильник-секундомер".

11.1. Тип данных Date

Все вы видели часы в правой части панели задач Windows. Если на них поставить мышь, они покажут дату. Давайте сделаем что-нибудь получше, а именно - большие, красивые часы-будильник, а заодно и секундомер. Для этого нам нужно познакомиться с новым типом данных - типом даты и времени суток - **Date**. Вы пока знакомы с числовыми и строковым типами. Тип Date тоже, в принципе, числовой, но, сами понимаете, специфический. Если, например, в нем к 0:40 прибавить 0:40, то получится 1:20.

Когда вы пишете в окне кода программу, в ней встречаются числа, строки, а теперь вы должны научиться писать в программе дату и время суток. Чтобы Visual Basic понял, что перед ним число, вы просто пишете число, и он понимает. Чтобы Visual Basic понял, что перед ним строка, вы пишете строку и берете ее в двойные кавычки, и он понимает, что это строка. Чтобы Visual Basic понял, что перед ним дата или время суток, вы правильно записываете дату и время и заключаете их между значками **#**, и он понимает. Например, так: #2/16/2002#. Это 16 февраля 2002 года. Как правильно записывать дату и время в других случаях, вы поймете из примеров:

```
Dim D As Date
Dim T As Date
Dim DT As Date
Private Sub Command1_Click()
    Debug.Print  #6/25/2001#                  '25 июня 2001 года
    Debug.Print  #2:22:57 PM#                 '2 часа 22 минуты 57 секунд после полудня (PM)
    Debug.Print  #2/28/1998  10:45:00 PM#     '10 часов 45 минут вечера 28 февраля 1998 года
    D =  #12/25/2044#
    T =  #2:00:32 AM#                         '2 часа 00 минут 32 секунды до полудня (AM)
    DT =  #1/15/2156  11:59:42 PM#
    Debug.Print  D, T, DT
End Sub
```

Эта процедура напечатает такие результаты:

```
25.06.01
14:22:57
28.02.98 22:45:00
25.12.2044   2:00:32      15.01.2156 23:59:42
```

Пояснения: Как видите, в окне кода мы обязаны писать дату и время по-американски, то есть месяц писать раньше числа и разделять все это косыми чертами, а в обозначении времени суток обязательно указывать до или после полудня было дело. А вот результаты по этим ненашим данным печатаются все равно по-нашему, вернее, так, как настроена Windows (а у большинства она настроена на Россию). Поэтому же, если вы захотите задавать дату или время компьютеру таким оператором:

D = InputBox("Введите дату")

то вводить ее по-американски нельзя и значки # тоже нельзя ставить.

Есть и другие способы задания дат и времени, но они сложнее и я их пропущу. Скажу только, что если присвоить переменной типа Date обычное число, то оно будет преобразовано в дату и время. Так, фрагмент

```
D = 26.5
Debug.Print  D
```

напечатает следующее:

```
25.01.1900 12:00:00
```

Пояснения: Число 26.5 считается количеством суток (двадцать шесть с половиной), прошедших с полуночи 30 декабря 1899 года.

Учитывая вышесказанное, вы можете наладить сложение и вычитание дат и времени. Однако, лучше это делать с помощью специальных функций, которые мы сейчас и рассмотрим.

Функции для работы с датами и временем суток

Функция	Результат
Debug.Print **Date**	*Печатается сегодняшнее число (то, что на панели задач Windows)*
Debug.Print **Time**	*Печатается сколько сейчас времени*
Debug.Print **Now**	*Печатается сегодняшнее число и сколько сейчас времени*

Пусть D = # 2/14/2009 4:45:07 *PM # (это суббота), тогда:*

Debug.Print **DatePart**("m", D)	2 *(функция выделяет из даты номер месяца в году)*
Debug.Print **DatePart**("q", D)	1 *(функция выделяет из даты номер квартала в году)*
Debug.Print **DatePart**("s", D)	7 *(функция выделяет из даты номер секунды в минуте)*
Debug.Print **DatePart**("w", D)	7 *(функция выделяет из даты номер дня в неделе, но делает это по-американски, а в США первый день недели - воскресенье, значит 7-й - суббота)*
Debug.Print **DatePart**("w", D, vbMonday))	6 *(а здесь мы попросили функцию считать первым днем недели понедельник (*vbMonday*) и поэтому результат она выдала привычный для нас - 6-й день - суббота)*

Значением функции DatePart является число типа Integer, а не дата.

Все возможные значения строкового параметра для функций работы с датами приведены в последней таблице этого параграфа.

Debug.Print **DateAdd**("s", 10, D)	14.02.09 16:45:17 *(функция добавляет к дате 10 секунд)*
Debug.Print **DateAdd**("m", -1, D)	14.01.09 16:45:07 *(функция вычитает из даты 1 месяц)*

Пусть D1 = #2/14/2009 4:45:07 PM# , D2 = #2/16/2009 11:32:43 AM# *, тогда:*

Debug.Print **DateDiff**("h", D1, D2)	43 *(количество часов, прошедших с момента даты D1 до момента даты D2)*

Debug.Print **Timer**	53861,97 *(количество секунд, прошедших с полуночи. Не путайте функцию Timer с элементом управления Timer. Результат выдается, как видите, с двумя знаками после запятой, но не верьте сотым долям, десятые же более менее точны)*
Debug.Print **MonthName** (5)	Май
Debug.Print **WeekdayName** (4)	Четверг *(как видите, работает по-нашему, а не по американски)*

Опасные операторы:

Date = #05/22/2004#	*Это не безобидная* функция *Date, которая только сообщает, какое сегодня число, это* оператор *Date, который устанавливает системные часы Windows на ту дату, какую вы указали. Ничего особо страшного, конечно, от установки неправильной даты не будет, компьютер не сломается, но вот, например, сохраняемые вами файлы будут иметь неправильную дату сохранения, а это приведет к путанице в вашем файловом хозяйстве. Да и некоторые программы могут начать жаловаться. Отличить функцию от оператора с тем же именем просто - они встречаются по разные стороны знака равенства в операторе присваивания, причем функция всегда справа.*
Time = #10:44:00 AM#	*Аналогично*

Значения строкового параметра для функций работы с датами:

Строковый параметр	Смысл
yyyy	Год
q	Квартал в году
m	Номер месяца в году
y	Номер дня в году
d	Номер дня в месяце
w	Номер дня в неделе (№1 - воскресенье)
ww	Номер недели в году
h	Час в сутках
n	Минута в часе
s	Секунда в минуте

Есть еще кое-какие функции, но пока вам хватит и этих.

Задание 95: Напишите программу, которая, ничего у вас не спрашивая, печатает, какое число будет через 52 недели.

Задание 96: Напишите одну-две строчки кода, которые, спросив у вас дату рождения и не спрашивая, какое сегодня число, печатают, сколько секунд вы живете на белом свете

Задание 97: Напишите программу, которая, спросив у вас дату рождения и не спрашивая, какое сегодня число и был ли у вас в этом году день рождения, печатает, сколько дней вам осталось до следующего дня рождения.

Задание 98: Я знаю, что високосных годов раз в четыре года ученым не хватает. Поэтому, не то где-то раз в много лет вклинивается лишний високосный год, не то иногда где-то в каком-то месяце бывает лишний день. Не знаю. Может быть Visual Basic подскажет?

11.2. Таймер

Создайте новый проект. Поместите на форму таймер. Установите его свойство Interval равным 10000. Сделайте двойной щелчок по таймеру. В появившуюся заготовку процедуры впишите одну строчку:

```
Private Sub Timer1_Timer()
    Debug.Print "Процедура сработала"
End Sub
```

Запустите проект. Подождите немного. Через 10 секунд в окне Immediate появится строчка "Процедура сработала". Еще через 10 секунд появится такая же строчка, через 10 секунд еще одна, и так далее.

Если бы мы установили Interval равным 5000, то строчки появлялись бы каждые 5 с, а если равным 500, то - каждые 0.5 с.

Вывод: **Таймер** Timer1 - объект, вся работа которого заключается в том, чтобы через каждые Interval секунд создавать событие (импульс), которое запускает процедуру Timer1_Timer.

Таймеров в проекте может быть несколько. Все они работают независимо друг от друга и каждый имеет свою собственную процедуру, которая реагирует только на его импульсы.

Цикл без цикла

В процедуре Timer1_Timer мы можем написать все, что угодно. Например:

```
Dim i As Long
Private Sub Timer1_Timer()
    Debug.Print "Процедура сработала", i
    i = i + 1
End Sub
```

Установите Interval в 1000. Запустите проект. Вы увидите через каждую секунду возникающие строчки:

```
Процедура сработала    0
Процедура сработала    1
Процедура сработала    2
Процедура сработала    3
……................
```

Пояснение: Поскольку мы нигде не придали переменной i никакого значения, Visual Basic при первой встрече с ней в операторе Debug.Print посчитал ее равной 0. После первого выполнения процедуры Timer1_Timer переменная i стала равной 1, после второго - 2. И так далее.

Мы видим, что у нас заработал цикл, несмотря на то, что операторов цикла мы не писали! Чуть позже вы догадаетесь, как задавать начальные значения переменным цикла и вовремя делать выход из цикла.

Установим свойство таймера Enabled (по русски - "в рабочем состоянии") в False. Запустим проект. Никаких строк не появляется. Значит таймер выдает импульсы только тогда, когда свойство Enabled установлено в True.

Добавим в проект две кнопки. Допишем две процедуры:

```
Private Sub Command1_Click()
    Timer1.Enabled = True
End Sub

Private Sub Command2_Click()
    Timer1.Enabled = False
End Sub
```

Запустите проект. Вы увидите, что новые строчки начинают появляться только после щелчка по первой кнопке, а после щелчка по второй прекращают.

Что будет, если процедура Timer1_Timer еще не успела закончить работу, а от таймера уже пришел новый импульс? Проверим. Установим Interval равным 100, то есть импульсы от таймера должны посылаться примерно 10 раз в секунду. Напишем простую процедуру Timer1_Timer, которая очень долго работает:

```
Private Sub Timer1_Timer()
    Debug.Print "Процедура таймера начала работу"
    For i = 1 To 20000000 :  Next
    Debug.Print "Процедура таймера закончила работу"
End Sub
```

Здесь долго работает пустой цикл For i = 1 To 20000000 : Next. Несмотря на то, что делать ему нечего (тело у цикла отсутствует), у него много времени уходит на то, чтобы просто досчитать до 20 миллионов. На моем компьютере это занимает секунды две, на вашем, наверно, по-другому. Это неважно, а важно то, что за эти две секунды в процедуру врежется порядка 20 импульсов от таймера. И все как об стену горох - процедура на них не реагирует. Пока не завершит свою работу, как положено, оператором End Sub. После этого первый же следующий импульс, которому повезло, снова ее запускает. Вы увидите, что на экране не спеша, где-то раз в две секунды появляется очередная пара строчек.

Точность таймера невелика, а на интервалах меньше 1 секунды вообще никуда не годится. Не ждите, что если вы установили интервал равный 10, то в секунду будет послано 100 импульсов. Причина в том, что таймер вообще не способен выдавать больше 18 импульсов в секунду.

Задание 99: Запрограммируйте с помощью таймера печать чисел от 100 до 110 через 1 секунду.

11.3. Проект "Будильник-секундомер"

Вы узнали о Visual Basic вполне достаточно, чтобы рваться в бой. Наверняка у вас в голове зреет идея создать некий элегантный и одновременно вполне "убойный" проект строчек на 200, способный поразить в самое сердце ваших друзей и, что не менее важно, подруг. Однако, спешу вам сказать, что вы спешите. Ибо! Друзья и подруги может быть и понесут вас на руках до ближайшего кафе-мороженого, но человек, мало-мальски разбирающийся в программировании, бросив скучающий взгляд на ваше жалкое детище, спросит: "Юноша, в каком году вы кончали церковно-приходскую школу?" И не потому, что проект работает плохо, он вполне может проделывать на экране что-нибудь любопытное. Но программа! Программа! В ней невозможно разобраться! 200 строк вы осилили, а сколько вы возились? Неделю? А веди могли все сделать за пару дней! Сколько у вас процедур? 8? А нужно было 40! И так далее. Когда придет пора делать проект из 1000 строк, вы не сможете его одолеть никогда!

Что же делать? Все очень просто. Вы ни в чем не виноваты. Ведь я пока только *объявил* вам, что надо писать много маленьких процедур. Но ведь *не научил* еще, как это делать правильно. А учиться надо на примерах.

Пришла пора создать проект, "правильно" составленный из процедур. Проект "Калькулятор", написанный нами ранее, не подходит в качестве учебного пособия, потому что логика его работы слишком проста: нажал кнопку - выполнилась соответствующая процедура и все. Нужна более сложная задача. Для ее решения мы создадим совсем небольшую программу из 80 строк, включающую 15 процедур. Поскольку создаваемая программа представляет для многих из вас неведомую страну, я буду применять метод изложения "за ручку", который я уже применял в 1.3. Повторяйте за мной все, что я буду делать.

Начнем с постановки задачи.

Постановка задачи

Создать часы-будильник следующего вида:

В постановке задачи нужно с максимальной подробностью описать, что должен делать ваш проект с точки зрения пользователя (а не программиста!). Ввиду очевидности того, что изображено на картинке, я поясню только то, чего не видно или может быть неправильно понято:

- На верхнем циферблате - текущее время суток (системное время Windows) в часах, минутах и секундах
- Под ним - дата и день недели (02.08.00 означает 2 августа 2000 года)
- Пользователь имеет возможность редактировать содержимое циферблата установки времени будильника
- При нажатии на кнопку "Выключить будильник" надпись на кнопке меняется на "Включить будильник", а надпись над циферблатом меняется на "Будильник отключен"
- При срабатывании будильника раздается какая-нибудь продолжительная мелодия, которая замолкает при нажатии на кнопку "Выключить сигнал"

- Секундомер измеряет время с точностью до 0.1 сек. На картинке вы видите секундомер в момент паузы. Если нажать на ПУСК, то отсчет времени продолжится с 1 минуты 15.4 сек, которые вы видите на картинке, а надпись на кнопке сменится на ПАУЗА. Если снова нажать на кнопку, цифры на секундомере снова замрут.
- При нажатии на кнопку "Обнулить" секундомер сбрасывается в ноль и останавливается. На циферблате - *0:00:00.0*.

Делим проект на части

Нельзя делать сразу весь проект одновременно. То есть было бы фатальной ошибкой сразу все нарисовать и пытаться все сразу программировать. Не пытайтесь ломать весь веник сразу, ломайте по прутикам. Когда Наполеон видел, что его армия слишком мала, чтобы сразу одолеть противника, он весь удар сосредотачивал на одной какой-то части вражеской армии и, разбив ее, нападал на следующую.

Надо разделить проект на части. Это не всегда легко. Некоторые проекты, особенно игры, представляют на первый взгляд единое неразрывное целое, так что нужен некоторый опыт, чтобы увидеть, "из каких кубиков построен домик". В нашем случае все более-менее просто. Три части просматриваются сразу, это

- Часы (с датой и днем недели)
- Будильник
- Секундомер

Отлично! За какую из этих частей браться вначале? Для этого нужно сначала решить, зависят ли друг от друга отдельные части. Здесь очень пригодился бы некоторый опыт программирования. А если его нет, подойдет житейский опыт. Действительно, если бы вы мастерили будильник-секундомер из шестеренок, что бы в нем от чего зависело? Ну, ясно, что будильник не сработал бы без часов, ведь он должен чувствовать, когда время на циферблатах часов и будильника совпадает. А вот часы ходят и без будильника, они от него не зависят. Секундомер же, видимо, представляет собой полностью независимую часть со своим механизмом.

Итак, проект распался на две независимые части:

- Часы и будильник
- Секундомер

Какой частью заняться вначале? Дело вкуса. Часы с будильником попроще, поэтому начнем с них. Ну а между ними что выбрать вначале - часы или будильник? Здесь сомнения неуместны - *раньше нужно делать ту часть, которая от другой не зависит* - это часы.

Итак, мы разделили проект на части и определили порядок выполнения частей:

1. Часы
2. Будильник
3. Секундомер

Беремся за часы. И тут опять пошла дележка. Чем раньше заниматься - временем суток (будем называть это просто часами), датой или днем недели? Шестеренки смутно подсказывают нам, что дата переключается в полночь, а значит момент переключения зависит от часов. Значит дату будем делать позже часов. А день недели, ясно, определяется датой.

Итак, окончательная последовательность такая:

1. Часы (время суток)
2. Дата
3. День недели
4. Будильник
5. Секундомер

Лирическое отступление (утешение): Если вам не удалось разделить ваш проект на части, или вы чувствуете, что разделили неправильно, это не значит, что нужно от проекта отказываться. Программируйте напропалую! В процессе программирования отдельные части постепенно (не без мучений и многократных досадных переделок) встанут на свои места.

Ну что ж, "Задачи ясны, цели определены, за работу, товарищи!", как говорили при социализме.

Делаем часы

Создаем новый проект. Первым делом берем элемент управления Shape и рисуем вокруг будущего циферблата часов красивую рамочку. Если вы хотите сделать что-нибудь более выдающееся, чем скромная рамочка на картинке, и украсить часы необыкновенными шедеврами графики, то на здоровье, пожалуйста!. Но не сейчас. Потом. Когда все заработает. Иначе потонете в подробностях и не откопаете в золотых лепесточках нужную вам стальную шестеренку.

Я не дал объекту Shape1 никакого другого имени, потому что не собираюсь упоминать его в программе. Все свойства я ему установил в режиме проектирования и не хочу, чтобы рамочка как-то себя "вела" в режиме работы.

Внутрь рамочки я поместил текстовое поле для того, чтобы компьютер показывал в нем время. Для красоты я свойством BorderStyle убрал у него бордюрчик, а свойство Alignment настроил так, чтобы любой текст внутри поля располагался по центру. Настроил название, размер, стиль и цвет шрифта.

Дадим текстовому полю имя Циферблат_часов. Еще раз предупреждаю: не надо сокращенных имен типа Циф_час. Сэкономив сейчас копейку, вы потом проиграете рубль.

Пришла пора программировать. Прежде всего нужна идея, как сделать так, чтобы часы показывали текущее время. Мы знаем, что есть функция Time, значение которой в любой момент равно текущему времени. Стоит выполнить оператор

```
Циферблат_часов.Text = Time
```

и на часах оно будет. Попробуйте. Но как сделать, чтобы оно менялось? Нужно, чтобы этот оператор выполнялся не реже раза в секунду. Может быть, использовать оператор цикла? Но я должен вас предупредить, что на этом пути вас ждут разнообразные трудности, в суть которых не так-то легко вникнуть. Назову только одну. Вот, предположим, у вас заработал оператор цикла для часов. Тут вам захотелось включить секундомер. Значит, нужно, чтобы заработал другой оператор цикла - для секундомера. Значит, нужно выходить из цикла для часов. Так что же - часам останавливаться?

Ввиду вышеизложенного программисты для таких дел используют таймеры. Для часов свой, для секундомера свой.

Поместим на форму таймер и дадим ему имя Таймер_часов. Зададим ему интервал = 2500. Напишем такую программу:

```
Private Sub Таймер_часов_Timer()
   Циферблат_часов.Text = Time
End Sub
```

Запустите проект. Вы увидите, что часы показывают правильное время, но обновляется оно раз в две-три секунды. Зададим интервал = 100. В этом случае процедура будет выполняться около 10 раз в секунду и не пропустит момента смены системного времени. Проверьте.

Ну что, все! Часы готовы.

Занимаемся датой

Размещаем на форме рамочку и текстовое поле для даты. Даем полю имя Циферблат_даты. Чтобы там появилась дата, достаточно выполнить оператор

Циферблат_даты.Text = Date

Если его поместить в ту же процедуру таймера часов, то задача будет решена. Но мне жаль компьютер. 24 часа в сутки по 10 раз в секунду он будет спрашивать у Windows, какое нынче число, и стараться вывести в текстовое поле одну и ту же дату, хотя делать это нужно только два раза - при запуске проекта и в полночь. Здесь еще и вопрос экономии: бегая, как белка в колесе, компьютер тратит ресурсы (силы), нужные в это же время для другого дела, для того же секундомера хотя бы. Вспоминаем: при запуске проекта вырабатывается событие FormLoad, а полночь - это когда показания часов равны нулю. Ага. Дописываем программу:

```
Private Sub Form_Load()
   Циферблат_даты.Text = Date        'Это чтобы дата появлялась на циферблате при запуске проекта
End Sub

Private Sub Таймер_часов_Timer()
   Циферблат_часов.Text = Time
   If Time = 0 Then Циферблат_даты.Text = Date        'Это чтобы дата менялась в полночь
End Sub
```

Чтобы проверить, как работает проект, переставьте системные часы Windows на "Двенадцать без пяти". Только потом не забудьте переставить обратно.

Все работает, но мы начинаем допускать погрешности против правильного стиля программирования, которые в будущем могут выйти нам боком:

Первое. Показания часов напрашиваются быть переменной величиной. Ведь они нам еще понадобятся и для будильника. Мы их анализируем, а это лучше делать с переменной величиной, а не со свойством Циферблат_часов.Text или функцией Time. Поэтому придумаем переменную Время_на_часах, объявим ее, как имеющую тип Date и будем пользоваться только ей.

Второе. Как при запуске проекта, так и в полночь нам придется менять не только дату, но и день недели. Я предвижу повторяющийся фрагмент как минимум из двух операторов (пока это только один оператор Циферблат_даты.Text = Date). Поэтому оформим его, как процедуру с именем Смена_даты_и_дня_недели.

Третье. Поскольку у нас появились переменные, поручим Бэйсику при помощи оператора Option Explicit присматривать, не забыли ли мы какую-нибудь переменную объявить.

С учетом всего вышесказанного перепишем программу:

```
Option Explicit
Dim Время_на_часах As Date

Private Sub Form_Load()
   Смена_даты_и_дня_недели
End Sub

Private Sub Таймер_часов_Timer()
   Время_на_часах = Time
   Циферблат_часов.Text = Время_на_часах
   If Время_на_часах = 0 Then Смена_даты_и_дня_недели
End Sub

Private Sub Смена_даты_и_дня_недели()
   Циферблат_даты.Text = Date
End Sub
```

В этот момент вы совершенно искренне и с большим чувством можете сказать: "Ну зачем все эти усложнения? Ведь все и так работает!" В ответ на это я могу только отослать вас к началу раздела.

Занимаемся днем недели

Копируем объекты. Нам не хочется трудиться, размещая на форме, а потом еще и настраивая рамку и текстовое поле для дня недели. Замечаем, что они почти такие же, как для даты. Скопируем их, как это принято в Windows. Можно копировать по-отдельности, а удобнее вместе. Для этого обведите их рамочкой, при этом оба объекта выделятся, после чего скопируйте их хотя бы при помощи пунктов Copy, Paste меню Edit. Visual Basic при этом задаст вам про каждый объект вопрос, смысл которого вы поймете позже, а пока он вам не нужен. Отвечайте No.

Даем полю имя Циферблат_дня_недели. Чтобы там появился правильный день недели, нужно выполнить оператор

Циферблат_дня_недели.Text = WeekdayName(DatePart("w", Date, vbMonday))

Попробуйте разобраться в нем сами в качестве упражнения к функциям работы с датами. Если не удалось, то вот вам пояснение:

Сначала хорошенько еще раз разберитесь с функциями WeekdayName и DatePart из 11.1. А теперь нужно разобраться со скобками. Вы поняли, какая пара скобок к какой функции относится? Если нет, то я перепишу один оператор в два:

A = DatePart("w", Date, vbMonday)
Циферблат_дня_недели.Text = WeekdayName(A)

Здесь A будет равняться номеру дня недели (число от 1 до 7). Функция WeekdayName по номеру дня недели получает строковое название этого дня.

Раз мы организовали процедуру Смена_даты_и_дня_недели, то нам нет нужды вспоминать, в какие моменты времени проект должен менять день недели. Вставляем наш оператор в эту процедуру:

```
Private Sub Смена_даты_и_дня_недели()
  Циферблат_даты.Text = Date
  Циферблат_дня_недели.Text = WeekdayName(DatePart("w", Date, vbMonday))
End Sub
```

В качестве последнего штриха исправим одну неприятную вещь. Дело в том, что наши циферблаты совершенно беззащитны перед шаловливым пользователем. В режиме работы проекта вы можете щелкнуть мышкой внутри текстового окна и с клавиатуры безнаказанно как угодно менять его содержимое. Чтобы прекратить это, воспользуйтесь свойством **Locked**.

Итак, часы с датой и днем недели готовы.

Знакомимся с типом Boolean

Чтобы грамотно запрограммировать будильник, нам нужно познакомиться с новым типом переменных величин - Boolean. Это так называемый логический тип. В 5.7 вы уже позакомились с логическими выражениями. Их главная черта в том, что они могут принимать всего два значения - **True** и **False** (истина и ложь). Логическая переменная величина отличается той же чертой. Объявляется она так:

Dim C As **Boolean**

А зачем она нужна, выяснится из следующего параграфа.

Делаем будильник

Поместим на форму все объекты, необходимые для будильника. Дадим им имена:

- Метка_будильника
- Циферблат_будильника
- Кнопка_включения_выключения_будильника
- Кнопка_выключения_сигнала

В циферблат занесем в режиме проектирования текст 12:00:00. Просто для того, "чтобы было".

Давайте пока забудем о кнопках, метках и прочих подробностях, а возьмем быка за рога и заставим будильник исполнять мелодию при совпадении времени на часах и на будильнике.

Вспомните, как вы занимались музыкой в "Калькуляторе". Поместите на форму Microsoft Multimedia Control 6.0 и дайте ему имя "Плеер". Для работы с мелодией нам нужно 5 операторов:

```
Плеер.DeviceType = "Sequencer"
Плеер.FileName = "c:\Windows\Media\Canyon.mid"
```

Эти два достаточно выполнить один раз за время работы проекта при его запуске.

```
Плеер.Command = "Open"
Плеер.Command = "Play"
```

Эти два нужно выполнять каждый раз, когда нужно начать воспроизведение сигнала. А этот оператор:

```
Плеер.Command = "Close"
```

каждый раз, чтобы его закончить.

Вторая парочка, несмотря на то, что встретится в программе скорее всего только один раз, в глазах программиста просится стать процедурой. Здесь принцип такой: *Если группа операторов (или даже один сложный оператор) представляет собой некое единое целое в том смысле, что решает какую-то свою задачу, то оформляйте ее процедурой*. Это улучшит понятность и читабельность программы - один из важнейших факторов ее качества.

Вот как дополнился теперь наш проект (я показываю только те процедуры, которых коснулись дополнения):

```
Private Sub Form_Load()
  Плеер.DeviceType = "Sequencer"
  Плеер.FileName = "c:\Windows\Media\Canyon.mid"
  Смена_даты_и_дня_недели
End Sub

Private Sub Таймер_часов_Timer()
  Время_на_часах = Time
  Циферблат_часов.Text = Время_на_часах
  If Время_на_часах = 0 Then Смена_даты_и_дня_недели
  If Время_на_часах = Циферблат_будильника.Text Then Включить_сигнал_будильника
End Sub
```

```
Private Sub Включить_сигнал_будильника()
   Плеер.Command = "Open"
   Плеер.Command = "Play"
End Sub
```

Здесь жирным шрифтом выделены новые по сравнению с предыдущим вариантом строки проекта.

Запустите проект и установите время на циферблате будильника. Устанавливайте аккуратно, по одной цифре за раз. Не трогайте двоеточия. Почему - скажу чуть позже. Дождитесь, когда будильник зазвонит. Завершите работу проекта, не дожидаясь конца мелодии.

Пора заняться кнопками и меткой. Подумаем вот о чем. Во время работы проекта будильник в каждый момент времени может находиться в одном из двух состояний: установлен или не установлен. Компьютер должен в любой момент времени знать, в каком именно, чтобы принять решение - звонить или не звонить. В таких случаях, когда компьютер должен в любой момент что-то знать или помнить, программист всегда придумывает переменную величину, дает ей подходящее имя и тип и организует хранение в этой переменной нужной компьютеру информации. Так было у нас с переменной Время_на_часах, которая нужна была компьютеру для перестановки дат.

Придумаем переменную и дадим ей имя Будильник_установлен. Каким типом ее объявить? Для этого надо понять, а какие значения будет принимать эта переменная? По смыслу задачи значений два - "да" и "нет". Можно придать ей тип String, но хорошая практика программирования диктует тип Boolean. Вы скоро поймете, что он удобнее.

Теперь попробуем вообразить, что должно происходить при включении будильника:

- Нужно сообщить компьютеру, что будильник установлен
- Нужно, чтобы текст над циферблатом был такой: "Будильник установлен на:"
- Нужно, чтобы текст на кнопке был такой: "Выключить будильник"

В соответствии с этим пишем процедуру:

```
Private Sub Включить_будильник()
   Будильник_установлен = True
   Метка_будильника.Caption = "Будильник установлен на:"
   Кнопка_включения_выключения_будильника.Caption = "Выключить будильник"
End Sub
```

Теперь попробуем вообразить, что должно происходить при выключении будильника:

- Нужно сообщить компьютеру, что будильник не установлен
- Нужно, чтобы текст над циферблатом был такой: "Будильник отключен"
- Нужно, чтобы текст на кнопке был такой: "Включить будильник"

В соответствии с этим пишем процедуру:

```
Private Sub Выключить_будильник()
   Будильник_установлен = False
   Метка_будильника.Caption = "Будильник отключен"
   Кнопка_включения_выключения_будильника.Caption = "Включить будильник"
End Sub
```

Теперь нам удивительно легко написать процедуру щелчка по кнопке включения-выключения будильника:

```
Private Sub Кнопка_включения_выключения_будильника_Click()
   If Будильник_установлен Then Выключить_будильник Else Включить_будильник
End Sub
```

Обратите внимание, что благодаря удачному выбору имен и структуры программы язык программы по своей понятности приближается к естественному, человеческому.

Поскольку переменная Будильник_установлен имеет логическое значение True или False, ее можно использовать подобно функции IsNumeric из 5.9 в качестве условия оператора If. Поэтому совсем необязательно было писать If Будильник_установлен = True Then ….

Процедуру щелчка по кнопке выключения сигнала приведу без пояснений:

```
Private Sub Кнопка_выключения_сигнала_Click()
   Плеер.Command = "Close"
End Sub
```

В последних четырех процедурах заключена вся механика работы кнопок и метки. Последнее, о чем надо позаботиться, это о том, чтобы будильник звенел только тогда, когда включен:

If Будильник_установлен **And** Время_на_часах = Циферблат_будильника.Text **Then** Включить_сигнал_будильника

Будильник готов. Приведу полностью программу всего нашего проекта, причем строки, касающиеся секундомера, выделю курсивом, чтобы вы пока не обращали на них внимания:

```
Option Explicit

Private Enum типРежим_работы_секундомера
   считает
   пауза
   в_нуле
End Enum
Dim Режим_работы_секундомера As типРежим_работы_секундомера
```

```
Dim Время_на_часах As Date
Dim Будильник_установлен As Boolean
Dim Время_на_секундомере As Single
Dim Время_запуска_секундомера As Single
Dim Время_на_паузе_секундомера As Single
Dim Цифра_десятых As Long


                    'НАЧАЛЬНАЯ УСТАНОВКА МЕХАНИЗМА

Private Sub Form_Load()
   Плеер.DeviceType = "Sequencer"
   Плеер.FileName = "c:\Windows\Media\Canyon.mid"
   Смена_даты_и_дня_недели
   Выключить_будильник
   Секундомер_обнулить
End Sub


                    'ПРОЦЕДУРЫ РАБОТЫ ЧАСОВ

Private Sub Таймер_часов_Timer()
   Время_на_часах = Time
   Циферблат_часов.Text = Время_на_часах
   If Время_на_часах = 0 Then Смена_даты_и_дня_недели
   If Будильник_установлен And Время_на_часах = Циферблат_будильника.Text Then Включить_сигнал_будильника
End Sub

Private Sub Смена_даты_и_дня_недели()
   Циферблат_даты.Text = Date
   Циферблат_дня_недели.Text = WeekdayName(DatePart("w", Date, vbMonday))
End Sub


                    'ПРОЦЕДУРЫ РАБОТЫ БУДИЛЬНИКА

Private Sub Кнопка_включения_выключения_будильника_Click()
   If Будильник_установлен Then Выключить_будильник Else Включить_будильник
End Sub

Private Sub Включить_будильник()
   Будильник_установлен = True
   Метка_будильника.Caption = "Будильник установлен на:"
   Кнопка_включения_выключения_будильника.Caption = "Выключить будильник"
End Sub

Private Sub Выключить_будильник()
   Будильник_установлен = False
   Метка_будильника.Caption = "Будильник отключен"
   Кнопка_включения_выключения_будильника.Caption = "Включить будильник"
End Sub

Private Sub Включить_сигнал_будильника()
   Плеер.Command = "Open"
   Плеер.Command = "Play"
End Sub

Private Sub Кнопка_выключения_сигнала_Click()
   Плеер.Command = "Close"
End Sub

Private Sub Form_Terminate()
   Кнопка_выключения_сигнала_Click
End Sub


                    'ПРОЦЕДУРЫ РАБОТЫ СЕКУНДОМЕРА

Private Sub Таймер_секундомера_Timer()
   Время_на_секундомере = Timer - Время_запуска_секундомера + Время_на_паузе_секундомера
   Цифра_десятых = Int(10 * (Время_на_секундомере - Int(Время_на_секундомере)))
   Циферблат_секундомера.Text = DateAdd("s", Время_на_секундомере, #12:00:00 AM#) & "." & Цифра_десятых
End Sub
```

```
Private Sub Кнопка_пуска_паузы_секундомера_Click()
   If Режим_работы_секундомера <> считает Then Секундомер_запустить Else Секундомер_остановить
End Sub

Private Sub Кнопка_обнуления_секундомера_Click()
   Секундомер_обнулить
End Sub

Private Sub Секундомер_запустить()
   Время_запуска_секундомера = Timer
   Режим_работы_секундомера = считает
   Таймер_секундомера.Enabled = True
   Кнопка_пуска_паузы_секундомера.Caption = "ПАУЗА"
End Sub

Private Sub Секундомер_остановить()
   Время_на_паузе_секундомера = Время_на_секундомере
   Режим_работы_секундомера = пауза
   Таймер_секундомера.Enabled = False
   Кнопка_пуска_паузы_секундомера.Caption = "ПУСК"
End Sub

Private Sub Секундомер_обнулить()
   Время_на_паузе_секундомера = 0
   Режим_работы_секундомера = в_нуле
   Таймер_секундомера.Enabled = False
   Кнопка_пуска_паузы_секундомера.Caption = "ПУСК"
   Циферблат_секундомера.Text = "0:00:00.0"
End Sub
```

Обратите внимание, что при запуске проекта я в процедуре Form_Load выключаю для удобства будильник, а в процедуре Form_Terminate закрывается звуковой файл, если пользователь не удосужился закрыть его кнопкой. В ней мы обращаемся к процедуре обработки события Кнопка_выключения_сигнала_Click, как к процедуре пользователя. Это вполне допустимо.

Напоминаю, что событие Form_Terminate наступает только тогда, когда мы завершаем работу проекта, щелкнув по кнопке **Close** (крестику в правом верхнем углу формы), а не кнопкой End на панели инструментов.

Знакомимся с перечислимым типом данных

Чтобы грамотно запрограммировать секундомер, нам нужно познакомиться с новым типом переменных величин. Необходимость в нем в нашем проекте вытекает вот откуда. Если будильник в каждый момент времени может находиться в одном из двух состояний (установлен или не установлен), то секундомер - в трех: считает, стоит в паузе, стоит в нуле. Придумаем переменную и дадим ей имя Режим_работы_секундомера. Объявить ее типом Boolean явно недостаточно, ведь в типе Boolean всего два возможных значения, а нам нужно три. Можно придать ей тип String, но хорошая практика программирования диктует другое.

В Visual Basic нет типа данных, в котором переменная имела бы ровно три значения, зато Visual Basic, как и многие языки, позволяет программисту создавать собственные типы. Наша задача - создать тип, в котором переменная принимает ровно три значения: считает, пауза, в нуле - а затем объявить этим типом переменную Режим_работы_секундомера.

В нашем случае для создания такого типа достаточно записать выше процедур конструкцию:

```
Private Enum типРежим_работы_секундомера
   считает
   пауза
   в_нуле
End Enum
```

Это не процедура, хоть и похожа. Слово **Enum** означает, что тип - **перечислимый**. Это означает, что мы обязаны были придумать и в этой конструкции *перечислить* имена всех возможных значений переменной этого типа, что мы и сделали. Поскольку каждый тип должен иметь свое имя (Integer, String, ...), нам тоже пришлось придумать имя новому типу (типРежим_работы_секундомера) и указать его в заголовке.

Теперь, когда новый тип определен, можно любую переменную объявить этим типом, что мы и делаем:

```
Dim Режим_работы_секундомера As типРежим_работы_секундомера
```

Замечание для новичков: Новички с трудом воспринимают смысл строк со стоящими рядом похожими именами, таких, например, как только что написанная. Новичку кажется, что эти имена означают одно и то же. Должен сказать, что профессиональные программисты специально обозначают близкие вещи похожими именами, им так больше нравится (можете себе вообразить?!). Например, у них может встретиться кнопка с именем cmdПодать_сюда_Ляпкина_Тяпкина и в этом же проекте переменная с именем strПодать_сюда_Ляпкина_Тяпкина. Для того, чтобы отличать одно от другого, они начинают каждое имя с *префикса*, который говорит программисту (не компьютеру!) о том, кому принадлежит имя (кнопке (префикс cmd), строковой переменной (префикс str) или кому-нибудь другому). Со временем вы поймете неизбежность такого подхода, а пока вам придется быть внимательным и не путать близкие имена.

Делаем секундомер

Поместим на форму элементы управления и дадим им имена:

- Таймер_секундомера
- Циферблат_секундомера
- Кнопка_пуска_паузы_секундомера
- Кнопка_обнуления_секундомера

а также не забудем метку и рамочку.

Будем использовать следующие переменные:

```
Dim Режим_работы_секундомера As типРежим_работы_секундомера
Dim Время_на_секундомере As Single
Dim Время_запуска_секундомера As Single
Dim Время_на_паузе_секундомера As Single
Dim Цифра_десятых As Long
```

Обратите внимание, что все переменные времени объявлены, как дробные числовые, а не как Date. Сейчас вам станет ясно, почему.

Давайте по порядку. Таймер нужен секундомеру для того же, для чего и часам, а именно - чтобы вовремя менялись цифры на циферблате, а *собственный* таймер нужен для того, чтобы не зависеть от часов. Поскольку нам нужно отслеживать десятые доли секунды, установим ему интервал поменьше, например 10 или 5 - не играет роли. Когда секундомер считает, таймер секундомера должен работать:

Таймер_секундомера.Enabled = True

а когда он в паузе или в нуле, таймер должен стоять

Таймер_секундомера.Enabled = False

Для отсчета времени на секундомере мы будем использовать известную вам функцию Timer (не путайте с элементом управления Timer), потому что она выдает с приемлемой для нас точностью десятые доли секунды (какой же секундомер без десятых). Поскольку она выдает число секунд, прошедших с полуночи, а нам нужно число секунд, прошедших с момента запуска секундомера, да еще с учетом того, что во время паузы на секундомере уже стояли какие-то показания, нам придется поразмыслить, как это сделать.

Засечем в момент пуска секундомера значение функции Timer оператором

Время_запуска_секундомера = Timer

Тогда в каждый момент времени после запуска секундомера выражение (Timer - Время_запуска_секундомера) как раз и будет равняться числу секунд, прошедших с момента запуска секундомера. А если нам удастся правильно засечь Время_на_паузе_секундомера (выраженное в секундах, прошедших с момента пуска), то дело решит оператор

Время_на_секундомере = (Timer - Время_запуска_секундомера) + Время_на_паузе_секундомера

Если поместить его в процедуру таймера, то он будет оперативно выдавать нужное Время_на_секундомере. Задача решена.

Теперь займемся внешним видом показаний секундомера. Время_на_секундомере - это дробное число - количество секунд. Например, такое - 67,2. А нам хотелось бы получить его в таком виде - 00:01:07.2. Для этого нам нужно как-то преобразовать число секунд в стандартный формат времени. Дело решает оператор:

Циферблат_секундомера.Text = DateAdd ("s", Время_на_секундомере, #12:00:00 AM#)

Здесь время #12:00:00 AM# обозначает, как ни странно, полночь по-американски. Задача решена.

Но функция DateAdd оставляет за бортом десятые доли секунды. Попробуем выделить их из числа Время_на_секундомере. Для этого мысленно проведем такую цепочку операций с использованием функции Int:

Целая часть = Int (Время_на_секундомере)
Дробная часть = Время_на_секундомере - Целая часть
Цифра_десятых = Int (10 * Дробная часть)

Сведем эти три оператора в один:

Цифра_десятых = Int(10 * (Время_на_секундомере - Int(Время_на_секундомере)))

и дополним оператор вывода времени на циферблат секундомера:

Циферблат_секундомера.Text = DateAdd ("s", Время_на_секундомере, #12:00:00 AM#) & "." & Цифра_десятых

Здесь знак **&** является удобным заменителем знака **+** для сборки данных в одну строку. Я рекомендую пользоваться именно им, так как компьютер его уж никак не спутает со сложением чисел. Между целой и дробной частью секунд я решил поставить точку. Обратите внимание, что знак **&** без проблем соединяет данные трех разных типов: функцию DateAdd, строку "**.**" и число Цифра_десятых.

Ну вот, пожалуй, и все объяснения. В остальном вы сможете разобраться сами, прочитав текст программы, так как ничего нового по сравнению с будильником не обнаружите.

После этого самые дотошные скажут мне, что нечего было огород городить - создавать новый тип данных, когда можно было обойтись логической переменной Секундомер_считает. Верно, но неправильно. Потому что нужно оставлять простор для дальнейшего развития проекта. Например, вы можете захотеть, чтобы во время паузы цифры на секундомере мигали, а в нуле – нет. Отличить одно состояние от другого вам и поможет переменная Режим_работы_секундомера.

Недостатки проекта

Мой проект работает на первый взгляд нормально, я проверял и часы, и будильник, и секундомер. Но делал я это не так тщательно, как положено при тестировании, поэтому в нем вполне возможны ошибки. Вот те, что я сумел заметить, но не стал исправлять, предоставив это вам:

- В будильнике, если сигнал завершился сам, без нажатия кнопки "Выключить сигнал", то в следующий раз он не про-

звучит, так как файл не закрыт. Справиться с этим недостатком вам поможет событие Плеер_Done, которое возникает при завершении воспроизведения мелодии. А можно, наверное, выкрутиться еще проще, как мы сделали в Калькуляторе.

- В будильнике пользователь может изменять время сигнала, не дождавшись конца мелодии, что может привести к необходимости нового запуска мелодии, пока она еще не отзвучала. Здесь вам поможет событие Циферблат_будильника_GotFocus.
- Самое досадное то, что если вы во время установки времени будильника случайно хоть на мгновение сотрете двоеточие или напишете вместо цифры букву или как-нибудь по-другому нарушите правильный формат времени, то проект аварийно завершит работу, потому что оператор (If Будильник_установлен And Время_на_часах = Циферблат_будильника.Text Then Включить_сигнал_будильника) не сможет понять, чему равно время Циферблат_будильника.Text. Здесь вам понадобятся средства, предотвращающие ввод в текстовое окно неправильных данных. Самое простое из них - функция **IsDate**, которая подобно функции IsNumeric из 5.9 определяет, можно ли считать ее аргумент правильным временем и правильной датой или это белиберда. Подумайте, вам вполне по силам справиться с этой проблемой.
- В течение той секунды, когда время на часах и будильнике одинаково, процедура таймера выполняется с десяток раз, а значит и сигнал будильника запускается с десяток раз. Это нехорошо.
- В циферблате даты месяц хорошо бы показывать не числом, а текстом.
- Я не запускал секундомер в полночь.

Таймер и моделирование

Теперь о роли таймера в компьютерном конструировании поведения реальных механизмов. Заглянем-ка еще раз в процедуру таймера часов:

```
Private Sub Таймер_часов_Timer()
   Время_на_часах = Time
   Циферблат_часов.Text = Время_на_часах
   If Время_на_часах = 0 Then Смена_даты_и_дня_недели
   If Будильник_установлен And Время_на_часах = Циферблат_будильника.Text Then Включить_сигнал_будильника
End Sub
```

Вы видите, что эта процедура является главным мотором, приводящим в движение весь механизм часов с будильником, главным штабом, планирующим всю их работу. Таймер, как сердце, несколько раз в секунду посылает импульсы, каждый из которых заставляет выполнится эту процедуру. А что за операторы составляют тело процедуры? Это как раз главные операторы проекта, которые, выполняясь, в свою очередь управляют выполнением вспомогательных процедур Смена_даты_и_дня_недели и Включить_сигнал_будильника. За один импульс таймера механизм совершает весь цикл своего функционирования с начала до конца и выполняет всю работу, которую положено выполнить: время на часах обновлено; если подошел момент, то именно на этом импульсе заменены дата и день недели; и если подошел момент, то именно на этом же импульсе отдан приказ на включение сигнала будильника. На следующем импульсе все повторяется. И так беспрерывно, бесконечно.

Этот принцип применения таймера подойдет для компьютерного конструирования работы бесчисленного множества других механизмов и аппаратов: холодильника, ситезатора, автоматической метеостанции, беспилотного космического аппарата и пр. и пр. Нужно только знать принцип их работы и вы сможете создать проект, "вживую" показывающий их деятельность на кухне или в космосе. Это называется **моделированием**. Вот, например, мы с вами создали **модель** будильника, которая настолько хороша, что работает не хуже оригинала. Однако, мы с вами не копировали работу реальных шестеренок бабушкиных ходиков, нам это было не нужно. Таким образом, нужно понимать, что мы создали модель только внешнего, а не внутреннего поведения механического будильника. Внутреннее поведение нашего будильника совсем другое.

Можно вполне моделировать и работу механизмов под управлением человека: автомобиля, корабля, самолета и др. В этом случае роль человека будете исполнять вы, щелкая мышкой по созданным вами кнопкам и рычагам управления автомобиля. Кстати, вы уже создали такой проект. Ведь будильник - это тоже механизм, работающий под управлением человека.

Задание 99-1: *Это задание - на любителя. Его делать не нужно, если вы выполните проект "Шахматные часы" (см. ниже).* Усовершенствуйте часы. Пусть они по вашему желанию показывают время в любом из нескольких десятков городов мира. Пользователю достаточно ввести название города в текстовое поле. Вам для программирования нужно самим знать поясное время в других городах. Для этого можете сделать двойной щелчок мышкой по индикатору времени на панели задач и в открывшемся окне загляните в список Time Zone. Основой для программирования можете сделать большой оператор Select Case. Будильник не должен обращать внимание на чужое время на циферблате, он все равно должен звонить по нашему местному времени.

Задание 99-2: *"Шахматные часы". Это задание нужно сделать обязательно.* По шахматным правилам шахматист не может думать над ходом бесконечно. На обдумывание первых, скажем, 40 ходов ему дается, скажем, 2 часа. Таким образом, партия, дошедшая до 41 хода, не может продолжаться дольше 4 часов. На следующие ходы тоже отводится какой-то лимит времени. Чтобы шахматисты могли следить за тем, сколько времени им осталось на обдумывание, существуют специальные шахматные часы. Они представляют собой единый корпус с двумя циферблатами - счетчиками времени, двумя кнопками и двумя флажками. Перед началом партии шахматистов А и В каждый счетчик показывает 2 часа. Пусть шахматисту А выпало начинать. Как только партия начинается, судья запускает счетчик А, который начинает обратный отсчет времени, уменьшая свои показания. Таким образом, в каждый момент партии счетчик показывает, сколько времени осталось шахматисту на обдумывание. Пока работает счетчик А, счетчик В, естественно, стоит. Как только шахматист А сделал ход, он тут же нажимает кнопку А, которая останавливает его счетчик и запускает счетчик В. За обдумывание принимается шахматист В. Сделав ход, он нажимает кнопку В, которая останавливает его счетчик и запускает счетчик А. И так далее. Если шахматист просрочит время, его флажок падает - он проиграл.

Для удобства шахматистов добавьте к часам счетчик ходов.

11.4. Анимация

Анимация означает придание неподвижному предмету движения. Еще одно значение слова анимация - мультфильм.

Анимация при помощи графических методов

В Глава 6 я объяснил идею создания иллюзии движения картинок по экрану. Там же мы двигали по форме объекты. Попробуем заставить двигаться по экрану не объекты, а геометрические фигуры, полученные графическими методами. Пусть слева направо движется окружность. Для этого мы должны сначала нарисовать ее слева и быстро же стереть, для чего нарисовать ее на том же месте, но цветом фона. Несмотря на то, что мы окружность быстро стерли, она успеет мелькнуть на экране, и глаз это заметит. Затем нужно нарисовать и стереть такую же окружность чуть правее, затем еще правее и т.д.

Ввиду причин, упомянутых в 11.3, откажемся от операторов цикла. Будем использовать таймеры. Создадим проект. Поместим в него таймер. Установим его интервал в любое число < 50, при этом он будет выдавать максимально возможное число импульсов в секунду - 18. Вот программа:

```
Dim x As Integer                        'Координаты и радиус окружности
Dim y As Integer
Dim R As Integer
Dim Цвет_окружности As Long
Dim Цвет_фона As Long

Private Sub Form_Load()
   x = 1000
   y = 1500
   R = 200
   DrawWidth = 5                        'Толщина линии
   Цвет_окружности = vbBlack
   Цвет_фона = BackColor
End Sub

Private Sub Timer1_Timer()
   Circle (x, y), R, Цвет_окружности    'Рисуем окружность
   For i = 1 To 500000:    Next         'Пустой цикл для задания паузы
   Circle (x, y), R, Цвет_фона          'Стираем окружность
   x = x + 30                           'Перемещаемся немного направо
End Sub
```

Пояснения: Когда вы попробуете выполнить эту программу на компьютере, изображение движущейся окружности может получиться некачественным - окружность в процессе движения будет мерцать и пульсировать. Это связано с разверткой электронно-лучевой трубки вашего монитора. Если создать маленькую паузу между рисованием и стиранием окружности, нежелательные эффекты сойдут на нет. Эту паузу я создаю пустым циклом. Поэкспериментируйте с радиусом, толщиной окружности, продолжительностью паузы или шагом движения по горизонтали. Последние две величины определяют скорость движения.

Задание 100: Пусть по экрану движется "вагон" - прямоугольник и два колеса.

Движем объекты

Мы больше не будем заниматься анимацией при помощи графических методов, потому что объекты двигать гораздо приятнее.

Поместим на форму объект Shape в виде той же окружности и таймер. Пусть окружность движется вверх. Программа:

```
Private Sub Timer1_Timer()
   Shape1.Top = Shape1.Top - 20
End Sub
```

Как видите, она гораздо проще, чем программа анимации при помощи методов.

Задание 101: Пусть одновременно движутся две окружности. Таймер - один.

Задание 102: Одна вниз, другая направо.

Задание 103: Покажите своим друзьям фильм, снятый "секретной видеокамерой": В небе над вашим домом летит летающая тарелка. Для этого вам понадобится ввести в компьютер фотографию, на которой были бы видны ваш дом и небо. Роль летающей тарелки с успехом выполнит фигура эллипса с заливкой. Можете сделать ей окна (фигуры окружностей с желтой заливкой). Все эти фигуры должны лететь с одинаковой скоростью, не то окна "уплывут" с тарелки.

Заставим какую-нибудь фигуру двигаться направо, а затем самостоятельно отскочить от правого края формы:

```
Dim Шаг As Integer
Dim x As Integer

Private Sub Form_Load()
   x = Shape1.Left
   Шаг = 50
End Sub

Private Sub Timer1_Timer()
```

```
    x = x + Шаг
    Shape1.Left = x
    If x > Width Then Шаг = -50     'Если фигура улетела за правый край формы, то лететь обратно
End Sub
```

Задание 104:. Заставьте фигуру бесконечно двигаться, отскакивая от правого и левого краев формы.

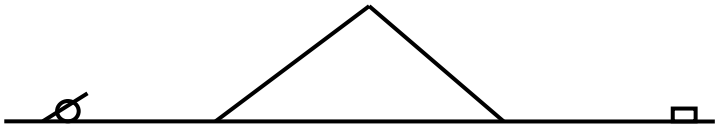
Задание 105:. *"Биллиардный шар".* Нарисуйте «биллиардный стол» – большой прямоугольник. Шар под углом летает по столу, отскакивая от его краев по закону отражения. Попав "в лузу" (любой из четырех углов стола), останавливается. Объектом здесь удобно взять Image с загруженной иконкой в виде шарика (подходящие иконки есть в Visual Basic).

Задание 105-1(сложное):. *"Часы со стрелками".* Если вы в ладах с тригонометрией, сделайте часы со стрелками: часовой, минутной, секундной. Задача упростится, если вы выберете в качестве стрелок тоненькие сектора окружностей.

Задание 106: Изобразите полет камня, брошенного с башни, для задания 45 из 6.3. Напоминаю условие задания. Камень бросили горизонтально со 100-метровой башни со скоростью v=20м/с. Его расстояние от башни по горизонтали (s) выражается формулой s=v*t, где t – время полета камня в секундах. Высота над землей h выражается формулой $h=100 - 9.81*t^2/2$. Нарисуйте башню, землю. Камнем может служить Image с подходящей загруженной иконкой. Затем камень летит. Добейтесь, чтобы время полета камня на экране примерно соответствовало реальному времени, полученному при решении задания 45. Нарисуйте траекторию полета камня. Для этого достаточно, чтобы камень оставлял за собой следы в виде точек.

Указание: В задаче говорится о метрах, а на экране расстояние измеряется в твипах. Поэтому вам придется задать масштаб, то есть вообразить, что один твип равен, скажем, одному метру. Тогда высота башни будет равна 100 твипов, а скорость камня – 20 твипов в секунду Правда, картинка на экране в этом случае будет слишком маленькой. Тогда можете задать другой масштаб – 1 метр равен, скажем, 40 твипам. Тогда высота башни будет равна 4000 твипам, а формулы изменятся: s = 40*v*t и $h=40*(100 - 9.81*t^2/2)$.

Задание 107 (сложное): Сделайте игру: Пушка на экране стреляет в цель ядрами. С какого выстрела она поразит противника? Между пушкой и целью расположена небольшая гора. Перед началом игры случайно задается горизонтальная координата цели. Затем рисуется картинка.



Перед каждым выстрелом компьютер отображает в текстовом поле номер выстрела и запрашивает у человека стартовую скорость ядра v и угол α наклона ствола пушки к земле. Затем летит ядро. Полет ядра подчиняется двум уравнениям: $s=v*t*\cos\alpha$ и $h=v*t*\sin\alpha - 9.81*t^2/2$ (см. предыдущее задание). Считается, что цель поражена, если ядро ее коснулось, не коснувшись горы. Указание: Вы можете запрограммировать автоматическое определение попадания в цель. Для этого нужно в момент, когда ядро при падении пересекло уровень земли, сравнить горизонтальные координаты ядра и цели. Если они достаточно близки, то фиксируйте попадание. Определение прикосновения к горе - более хлопотное занятие, но идея та же.

"Движем" свойства объектов

Сделаем рекламный ролик: Из черной глубины экрана на нас наплывают, увеличиваясь, красные слова "Съешьте Марс!"

Покрасим форму. Поместим на форму метку и сделаем ее большой и прозрачной. Придадим ее свойству Caption значение нужного текста. Сменим название шрифта на "Times" или какой-нибудь другой красивый, настроим стиль и цвет шрифта. Программа:

```
Private Sub Timer1_Timer()
        Label1.FontSize = Label1.FontSize + 1
End Sub
```

Задание 108: Пусть текст также непрерывно меняет цвет.

Пусть теперь на нас надвигается, увеличиваясь в размерах, фото. Для этого придадим это фото объекту Image. Установив в True его свойство Strech, будем увеличивать размеры объекта, стараясь сохранять его пропорции:

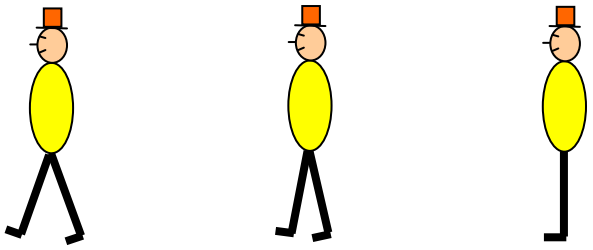
```
Private Sub Timer1_Timer()
        Image1.Width = Image1.Width + 10
        Image1.Height = Image1.Height + 7
End Sub
```

Таким образом можно оживлять объект, меняя любое его свойство, имеющее численное значение.

Мультфильм

Когда мы снимаем видеокамерой идущего человека, то на разных кадрах у него разное положение ног. Когда мультипликаторы на студии создают мультик с идущим человечком, они карандашом и красками прорисовывают все кадры, а их много - по нескольку на каждую секунду фильма - адский труд.

Сейчас мы с вами создадим сверхпростой мультфильм, где вот такой вот человечек



идет по белой форме справа налево. Для создания приемлемой иллюзии ходьбы нам достаточно смены этих трех кадров. Я пронумеровал их справа налево.

Зайдите в графический редактор Paint и на белом фоне нарисуйте первый кадр. Вы можете сделать человечка гораздо красивее и подробнее, чем это сделал я. Если вы отлично работаете в солидном графическом редакторе, то можете даже взять фотографию вашего знакомого. Сохраните человечка под именем Кадр1. Теперь, не стирая человечка, измените ему положение ног и сохранитесь как Кадр2 (не Save, а Save as...). Аналогично создайте и сохраните Кадр3.

Теперь зайдите в Visual Basic. Сделайте форму белой. Поместите на форму куда-нибудь в сторонку три объекта Image. Каждому в качестве картинки придайте свой кадр. Теперь поместите на форму объект Image4. Наша идея - придавать объекту Image4 по очереди с достаточной скоростью картинки из трех других Image. Тогда мы увидим, что человечек в Image4 передвигает ногами. Менять кадры нужно в такой последовательности: 1-2-3-2-1-2-3-2-1-2-3-2-1-2-…. Если при этом объект Image4 будет еще и двигаться налево по белой форме, то результат будет приятным. Давайте-ка для дальнейшего понимания разобьем эту последовательность на одинаковые участки: 1-2-3-2----1-2-3-2-----1-2-3-2-----1-2-…. Длина участка равна 4.

Поместите на форму таймер и придайте ему интервал 100 (потом, если движение будет слишком быстрым или медленным, вы его измените). Наша задача - сделать так, чтобы при каждом выполнении процедуры таймера мы видели один очередной кадр из приведенной мной последовательности. Для этого я организовал переменную N и заставил ее пробегать значения 0-1-2-3-----0-1-2-3-----0-1-2-…. Как видите, длину участка на ней я подобрал тоже = 4. Сделайте три объекта Image невидимыми, чтобы не мешались. Вот программа:

```
Dim N As Integer

Private Sub Form_Load()
    N = 0                                   'Начинаем с 0
End Sub

Private Sub Timer1_Timer()
    Select Case N
        Case 0:     Image4.Picture = Image1.Picture
        Case 1:     Image4.Picture = Image2.Picture
        Case 2:     Image4.Picture = Image3.Picture
        Case 3:     Image4.Picture = Image2.Picture
    End Select
    N = N + 1                               'Увеличиваем N на 1
    If  N = 4  Then  N = 0                  'После 3 должен идти 0, а не 4
    Image4.Left = Image4.Left  -  60        'Движем человечка налево
End Sub
```

Пару операторов

```
N = N + 1
If  N = 4  Then  N = 0
```

можно заменить одним изящным оператором N = (N + 1) Mod 4

Вы можете как угодно улучшать мультик, например, пусть сзади человечка медленно едет автомобиль.

Задание 109: "*Улыбка*". Для тех, кто умеет рисовать. Попросите у своей знакомой ее фотографию, где она снята с серьезным выражением лица. Введите фото в компьютер. Сделайте в Paint, а лучше в FotoShop еще два-три кадра этого фото, аккуратно понемножку приподнимая уголки губ на изображении. Подумайте, в какой последовательности нужно показывать кадры, чтобы улыбка постепенно возникала и исчезала.

О прозрачном цвете

Конечно, вам бы хотелось, чтобы человечек шел не по белому экрану, а по улице (фотография улицы). Но здесь вы столкнетесь с проблемой: человечек будет обрамлен белым прямоугольником фона, в котором вы его рисовали. Хорошо бы можно было сделать белый (или любой другой) цвет прозрачным. Но это нетривиальная проблема и в курсе для начинающих ее не стоит решать.

Глава 12. Работа с мышью и клавиатурой

До сих пор в режиме работы проекта мы пользовались мышкой только для того, чтобы примитивно нажимать на кнопки, а клавиатурой - только для ввода текста в текстовые поля. Однако, Visual Basic позволяет мышью и клавиатурой делать все те вещи, которые мы делаем ими в любых графических и текстовых редакторах, играх и других приложениях Windows. В том числе, мы можем с их помощью управлять поведением и движением объектов на форме.

12.1. Работа с мышью

Создадим программу на определение точности руки и глаза: При нажатии кнопки возникает в случайном месте экрана и тут же исчезает маленькая окружность. Вы должны поточнее щелкнуть мышкой там, где она была. После щелчка компьютер сообщает вам, на каком расстоянии от центра окружности было острие мышиного курсора во время щелчка.

Для создания программы нам необходимо поближе познакомиться с событиями, возникающими при работе с мышью. Заглянем в "универсальный справочник" - Object Browser. Поскольку щелкать мышью мы будем над формой, то в левой части Object Browser выберем объект Form. В правой части отыщем события, связанные с мышью. Нас пока интересует пять событий: **Click** (щелчок), **DblClick** (двойной щелчок), **MouseDown** (нажали клавишу мыши), **MouseUp** (отпустили клавишу мыши), **MouseMove** (сдвинули мышь).

События Click и DblClick нам не подойдут, так как они ничего не говорят о координатах мыши во время щелчка. А вот MouseDown подойдет, так как координаты сообщает. А при щелчке события MouseDown и MouseUp обязательно наступают, так как любой щелчок это не что иное, как нажатие и отпускание.

События MouseDown и MouseUp

Зайдем в окно кода и выберем для объекта Form событие **MouseDown** (как это делать, я вас учил в 3.6). В окне кода появится следующая заготовка:

```
Private Sub  Form_MouseDown   (Button As Integer,   Shift As Integer,   X As Single,   Y As Single)

End Sub
```

4 параметра в скобках - это 4 вещи, которые компьютер сообщает процедуре в момент события:

- Button - какая из трех кнопок мыши была нажата
- Shift - были ли при этом в нажатом состоянии служебные клавиши на клавиатуре и какие именно
- X , Y - координаты острия курсора мыши во время нажатия

Для того, чтобы понять и проверить смысл этих параметров, прочтите (чтобы понять) и запустите (чтобы проверить) такую программу:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)

    'Определяем, какая клавиша мыши нажата:
  Select Case Button
    Case 1:    Debug.Print "Нажата левая клавиша"
    Case 2:    Debug.Print "Нажата правая клавиша"
    Case 4:    Debug.Print "Нажата средняя клавиша"
  End Select

    'Определяем, какие из трех клавиш клавиатуры (Shift, Ctrl, Alt) были при этом в нажатом состоянии:
  Select Case Shift
    Case 0:    Debug.Print "Не нажата ни одна клавиша Shift, Ctrl, Alt"
    Case 1:    Debug.Print "Нажата клавиша Shift"
    Case 2:    Debug.Print "Нажата клавиша Ctrl"
    Case 3:    Debug.Print "Нажаты клавиши Shift, Ctrl"
    Case 4:    Debug.Print "Нажата клавиша Alt"
    Case 5:    Debug.Print "Нажаты клавиши Shift, Alt"
    Case 6:    Debug.Print "Нажаты клавиши Ctrl, Alt"
    Case 7:    Debug.Print "Нажаты клавиши Shift, Ctrl, Alt"
  End Select

    'Определяем координаты острия курсора мыши во время нажатия:
  Debug.Print "X="; X, "Y="; Y

End Sub
```

В численном значении Shift есть система. Посмотрите в процедуре, чему "равны" клавиши Shift, Ctrl, Alt по одиночке. 1, 2 и

4. Так вот, их совместное нажатие "равно" их сумме. Убедитесь.*

Поместите на форму несколько разных объектов. Обратите внимание, что при щелчке по ним процедура не срабатывает. Это естественно, у каждого объекта есть свое событие MouseDown.

Событие **MouseUp** работает аналогично.

Пример программы

Вот программа для поставленной выше задачи на точность руки и глаза:

```
Dim X_кружка As Integer
Dim Y_кружка As Integer
Dim Расстояние_до_кружка As Double

Private Sub Form_Load()
   Randomize
End Sub

            'Процедура для создания мелькнувшего кружка:
Private Sub Command1_Click()
   X_кружка = 4000 * Rnd            'Определяем координаты кружка (центра кружка)
   Y_кружка = 4000 * Rnd
   ForeColor = vbBlack              'Чертим кружок черным цветом
   Circle (X_кружка, Y_кружка), 50
   For i = 1 To 3000000: Next       'Пауза, чтобы мы успели заметить кружок
   ForeColor = BackColor            'Стираем кружок цветом фона
   Circle (X_кружка, Y_кружка), 50
End Sub

            'Процедура для определения расстояния от щелчка до кружка:
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Расстояние_до_кружка = Sqr((X - X_кружка) ^ 2 + (Y - Y_кружка) ^ 2)
   MsgBox ("Промах на " & Int(Расстояние_до_кружка) & " твип.")
End Sub
```

Пояснение того, как вычислялось расстояние (для тех, кто знает теорему Пифагора): Мысленно соедините отрезком прямой центр окружности и точку щелчка. Это будет гиппотенуза прямоугольного треугольника, катеты которого проведите вертикальной и горизонтальной линией. Нетрудно заметить, что горизонтальный катет равен X - X_кружка, а вертикальный равен Y - Y_кружка (знак я не учитываю). Гиппотенуза же равна нужному нам расстоянию. Теорема Пифагора гласит, что квадрат гиппотенузы равен сумме квадратов катетов. Отсюда, гиппотенуза равна корню квадратному из суммы квадратов катетов (каковая формула и записана в программе).

Событие MouseMove

Это событие возникает, как только мы сдвигаем мышку с места, а во время движения мыши оно возникает постоянно и многократно. Оно похоже на событие MouseDown, но есть и отличия. Запустите и проверьте эту поясняющую программу:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
      'Определяем, какие клавиши мыши удерживаются нажатыми во время движения:
   Select Case Button
      Case 0:   Debug.Print "Не нажата ни одна клавиша мыши"
      Case 1:   Debug.Print "Нажата левая клавиша"
      Case 2:   Debug.Print "Нажата правая клавиша"
      Case 3:   Debug.Print "Нажаты левая, правая клавиши"
      Case 4:   Debug.Print "Нажата средняя клавиша"
      Case 5:   Debug.Print "Нажаты левая, средняя клавиши"
      Case 6:   Debug.Print "Нажаты правая, средняя клавиши"
      Case 7:   Debug.Print "Нажаты левая, правая, средняя клавиши"
   End Select

      'Определяем, какие из трех клавиш клавиатуры (Shift, Ctrl, Alt) удерживаются нажатыми во время движения:
   Select Case Shift
      Case 0:   Debug.Print "Не нажата ни одна клавиша Shift, Ctrl, Alt"
      Case 1:   Debug.Print "Нажата клавиша Shift"
      Case 2:   Debug.Print "Нажата клавиша Ctrl"
      Case 3:   Debug.Print "Нажаты клавиши Shift, Ctrl"
      Case 4:   Debug.Print "Нажата клавиша Alt"
      Case 5:   Debug.Print "Нажаты клавиши Shift, Alt"
      Case 6:   Debug.Print "Нажаты клавиши Ctrl, Alt"
      Case 7:   Debug.Print "Нажаты клавиши Shift, Ctrl, Alt"
   End Select

      'Определяем, координаты острия курсора мыши во время движения:
```

* Когда я проверял этот пример, оказалось, что правая клавиша Alt ведет себя не так, как положено.

```
    Debug.Print "X="; X, "Y="; Y
End Sub
```

Если событие MouseDown сообщает о нажатии какой-то *одной* клавиши мыши, то MouseMove сообщает о *любой комбинации* мышиных клавиш. Если событие MouseDown сообщает о *событии* нажатия какой-то клавиши, то MouseMove сообщает о *состоянии* мышиных клавиш (удерживаются нажатыми или нет).

Мышь рисует

Вот программа, превращающая мышку в карандаш:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    PSet (X, Y)
End Sub
```

Запустите ее и медленно ведите мышкой по форме. За мышкой остается нарисованный след.

Задание 110: Сделайте так, чтобы мышь рисовала только при нажатой левой клавише, что более привычно для всех, кто работал в графических редакторах.

Задание 111: Сделайте так, чтобы при щелчке по правой клавише толщина линии возрастала на 1.

12.2. Работа с клавиатурой

Поставим задачу сделать игру, где наш миниатюрный гоночный автомобиль будет под управлением клавиш клавиатуры нестись от старта до финиша. Для этого вам нужно познакомиться с событиями, связанными с клавиатурой. Их три: **KeyDown** (клавиша нажата), **KeyUp** (клавиша отпущена) и **KeyPress**.(по клавише щелкнули). Нас пока интересуют только первые два.

События KeyDown и KeyUp

Создайте проект из одной формы, без элементов управления. Зайдите в окно кода и выберите для объекта Form событие **KeyDown**. В появившуюся заготовку процедуры запишите сдедующий код:

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)

    'Определяем, какая клавиша клавиатуры была нажата:
  Select Case KeyCode
    Case vbKeyUp:        Debug.Print "Нажата стрелка вверх"
    Case vbKeyDown:      Debug.Print "Нажата стрелка вниз"
    Case vbKeyLeft:      Debug.Print "Нажата стрелка налево"
    Case vbKeyRight:     Debug.Print "Нажата стрелка направо"
    Case vbKeyW:         Debug.Print "Нажата клавиша W"
    Case vbKey7:         Debug.Print "Нажата клавиша 7"
    Case vbKeySpace:     Debug.Print "Нажата клавиша пробела"
    Case vbKeyDelete:    Debug.Print "Нажата клавиша Delete"
    Case vbKeyF4:        Debug.Print "Нажата клавиша F4"
    Case vbKeyAdd:       Debug.Print "Нажата клавиша +"
    Case vbKeyEscape:    Debug.Print "Нажата клавиша Esc"
  End Select

    'Определяем, какие из трех клавиш клавиатуры (Shift, Ctrl, Alt) при этом были в нажатом состоянии:
  Select Case Shift
    Case 0:  Debug.Print "Не нажата ни одна клавиша Shift, Ctrl, Alt"
    Case 1:  Debug.Print "Нажата клавиша Shift"
    Case 2:  Debug.Print "Нажата клавиша Ctrl"
    Case 3:  Debug.Print "Нажаты клавиши Shift, Ctrl"
    Case 4:  Debug.Print "Нажата клавиша Alt"
    Case 5:  Debug.Print "Нажаты клавиши Shift, Alt"
    Case 6:  Debug.Print "Нажаты клавиши Ctrl, Alt"
    Case 7:  Debug.Print "Нажаты клавиши Shift, Ctrl, Alt"
  End Select
End Sub
```

Два параметра в скобках заголовка процедуры - это две вещи, которые компьютер сообщает процедуре в момент события:

- **KeyCode** - код нажатой клавиши. На клавиатуре - сто с лишним клавиш. На каждой - по две буквы или один, два, три других символа. Компьютер различает клавиши независимо от того, какие значки на них нанесены. Так мать различает сыновей независимо от того, что написано у них на майках. У каждого сына есть имя, у каждой клавиши есть код (KeyCode). Итак, попросту говоря, компьютер сообщает процедуре, какая клавиша была нажата. Полный список кодов находится в Object Browser (класс KeyCodeConstants).
- **Shift** - были ли при этом в нажатом состоянии клавиши клавиатуры (Shift, Ctrl, Alt) и какие именно. В численном значении Shift та же система, что и в событии MouseDown. Совместное нажатие клавиш Shift, Ctrl, Alt "равно" сумме их нажатий поодиночке (1, 2 и 4).

Запустите проект и проверьте, как работает программа. Обратите внимание, что при удержании клавиш в нажатом состоя-

нии событие генерируется (создается компьютером) несколько раз в секунду. Положение несколько меняется, когда при нажатой Shift, Ctrl или Alt щелкается обычная клавиша. Впрочем, пока это неважно.

События, связанные с клавиатурой, имеются у многих объектов. Поместим на форму, к примеру, пару кнопок и текстовое поле. Предположим, процедуру Private Sub Form_KeyDown мы стерли, а написали три процедуры:

Private Sub Command1_KeyDown…

Private Sub Command2_KeyDown…

Private Sub Text1_KeyDown…

Запустим программу и щелкнем по какой-нибудь клавише. Какая из трех процедур сработает? Та, чей объект находится в фокусе. Для нашей игры это неудобно: если мы для программирования реакции автомобиля на нажатия клавиш выберем, например, процедуру Private Sub Command2_KeyDown, то во время гонки мы не сможем щелкать по другим кнопкам, кроме Command2, так как иначе Command2 выйдет из фокуса и автомобиль перестанет реагировать на клавиши. Не надо было стирать процедуру Private Sub Form_KeyDown, восстановим ее. Но это не помогает. По простой причине - один какой-нибудь объект на форме всегда находится в фокусе, так что до процедуры Private Sub Form_KeyDown дело никак не доходит. Против этого в Visual Basic есть специальный прием - свойство формы **KeyPreview** устанавливается в True. Это означает приказ компьютеру почти каждое нажатие на клавиши считать событием формы, а не другого объекта. Вдобавок к этому советую поместить в самый конец процедуры Private Sub Form_KeyDown оператор KeyCode=0. Тоже хорошо помогает (без комментариев).

Что значит "почти каждое нажатие"? Есть исключения - клавиши Enter, Esc, Tab в некоторых случаях.

Событие **KeyUp** работает аналогично.

Задание 112 “Светофор”: Нарисуйте светофор: прямоугольник и три круга. При нажатии на клавиатуре клавиши R светофор должен загораться красным светом, G - зеленым, Y - желтым. Здесь таймер не нужен.

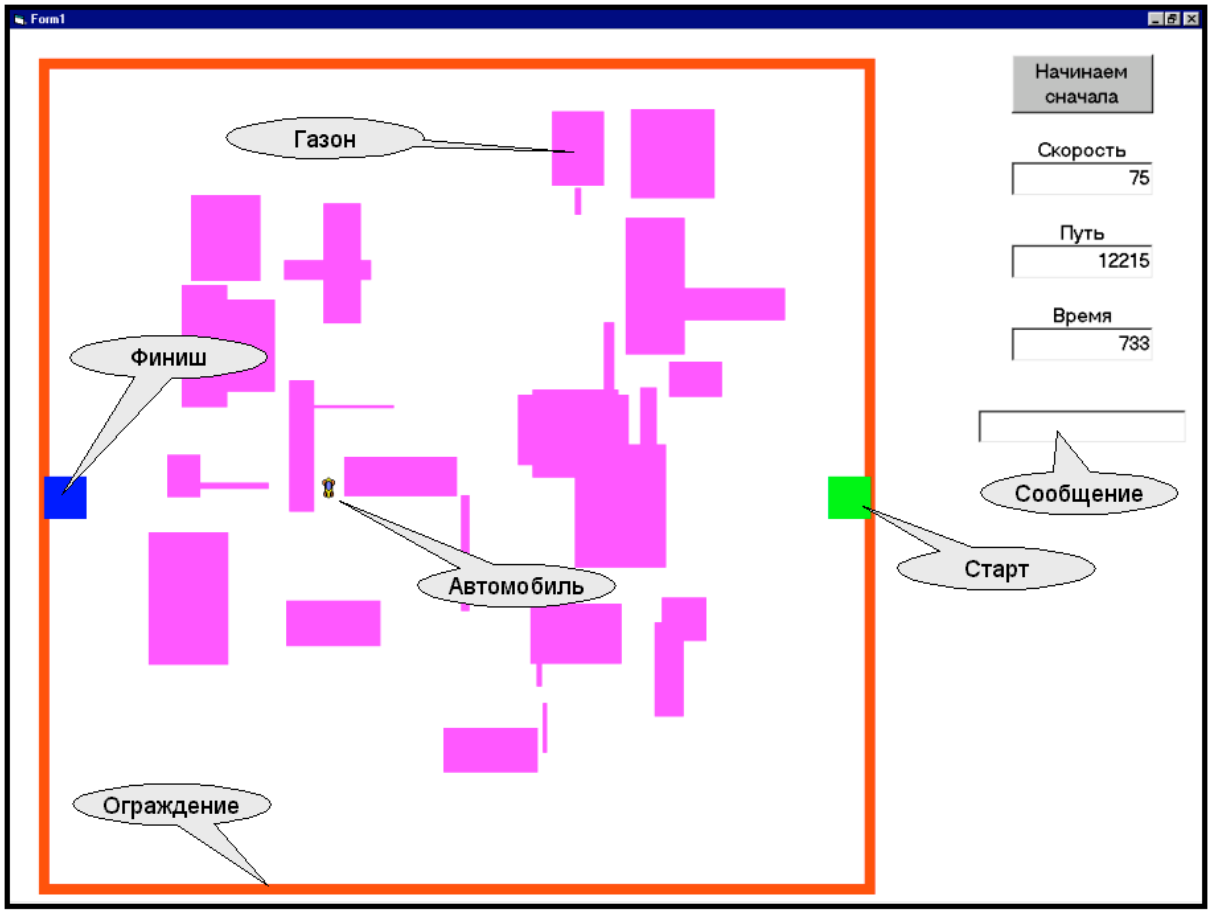
Задание 113 “Зенитка”: Вверху справа налево медленно движется вражеский самолет. В подходящий момент вы нажатием любой клавиши запускаете снизу вверх зенитный снаряд. Здесь нужны два таймера.

12.3. Проект - Гонки (игра)

Мы с вами разобрали уже два больших проекта: "Калькулятор" и "Будильник". Но "Калькулятор" можно не считать, на нем мы просто знакомились с Visual Basic. Выходит, "Будильник" - единственный большой проект, который написан более-менее правильно. И этого, конечно, мало. Вам нужен опыт создания проектов, в частности проектов с движением объектов по форме, да к тому же под управлением мыши и клавиатуры. Поэтому я решил разобрать с вами еще один проект (строк на 180). На этот раз это будет игра.

Постановка задачи

Вот внешний вид игры (о красотах я не заботился, красоты вы всегда сможете добавить по вкусу):



Процесс игры таков. Вы нажимаете на кнопку Начинаем сначала. На форме рисуется квадратное поле для гонки со случайно расположенными газонами. Нажатием на клавишу пробела вы даете газ и машина со старта набирает скорость. Ваша цель - любым путем побыстрее добраться до финиша. На белом асфальте вы можете газовать до любой скорости. Если же ненароком попадете на газон, то на газоне ваша скорость будет очень мала (я выбрал 15). Поэтому имеет смысл по газонам ездить пореже. Направление движения может быть только горизонтальное и вертикальное, наискосок машина не движется. Выбор направления (руль) - это клавиши со стрелками. Тормоз - клавиша B (английская). Тормозить надо по той причине, что на слишком большой скорости руль отказывает (это не потому, что проект плохой, а потому что я так придумал). В ограждение врезаться нельзя - катастрофа. Когда приедете на финиш, посмотрите на счетчик времени - это ваш результат. Снова нажимаете на кнопку Начинаем сначала. Теперь расположение газонов будет другим. Сажайте за клавиатуру приятеля и смотрите, не покажет ли он время лучшее, чем ваше. Можете посмотреть, кто из вас покажет лучшее время из 10 заездов. И тому подобное.

Не удивляйтесь, что скорость на спидометре не равна пути, деленному на время. Ведь это мгновенная (то есть настоящая скорость в данное мгновение), а не средняя скорость автомобиля (см. Физика, 9 класс).

Все эти правила я старался сделать как можно проще, чтобы не усложнять проект. Разобравшись в проекте, вы всегда сможете дописать процедуры, делающие процесс гонки более для вас привлекательным.

Для проекта я выбрал вариант игры с одним автомобилем. Я вам намекну, как модифицировать проект, чтобы получилась игра с двумя автомобилями. Однако помните, что для грамотного создания игры с несколькими автомобилями вам нужно будет подняться на новый уровень программирования - научиться создавать собственные классы объектов.

Делим проект на части

Начнем создавать проект. Прежде всего, как положено, разделим его по возможности на части. Мы уже делили проект на части, когда создавали будильник. Сейчас вам очень полезно перечитать тот материал. И проглядеть ту программу.

Сразу же бросается в глаза, что наша задача распадается на две последовательные почти независимые части:

1. Сначала рисование поля и всех его элементов, выкатывание машины на старт, обнуление скорости, времени и пути - в общем, все то, что должно быть сделано после нажатия на кнопку Начинаем сначала, но до начала движения. Этим будет заниматься одна группа процедур.
2. Затем управление машиной и событиями во время гонки. Этим будет заниматься другая группа процедур. Мы полностью задействуем идею использования таймера, так, как я ее изложил в 11.3. На каждом импульсе таймера автомобиль должен будет проделывать весь цикл своего функционирования.

Первая часть

Поскольку в первой части нет движения, то, в общем, все равно, что именно рисовать сначала, что потом, но и здесь следование житейской логике дает некоторую экономию кода и смысла:

1. Сначала нарисуем большой квадрат поля (ограждение), затем
2. Рисуем старт

3. Рисуем финиш
4. Рисуем газоны
5. Ставим машину на старт и обнуляем показания приборов

В общем, идеология первой части ясна. Пора разбираться в полном тексте программы, приведенном ниже. Здесь вам бы очень помогло, если бы программа уже находилась в вашем компьютере. Вам было бы легко и приятно в такт моим словам нажимать клавишу F8.

Начнем сверху. Разберитесь в объявленных переменных, не вникая пока в те, что относятся к движению.

Разобрались? Теперь идем дальше. Первая процедура, которая выполняется в проекте, это, конечно, Form_Load. В каждом проекте в ней удобно программировать все действия и вычисления, которые нужно выполнить один раз за все время работы проекта, в самом начале. В нашем случае это в основном задание и вычисление значений переменных величин - размеров элементов поля игры. Кое-что здесь нуждается в пояснении:

WindowState - это не переменная, а свойство формы. Вы помните, что свойства формы, в отличие от свойств других объектов, можно писать, не указывая хозяина (Form1.WindowState).

ScaleLeft и **ScaleTop**. Это тоже свойства формы. Мы их устанавливаем, если нам не нравится, что начало системы координат находится в левом верхнем углу формы. ScaleLeft занимается смещением начала координат по горизонтали, ScaleTop - по вертикали. Примеры:

Оператор	Смысл
ScaleLeft = 1000	Начало системы координат смещается налево на 1000 твипов
ScaleLeft = -500	Начало системы координат смещается направо на 500 твипов
ScaleTop = 800	Начало системы координат смещается вверх на 800 твипов
ScaleTop = -500	Начало системы координат смещается вниз на 500 твипов

В нашем проекте мы, конечно, могли бы обойтись и без этого, но многие операторы рисования у нас запишутся гораздо проще, если начало координат находится в левом верхнем углу поля игры, а не формы.

Все числа, которые встречаются в программе, я подобрал на опыте, исходя из размеров экрана и быстродействия своего компьютера. Вы можете эти числа изменить и посмотреть, что получится.

После выполнения процедуры Form_Load вы видите распахнутую на весь экран пустую форму с четырьмя текстовыми полями, тремя метками и кнопкой, ждущей нажатия. После щелчка по этой кнопке должна в полном объеме выполнится первая часть нашего проекта, так, как она описана выше. Заглянем в процедуру Кнопка_начинай_сначала_Click.

Здесь процедуры рисования ограждения, старта и финиша ясны без пояснений. Поясню процедуру Рисуем_газоны. Вы видите, что некоторые переменные я объявил внутри процедуры. Так поступают, когда знают, что значения этих переменных нигде, кроме как в этой процедуре, не нужны. Значения чисел и вид формул для X_газона и Y_газона я выбрал так, чтобы газоны не накладывались ни на ограждения, ни на старт с финишем.

Процедуру Ставим_машину_на_старт я поясню во второй части, так как она больше относится к ней.

Оператор Спидометр.SetFocus, появился вот почему. При запуске проекта фокус автоматически устанавливается на кнопку "Начинаем сначала". Кнопка в фокусе перехватывает нажатия на стрелки, в результате чего первое нажатие во время гонки на клавишу со стрелкой приводило не к выбору направления машиной, а к перескакиванию фокуса с кнопки на другой объект формы. Оператор Спидометр.SetFocus просто увел фокус с кнопки.

Вот, в общем, и все, что касается первой части. После ее выполнения мы увидим все, что нужно для старта, процедура Ставим_машину_на_старт поставит машину на старт, мотор будет заведен, вам останется только коснуться руля или педалей, чтобы началась часть вторая. Перейдем к ней.

Вторая часть

Если с первой частью все просто, то про вторую стоит поговорить подробнее. Фактически, нам нужно будет создавать автомобиль, как раньше мы создавали будильник. Не имея программистского опыта, мы попытаемся использовать житейский опыт касательно того, как автомобиль устроен. Причем применительно к задачам проекта. Так, цвет сиденья нам не очень важен в этом смысле. А важно нам управлять скоростью и направлением движения (этим в обычном автомобиле занимаются руль и педали газа и тормоза). А еще важно, чтобы автомобиль чувствовал, "куда он въехал" (газон, ограждение, финиш) и вел себя соответственно.

Итак, на каждом импульсе таймера автомобиль должен будет проделать весь цикл своего функционирования:

- Определить, где он находится (асфальт, газон, ограждение, финиш) и действовать соответственно.
- Изменить или не изменить скорость в соответствии с приказами клавиатуры.
- Изменить или не изменить в соответствии с приказами клавиатуры направление движения и сделать очередной шаг в нужном направлении.
- Изменить нужным образом показания приборов на пульте управления.

Но прежде, чем заниматься всем этим, давайте возьмем быка за рога и сразу посмотрим, как с клавиатуры примитивно управлять движением автомобиля.

Создайте отдельный пустой проект и поместите на форму объект Image. Дайте ему имя Image_авто. Чтобы он был виден в режиме работы, придайте значение его свойству BorderStyle. Он будет слушаться клавиш со стрелками при помощи такой программы:

```
Dim x As Integer                    'Горизонтальная координата
Dim y As Integer                    'Вертикальная координата

Private Sub Form_Load()
   x = Image_авто.Left
   y = Image_авто.Top
```

```
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
  Select Case KeyCode
    Case vbKeyRight:        x = x + 100
    Case vbKeyLeft:         x = x - 100
    Case vbKeyDown:         y = y + 100
    Case vbKeyUp:           y = y - 100
  End Select
  Image_авто.Left = x
  Image_авто.Top = y
End Sub
```

Эта программа только иллюстрирует идею, но для проекта она не годится. Во-первых, потому, что для непрерывного движения объекта необходимо клавишу держать все время нажатой, что создает трудности при обработке нажатия в это время других клавиш. Во-вторых, сложные проекты с таймерами всегда должны помнить, куда они едут, а для этого нужна переменная величина. Есть и другие причины.

Создадим переменную, назовем ее Руль и сделаем перечислимой (см. объявления в тексте программы). Она выполнит предназначенную ей роль - помнить, куда мы едем. Создадим также перечислимую переменную Педаль (см. там же). Она будет хранить информацию о том, нажали ли мы педаль газа, тормоза или вообще на педаль не нажимали. Теперь разберемся с процедурой Form_KeyDown. Вы видите, что переменная Руль будет чувстврвать нажатие на стрелки только тогда, когда скорость движения автомобиля меньше пороговой. О том, что на педали пока не нажимали, сообщает компьютеру процедура Ставим_машину_на_старт. Именно поэтому после нажатия кнопки Начинаем сначала машина сама не срывается со старта, несмотря на работающий мотор. Оператор Клавиатура_сработала = True нужен для того, чтобы запустить счетчик времени

If Клавиатура_сработала Then Время = Время + 1

в процедуре Отображаем_информацию).

Итак, при нажатии нужных клавиш меняются значения нужных переменных. Но дела никакого не происходит. О деле чуть позже, а сейчас давайте подумаем, как сделать так, чтобы автомобиль глядел в ту же сторону, куда он едет. Есть всего 4 направления движения, значит и 4 ориентации автомобиля. Нужно всего лишь взять 4 картинки автомобиля с разной ориентацией и вовремя их менять. Откуда взять картинки? Нарисовать одну в графическом редакторе, а там ее легко поворачивать в нужном направлении и сохраняться. Сохранитесь в 4 файлах. Если вы не хотите рисовать, возьмите из папки с иконками Visual Basic, о которой я уже говорил, 4 иконки в виде стрелок, направленных в разные стороны. Вот вам и автомобиль.

Затем поместите на форму 4 объекта Image, дайте им имена Image_вверх, Image_вниз, Image_налево, Image_направо и придайте им соответствующие картинки. Чтобы эти объекты не мозолили глаза, сделайте их невидимыми. Вся механика будет заключаться в том, что когда вы нажимаете на клавиатуре стрелку вверх, должен выполняться оператор

Image_авто.Picture = Image_вверх.Picture

а когда вниз - то оператор

Image_авто.Picture = Image_вниз.Picture

И так далее.

Теперь можно разбираться в тексте процедур.

Мы с вами чуть выше определили четыре вещи, которые должен проделать автомобиль на каждом импульсе таймера. Посмотрите в процедуру Timer1_Timer и найдите там обращения к соответствующим процедурам. Из них нас сейчас интересует процедура Выбираем_куда_ехать_и_делаем_шаг. Именно она задает направление движения. Загляните в нее. Ее дело - чувствовать одно из 4 значений переменной Руль и запускать соответствующую из 4 процедур. Каждая из этих процедур делает две вещи: поворачивает машину в нужном направлении и меняет в этом направлении ее координату. Само же изображение автомобиля прыгнет на указанную координату чуть позже - во время выполнения процедуры Показываем_автомобиль.

Теперь посмотрим, как регулируется скорость. Прежде всего, при нажатии кнопки Начинаем сначала скорость устанавливается в ноль процедурой Ставим_машину_на_старт. В процессе гонки скорость регулируется процедурой Изменяем_скорость. Действие ее полностью определяется значением переменной Педаль. Если это газ, то на данном такте таймера скорость возрастет на 5. Если тормоз - упадет на 10 (потому что тормоз обычно действует сильнее газа). Как видите, значение переменной Педаль в конце процедуры принудительно приравнивается никакой педали. Это значит, что на следующем такте таймера скорость не изменится. Чтобы она изменилась, нам нужно еще раз нажать на клавишу газа или тормоза. Обычно поступают по-другому - просто удерживают клавишу нажатой, при этом событие KeyDown возникает несколько раз в секунду и скорость меняется достаточно быстро. Это соответствует механике реального автомобиля - чтобы набирать скорость, нужно непрерывно держать нажатой педаль газа, а чтобы тормозить - тормоза.

Процедура Отображаем_информацию заботится о том, чтобы на каждом такте в текстовых полях Спидометр, txtПуть и txtВремя были правильные цифры. В нашем проекте переменная Скорость только называется скоростью, а на самом деле это расстояние, на которое перемещается машина на одном такте таймера. Поэтому для вычисления суммарного пути, пройденного автомобилем, вполне уместен оператор Путь = Путь + Скорость. Переменная Время тоже является не временем в секундах, а количеством тактов таймера, прошедших со старта.

Теперь о процедуре Определяем_где_мы. Ее задача - задать реакцию автомобиля на три ситуации: попадание на газон, на финиш и на полосу ограждения. Метод, при помощи которого автомобиль определяет, где он, самый простой - Point. Его мы разобрали в 9.7. Поскольку газон я рисовал сиреневым (vbMagenta), финиш - синим, а ограждение - красным, то метод Point выдает на них разные результаты. Оператор Select эти результаты анализирует и задает реакцию автомобиля. Как видите, на сиреневое компьютер реагирует только установкой скорости = 15, на два других цвета он выдает сообщение в текстовое поле Сообщение, выключает мотор и устанавливает переменную Приехали, чтобы компьютер ее тут же проанализировал (см. процедуру таймера) и вышел из процедуры таймера, так как, когда приехали, делать, естественно, больше нечего.

В процедуре Ставим_машину_на_старт вы вполне сможете разобраться по комментариям самостоятельно.

Ну вот, кажется, и все пояснения.

Текст программы

```
Dim x As Integer                                    'Горизонтальная координата автомобиля
Dim y As Integer                                    'Вертикальная координата автомобиля
Dim X_старта As Integer                             'Координаты старта и финиша
Dim Y_старта As Integer
Dim X_финиша As Integer
Dim Y_финиша As Integer
Dim Размер_старта As Integer                        'Старт - квадрат, это сторона квадрата
Dim Размер_финиша As Integer                        'Финиш - квадрат, это сторона квадрата
Dim Число_газонов As Integer                        'Каждый газон - это случайный прямоугольник в случайном месте
Dim Максимальный_размер_газона As Integer
Dim Размер_поля As Integer                          'Поле - квадрат, это сторона квадрата
Dim Отступ_поля_от_края_формы As Integer            'Имеется в виду отступ слева и сверху
Dim Путь As Long                                    'Путь автомобиля с момента старта (в твипах)
Dim Время As Integer                                'Время измеряется количеством импульсов таймера с момента щелчка по кнопке начала
Dim Скорость As Integer                 'Скорость - это не скорость, а расстояние. Она численно равна шагу автомобиля на каждом такте таймера
Dim Пороговая_скорость As Integer                   'Скорость, выше которой не работает руль
Dim Цвет_под_автомобилем As Long                    'Нужен чтобы знать, где находится автомобиль - на газоне, на финише, врезался в ограждение

Private Enum типРуль
   вверх
   влево
   вниз
   вправо
End Enum
Dim Руль As типРуль

Private Enum типПедаль
   тормоз
   газ
   ни_та_ни_эта
End Enum
Dim Педаль As типПедаль

Dim Приехали As Boolean                             'Приехали = True, когда приехали на финиш или врезались в ограждение
Dim Клавиатура_сработала As Boolean                 'Чтобы секундомер судьи запускался автоматически, когда мы стартуем, не раньше

Private Sub Form_Load()
   Timer1.Enabled = False                           'Нечего мотору зря работать до начала гонки
   Randomize                                        'Нам придется рисовать случайные газоны
   WindowState = 2                                  'Окно игры распахнуто на весь экран
   BackColor = vbWhite                              'Асфальт белый
                        'Задаем и вычисляем размеры элементов поля (все числа подбирайте еа опыте):
   Отступ_поля_от_края_формы = 500
   ScaleLeft = -Отступ_поля_от_края_формы           'Смещаем для удобства начало системы координат в левый верхний угол поля
   ScaleTop = -Отступ_поля_от_края_формы
   Размер_поля = Screen.Height - 2 * Отступ_поля_от_края_формы          'Отступ от края экрана должен быть и снизу
   Размер_старта = 600
   Размер_финиша = 600
   X_старта = Размер_поля - Размер_старта           'Чтобы старт находился у правой кромки поля
   X_финиша = 0                                     'Чтобы финиш находился у левой кромки поля
   Y_старта = Размер_поля / 2                       'Чтобы старт по высоте находился посредине поля
   Y_финиша = Размер_поля / 2                       'Чтобы финиш по высоте находился посредине поля
   Число_газонов = 30
   Максимальный_размер_газона = Размер_поля / 6
   Пороговая_скорость = 200
End Sub

Private Sub Кнопка_начинай_сначала_Click()
   Клавиатура_сработала = False                     'Мы еще не стартовали
   Timer1.Enabled = False                           'Нечего мотору зря работать до начала гонки
                 'Рисуем поле игры со всеми элементами:
   Cls                                              'Перед тем, как заново рисовать, надо все стереть
   Рисуем_границы_поля
   Рисуем_старт
   Рисуем_финиш
   Рисуем_газоны
                 'Необходимые начальные установки:
   Ставим_машину_на_старт
   Приехали = False
```

```
    Timer1.Enabled = True                          'Заводим мотор, скоро начало гонки
    Спидометр.SetFocus                             'Это чтобы фокус ушел с кнопки, а то первое нажатие на стрелку действует не так, как нам надо
End Sub

Private Sub Рисуем_границы_поля()
    DrawWidth = 10
    Line (0, 0)-(Размер_поля, Размер_поля), vbRed, B
    DrawWidth = 1                                  'Возвращаем нормальную толщину карандаша
End Sub

Private Sub Рисуем_старт()
    Line (X_старта, Y_старта)-(X_старта + Размер_старта, Y_старта + Размер_старта), vbGreen, BF
End Sub

Private Sub Рисуем_финиш()
    Line (X_финиша, Y_финиша)-(X_финиша + Размер_финиша, Y_финиша + Размер_финиша), vbBlue, BF
End Sub

Private Sub Рисуем_газоны()                        'Каждый газон - это прямоугольник случайного размера в случайном месте
    Dim i As Integer
    Dim X_газона As Integer                        'Горизонтальная координата верхнего левого угла газона
    Dim Y_газона As Integer                        'Вертикальная координата верхнего левого угла газона
    For i = 1 To Число_газонов                     'Числа в формулах подобраны на опыте:
        X_газона = 2 * Размер_финиша + Размер_поля * Rnd * 2 / 3
        Y_газона = Размер_финиша + (Размер_поля - Размер_финиша - Максимальный_размер_газона) * Rnd
        Line (X_газона, Y_газона)-(X_газона + Максимальный_размер_газона * Rnd, Y_газона + Максимальный_размер_газона * Rnd), vbMagenta, BF
    Next
End Sub

Private Sub Ставим_машину_на_старт()
    x = X_старта:              y = Y_старта        'Координаты машины приравниваются координатам точки старта
    Скорость = 0:    Путь = 0:    Время = 0
    Руль = влево                                   'Это чтобы машина знала, куда ехать, когда стартуем нажатием на газ
    Едем_влево                                     'Отсюда нам нужна только ориентация машины налево, а шага не будет, так как скорость=0
    Педаль = ни_та_ни_эта                          'Газовать пока нельзя, а тормозить бессмысленно
    Image_авто.Visible = True                      'До первого старта я сделал машину невидимой, теперь пришла пора ей появиться
    Показываем_автомобиль
End Sub

Private Sub Timer1_Timer()
    Определяем_где_мы
    If Приехали Then Exit Sub
    Изменяем_скорость
    Выбираем_куда_ехать_и_делаем_шаг
    Показываем_автомобиль
    Отображаем_информацию
End Sub

Private Sub Определяем_где_мы()
    Цвет_под_автомобилем = Point(x, y)
    Select Case Цвет_под_автомобилем
        Case vbMagenta:    Скорость = 15                                  'На газоне скорость мала
        Case vbBlue:       Сообщение.Text = "Финиш!":                     Timer1.Enabled = False:    Приехали = True
        Case vbRed:        Сообщение.Text = "Врезались в ограждение!":    Timer1.Enabled = False:    Приехали = True
    End Select
End Sub

Private Sub Изменяем_скорость()
    Select Case Педаль
        Case газ
            Скорость = Скорость + 5
        Case тормоз
            If Скорость > 0 Then Скорость = Скорость - 10     'потому, что тормоз быстрее газа
            If Скорость < 0 Then Скорость = 0         'В результате быстрого торможения скорость может стать отрицательной, что и предотвращается
    End Select
    Педаль = ни_та_ни_эта                          'Это чтобы во время набора скорости и торможения приходилось без перерыва жать на педаль
End Sub

Private Sub Выбираем_куда_ехать_и_делаем_шаг()
    Select Case Руль
        Case вверх:        Едем_вверх
        Case вниз:         Едем_вниз
        Case влево:        Едем_влево
        Case вправо:       Едем_вправо
```

```
    End Select
End Sub

Private Sub Едем_вверх()
    Image_авто.Picture = Image_вверх.Picture
    y = y - Скорость
End Sub

Private Sub Едем_вниз()
    Image_авто.Picture = Image_вниз.Picture
    y = y + Скорость
End Sub

Private Sub Едем_влево()
    Image_авто.Picture = Image_налево.Picture
    x = x - Скорость
End Sub

Private Sub Едем_вправо()
    Image_авто.Picture = Image_направо.Picture
    x = x + Скорость
End Sub

Private Sub Показываем_автомобиль()
    Image_авто.Left = x
    Image_авто.Top = y
End Sub

Private Sub Отображаем_информацию()
    Спидометр.Text = Скорость
    Путь = Путь + Скорость                          'Нужно помнить, что именем Скорость назван шаг автомобиля
    txtПуть.Text = Путь
    If Клавиатура_сработала Then Время = Время + 1  'Счетчик времени запускается только тогда, когда мы стартуем
    txtВремя.Text = Время
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    Клавиатура_сработала = True
    Select Case KeyCode
        Case vbKeyUp:       If Скорость < Пороговая_скорость Then Руль = вверх    'Поворот на большой скорости запрещен
        Case vbKeyLeft:     If Скорость < Пороговая_скорость Then Руль = влево
        Case vbKeyDown:     If Скорость < Пороговая_скорость Then Руль = вниз
        Case vbKeyRight:    If Скорость < Пороговая_скорость Then Руль = вправо
        Case vbKeySpace:    Педаль = газ
        Case vbKeyB:        Педаль = тормоз
    End Select
    KeyCode = 0                                     'Это чтобы объекты на форме не реагировали на клавиатуру
End Sub
```

Недоработки проекта

Замеченные мной недоработки вызваны в основном нежеланием увеличивать размер кода и заключаются в следующем:

- Координатами (x,y) машины считается ее левый верхний угол, а не центр. Причина этого - слишком простые операторы в процедуре Показываем_автомобиль. Это приводит к некоторой ассиметрии поведения машины при переезде с цвета на цвет. Немного повозившись, вы сможете исправить положение.
- При большой скорости машина делает огромные шаги от одного такта таймера к другому. А ведь цвет под собой она ощущает не непрерывно, а только на тактах таймера. Значит она может перепрыгнуть через тоненький газон, не заметив его. То же относится к финишу и особенно к ограждению. Сделайте их потолще, чем у меня на картинке, или ограничьте скорость.
- Сам бог велел вам добавить в проект управление машиной с помощью мыши. Например, щелчок мышью в стороне от машины вызывает поворот машины в эту сторону, удерживание нажатой левой клавиши мыши - газ, правой - тормоз. Для этого вам нужно будет написать процедуру MouseDown, придавая в ней нужные значения переменным Руль и Педаль аналогично тому, как это делает процедура KeyDown. Вам придется поразмыслить, как правильно сравнить координаты машины и мыши, чтобы добиться нужного результата.

Гонки двух автомобилей

Если вы хотите играть вместе с другом, вам нужны гонки двух машин. Несмотря на то, что ничего нового вам придумывать не придется, размер программы вырастет раза в полтора. Для второй машины нужен будет второй таймер. Вам придется иметь по два экземпляра переменных, описывающих поведение каждой машины. Например, вместо переменной Скорость вам придется иметь две переменные - Скорость1 и Скорость2. Вам придется иметь по два экземпляра процедур, задающих поведе-

ние каждой машины. Процедуру KeyDown вам придется дополнить новыми клавишами для управления второй машиной. При этом нужно будет внести серьезные изменения в педали, так как удерживать клавиши нажатыми теперь будет нельзя без усложнения программы. Вы могли бы сократить рост программы, используя процедуры с параметрами, но пока у вас для этого нет достаточных знаний. А если у вас гонки 10 автомобилей? Проблемы нарастают.

Visual Basic конечно же предлагает универсальное решение этой проблемы. Он, как и все серьезные и мощные языки программирования, является объектно-ориентированным языком. В частности, имеется в виду, что он позволяет программисту создавать **собственные классы объектов**. Вот у вас имеется класс объектов "кнопка" - CommandButton и вы пользуетесь в проекте отдельными экземплярами этого класса - объектами. Поведение кнопок запрограммировано глубоко внутри Visual Basic и вы видите только результаты этого программирования. Например, когда выполняется оператор

```
Command1.Visible = True
```

вы видите, что кнопка легко возникает на форме. Однако, чтобы она возникла, нужно чтобы компьютер ее нарисовал из прямоугольников и других элементов и написал что-то на ней, а для этого нужна программа.

Научившись создавать собственные классы, вы сможете создать класс "автомобиль" и пользоваться каким угодно числом экземпляров этого класса. Программу поведения автомобиля вы пишете сами, причем один раз и только для одного автомобиля. Остальные автомобили будут автоматически пользоваться ей же. Как все это делается, описывается в Глава 20.

12.4. Задание на игру “Торпедная атака

Задание 114: “*Торпедная атака*”. Начинается все красивой заставкой с названием игры. Компьютер спрашивает ваше имя, чтобы во время игры оно горело красивыми яркими буквами в углу экрана. В меню вы можете посмотреть правила игры, сведения об авторе, выбрать игру со звуком или без. Но вот игра началась. На экране морской пейзаж (нарисованный вами или взятый готовым). Вдали экрана по горизонтали плывет вражеский корабль. У нижнего края экрана притаился ваш торпедный аппарат. В подходящий момент времени вы нажимаете клавишу - и торпеда плывет вдаль, уменьшаясь по мере удаления. Если вы попали, то экран озаряется вспышкой от взрыва, на мгновение виден и сам взрыв, раздается коротенькая радостная мелодия, на экране - коротенький поздравительный текст, счетчик подбитых кораблей на экране увеличивается на 1. Если не попали, то зрительные и звуковые эффекты - совсем другие. В любом случае уменьшается на 1 счетчик оставшихся торпед. Затем плывет следующий корабль. И так далее. Когда торпеды у вас кончатся (скажем, их было 10), игра заканчивается. Программа анализирует ваши успехи и в зависимости от них выдает на экран текст, скажем “Мазила!”, если вы не попали ни разу из 10, или “Профи!”, если вы попали 8 раз. Затем спрашивает, будете ли вы играть еще.

Помощь: Как компьютер определит, попал или не попал? Нужно в тот момент, когда торпеда доплывет до линии движения корабля, сравнить горизонтальные координаты корабля и торпеды, и если они достаточно близки, считать, что попал.

Улучшение. Если у всех кораблей будет одинаковая скорость, то попадать будет слишком просто, а значит и играть неинтересно. Сделайте скорость кораблей случайной. Конечно, не совсем уж (скажем, в условных единицах скорости диапазон от 0 до 10 – это слишком), а в пределах разумного (скажем, от 4 до 8 – это нормально). Причем не нужно менять скорость одного и того же корабля в процессе движения. Пусть она остается постоянной, а то все будет зависеть не от вашего мастерства, а от везения. Различаются скорости только разных кораблей.

Пусть игрок сможет выбирать из нескольких уровней трудности. Трудность удобнее всего увеличивать, уменьшая размеры корабля, то есть требуемую величину близости координат корабля и торпеды при определении попадания.

Мы с вами закончили первый из двух циклов знакомства с Visual Basic. Если вам удалась «Торпедная атака» и она работает не хуже, чем указано в задании, то у вас должна появиться уверенность, что теперь, умело разбивая программы проекта на небольшие процедуры, вы можете создавать проекты любой сложности, а значит цель этого цикла достигнута. Я вас поздравляю - вам присваивается звание “*Программист-любитель III ранга*”!

Часть III. Программирование на Visual Basic - второй уровень

Если вам кажется, что вы уже все можете, то вы правы и неправы. Правы потому, что вы можете написать сколь угодно большую программу, разбив ее на разумное число процедур. Неправы потому, что ваша способность манипулировать данными разных типов еще очень ограничена. А без нее вам не поддадутся «умные» задачи. Например, не познакомившись с так называемыми массивами, вы не сможете запрограммировать игру в крестики-нолики или морской бой. Не освоив работу со строками, вы не сможете решить задачу о мало-мальски серьезной шифровке и расшифровке секретных сообщений. Не умея работать с файлами, вы не сможете сохраняться в созданных вами играх. Вам нужно освоить начала объектного программирования, познакомиться с базами данных, с работой в Интернете и с другими вещами.

Если в предыдущей части главной целью было научить вас программировать сложные проекты для работы с простыми данными, то теперь главная цель - программировать простые проекты для работы со сложными данными. В этой части я предполагаю, что предыдущую часть вы прочитали и задания выполнили.

Если в предыдущей части я вводил новые элементы Visual Basic бессистемно, только тогда, когда они нужны были в процессе изучения программирования, то сейчас я буду вводить их систематически и рассматривать их более полно, независимо от того, нужны они вам сейчас или пригодятся позже.

Глава 13. Массивы

Массивы - одно из главных средств хранения в памяти компьютера больших объемов информации.

Для того, чтобы понять массивы, нужно обладать некоторой культурой математического мышления. Если этот материал покажется вам трудным, не поддавайтесь искушению пропустить его. Настоящего программирования без массивов не бывает, да и большая часть дальнейшего материала без массивов не будет понятна.

В основе массивов лежит понятие индекса.

13.1. Переменные с индексами

В математике широко применяются так называемые **индексированные переменные**. На бумаге они записываются так:

x_1 x_2 b_8 y_i y_{i-6} z_{ij} z_{i+1j}

а читаются так: икс первое, икс второе, бэ восьмое, игрек итое, игрек и минус шестое, зет итое житое, зет и плюс первое житое. Все эти маленькие подстрочные цифры и выражения называются **индексами**. Поскольку в алфавите Visual Basic нет подстрочных букв и цифр, то те же индексированные переменные в Visual Basic приходится обозначать так:

X(1) X(2) B(8) Y(i) Y(i-6) Z(i,j) Z(i+1,j)

Зачем математикам нужны индексированные переменные? Их удобно применять хотя бы при операциях над числовыми рядами. Числовой ряд – это просто несколько чисел, выстроенных по порядку одно за другим. Чисел в ряду может быть много и даже бесконечно много.

Возьмем, например, бесконечный ряд *чисел Фибоначчи*: 1 1 2 3 5 8 13 21 34..... Попробуйте догадаться, по какому закону образуются эти числа. Если вы сами не догадались, то я подскажу: каждое из чисел, начиная с третьего, является суммой двух предыдущих. А теперь попробуем записать это утверждение с помощью языка математики. Для этого обозначим каждое из чисел Фибоначчи индексированной переменной таким образом:

Первое число Фибоначчи обозначим так: f(1),

Второе число Фибоначчи обозначим так: f(2) и т.д.

Тогда можно записать, что f(1)=1 f(2)=1 f(3)=2 f(4)=3 f(5)=5 f(6)=8 .....

Очевидно, что f(3)=f(1)+f(2),

f(4)=f(2)+f(3),

f(5)=f(3)+f(4) и т.д.

Как математически одной формулой записать тот факт, что *каждое* из чисел является суммой двух предыдущих? Математики в индексном виде записывают это так:

f(i)=f(i-2)+f(i-1).

Для пояснения подставим вместо i любое число, например, 6. Тогда получится:

f(6)=f(6-2)+f(6-1) или

f(6)=f(4)+f(5), что соответствует определению чисел Фибоначчи.

Задание 115: Запишите в индексном виде, как получается из предыдущего числа ряда последующее:

1) 14 18 22 26 .....
2) 6 12 24 48 ....
3) 3 5 9 17 33 65 ....

Вот еще примеры, когда математики предпочитают использовать индексы. Пусть мы на протяжении года каждый день раз в сутки измеряли температуру за окном. Тогда вполне естественно обозначить через t(1) температуру первого дня года, t(2) - второго,....., t(365) - последнего. Пусть 35 спортсменов прыгали в высоту. Тогда через h(1) можно обозначить высоту, взятую первым прыгуном, h(2) - вторым и т.д.

13.2. Одномерные массивы переменных величин

Одна из типичных задач программирования формулируется примерно так. Имеется большое количество данных, например, тех же температур или высот. С этими данными компьютер должен что-нибудь сделать, например, вычислить среднегодовую температуру, количество морозных дней, максимальную взятую высоту и т.п. Раньше мы уже вычисляли подобные вещи, и при этом данные вводили в компьютер с клавиатуры одно за другим в одну и ту же ячейку памяти (см. Глава 8). Однако, программистская практика показывает, что удобно, а часто и необходимо иметь данные в оперативной памяти сразу все, а не по очереди. Тогда для задачи про температуру нам понадобится 365 ячеек. Эти 365 ячеек мы и назовем массивом. Итак, **массивом** можно назвать ряд ячеек памяти, отведенных для хранения значений индексированной переменной. Вопрос о том, как большое количество значений оказывается в памяти, отложим на будущее (16.2).

Знаете, что напоминает одномерный массив? Список ListBox, который мы проходили в 15.5. Такой же аккуратный ряд пронумерованных элементов.

Рассмотрим на простом примере, как Visual Basic управляется с массивами. Предположим, в зоопарке живут три удава. Известна длина каждого удава в сантиметрах (500, 400 и 600). Какая длина получится у трех удавов, вытянутых в линию?

Обозначим длину первого удава - dlina(1), второго - dlina(2), третьего - dlina(3). Прикажем Visual Basic отвести под эту индексированную переменную массив ячеек в памяти:

Dim dlina **(1 To 3)** As Integer

Здесь 1 - **нижняя граница индекса**, 3 - **верхняя граница индекса**. Слово **To** обозначает **до**. В целом эту строку можно перевести так: Отвести в памяти под переменную dlina ряд ячеек типа Integer, пронумерованных **от** 1 **до** 3.

Вот программа полностью:

```
Dim dlina(1 To 3)   As Integer
Dim summa           As Integer
Private Sub Command1_Click()
   dlina(1) = 500
   dlina(2) = 400
   dlina(3) = 600
         'В этот момент в трех ячейках памяти уже находятся числа
         'и с ними можно выполнять арифметические действия
   summa = dlina(1) + dlina(2) + dlina(3)
   Debug.Print summa
End Sub
```

Если смысл написанного выше вам неясен, запустите отладочный пошаговый режим выполнения программы и загляните в текущие значения dlina(1), dlina(2), dlina(3), summa.

Теперь запишем ту же программу в предположении, что длины удавов заранее неизвестны и мы их вводим при помощи InputBox:

```
Dim dlina(1 To 3)   As Integer
Private Sub Command1_Click()
   dlina(1) = InputBox("Введите длину 1-го удава")
   dlina(2) = InputBox("Введите длину 2-го удава")
   dlina(3) = InputBox("Введите длину 3-го удава")
   Debug.Print dlina(1) + dlina(2) + dlina(3)
End Sub
```

Вопрос: Что напечатает следующий фрагмент:

```
i = 2:    a(3) = 10:    a(i) = 100:    a(i + 6) = a(i) + a(5 - i):   i = 0:   Debug.Print a(i + 3) + a(2) + a(i + 8)
```

Ответ: 220

Пояснение: i = 2 ; a(3) = 10 ; a(2) = 100 ; a(2 + 6) = a(2) + a(5 - 2) ; i = 0 ; Debug.Print a(0 + 3) + a(2) + a(0 + 8)

Теперь решим ту же задачу про удавов в предположении, что удавов не три, а тысяча:

```
Dim dlina(1 To 1000)       As Integer
Dim summa                  As Integer
Private Sub Command1_Click()
      'Вводим длины тысячи удавов, хоть это и утомительно и никто так не делает:
   For i = 1 To 1000
      dlina(i) = InputBox("Введите длину " & i & "-го удава")
   Next
      'Здесь на первом выполнении цикла i=1 и поэтому компьютер вводит число в ячейку dlina(1),
      'на втором -  i=2 и поэтому компьютер  вводит число в ячейку dlina(2) и т.д.
            'Определяем суммарную длину тысячи удавов:
   summa = 0
   For i = 1 To 1000
      summa = summa + dlina(i)
   Next
   Debug.Print summa
End Sub
```

Отлаживая эту программу, возьмите, конечно, вместо числа 1000 число 3.

Решим еще одну задачу. Дан ряд из 10 произвольных чисел: a(1), a(2), ... , a(10). Подсчитать и напечатать суммы восьми троек стоящих рядом чисел: a(1)+a(2)+a(3), a(2)+a(3)+a(4), a(3)+a(4)+a(5), ... , a(8)+a(9)+a(10).

```
Dim a(1 To 10) As Integer
Private Sub Command1_Click()
   a(1) = 23:  a(2) = 28:  a(3) = 4:  a(4) = 0:   a(5) = 12
   a(6) = 10:  a(7) = 23:  a(8) = 2:  a(9) = 9:   a(10) = 1
   For i = 1 To 8
      Debug.Print a(i) + a(i + 1) + a(i + 2)
   Next
End Sub
```

Задание 116:. Напишите с использованием массива программу вычисления среднегодовой температуры (Для отладки в компьютере годом можно считать неделю).

Задание 117:. Подсчитайте количество теплых дней в году (когда температура выше 20 град.).

Задание 118:. Каким по порядку идет самый жаркий день?

Задание 119:. Вычислить и распечатать первые 70 чисел Фибоначчи.

13.3. Двумерные массивы

Поясним суть двумерных массивов на простом примере. Пусть на целом ряде метеостанций, расположенных в разных точках земного шара, в течение многих дней измеряли температуру воздуха. Показания термометров свели в таблицу. Ограничимся для экономии места тремя станциями и четырьмя днями.

	1-й день	2-й день	3-й день	4-й день
Метеостанция 1	-8	-14	-19	-18
Метеостанция 2	25	28	26	20
Метеостанция 3	11	18	20	25

Требуется (в порядке возрастания трудности):

1) Распечатать температуру на 2-й метеостанции за 4-й день и на 3-й метеостанции за 1-й день.
2) Распечатать показания термометров всех метеостанций за 2-й день
3) Определить среднюю температуру на третьей метеостанции
4) Распечатать всю таблицу
5) Распечатать, в какие дни и на каких метеостанциях температура была в диапазоне 24-26 градусов тепла

Для этого обозначим показания термометров индексированной переменной с двумя индексами по следующей схеме:

t(1,1) t(1,2) t(1,3) t(1,4)
t(2,1) t(2,2) t(2,3) t(2,4)
t(3,1) t(3,2) t(3,3) t(3,4)

Обратите внимание, что первый индекс в скобках обозначает номер строки (метеостанции), второй - номер столбца (дня) прямоугольной таблицы. Такую таблицу математики называют *матрицей*.

В памяти отводим массив из 3*4=12 ячеек под значения типа Integer индексированной переменной t. Будем называть его **двумерным массивом:**

Dim t (**1 To 3, 1 To 4**) As Integer

Программа:

```
Dim t(1 To 3, 1 To 4) As Integer
Private Sub Command1_Click()
        'Зададим значения элементов массива примитивным присваиванием:
  t(1, 1) = -8:   t(1, 2) = -14:   t(1, 3) = -19:   t(1, 4) = -18
  t(2, 1) = 25:   t(2, 2) = 28:    t(2, 3) = 26:    t(2, 4) = 20
  t(3, 1) = 11:   t(3, 2) = 18:    t(3, 3) = 20:    t(3, 4) = 25

        'Выполняем 1 пункт задания:
  Debug.Print t(2, 4), t(3, 1)

        'А теперь распечатаем второй столбец массива (2 пункт задания):
  For i = 1 To 3:    Debug.Print t(i, 2):    Next

        'Определим среднее значение элементов третьей строки (3 пункт задания):
  i = 3
  s = 0
  For j = 1 To 4:    s = s + t(i, j):    Next
  Debug.Print s / 4

        'Распечатаем всю таблицу (4 пункт задания):
  For i = 1 To 3
    For j = 1 To 4
      Debug.Print t(i, j),
    Next j
    Debug.Print
  Next i

        'Распечатаем станции и дни с температурой 24-26 градусов (5 пункт задания):
  For i = 1 To 3
    For j = 1 To 4
      If t(i, j) >= 24 And t(i, j) <= 26 Then Debug.Print "Станция"; i; "день"; j
    Next j
  Next i
End Sub
```

Задание 120: Вычислить разницу между максимальной и минимальной температурой во всей таблице.

13.4. Какие бывают массивы

Массивы бывают не только числовые, но и строковые и типа Date и прочие. Подходит любой известный нам тип. Например:

Dim s(1 To 50) As String

Это означает, что в каждой из 50 ячеек должно находиться не число, а произвольная строка. Вот элементарный пример использования строкового массива:

```
Dim s(1 To 50) As String
Private Sub Command1_Click()
    s(21) = "Привет":                    Debug.Print s(21)
End Sub
```

Вот пример работы с массивами других типов:

```
Dim b(1 To 30, 1 To 6) As Boolean
Dim DT(1 To 10) As Date
Private Sub Command1_Click()
    b(2, 3) = False:                     Debug.Print b(2, 3)
    DT(2) = #1/15/2156 11:59:42 PM#:     Debug.Print DT(2)
End Sub
```

Еще пример:

```
Private Enum типРуль
    вверх
    влево
    вниз
    вправо
End Enum
Dim Руль(1 To 300) As типРуль
Private Sub Command1_Click()
    Руль(200) = вправо:                  Debug.Print Руль(200)
    Руль(220) = влево:                   Debug.Print Руль(220)
End Sub
```

Здесь будут напечатаны числа 3 и 1, так как элементы перечислимого типа пронумерованы по порядку от 0.

Границы индексов в круглых скобках тоже могут быть разными, например:

Dim a(20 To 60) As Integer

Здесь под числа отводится 41 ячейка. Еще пример:

Dim b(0 To 9, -20 To 30) As Integer

Здесь под числа отводится 10*51=510 ячеек.

Раз вы объявили границы индексов, то должны их придерживаться. Так, неправильно было бы теперь написать a(15) = 0.

Если нижняя граница индекса вашего массива равна нулю, то вы можете сэкономить усилия и вместо

Dim f**(0 To 9)** As Integer

писать

Dim f**(9)** As Integer

так как Visual Basic по умолчанию считает нижней границей индекса всех массивов число 0. Если же вам хочется, чтобы эта льгота касалась массивов с нижней границей равной не **0**, а **1**, то наверху программы напишите

Option base 1

Массивы могут быть одномерные, двумерные, трехмерные, четырехмерные и т.д.:

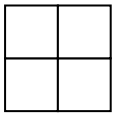
Dim a (1 To 10) As Integer	-одномерный массив	10 ячеек
Dim a (1 To 10, 1 To 5) As Integer	-двумерный массив	50 ячеек
Dim a (1 To 10, 1 To 5, 1 To 2) As Integer	-трехмерный массив	100 ячеек
Dim a (1 To 10, 1 To 5, 1 To 2, 1 To 3) As Integer	-четырехмерный массив	300 ячеек

13.5. Использование массивов при программировании игр

Какая польза от массивов при программировании игр? Вряд ли хоть одну «умную» игру можно запрограммировать без применения массивов. Возьмем хотя бы «крестики-нолики» на поле размером 3 на 3. Вам придется рисовать на экране большие клетки, а в них – нолики (кружочки) после ваших ходов и крестики (пересекающиеся косые линии) после ходов компьютера. Но этого недостаточно. Чтобы компьютер не поставил случайно крестик в занятом поле, он должен хотя бы знать, в каких клетках крестики и нолики уже стоят. Анализировать для этого информацию о пикселах экрана очень неудобно. Гораздо разумнее заранее организовать Dim a (1 To 3, 1 To 3) As Integer и записывать туда в нужные места нолики после ходов человека и, скажем, единички после ходов компьютера. Сразу же после записи в массив 0 или 1 программа должна рисовать в соответствующем месте экрана кружок или крестик. Мыслить компьютер мог бы при помощи примерно таких операторов – if a(1,1)=0 And a(1,2)=0 Then a(1,3)=1. Это очевидный защитный ход компьютера - на два кружочка в ряду он ставит в тот же ряд крестик.

Проиллюстрируем эту идею подробнее на специально придуманном примитивном примере (типа морского боя).

Задание на создание игры: Играют друг против друга два человека на квадратном поле размером 2 на 2:



Компьютер в игре участвует только как судья. Сначала первый игрок тайком от второго сообщает компьютеру, в каких двух клетках находятся его одноклеточные корабли (например, в правой верхней и правой нижней). Затем второй производит два выстрела (наугад – например, в левую верхнюю и правую нижнюю клетки). Если подбиты оба корабля - он выиграл, если один - ничья, если ни одного - проиграл.

Сначала запрограммируем эту игру без графики.

Поле боя после двух выстрелов будет показано на экране в следующем виде:

м к
о х

Здесь я использовал такие обозначения:

о - корабля здесь нет и сюда не стреляли
к - неподбитый корабль
х - подбитый корабль
м - мимо (стреляли и промахнулись)

Других вариантов быть не может. До окончания игры поле на экране не показывается.

Вот программа:

```
Dim a(1 To 2, 1 To 2) As String * 1        'Поле боя
Dim i As Integer
Dim j As Integer
Dim Подбито As Integer
_____________________________________________________________
Private Sub Command1_Click()               'Главная процедура
   a(1, 1) = "о":   a(2, 1) = "о"          'Поначалу на поле боя кораблей нет
   a(1, 2) = "о":   a(2, 2) = "о"

   Устанавливаем_корабль  1
   Устанавливаем_корабль  2

   Подбито = 0                             'Пока не стреляли
   Выстрел  1
   Выстрел  2
                                           'Показываем поле боя после битвы:
   For i = 1 To 2
      For j = 1 To 2
         Debug.Print a(i, j);
      Next j
      Debug.Print
   Next i
   Debug.Print Подбито                     'Показываем исход битвы
End Sub
_____________________________________________________________
Private Sub Устанавливаем_корабль(Номер_корабля As Integer)
   i = InputBox("Первый игрок, назовите номер строки корабля " & Номер_корабля)
   j = InputBox("Первый игрок, назовите номер столбца корабля " & Номер_корабля)
   a(i, j) = "к"
End Sub
_____________________________________________________________
Private Sub Выстрел(Номер_выстрела As Integer)
   i = InputBox("Второй игрок, назовите номер строки для выстрела " & Номер_выстрела)
   j = InputBox("Второй игрок, назовите номер столбца для выстрела " & Номер_выстрела)
   If a(i, j) = "к" Then                   'Если попал, то
      a(i, j) = "х"                        'ставим крестик
      Подбито = Подбито + 1                'и увеличиваем счетчик подбитых кораблей
   ElseIf a(i, j) = "о" Then               'иначе если промахнулся, то
      a(i, j) = "м"                        'ставим м
   End If
End Sub
```

Пояснения касательно Dim a(1 To 2, 1 To 2) As **String * 1** :

Конструкция String * 1 означает приказ компьютеру считать значения переменной a имеющими длину в 1 символ. Вообще-то можно было написать просто String, но так все-таки лучше, четче и экономнее. Теперь компьютер отведет в памяти под каждое значение a ровно 1 байт и не будет думать, хватит этого или не хватит.

Определение победителя оставляю вам.

Когда готова и работает программа без графики, можно подумать о том, как оживить ее графическими изображениями. Эффективнее всего создать единую процедуру, которая сначала оператором Cls все стирает с экрана, а затем все рисует снова согласно новым значениям массива a. Например, рисует во весь экран пустое поле из 4 клеток и тут же их заполняет: скажем, ставит маленькие кружочки туда, куда стреляли и промахнулись, красит красным клетки с подбитыми кораблями и синим

- с неподбитыми. Для заполнения достаточно в написанном уже фрагменте, показывающем поле

```
For i = 1 To 2
   For j = 1 To 2
      Debug.Print a(i, j);
   Next j
   Debug.Print
Next i
```

вместо Debug.Print a(i, j) написать некий код такого примерно смысла:

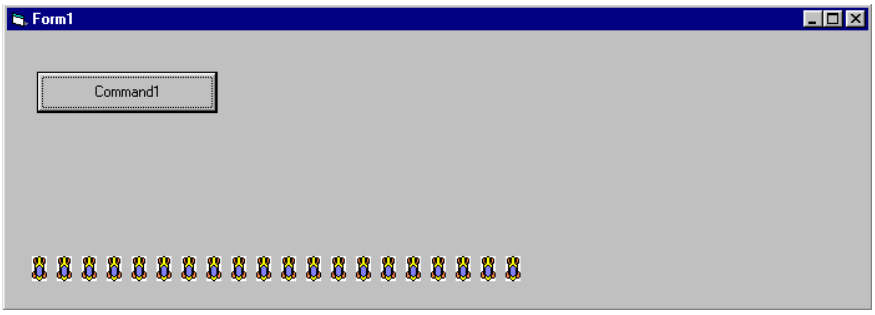
```
If a(i, j) = "х" Then Крась_красным  i  j
If a(i, j) = "к" Then Крась_синим  i  j
If a(i, j) = "м" Then Рисуй_кружок  i  j
```

Здесь мы видим три процедуры с параметрами i и j, от значения которых зависят координаты кружочков на форме и закрашиваемых квадратов. Зависимость эту продумайте сами.

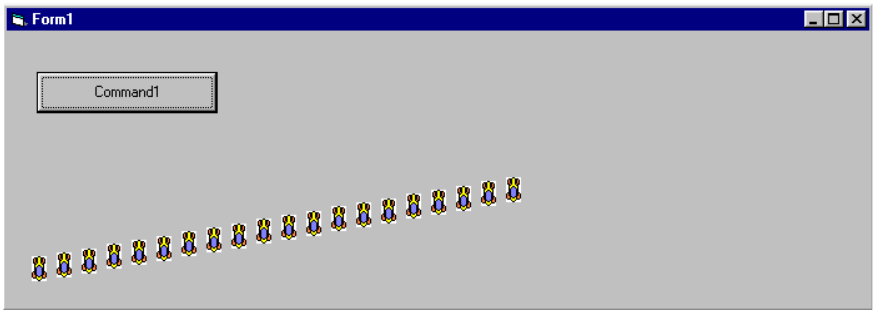
Что замечательно - в остальном программа от добавлении графики не изменится. Процедуру эту можно будет, не думая о графике, использовать всякий раз, когда мы хотим обновить поле на экране. Особенно это удобно в настоящем морском бое и настоящих крестиках-ноликах, когда поле надо обновлять после каждого хода.

13.6. Массивы элементов управления

Задача: Выстроить вдоль нижней кромки формы на равных расстояниях друг от друга 20 автомобилей, вот так:



При нажатии кнопки эти автомобили должны прыгнуть вверх, вот так:



(самый левый остается на месте).

Трудности: Нам нужно поместить на форму 20 элементов управления Image и загрузить в них картинку автомобиля, что само по себе утомительно. Расположить их вручную на равных расстояниях тоже нелегко. Visual Basic дает возможность изящно преодолеть все эти трудности программным путем.

Ваши действия: Поместите на форму один объект Image и дайте ему имя imgАвто. Загрузите в него картинку автомобиля. Обратите внимание на его свойство Index. Оно пусто. Это значит, что данный объект пока не является элементом массива объектов. Заголовок у окна свойств такой: Properties - imgАвто. Придайте свойству Index какое-нибудь целое неотрицательное значение, например 1. Тут же заголовок у окна свойств меняется: Properties - imgАвто(1). Это значит, что данный объект стал элементом **массива объектов**. Пока в этом массиве один элемент - imgАвто(1).

Как создать остальные объекты массива? Можно вручную, скопировав его обычным образом, как я объснял в 11.3, но отвечая на вопрос - Yes. При этом Visual Basic будет придавать новым объектам индексы от 0 и выше. Но быстрее это сделать программным путем. Так, оператор

Load imgАвто(4)

создаст (загрузит) на форме новый объект imgАвто с индексом 4. Значит, если применить оператор в цикле, он загрузит нам все нужные объекты и создаст их массив. Безо всякого объявления. Вот программа:

```
Dim i As Integer
Dim N As Integer          'Количество объектов
_____________________________________________________________
Private Sub Form_Load()
   N = 20
         'Загружаем на форму и делаем видимыми все объекты массива, кроме первого (который и так уже загружен):
   For i = 2 To N
```

```
        Load imgАвто(i)
        imgАвто(i).Visible = True
    Next
        'Помещаем на свои места все объекты массива:
    For i = 1 To N
        imgАвто(i).Top = Height - 1000
        imgАвто(i).Left = 300 * i
    Next
End Sub
_____________________________________________________________
Private Sub Command1_Click()
        'Прыжок всех объектов:
    For i = 1 To N
        imgАвто(i).Top = imgАвто(i).Top - 50 * (i - 1)
    Next
End Sub
```

Пояснения: Как обычно, первой выполняется процедура Form_Load, которая все создает и расставляет. Затем процедура Command1_Click выполняет прыжок автомобилей.

Вот более "правильный" вариант той же программы, использующий процедуры с параметрами:

```
Dim i As Integer
Dim N As Integer
_____________________________________________________________
Private Sub Form_Load()
    N = 20
        'Загружаем на форму и делаем видимыми все объекты, кроме первого (который и так уже загружен):
    For i = 2 To N
        Load imgАвто(i)
        imgАвто(i).Visible = True
    Next
        'Помещаем на место все объекты:
    For i = 1 To N
        Поместить_автомобиль_на_место   i
    Next
End Sub
_____________________________________________________________
Private Sub Command1_Click()
        'Прыжок всех автомобилей:
    For i = 1 To N
        Прыжок_автомобиля   i
    Next
End Sub
_____________________________________________________________
        'Процедура, помещающая на место один объект из массива с индексом Номер_автомобиля:
Private Sub Поместить_автомобиль_на_место(Номер_автомобиля As Integer)
        imgАвто(Номер_автомобиля).Top = Height - 1000
        imgАвто(Номер_автомобиля).Left = 300 * Номер_автомобиля
End Sub
_____________________________________________________________
        'Процедура прыжка одного объекта из массива с индексом Номер_автомобиля:
Private Sub Прыжок_автомобиля Прыжок_автомобиля (Номер_автомобиля As Integer)
        imgАвто(Номер_автомобиля).Top = imgАвто(Номер_автомобиля).Top - 50 * (Номер_автомобиля - 1)
End Sub
```

В процедуре Прыжок_автомобиля вы можете задавать разным автомобилям разное поведение. Например:

```
If  Номер_автомобиля = 8  Then   imgАвто(Номер_автомобиля).Top = 1000
If  Номер_автомобиля = 14  Then   imgАвто(Номер_автомобиля).Visible = False
```

Чтобы удалить (выгрузить) 4-й объект с формы, применяем оператор **Unload** imgАвто(4).

Обратите внимание, что теперь, работая в окне кода и пытаясь получить заготовку процедуры для обработки события, принадлежащего автомобилю, вы получите ее с параметром Index, значение которого равно индексу автомобиля, с которым происходит событие, в массиве. Это позволяет вам задавать разную реакцию автомобилей на одно и то же событие. Например, следующая процедура

```
Private Sub imgАвто_Click(Index As Integer)
    If Index = 2 Then imgАвто(Index).Top = imgАвто(Index).Top + 200 Else imgАвто(Index).Visible = False
End Sub
```

позволяет вам превращать любой автомобиль щелчком мыши в невидимку, а 2-й автомобиль вместо этого отскакивает.

Глава 14. Разные звери в одном ковчеге

В предыдущей главе я рассматривал массивы - структуры, состоящие из многих данных и объектов. В этой главе я продолжу рассматривать сложные структуры данных и объектов. Но если массивы - это одинаковые солдатики, стоящие стройными рядами в своих строгих батальонах, то те структуры, что мы рассмотрим в этой главе, больше напоминают натюрморт.

14.1. Тип Variant

Массив - это набор элементов. Если вы объявили массив некоторого типа, то он обязан состоять из элементов только этого типа. Например, каждый из десяти элементов массива, объявленного как

Dim a(1 To 10) As Integer

обязан быть целым числом. Ни один из них не имеет права быть строкой или дробным числом.

Имеется целый ряд задач, где хотелось бы, чтобы элементы в наборе имели разный тип. Этого можно достичь, если вообще не объявлять тип массива:

```
Dim a(1 To 3)
Private Sub Command1_Click()
   a(1) = 32
   a(2) = "кошка"
   a(3) = #8:56:00 AM#
End Sub
```

В этом случае Visual Basic считает, что массив объявлен по умолчанию специальным типом **Variant**. Это тип-хамелеон. Он не возражает, если объявленная им переменная будет иметь значение любого типа. Он присматривает, чтобы выполнение программы при этом проходило по возможности гладко и без неприятностей для программиста.

Вместо Dim a(1 To 3) можно было прямо написать **Dim a(1 To 3) As Variant**.

Еще один пример всеядности Variant (здесь я объявлю этим типом не массив, а переменную):

```
Dim b  As Variant                    'можно было просто написать   Dim b
Private Sub Command1_Click()
   b = 77
   b = "кот"
End Sub
```

Итак, если переменная не объявлена или объявлена без указания типа, Visual Basic считает ее принадлежащей типу Variant. Какой толк от этого типа? Иногда бывает так, что соблюдая приличия и объявляя тип каждой переменной, мы наталкиваемся во время выполнения программы на сообщение об ошибке Type mismatch (несовпадение типов). Это значит, что выполняя операции над данными разных типов и преобразовывая один в другой, Visual Basic не захотел преступать строгие рамки безопасности и приличий и потерпел крах. Не всегда у начинающего программиста хватает знаний, чтобы разобраться в причинах краха. В этом случае, если вы любите риск, объявите все данные одним типом - Variant или вообще не указывайте тип. Теперь вероятность сообщений об ошибке снизится, хотя несколько увеличится вероятность получения недостоверного результата.

Вернемся к нашему желанию иметь набор из элементов разных типов. Программисты не хотят для этого использовать массивы типа Variant. Они используют для этого так называемый пользовательский тип данных.

14.2. Пользовательский тип данных

В Паскале пользовательскому типу данных соответствуют *записи*, в языке Си - *структуры*. Рассмотрим простейший пример пользовательского типа:

Вы хотите занести в компьютер информацию о ваших любимых компьютерных играх, чтобы затем как-то ее анализировать, например, определить, какая игра занимает самое большое место на диске. Для простоты ограничим информацию об игре тремя элементами:

- Название игры
- Сколько места игра занимает на диске (в мегабайтах)
- Хорошая или плохая графика у игры (ваша оценка)

Также для экономии места ограничимся двумя играми. Ваша программа должна будет занести в память информацию об обеих играх, а затем выполнить три задания:

- Распечатать название первой игры
- Определить, сколько места займут на диске обе игры вместе
- Ответить, хороша или плоха графика у второй игры

Ваши действия: Для начала нужно создать пользовательский тип данных для размещения информации об одной игре. Нам

уже приходилось конструировать перечислимый тип (см.11.3). Здесь действуем аналогично - придумываем новому типу имя (типИгра) и пишем:

```
Private Type типИгра
   Название As String
   Объем As Integer
   Графика_хорошая As Boolean
End Type
```

Тип определен. (Объяснение слова Private отложим на будущее). Теперь Visual Basic знает, сколько места в памяти занимает информация об игре и какая у нее структура. Можно отводить место в памяти:

```
Dim Игра1 As типИгра
Dim Игра2 As типИгра
```

Вот полный текст программы:

```
        'Создаем пользовательский тип:
Private Type типИгра
   Название As String
   Объем As Integer
   Графика_хорошая As Boolean
End Type
        'Отводим в памяти место под информацию о двух играх:
Dim Игра1 As типИгра
Dim Игра2 As типИгра
_____________________________________________________________
Private Sub Form_Load()
        'Заносим в память компьютера информацию об играх:
   Игра1.Название = "StarC":                Игра2.Название = "Heroes III"
   Игра1.Объем = 90:                        Игра2.Объем = 200
   Игра1.Графика_хорошая = False:           Игра2.Графика_хорошая = True
        'Выполняем 1 задание:
   Debug.Print Игра1.Название
        'Выполняем 2 задание:
   Debug.Print Игра1.Объем + Игра2.Объем
        'Выполняем 3 задание:
   If Игра2.Графика_хорошая Then Debug.Print "Хорошая графика" Else Debug.Print "Плохая графика"
End Sub
```

Обратите внимание, что имя элемента отделяется от имени переменной точкой подобно тому, как название свойства объекта отделяется точкой от имени объекта.

Выстроенную подобным образом в памяти информацию о чем-либо часто называют **базой данных**. Всю информацию об одной переменной (в нашем случае об одной игре) называют **записью** в этой базе данных. Программу, которая извлекает информацию из базы данных, сортирует записи и производит другую обработку информации в базе данных, называют **системой управления базой данных**. В нашем примере роль системы управления базой данных играет процедура Form_Load.

Элементы пользовательского типа могут иметь любой известный вам тип, в том числе быть массивом или пользовательским типом. Пусть вы хотите дополнить информацию об игре именами персонажей игры (не более 20). Вот каким массивом дополнится определение типа:

```
Private Type типИгра
   Название As String
   Объем As Integer
   Графика_хорошая As Boolean
   Персонаж(1 To 20) As String
End Type
```

Теперь можно использовать такие операторы:

```
Игра2.Персонаж(4) = "Криган"
Игра1.Персонаж(12) = "Fenix"
Debug.Print   Игра1.Персонаж(10)
```

Усложним задание: Вы хотите создать базу данных о 30 играх. Для этого достаточно вместо объявлений

```
Dim Игра1 As типИгра
Dim Игра2 As типИгра
```

объявить массив:

```
Dim Игра(1 To 30) As типИгра
```

Теперь можно использовать операторы:

```
Игра(16).Название = "StarCraft"          'название 16-й игры
Игра(8).Персонаж(12) = "Солдат"          'двенадцатый персонаж восьмой игры
Debug.Print   Игра(16).Название,   Игра(8).Персонаж(12)
```

Еще усложним задание: Вы хотите задать более подробную информацию о персонажах, а не только их имя. Вас интересует имя персонажа, количество его здоровья (в численном виде), название его оружия. Для этого вы определяете еще один пользовательский тип. Вот программа:

```
        'Сначала создаем пользовательский тип персонажа,
        'так как без него нельзя определить пользовательский тип игры:
Private Type типПерсонаж
```

```
    Имя As String
    Здоровье As Integer
    Оружие As String
End Type
            'Затем создаем пользовательский тип игры:
Private Type типИгра
    Название As String
    Объем As Integer
    Графика_хорошая As Boolean
    Персонаж(1 To 20) As типПерсонаж            'Не больше 20 персонажей в одной игре
End Type
            'Отводим в памяти место под информацию об играх:
Dim Игра(1 To 30) As типИгра

Private Sub Form_Load()
    Игра(8).Персонаж(12).Имя = "Солдат"
    Игра(8).Персонаж(12).Здоровье = 140
    Debug.Print    Игра(8).Персонаж(12).Имя,     Игра(8).Персонаж(12).Здоровье
End Sub
```

Если вы захотите более подробно описать оружие, то можете создать еще один пользовательский тип. И так далее. Массивы и пользовательские типы могут вкладываться друг в друга, как матрешки, до бесконечности. Так создаются иерархии данных практически в любой области знаний.

Задание 121: Создайте базу данных о своих родственниках. О каждом родственнике должно быть известно:

- *Имя*
- *Год рождения*
- *Цвет глаз*

Массивы не используйте. Программа должна:

- Распечатать ваш возраст и цвет глаз
- Ответить на вопрос – правда ли, что ваш дядя старше тети.

Задание 122: Создайте базу данных о своих однокашниках. О каждом однокашнике должно быть известно:

- *Фамилия*
- *Имя*
- *Пол*
- *Год рождения*

Обязательно используйте массив не меньше, чем из 10 записей. Программа должна:

- Вычислить средний возраст ваших однокашников
- Определить, кого среди них больше – дам или кавалеров
- Ответить на вопрос – есть ли в вашей базе тезки (это нелегко).

Базы данных являются настолько распространенным средством хранения информации, что в Visual Basic есть специальные мощные инструменты для работы с ними. Мы рассмотрим их в Глава 22.

14.3. Коллекции

Объектные переменные

В предыдущих разделах вы видели, что значением переменной величины может быть сложная структура данных. Но переменная может иметь своим значением и еще более сложную вещь - объект.

Понятие объекта - сложное понятие программирования. Частными случаями объектов в Visual Basic являются формы и элементы управления. Исчерпывающее описание работы объектов выходит за рамки курса для начинающих. В этой главе мы только вскользь знакомимся с ними. В Глава 20 я попытался дать основные понятия об этой области программирования.

В Visual Basic существует несколько типов, к которым может принадлежать объект. Самый общий тип - это **Object**. Если мы напишем Dim A As Object, то переменная A сможет принимать значения любых объектов. В нижеприведенном примере она сначала принимает значение метки, а затем формы. Для того, чтобы присвоить переменной значение существующего объекта, простой оператор присваивания не подходит, нужно писать так: **Set** A **=** Label1.

Создайте проект с меткой.

```
Private Sub Command1_Click()
    Dim A As Object
    Set A = Label1
    A.FontSize = 20
    A.Caption = "Привет!"                'Меняется надпись на метке
    Set A = Form1
    A.Caption = "Еще один привет!"       'Меняется надпись на форме
```

```
End Sub
```

Вы видите, что как только мы присвоили переменной A значение метки Label1, мы смогли вместо Label1.FontSize = 20 писать A.FontSize = 20.

Объектный тип **Control** - это тип элементов управления. Если мы напишем Dim A As Control, то переменная A сможет принимать значения не любых объектов, а только элементов управления.

Зачем все это нужно? Один из примеров применения объектной переменной вы найдете чуть ниже в этом же разделе, где объектная переменная в цикле пробегает значения элементов управления на форме. Другой пример, когда объект является параметром процедуры, вы найдете в 17.7.

Коллекции

Коллекции так же относятся к пользовательским типам данных, как массивы элементов управления к массивам переменных величин. Коллекция - это набор элементов разного типа, таких как элементы управления, другие объекты. Коллекция и сама является объектом (а почему бы и нет? - коробки в коробке). Коллекции всеядны, они позволяют иметь в одном наборе и кнопку и метку и переменную величину типа Integer. Я ограничусь распространенным случаем, когда в коллекцию входят только элементы управления.

Задача: На вашей форме имеется несколько десятков кнопок, меток и текстовых полей, которые делают свое дело, скажем, помогают обрабатывать банковскую информацию - кто принес деньги, сколько принес, в долларах или рублях, да сколько денег у него стало, да сколько всего в банке денег и т.д. Вам нужно, чтобы в некоторый момент выполнения программы такие-то и такие-то кнопки, метки и текстовые поля из самых разных мест формы (всего, скажем, дюжина объектов) поменяли размер шрифта. Ну, скажем, для того, чтобы привлечь к себе внимание. А в какой-нибудь другой момент эта же дюжина должна сделать что-нибудь другое, например, кнопки из этой дюжины должны быть деактивированы, чтобы на них случайно не нажали. И так далее.

Решение: Создадим проект. Для простоты разместим на форме только шесть элементов управления: Label1, Text1, Command1, Label2, Text2, Command2. Пусть в нашу "дюжину" входят из них только Label2, Text2, Command2. Программным путем создадим **коллекцию** из этих объектов. Для этого сначала придумаем ей имя Моя_коллекция и объявим ее так:

Dim Моя_коллекция **As New Collection**

Слово **Collection** означает коллекцию. Слово **New** мы пишем тогда, когда создаем новый объект (а коллекция - объект).

Коллекция объявлена, но пока она пуста. Теперь будем по очереди добавлять объекты в коллекцию подобно тому, как в 13.6 мы добавляли объекты в массив. Но здесь другая грамматика. Смотрим программу:

```
Dim Моя_коллекция As New Collection                 'Объявляем коллекцию

Private Sub Command1_Click()
   Моя_коллекция.Add Label2                         'Добавляем в коллекцию 1-й элемент
   Моя_коллекция.Add Text2, "Текстик"               'Добавляем в коллекцию 2-й элемент
   Моя_коллекция.Add Command2, "Кнопочка"           'Добавляем в коллекцию 3-й элемент

   Debug.Print Моя_коллекция.Count                  'Печатается 3 - число элементов в коллекции
   Debug.Print Моя_коллекция (1).Caption            'Печатается "Label2"
   Моя_коллекция (3).Height = 800                   'Задается высота кнопки Command2
   Моя_коллекция ("Текстик").BackColor = vbYellow   'Задается цвет поля Text2

   For Each Мой_объект In Моя_коллекция             'ДЛЯ КАЖДОГО объекта В коллекции
      Мой_объект.FontSize = 18                              'задается размер шрифта
   Next

   For Each Мой_объект In Моя_коллекция             'ДЛЯ КАЖДОГО объекта В коллекции
      If TypeName(Мой_объект) = "CommandButton" Then        'если имя типа объекта - CommandButton, то
         Мой_объект.Enabled = False                         'деактивировать ее
      End If
   Next

   Моя_коллекция.Remove 2                           'Удаляем из коллекции второй элемент
   Моя_коллекция.Remove "Кнопочка"                  'Удаляем из коллекции кнопку
End Sub
```

Пояснения: У коллекций есть несколько методов, три из них (Add, Remove, Count) я сейчас поясню:

При помощи метода **Add** мы добавляем в коллекцию элементы. При этом обязательно указывается имя элемента. Также можно через запятую указать произвольную строку - **ключ** элемента, по которому можно к нему при желании обращаться, что мы и сделали в программе. Как видите, здесь для добавления каждого элемента мы использовали отдельный оператор, хотя есть ситуации, когда можно это делать в цикле.

Метод **Count** просто сообщает число элементов в коллекции:

Обращаться к отдельным элементам коллекции можно по индексу, как к элементу массива, а можно и по ключу, там, где он задан.

Чтобы удалить отдельный элемент из коллекции (не с формы), используется метод **Remove** с указанием индекса или ключа элемента.

Для коллекций удобно применять специальную разновидность оператора цикла - **For Each**. Для этого необходимо придумать имя переменной величине, значение которой будет пробегать все элементы коллекции. Мы придумали имя Мой_объект. Когда цикл выполняется в первый раз, Мой_объект "равняется" 1-му элементу коллекции, во второй раз - 2-му и т.д., пока элементы не будут исчерпаны. В остальном синтаксис и порядок выполнения оператора For Each такой же, как и у привычного нам For. Здесь я не объвлял переменную Мой_объект, а мог бы и объявить - Dim Мой_объект As Control.

Здесь мы использовали функцию **TypeName**, чтобы выбрать из всех элементов коллекции только элементы данного типа. Пояснять ее я не буду, используйте дальше по аналогии. Вообще, вы чувствуете, что с этими коллекциями мы ступили на зыбкую для нас почву работы с объектами? Зыбкая она потому, что мы пока про работу с объектами почти ничего не знаем. Ну что ж, получше узнаем мы ее в Глава 20, а пока вам хватит того, что вы уже узнали.

Кстати, оператор For Each можно использовать и при работе с массивами.

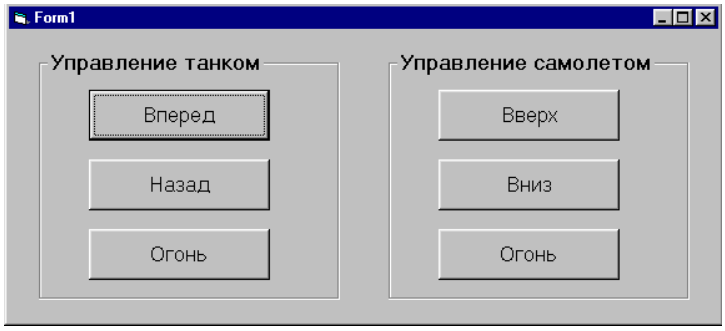
14.4. Рамка (Frame)

Рамка объединяет в себе разнотипные элементы управления. В этом рамка похожа на коллекцию. Но у рамок несколько иное назначение, чем у коллекций.

Поместите на форму элемент управления **Frame**, придав ему довольно большие размеры. Затем возьмите из Toolbox и поместите внутрь этой рамки несколько других элементов управления, в том числе еще один Frame. А теперь передвиньте рамку по форме. Вы видите, что объекты внутри рамки передвинулись вместе с ней. Посмотрите на значения свойств Top и Left любого объекта внутри рамки. Теперь это координаты не относительно формы, а относительно рамки.

Если вы захотите мышкой переместить в рамку уже размещенный на форме объект, у вас ничего не получится. Оказавшись внутри границ рамки, он не стал "своим". Убедитесь в этом, сдвинув рамку в сторону. Все уехали, объект остался на месте. Чтобы добиться желаемого, вырежьте (Cut) объект со старого места и наклейте (Paste) в рамку. Аналогичная ситуация, когда вы попытаетесь обнять новой или старой рамкой уже существующие объекты.

Рамка является удобным средством объединения элементов управления по смыслу. Пример:



В программе вы можете изменять значения свойств рамки Top, Left. Объекты, объединенные в рамке, будут при этом перемещаться вместе с ней. Вы можете изменять значения свойств рамки Visible, Enabled. При этом точно так же будут меняться значения этих свойств всех объектов, объединенных в рамке. Это полезно тогда, когда у вас на форме слишком много объектов и все они свободно не умещаются на ней. Вас может выручить то, что не все они нужны одновременно. Разделите их между наложенными друг на друга рамками и в каждый момент времени делайте видимой только одну из них.

Поэкспериментируйте с цветами, границей, шрифтом рамки.

Способностями, аналогичными рамке, обладает и объект PictureBox. Проверьте - в нем точно так же можно размещать элементы управления. Мы знаем и еще один такой объект. Это, конечно же, сама форма. Все подобные объекты называются **контейнерами**.

Глава 15. Элементы управления

Вот некоторые элементы управления, с которыми мы уже познакомились:

- *Кнопка*, *метка*, *текстовое поле* - в Глава 1 и Глава 2.
- *Microsoft Multimedia Control* - в 2.8
- *Animation* - в 2.12
- *Image*, *Picture* - в 9.3
- *Line*, *Shape* - в 9.4
- *Таймер* - в 11.2
- *Рамка* (*Frame*) - в 14.4

А эти элементы управления мы пройдем позже, в следующих главах:

- *Common Dialog* - в 18.2
- *Панель инструментов* и *ImageList* - в 18.3
- *Web-броузер* - в 21.5
- *Data* и *DBGrid* - в 22.3

В этой главе я познакомлю вас с остальными нужными, с моей точки зрения, элементами управления. Я буду описывать только главное действие и главные свойства этих элементов, но этого в подавляющем большинстве случаев вполне достаточно. С другими любопытными и удобными их свойствами вы можете познакомиться в более толстых книжках.

Я не буду приводить примеры реальных программ с использованием этих элементов. Вам достаточно будет знать, как меняются в процессе функционирования значения их основных свойств. Вы уже достаточно опытны, чтобы при некоторой фантазии самостоятельно придумать проекты с их использованием.

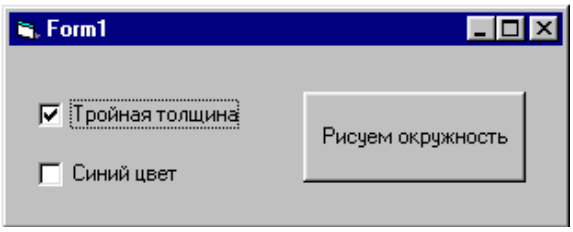
Многих из описываемых элементов управления нет на Toolbox. Чтобы их добыть, поставьте флажки в **Project → Components → Microsoft Windows Common Controls 6.0** и **Microsoft Windows Common Controls 6.0-2.**

15.1. Флажок (CheckBox)

Поместите на форму несколько **флажков** (**CheckBox**). Запустите проект. Пощелкайте внутри квадратиков. Флажок мы *устанавливаем* (ставим галочку мышкой или с клавиатуры) тогда, когда хотим сказать "да", а *снимаем установку* (снимаем галочку мышкой или с клавиатуры), когда хотим сказать "нет".

Задача: Нажатием на кнопку нужно нарисовать окружность. Но перед нажатием кнопки мы хотим иметь возможность немножко настроить вид окружности, а именно: одним флажком установить или не установить тройную толщину линии этой окружности, а другим флажком установить или не установить синий цвет этой линии. Если флажки мы не установим, то толщина и цвет будут выполнены по умолчанию (одинарная и черный).

Ваши действия: Создайте проект с кнопкой и двумя флажками:



Имена флажков - Check1 и Check2.

Программа:

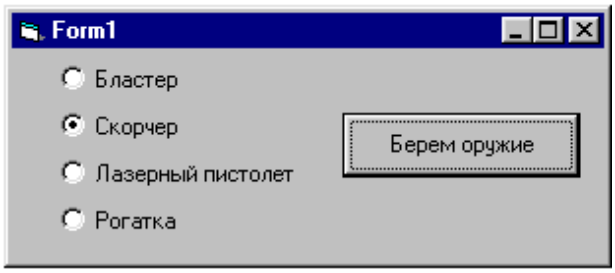
```
Private Sub Command1_Click()
  Cls
  If Check1.Value = vbChecked Then DrawWidth = 3 Else DrawWidth = 1          'Если флажок 1 установлен, то толщина тройная, иначе одинарная
  If Check2.Value = vbChecked Then ForeColor = vbBlue Else ForeColor = vbBlack  'Если флажок 2 установлен, то цвет синий, иначе черный
  Circle (500, 500), 300
End Sub
```

Пояснения: У многих элементов управления есть свойство **Value** (значение), характеризующее главное назначение элемента. Свойство Value флажка имеет 3 значения: **vbChecked** (установлен, галочка), **vbUnchecked** (не установлен, пустой белый квадрат), и **vbGrayed** (галочка на сером фоне - используется нечасто, например, в процессе установки Microsoft Office). Все три значения можно устанавливать в режиме проектирования и программным способом. В режиме работы можно вручную, мышкой или с клавиатуры, устанавливать только первые два значения, причем переключать между этими двумя значениями можно сколько угодно раз. В этом и состоит удобство флажков: перед тем, как совершить решающее нажатие на кнопку, можно поддаться сомнениям и как угодно устанавливать флажки или, передумав, снимать установку.

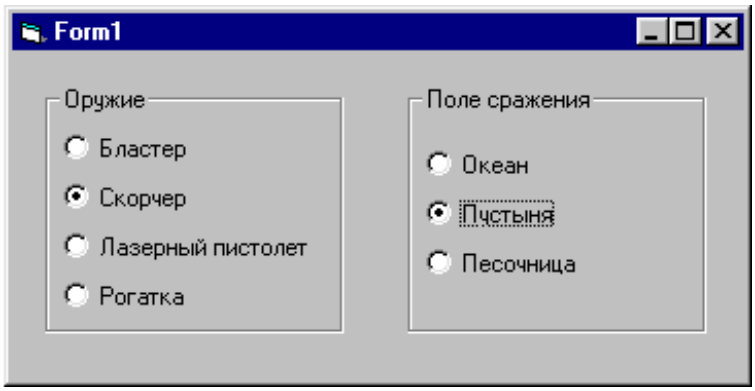
Поэкспериментируйте с цветами, шрифтом флажка. Флажок может иметь и другой внешний вид - графический (свойство Style) с участием картинок.

15.2. Переключатель (OptionButton)

Поместите на форму три элемента управления **OptionButton**. Запустите проект. Пощелкайте внутри кружочков. Вы видите, что из всех переключателей в любой момент времени только один может быть выбран, остальные автоматически отключаются. Это полезное свойство удобно применять там, где нужно выбрать только одну из нескольких возможностей. Например, если персонажу вашей игры нужно выбрать одно оружие из четырех, вы организуете такую группу переключателей:



А что делать, если вам на той же форме нужно организовать еще одну группу переключателей, например, для выбора поля сражения? Для этого нужно каждую группу поместить в свой *контейнер*, например, рамку:



В противном случае из всех семи переключателей в любой момент времени будет выбран только один.

Вы скажете, что переключатели и флажки - это одно и то же. Неверно. Флажков в любой момент времени может быть установлено сколько угодно, хоть все, а из переключателей может быть выбран только один.

Так же, как и у флажка, главным свойством переключателя является Value. Только тип этого свойства логический - когда переключатель выбран, оно равно True. Компьютер может узнать, выбран ли переключатель Option1, при помощи следующего оператора:

```
If  Option1.Value = True  Then ........
```

Учитывая, что тип свойства логический, можно сократить эту запись:

```
If  Option1.Value  Then ........
```

А можно и еще сократить. Visual Basic позволяет обращаться к главному свойству элемента управления (в нашем случае это Value) сокращенно - **вместо Option1.Value - просто Option1**. Получается:

```
If  Option1  Then ........
```

Вот фрагмент, позволяющий узнать, какой из переключателей группы выбран (имена переключателей: Option1, Option2, …):

```
If Option1 Then
    Debug.Print "Выбран бластер"
ElseIf Option2 Then
    Debug.Print "Выбран скорчер"
    ....................
End If
```

Если вам нужно, чтобы некоторое действие было выполнено не потом, а сразу же после щелчка по переключателю, вы можете для каждого переключателя написать процедуру обработки события Click, но гораздо удобнее объединить все переключатели группы в массив, тогда процедуру придется писать только одну:

```
Private Sub Option1_Click(Index As Integer)
    Select Case Index
        Case 0
            Debug.Print "Выбран бластер"
        Case 1
            Debug.Print "Выбран скорчер"
        ....................
```

```
    End Select
End Sub
```

Поэкспериментируйте с цветами, шрифтом переключателя. Переключатель, как и флажок, может иметь графический внешний вид (свойство Style) - с участием картинок.

15.3. Полосы прокрутки (HScrollBar и VScrollBar)

Для определенности поговорим о горизонтальной полосе прокрутки (**HScrollBar**). Все сказанное будет полностью относиться и к вертикальной полосе (**VScrollBar**).

Поместите на форму горизонтальную полосу. Запустите проект. Потаскайте мышкой бегунок. Вы, безусловно, знакомы с полосой прокрутки по другим приложениям Windows. Она используется в основном для того, чтобы прокручивать информацию в окне или же просто менять значение какой-нибудь величины.

Задача: Изменять с помощью полосы прокрутки значение переменной величины W в пределах от 20 до 50. При щелчке по стрелкам полосы или клавиатуры значение переменной должно меняться на 2, а при щелчке по полосе слева или справа от бегунка значение переменной должно меняться на 5. При запуске проекта бегунок должен стоять на отметке 27.

Ваши действия: Создайте проект и поместите на форму горизонтальную полосу прокрутки. Ее имя HScroll1. Установите в соответствии с числами из задания следующие свойства полосы:

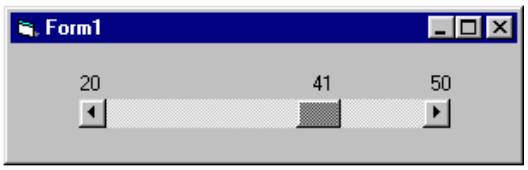
- Min - 20
- Max - 50
- SmallChange - 2
- LargeChange - 5
- Value - 27

Проверьте правильность работы полосы, запустив программу:

```
Private Sub HScroll1_Change()
  W = HScroll1.Value
  Debug.Print W
End Sub
```

Событие HScroll1_Change возникает при любом перемещении бегунка.

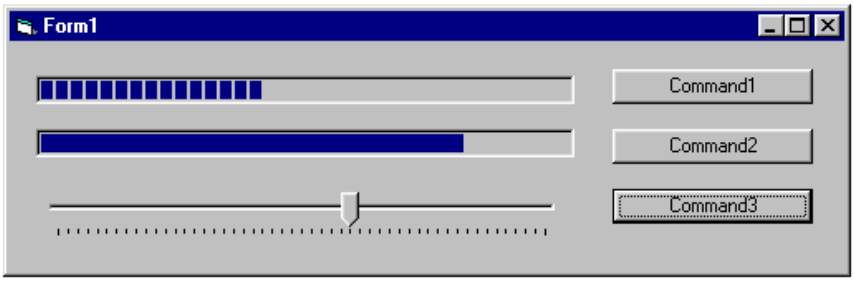
Задание 123: Поместите рядом с полосой три метки:



Левая метка должна указывать минимальное значение, правая - максимальное, средняя - текущее. Усложнение: Хорошо бы средняя метка бегала рядом с бегунком.

15.4. Slider, ProgressBar

Slider изображен на картинке внизу. Его действие и свойства (Min, Max, SmallChange, LargeChange, Value) аналогичны действию и свойствам полосы прокрутки HScrollBar.



ProgressBar в двух видах изображена сверху. Вы наверняка видели такую при инсталляции программ. Зачем она нужна? Пусть ваша программа запускает длительный процесс, во время которого на экране ничего не происходит (например, считывает информацию из 400 файлов). У пользователя может возникнуть тревожное ощущение, что программа зависла. Чтобы этого ощущения не было, вы можете выдать на экран текст "Я занята. Подождите минутку.", которое пропадет, когда дело сделано. Но опять же, пока дело делается, этот текст так долго и неподвижно красуется посреди экрана, что в душу опять закрадываются подозрения. Гораздо лучше создать ProgressBar, полоса которой после считывания информации из каждого файла будет продвигаться на 1/400 часть длины ProgressBar. Поскольку компьютер работает быстро, у пользователя создастся впечатление, что полоса плавно ползет направо. А раз движение есть, значит компьютер не завис! К тому же можно в процес-

се работы примерно представлять, какая часть ее выполнена.

Свойства ProgressBar - Min, Max - аналогичны свойствам полосы прокрутки. Внешний вид определяется свойством Scrolling.

Для иллюстрации сказанного создайте проект. Разместите элементы управления, как на картинке. Установите свойство Max слайдера в 50, у остальных - в 100. Запустите следующие процедуры:

```
Private Sub Command1_Click()
   For a = 0 To 40 Step 0.01
      ProgressBar1 = a
   Next
End Sub

Private Sub Command2_Click()
   For a = 0 To 80 Step 0.1
      ProgressBar2 = a
   Next
End Sub

Private Sub Command3_Click()
   For a = 0 To 30 Step 0.001
      Slider1 = a
   Next
End Sub
```

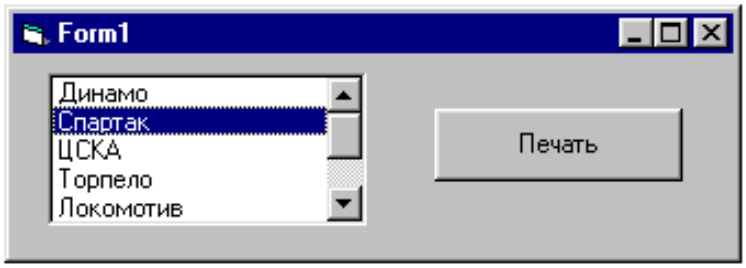
Во время выполнения процедур полосы и бегунок слайдера движутся. Если движение слишком быстрое или слишком медленное, измените шаг цикла. На картинке вы видите состояние элементов управления после того, как процедуры отработали.

В реальной задаче о копировании 400 файлов вы могли бы сделать так, чтобы при закрытии очередного файла вызывалась процедура пользователя, продвигающая чуть-чуть полосу.

15.5. Список (ListBox) и поле со списком (ComboBox)

Список (ListBox)

Поместите на форму **список**. Его имя - List1. Заполните список. Для этого зайдите в его свойство List и введите слово "Динамо" - первый элемент списка, после чего нажмите Ctrl-Enter. Теперь введите слово "Спартак" - второй элемент списка, нажмите Ctrl-Enter. И так далее. Запустите проект. Пощелкайте по элементам списка.



Если все элементы списка не умещаются в его видимом пространстве, то у списка автоматически возникает полоса прокрутки. А кнопку "Печать" я поместил на форму вот для чего:

Задача: Пусть вам нужно напечатать текст "Следующим соперником нашей команды будет команда …". На месте многоточия должно стоять название команды, выбранной вами из списка. Запустив проект, вы щелчком мыши выбираете нужную команду (на картинке выбран Спартак), а затем нажатием на кнопку печатаете текст.

Вот программа:

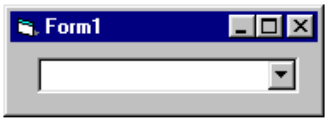
```
Private Sub Command1_Click()
    Debug.Print "Следующим соперником нашей команды будет команда "; List1.Text
End Sub
```

Свойство **Text** списка List1 - это значение выбранного элемента списка List1. Пощелкайте по элементам списка, после каждого щелчка нажимая кнопку "Печать". Итак, мы нашли одно из применений списка: он облегчает ввод в компьютер часто встречающихся слов.

ComboBox (вариант "Раскрывающийся список")

ComboBox - это обогащенный новыми возможностями ListBox. Он существует в трех вариантах. Рассмотрим их по мере усложнения.

Поместите на форму ComboBox. Его имя - Combo1. Вариант ComboBox определяется значением его свойства Style. Выберите вариант Dropdown List - "Раскрывающийся список". Заполните ComboBox так же, как вы заполняли список ListBox. Запустите проект. Список выглядит так:



Щелкните по черной треугольной стрелке в правой части списка. Список раскроется и вы сможете выбрать из него любой элемент так же, как вы выбирали элементы из списка ListBox. Программа для ComboBox также будет совершенно аналогична программе для ListBox.

```
Private Sub Command2_Click()
    Debug.Print "Следующим соперником нашей команды будет команда "; Combo1.Text
End Sub
```

Как видите, преимуществом раскрывающегося списка перед обычным является экономия места на форме.

ComboBox (вариант "Простой Combo")

Поместите на форму ComboBox (а проще - скопируйте старый). В его свойстве Style выберите вариант Simple Combo - "Простой Combo". Заполните ComboBox так же, как вы заполняли список ListBox. Запустите проект. Combo выглядит так:

Отличие ComboBox от ListBox в том, что сверху от списка имеется текстовое поле. Пока не вводите туда ничего, а просто выбирайте элементы из списка и печатайте, как в проекте с ListBox. Все получается по-старому. Зачем же нужно текстовое поле? А затем, что если вам вдруг понадобится распечатать команду, которой нет в списке, вы всегда можете ввести ее в текстовое поле и нажать кнопку печати.

Вариант "Раскрывающийся список" тоже имеет наверху текстовое поле, только вводить туда ничего нельзя.

ComboBox (вариант "Раскрывающийся Combo")

Раскрывающийся Combo объединяет в себе преимущества двух других вариантов ComboBox: он выглядит так же компактно, как раскрывающийся список, и позволяет редактировать текстовое поле, как простой Combo.

Свойства, события и методы элементов управления ListBox и ComboBox.

Все примеры, приведенные здесь относительно ListBox, относятся также и к любому варианту ComboBox.

Элементы списков нумеруются с нуля.

Оператор	Смысл
List1.**AddItem** "Бавария"	В конец списка List1 метод AddItem добавляет элемент Бавария
List1.**AddItem** "Боруссия", **3**	Метод AddItem добавляет элемент Боруссия на место №3 в список List1.
List1.**RemoveItem 4**	Метод RemoveItem удаляет элемент №4 из списка List1.
List1.**Clear**	Опустошить список List1
Debug.Print List1.**Text**	Напечатать значение выбранного элемента в списке List1
Debug.Print Combo1.**Text**	Напечатать значение текстового поля из ComboBox
Debug.Print List1.**List (5)**	Напечатать значение элемента №5 в списке List1
List1.**List(0)** = "Барселона"	Присвоить элементу №0 в списке List1 значение Барселона
Debug.Print List1.**ListIndex**	Напечатать номер выбранного элемента в списке List1. Если элемент не выбран, то печатается -1.
Debug.Print List1.**ListCount**	Напечатать количество элементов в списке List1

Если вы хотите, чтобы элементы списка были отсортированы по алфавиту, то в режиме проектирования установите в True свойство **Sorted**. В режиме работы его менять нельзя. Сортировка нарушится, если вы будете использовать метод AddItem с индексом. Не забывайте, что сортировка - текстовая, а не числовая, поэтому 28 будет стоять выше, чем 5.

У ListBox есть некоторые преимущества перед ComboBox: он может состоять из нескольких столбцов и допускает множественный выбор. Но на этом мы останавливаться не будем.

Щелчок или двойной щелчок по элементу списка вызывают, как водится, события Click и DblClick.

Задача: Создать на форме простой Combo и 5 кнопок для разнообразной работы с его списком. Вот функции кнопок:

- Кнопка "Печать" распечатывает содержимое текстового поля Combo
- Кнопка "Удаление" удаляет выбранный элемент из списка
- Кнопка "Добавить" добавляет содержимое текстового поля в список (удобно для быстрого внесения дополнений в список)

- Кнопка "Перестановка" переставляет выбранный элемент в конец списка (удобно для произвольной сортировки списка)
- Кнопка "Заменить" заменяет выбранный элемент содержимым текстового поля (удобно для небольших исправлений в написании элемента)

Кроме этого, элемент должен распечатываться двойным щелчком.

Попробуйте выполнить это задание самостоятельно. Если не получится, то вот программа:

```
Dim номер As String     'Номер выбранного элемента в списке

Private Sub Печать_Click()
   Debug.Print  Combo1.Text
End Sub

Private Sub Удаление_Click()
   Combo1.RemoveItem  Combo1.ListIndex
End Sub

Private Sub Добавить_Click()
   Combo1.AddItem  Combo1.Text
End Sub

Private Sub Перестановка_Click()
   Combo1.AddItem  Combo1.Text
   Combo1.RemoveItem  Combo1.ListIndex
End Sub

Private Sub Заменить_Click()
   Combo1.List(номер) = Combo1.Text
End Sub

Private Sub Combo1_Click()
   номер = Combo1.ListIndex
End Sub

Private Sub Combo1_DblClick()
   Печать_Click
End Sub
```

Задание 123-1: "*Англо-русский словарь*". Поместите на форму два раскрывающихся списка. В левый запишите несколько десятков английских слов и упорядочте их по алфавиту. В правый запишите в том же порядке переводы этих слов на русский. При выборе слова в левом списке в правом должен появляться перевод.

15.6. Знакомство с другими элементами управления

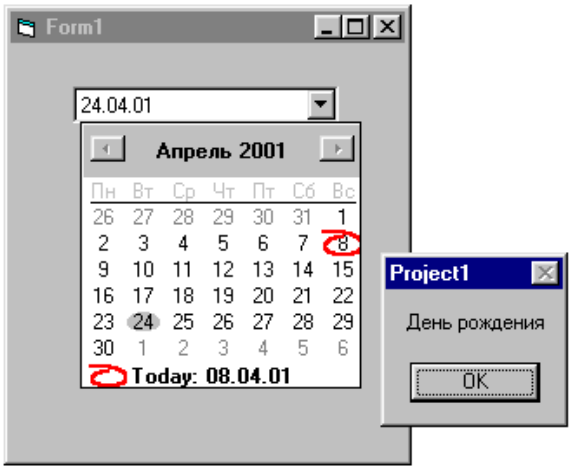
А теперь просто познакомимся, не вникая, с некоторыми другими элементами управления, имеющимися в Visual Basic.

Элементы MonthView и DTPicker

Оба они похожи друг на друга, поэтому остановлюсь только на **DTPicker**. На рисунке вы можете видеть его после запуска проекта с такой процедурой в окне кода

```
Private Sub DTPicker1_Change()
   If  DTPicker1 = #4/24/2001#   Then   MsgBox "День рождения"
End Sub
```

и щелчка мышкой по 24 апреля 2001 года.



Тут же на картинке вы видите и окно сообщения.

Вы можете щелкать по любой дате мышкой, она появится в текстовом поле и будет считаться значением DTPicker1. Можете щелкнуть по Today. Стрелками влево и вправо вы меняете месяцы. Вы можете вручную писать любую дату в текстовом поле. DTPicker1 воспринимает любые даты с 1601 по 9999 годы.

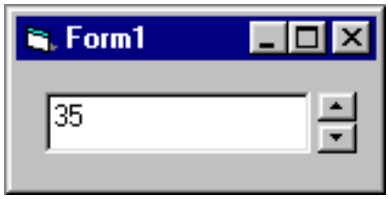
Этими элементами удобно пользоваться для ручного заполнения датами списков и баз данных.

UpDown

Разместите на форме элемент управления **UpDown**. Он имеет вид двух кнопок со стрелками. Рядом с ним разместите текстовое поле или другой элемент управления, способный показывать числа. Мы разместили текстовое поле Text1.

Задача: При помощи элемента UpDown изменять в текстовом поле числа от 0 до 100 с шагом 5 и обратно.

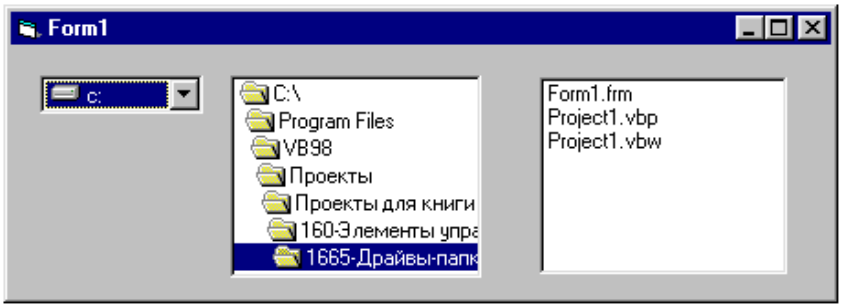
Установите следующие свойства элемента UpDown: Min - 0, Max - 100, Increment - 5, BuddyControl - Text1, BuddyProperty - Default. Запустите проект.



Убедитесь в правильности его работы.

Элементы DriveListBox, DirListBox, FileListBox

Все эти три элемента управления показаны в порядке слева-направо на рисунке.



Поместите их на форму, запустите проект. Пощелкайте по всем трем элементам. Вы видите, что в них отражена информация о вашем компьютере. **DriveListBox** показывает список логических дисков вашего компьютера, **DirListBox** показывает вам папки логического диска и позволяет путешествовать по ним, **FileListBox** показывает файлы в папке. Как видите, Visual Basic "чувствует" структуру дисков и папок вашего компьютера. Однако, щелчки по элементам не приводят ни к каким конкретным действиям. Вы даже не можете, выбирая логический диск в DriveListBox, изменить логический диск в DirListBox, и выбирая папку в DirListBox, изменить содержимое FileListBox. Не говоря уже о том, чтобы сохраняться или открывать файлы, как это вы привыкли делать при помощи похожих элементов в приложениях Windows. Для всего этого, как вы уже догадались, надо будет писать программу.

Но делать этого мы не будем, так как я планирую использовать для этих целей другой, более универсальный и удобный элемент управления - CommonDialog (см. 18.2).

RichTextBox

Это нечто среднее между текстовым полем и редактором Microsoft Word. Другими словами, это текстовое поле, которое вы можете обогатить многими возможностями солидного текстового редактора. Вы сможете создавать в нем вполне приличные документы. Отдельные фрагменты текста в окне RichTextBox вы сможете форматировать по-разному, то есть придавать им разный размер шрифта, разный цвет шрифта, придавать разные стили абзацам и т.д. Созданные документы вы можете сохранять в формате RTF, который воспринимается редактором Word. И открывать, естественно, тоже.

Найдете вы его в **Project → Components → Microsoft RichTextBox Control 6.0**.

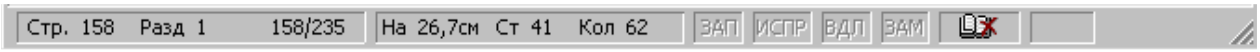
ListView и TreeView

Если вы хотите увидеть одно из применений ListView и TreeView, то откройте Проводник Windows (см. 0). Левая панель проводника это TreeView, а правая - ListView. Элемент **TreeView** приспособлен для отображения любых древовидных структур, не обязательно структуры папок на диске. Например, вы можете отобразить собственное генеалогическое древо. Элемент **ListView** приспособлен для удобного отображения списков, включая пиктограммы (значки, иконки) элементов списка. Списки могут отображаться и в табличном виде, подобно таблицам баз данных.

Закладка (TabStrip) и строка состояния (StatusBar)

Что такое **закладка**, вы можете посмотреть так: **File → Add Project**. Там вы увидите три закладки: New, Existing, Recent. Эти же три закладки вы можете видеть на картинке в 1.3.

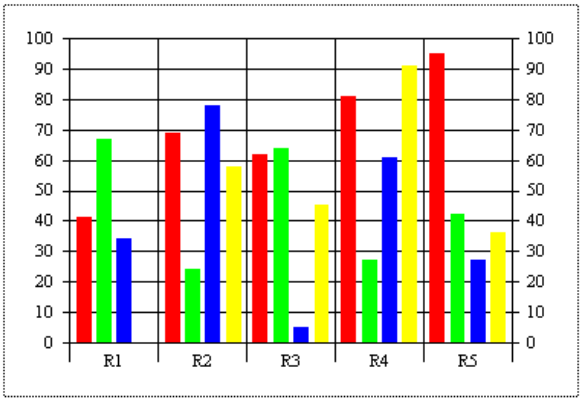
Строка состояния присутствует во многих приложениях Windows. Вот пример строки состояния редактора Microsoft Word:



Строка состояния всегда присутствует на экране и применяется поэтому для отображения информации, которую всегда удобно держать перед глазами.

MSChart

Если у вас в проекте есть числовые данные, то элемент управления MSChart поможет вам построить по этим данным красивую диаграмму. Найдете вы этот элемент в **Project → Components → Microsoft Chart Control 6.0**. Разместите его на форме. Он имеет такой вид:



Выбирая разные значения свойства chartType, вы изменяете тип диаграммы. Должен сказать, что источник данных для диаграммы подойдет далеко не всякий.

PictureClip

Применяется для хранения большого числа кадров графической информации. Эти кадры хранятся, как прямоугольные фрагменты одной растровой картинки. Из этих кадров можно делать мультфильмы. В 11.4 мы уже делали мультфильмы. PictureClip позволяет это делать с меньшей затратой ресурсов компьютера.

Найдете вы его в **Project → Components → Microsoft PictureClip Control 6.0**.

ImageCombo

Похож на ComboBox, но для каждого элемента списка показывает еще и пиктограммку.

MSComm

Этот элемент обеспечивает передачу и прием информации через последовательный порт компьютера.

Глава 16. Строки, файлы, обработка ошибок

В этой главе я собрал совершенно необходимые, но разнокалиберные вещи, которые по тематике не подходили к другим главам.

16.1. Строки

Со строковым типом String мы познакомились в 4.7. Какие интересные задачи связаны с работой со строками? Шифровка-дешифровка информации. Поиск в длинном тексте (например, в словаре или в инструкции по игре) нужного слова, которое просто так, глазами, искать очень долго. Автоматическое исправление орфографических ошибок в диктанте по русскому языку. И так далее.

Для решения этих задач мы должны уметь "влезать внутрь строк", то есть анализировать и преобразовывать строки. Visual Basic предоставляет для этого богатый набор функций. Вот некоторые из них:

Функция	Результат	Пояснение
"Мото" **+** "роллер"	Мотороллер	Операция + над строками просто соединяет строки в одну. Visual Basic может ее спутать со сложением чисел
"Мото" **&** "рол" **&** "лер"	Мотороллер	Операция & тоже соединяет строки в одну. Ее рекомендуется всегда применять вместо +, так как со сложением ее не спутаешь
Len ("Чук и Гек")	9	Длина строки, включая пробелы
Mid ("Астроном" , 3, 4)	трон	Часть строки длиной 4, начиная с 3-го символа
Mid ("Чук и Гек", 5, 1)	и	5-й символ в строке

При помощи функции Mid вы можете добраться до каждой буквы в тексте. Следующий фрагмент распечатывает в столбик слово "Телепортация":

```
s = "Телепортация"
For  i = 1 To Len(s)
   Debug.Print  Mid(s, i, 1)                    'Это i-я буква в строке
Next
```

Следующая функция позволяет искать в тексте нужное слово:

InStr ("Астроном", "трон")	3	Позиция (номер символа), начиная с которой строка "трон" находится в строке "Астроном"
InStr ("Астроном", "Трон")	0	Строка "Трон" не найдена в строке "Астроном"
Left ("Победа", 2)	По	2 левых символа в строке
Right ("Победа", 3)	еда	3 правых символа в строке
Ucase ("астРОнОм")	АСТРОНОМ	Все символы строки переводятся в верхний регистр
Lcase ("астРОнОм")	астроном	Все символы строки переводятся в нижний регистр

В 2.5 (Калькулятор) жизнь заставила нас познакомиться с функцией Val, которая преобразует строку в число, а в 4.10 (Переменные) - с функцией Str, которая преобразует число в строку. Напомним их вам:

Val ("20 груш и 8 яблок")	20	Функция читает строку слева направо, пока не натолкнется на символы, никакого отношения к числам не имеющие
Val (" - 1 0груш")	-10	На пробелы при этом внимание не обращается
3 * **Val** ("2" & "0")	60	Выражение "2" & "0" равняется строке "20", ну а Val ("20") равняется числу 20
Str (5 * 5)	25	Число 25 преобразуется в строку "25". Хотя, надо сказать, что Visual Basic при работе с данными во многих случаях сам, без всякого вмешательства, услужливо преобразовывает данные к удобному с его точки зрения типу.

Когда мы вводим текст в текстовое окно, мы часто не замечаем, что лишний раз нажали на клавишу пробела, тем более, что лишние пробелы, особенно в самом начале и в самом конце строки, заметить трудно. Мы не всегда заботимся о том, чтобы избавиться от них. А зачем? А затем, что компьютер пробелы видит не хуже любой буквы и считает их полноправными символами. Мы склонны считать строки "Африка" и "Африка " вполне одинаковыми. Компьютер же не может позволить себе такой вольности, он прекрасно видит, что во второй строке в конце стоит пробел. Значит строки не равны и это может привести к неожиданным для нас результатам. Следующие три функции позволяют нам справиться с невнимательностью:

Функция	Результат	Пояснение
"Ж" & **LTrim**(" Бутевни матлны ") & "Ж"	ЖБутевни матлны Ж	Функция LTrim отсекает ведущие слева пробелы
"Ж" & **RTrim**(" Бутевни матлны ") & "Ж"	Ж Бутевни матлныЖ	Функция RTrim отсекает волочащиеся справа пробелы
"Ж" & **Trim** (" Бутевни матлны ") & "Ж"	ЖБутевни матлныЖ	Функция Trim отсекает пробелы и слева и справа

Таблица ASCII

Символы, используемые в работе с компьютерами, сведены в так называемую таблицу ASCII. У каждого из них есть порядковый номер в этой таблице. Его нам сообщает функция Asc. Функция же Chr наоборот - по номеру сообщает символ:

Функция	Результат	Пояснение
Asc ("Ы")	219	Буква Ы стоит в таблице ASCII под 219 номером
Chr (219)	Ы	Под 219 номером в таблице ASCII стоит буква Ы

Всего в таблице ASCII 256 символов, пронумерованных от 0 до 255. Вот фрагмент, распечатывающий эти символы, начиная с 32-го:

```
For i = 32 To 255
   Debug.Print Chr(i);
Next
```

Вот результат работы фрагмента:

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
{|}~  ЂЃ‚ѓ„…†‡€‰Љ‹ЊЌЋЏ‘’“”•——  ™љ›њќћ ЎўJ¤Ґ¦§Ё©Є«¬-®Ї°±Ііґµ¶·ё№є»jSsї
АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюя
```

Задание 124: Определите без компьютера, что напечатает оператор Debug.Print Chr(Asc("Ю") + 1) ?

Задание 125: *"Детская шифровка"*. Среди детей встречается игра, заключающаяся в зашифровке своей речи “для секретности” за счет вставки в произносимые слова какого-нибудь звукосочетания, например, “быр”. Тогда вместо слова “корова” будет произнесено “кобырробырвабыр”. Составьте программу, которая распечатывает любую строку из 6 букв, после каждой второй буквы вставляя “быр”. Если получилось, то решите эту задачу для строки произвольной длины.

Задание 126: Давайте поставим задачу шифрования текста более серьезно. Имеется строка текста. Требуется написать программу, которая зашифровывала бы ее в другую строку. Способов шифровки вы можете придумать сколько угодно. Попробуйте такой – заменять каждый символ текста символом, следующим по порядку в алфавите (в таблице ASCII). Тогда слово кот превратится в слово лпу. Составьте, пожалуйста, и программу дешифровки. Когда вы познакомитесь с файлами, вы сможете уже зашифровывать и дешифровывать не отдельные строки, а целые тексты. В том числе и ваши программы.

Один из возможных путей программирования таких задач: Начните с того, что объявите массив, состоящий из строк длины 1, то есть, по сути, массив символов. Вот как это сделать: Dim s(20) As String * 1. Заполните массив символами исходной строки. Теперь вам будет удобней с ними работать. Придумайте, что делать с буквой "я". В ответе же эта задача решена другим, более коротким способом.

16.2. Файлы

Если вы играли в компьютерные игры, то наверняка сохранялись. А задумывались ли вы над тем, что значит сохраниться? Вы знаете, что в результате сохранения игра в следующий раз начинается с того места, где вы остановились раньше. А как компьютер помнит, где вы остановились? В каком месте компьютера хранится эта информация? В персональном компьютере два вида памяти - оперативная и на диске (смотри Приложение 1). Оперативная память стирается в момент выключения компьютера, а поскольку компьютер мы выключаем чуть ли не каждый день, то использовать ее для сохранения нельзя. Поэтому все, что нужно сохранить, компьютер запоминает на диске. Когда в следующий раз вы запускаете игру, то программа игры считывает с диска сохраненную информацию и с ее помощью позволяет вам продолжить игру с того места, где вы остановились. Какую именно информацию об игре нужно для этого сохранять, вам станет ясно позже.

Как книжка состоит из рассказов, так диск состоит из **файлов**. Файлов на диске множество. Каждая игра и вообще любая программа сохраняется в своем файле. Сколько игр, столько и файлов. Говорят, что когда происходит сохранение информации в файле, то информация **выводится** или **записывается** из оперативной памяти в файл, а когда игра или другая программа **читает** эту информацию из файла, то говорят, что информация **вводится** или **загружается** из файла в оперативную память.

Если вы еще не знакомы с понятием файла или папки, прочтите Приложение 2. Для определенности мы будем считать, что файл расположен именно на магнитном диске, хотя файл - достаточно общее понятие, которое может применяться к различным устройствам ввода, вывода и хранения информации.

В Visual Basic существует несколько типов файлов. Мы познакомимся с самым простым из них - текстовым файлом. Вам совершенно не обязательно знать, как физически устроен носитель, на который файл будет записываться. При работе с текстовым файлом удобно воображать, что носитель - не диск, состоящий из дорожек, а подобен листу бумаги или экрану монитора, файл же - это строки информации на этом листе или экране. Запись в файл и считывание из файла осуществляются магнитной головкой, которая движется по строкам файла строго последовательно, не пропуская ни строки, подобно авторучке, когда вы пишете письмо, или глазу, когда его читаете. Но у магнитной головки нет свободы глаза или авторучки, которые по вашему желанию могут "прыгать" по листу, как хотят. Головка движется строго последовательно, не пропуская ни символа, поэтому такие файлы еще называют **файлами с последовательным доступом**.

Данные в текстовых файлах могут быть числами, строками или иметь другой тип.

Сейчас на шести задачах мы научимся работать с текстовыми файлами.

Задача 1: Записать слово "Азия" и число 1998 на магнитный диск c: в текстовый файл с именем Filimon.txt, располагающийся в папке VB.

Решение: Прежде всего вы должны убедиться, что такая папка действительно существует по указанному адресу. Файла же там может и не быть. Затем придумаем файлу Filimon.txt номер, которым мы будем пользоваться в программе. Пусть это будет 1.

Прежде чем начать работать с файлом, он должен быть **открыт**. При этом компьютер выполняет определенные подготовительные действия для работы с файлом. Вот как выглядит оператор, открывающий текстовый файл для записи:

Оператор:	**Open**	"C:\VB\Filimon.txt"	**For**	**Output**	**As**	#1
Перевод:	Открыть файл	"C:\VB\Filimon.txt"	для	вывода	как	№1

Если файл раньше не существовал, то он создается. Если файл открывается для записи, то магнитная головка перемещается в начало файла. Это означает, как вы можете догадаться, что если в файле было раньше что-то записано, то все сотрется. Вот программа:

```
Private Sub Command1_Click()
  Open "C:\VB\Filimon.txt" For Output As #1  'Открыть для записи файл  Filimon.txt  в папке VB диска C под номером 1
  Write #1, "Азия"                           'Записать в файл №1 строку "Азия"
  Write #1, 1998                             'Записать в файл №1 в следующую строку число 1998
  Close #1                                   'Закрыть файл №1
End Sub
```

Вы видите, что собственно запись осуществляется оператором **Write**. После работы с файлом его нужно закрыть оператором **Close**.

Давайте убедимся, что все действительно правильно записалось. Для этого выйдем из Visual Basic в Windows, найдем нужную папку и обнаружим, что там действительно находится файл Filimon.txt. Чтобы заглянуть в него, щелкните по нему дважды мышкой, он откроется программой Notepad и вы увидите, что он содержит две строки:

```
"Азия"
1998
```

Если вместо двух операторов Write вы напишете один:

Write #1, "Азия", 1998

то в файл будет записана одна строка:

```
"Азия",1998
```

Если вы не хотите стирать содержимое файла, а просто хотите дописать что-нибудь в его конец, то вместо слова Output в операторе Open нужно использовать слово ***Append***.

Удобно держать текстовый файл, в который ваша программа сохраняет свои данные, в той же папке, где находятся файлы вашего проекта. В этом случае возникает проблема - при переносе папки с проектом в другое место диска изменяется ее адрес, а значит и адрес файла данных. В операторе же Open указан старый адрес. Visual Basic выдаст сообщение об ошибке. В этой ситуации нужно в операторе Open дать понять компьютеру, что файл находится именно в папке с файлами проекта. Делается это так:

Open **App.Path** & "\Filimon.txt" For ...

App.Path - это адрес папки вашего проекта, где бы он ни путешествовал.

Информация в наш файл может записываться только по порядку, последовательно. Мы не можем записать что-то сперва в начало файла, потом в конец, потом в середину. То же самое относится и к считыванию, о котором сейчас пойдет речь.

Задача 2: В файле Filantrop.txt в папке проекта записаны следующие строки:

```
1999
"Азия"
"Африка", 2000
….
```

Вывести третью строку на экран монитора.
Вот программа:

```
Private Sub Command3_Click()
  Dim a1 As Integer                          'Четыре переменные в оперативной памяти,
  Dim a2 As String                           'в которые будут загружены данные
  Dim a3 As String                           'из первых трех строк
  Dim a4 As Integer                          'файла
  Open App.Path & "\Filantrop.txt" For Input As #1   'Открыть для чтения под номером 1 файл Filantrop.txt из папки проекта
  Input #1, a1        'чтение 1-й строки
  Input #1, a2        'чтение 2-й строки
  Input #1, a3, a4    'чтение 3-й строки
  Close #1                                   'Закрыть файл 1
  Debug.Print a3, a4
End Sub
```

Вот результаты в окне Immediate:

```
Африка    2000
```

Вы видите, что собственно чтение осуществляется оператором **Input**. После работы с файлом его нужно закрыть оператором **Close**. Обратите внимание, что для того, чтобы добраться до третьей строки, нам пришлось "зря" прочитать первые две.

Задача 3: В предыдущей задаче мы еще до считывания знали, как располагаются по строкам и внутри строк данные в файле *Filantrop.txt*. Однако, нередки случаи, когда мы не имеем об этом представления. Все, что мы знаем, это то, что файл состоит из строк. Тем не менее, хочется прочитать данные из файла, чтобы хотя бы посмотреть на них. Для этого можно приказать Бэйсику загружать каждую строку файла, независимо от того, из каких данных она состоит, в переменную типа String.

Вот программа, распечатывающая три первые строки файла Filantrop.txt:

```
Private Sub Command4_Click()
   Dim s1 As String                                  'Три переменные в оперативной памяти,
   Dim s2 As String                                  'в которые будут загружены три
   Dim s3 As String                                  'первые строки файла
   Open App.Path & "\Filantrop.txt" For Input As #1  'Открыть для чтения под номером 1 файл Filantrop.txt из папки проекта
   Line Input #1, s1                                 'чтение очередной строки файла
   Debug.Print s1
   Line Input #1, s2                                 'чтение очередной строки файла
   Debug.Print s2
   Line Input #1, s3                                 'чтение очередной строки файла
   Debug.Print s3
   Close #1                                          'Закрыть файл 1
End Sub
```

Загрузку очередной строки в переменную типа String выполняет оператор **Line Input**. Вот результаты в окне Immediate:

```
1999
"Азия"
"Африка", 2000
```

Вы видите, что кавычки и запятая не исчезли, как это было в распечатке задачи 2. Это потому, что Line Input не обращает внимания на символы, из которых состоит строка файла. Он просто загружает их подряд в строковую переменную. В нашем случае, например, строка s3 состоит из символов "Африка", 2000, включая кавычки и запятую.

Если вы хотите как-то изменить содержимое текстового файла на диске, то вам целесообразно полностью загрузить его в память, изменить его там, как вам надо, а затем то, что получилось, целиком записать обратно в файл.

Задача 4: Файл f состоит из 10 строк. Дописать в конец каждой строки восклицательный знак.

Программа:

```
Dim s(1 To 10) As String
Private Sub Command1_Click()
        'Загружаем файл в память:
   Open App.Path & "\f.txt" For Input As #1      'Открыть для чтения под номером 1 файл f.txt из папки проекта
   For i = 1 To 10
      Line Input #1, s(i)                        'чтение очередной строки файла
   Next
   Close #1                                      'Закрыть файл 1
        'Преобразовываем строки в памяти:
   For i = 1 To 10
      s(i) = s(i) & "!"                          'Добавляем восклицательный знак в каждую строку
   Next
        'Записываем преобразованные строки в файл:
   Open App.Path & "\f.txt" For Output As #1      'Открыть для записи под номером 1 файл f.txt из папки проекта
   For i = 1 To 10
      Print #1, s(i)                             'Запись очередной строки в файл
   Next
   Close #1                                      'Закрыть файл 1
End Sub
```

Вы видите, что в этой программе запись осуществляется оператором **Print**, а не Write. Иначе бы строки в результирующем файле брались в кавычки, как это было в задаче 2.

Таким образом, оператор Print применяется для вывода не только на монитор, но и в файл.

Задача 5: Вам нужно прочесть все строки файла, а сколько строк в файле вы не знаете.

Ваши действия: В этом случае оператор цикла For не подойдет, так как там нужно указывать точное число строк. Хорошо бы Бэйсику можно было приказать: "Читай, пока файл не кончился". И такой приказ есть. В его основе лежит функция **EOF** (End of File), которая в процессе последовательного считывания файла все время "чувствует", кончился ли он, и если кончился, принимает значение True, иначе - False.

Упомянутый приказ реализуется фрагментом:

```
Do While  Not  EOF(1)     'Выполняй, пока НЕ наступил КОНЕЦ ФАЙЛА 1
     Line Input #1, s(i)
     i = i + 1
Loop
```

А теперь на примере примитивной задачи продемонстрируем, как программа может сохранять нужные ей данные, чтобы в следующий раз воспользоваться ими:

Задача 6: Пусть в файле Данные.txt записано число 10. После запуска программа должна раз в секунду печатать последовательные целые числа, начиная с числа, записанного в этом файле. Больше ничего делать ей не нужно. Понаблюдав некоторое время за печатью чисел, вы завершаете выполнение проекта. Вся соль задачи в том, чтобы при следующем запуске программа начала печатать не с 10, а с того числа, на котором завершился проект.

Ваши действия:

1. Вам необходимо сделать так, чтобы при запуске программы число считывалось из файла и счет начинался с него.
2. Вам необходимо сделать так, чтобы при завершении работы последнее из напечатанных чисел записывалось в файл на место десятки. Тогда в следующий раз счет сам собой начнется с него.

Создайте таймер. Настройте его интервал на 1 секунду.

Программа:

```
Dim Число As Integer

Private Sub Form_Load()
   Open App.Path & "\Данные.txt" For Input As #1      'Открыть для чтения под номером 1 файл Данные.txt из папки проекта
   Input #1, Число                                    'Чтение числа, с которого начать счет
   Close #1                                           'Закрыть файл №1
End Sub

Private Sub Timer1_Timer()                            'Процедура таймера, выполняемая раз в секунду
   Debug.Print Число                                  'Печать числа
   Число = Число + 1                                  'Получаем следующее число
End Sub

Private Sub Form_Unload(Cancel As Integer)
   Open App.Path & "\Данные.txt" For Output As #1     'Открыть для записи под номером 1 файл Данные.txt из папки проекта
   Write #1, Число                                    'Запись в файл числа, на котором закончен счет
   Close #1                                           'Закрыть файл №1
End Sub
```

Здесь я использовал событие **Form_Unload**. Оно так же, как и событие Form_Terminate, наступает при завершении работы проекта нажатием на крестик в правом верхнем углу формы. Для нас между этими событиями нет разницы.

Кроме файлов с последовательным доступом существуют еще **файлы с произвольным доступом**. Информация в этих файлах записывается упорядоченно и регулярно, благодаря чему к ней возможен более гибкий и быстрый доступ. На них мы останавливаться не будем.

Задание 127: Создайте игру "Угадай число". Компьютер загадывает число из диапазона от 1 до миллиарда. Человек должен его отгадать. Причем за наименьшее число попыток. При каждой попытке компьютер печатает номер попытки, число, предложенное человеком, и подсказку - "*мало*" или "*много*".

Поскольку даже у самых умных математиков на угадывание уйдет несколько десятков попыток, то в процессе угадывания может возникнуть желание сохраниться до лучших времен. При запуске игры в следующий раз компьютер должен спросить, начинать ли новую игру или продолжать старую. Если старую, то компьютер должен распечатать все то, что было распечатано в прошлый раз - все номера попыток, числа, подсказки.

Что еще можно делать с файлами

Visual Basic может выполнять над файлами и папками те же действия, что вы вручную выполняете в Проводнике Windows, а именно: создание папок, копирование, перемещение и уничтожение папок и файлов.

Пусть у вас на диске с в папке temp расположены папки 222, 333, 666, 999 и файл 1.txt.

Действия над файлами:

Оператор	Смысл
FileCopy "c:\temp\1.txt", "c:\temp\2.txt"	Оператор FileCopy копирует файлы. У него два параметра: первый - адрес копируемого файла, второй - адрес файла, в который производится копирование. В данном примере Файл 1.txt копируется в свою же папку под именем 2.txt
FileCopy "c:\temp\1.txt", "c:\temp\222\4.txt"	Файл 1.txt копируется в папку c:\temp\222 под именем 4.txt
Name "c:\temp\222\4.txt" **As** "c:\temp\222\5.txt"	Оператор Name As переименовывает и перемещает файлы и папки. Два его параметра имеют тот же смысл, что и у оператора FileCopy. В данном примере Файл 4.txt переименовывается в 5.txt и остается в своей папке.
Name "c:\temp\222\5.txt" As "c:\temp\5.txt"	Файл 5.txt перемещается в папку temp.
Kill "c:\temp\5.txt"	Оператор Kill уничтожает файл.

Действия над папками:

Name "c:\temp\333" As "c:\temp\444"	Папка 333 переименовывается в 444.
Name "c:\temp\666" As "c:\temp\222\666"	Папка 666 перемещается вместе со всем своим содержимым внутрь папки 222.
MkDir "c:\temp\888"	Создается пустая папка 888.
RmDir "c:\temp\888"	Уничтожается пустая папка 888.

В программировании широко используется понятие **текущей папки**. В данный момент времени текущая папка может быть только одна. Назначьте текущей папку, с файлами которой вы в данный момент работаете, и запись адресов в операторах существенно сократится. Например, работая в папке c:\temp\999, вы вместо того, чтобы все время писать операторы вида

FileCopy "c:\temp\999\22.txt", "c:\temp\999\44.txt"

один раз задаете текущую папку, вот так:

ChDir "c:\temp\999"

а затем уже все время пишете операторы такого вида:

FileCopy "22.txt", "44.txt"

Функция **CurDir** позволяет вам узнать, какая папка в настоящий момент является текущей. Для этого вам достаточно выполнить оператор Debug.Print CurDir.

16.3. Функция Shell

С помощью функции Shell вы можете, не выходя из проекта, запускать другие программы Windows. Например, строка

y = **Shell** ("C:\WINDOWS\Notepad.exe")

запускает стандартный текстовый редактор Windows - Блокнот (Notepad). В скобках вы должны указать адрес запускающего файла нужного вам приложения. Вместо адреса можно писать так называемую командную строку:

y = Shell ("C:\WINDOWS\Notepad.exe **C:\Untitled.txt**")

Здесь Блокнот откроется с загруженным файлом C:\Untitled.txt.

Вы можете управлять видом окна, в котором откроется программа:

y = Shell ("C:\WINDOWS\Notepad.exe C:\Untitled.txt", **vbMaximizedFocus**)

Здесь окно будет развернуто на весь экран. Попробуйте другие константы: **vbHide, vbNormalFocus, vbMinimizedFocus, vbNormalNoFocus, vbMinimizedNoFocus**.

16.4. Обработка ошибок. Оператор On Error

Многие функции и операторы, которыми вы пользуетесь в Visual Basic, вовсе не обязаны при любых обстоятельствах успешно завершать свою работу. Например, вы запустили процедуру для чтения файла с дискеты:

```
Private Sub Command1_Click()
   Open "a:\Файлик.txt" For Input As #1
   Input #1, a
   Close #1
   Debug.Print a
End Sub
```

но забыли вставить дискету в дисковод. Оператор Open не может выполнить свою работу, Visual Basic выдает сообщение об ошибке и работа приложения прерывается. Рядовой пользователь, работающий с вашим приложением, окажется в затруднительном положении. Он совсем не обязан разбираться в английском тексте сообщения, и если он даже догадается, в чем дело, и вставит дискету, все равно приложение надо будет запускать заново, так как оно прервано. Вы, как программист, должны учитывать, что с вашим приложением будут работать рядовые пользователи, причем даже не очень квалифицированные. Поэтому при программировании вы должны предугадать <u>все</u> возможные неправильные действия пользователя, чтобы при таких действиях приложение не прерывалось, а выдавало вразумительное сообщение на русском языке и советы по выходу из затруднительной ситуации.

Кое-что в этом направлении мы уже делали в 5.9, предохраняя калькулятор от арифметических действий над текстом и от деления на ноль.

Самым примитивным способом защиты вышеприведенной программы будет такой:

```
Private Sub Command1_Click()
   On Error GoTo m1

   Open "a:\Файлик.txt" For Input As #1
   Input #1, a
   Close #1
   Debug.Print a
   Exit Sub

m1: MsgBox ("Ошибка при вводе файла. Возможно, в дисководе нет дискеты.")
End Sub
```

<u>Пояснения</u>: Полужирным шрифтом я выделил новые элементы по сравнению с предыдущей программой. Оператор **On Error** GoTo m1 переводится так: "При возникновении ошибки иди к метке m1". Оператор On Error с момента своего выполнения заставляет компьютер быть начеку, не появится ли ошибка при выполнении программы. При ее появлении сообщение Visual Basic об ошибке уже не возникает, программа не прерывается, а управление передается на метку. Поэтому мы должны ставить оператор On Error раньше, чем операторы, могущие дать ошибку (в нашем случае это Open и Input). После метки вы должны поставить операторы, объясняющие пользователю, в чем дело, и помогающие как-то исправить ситуацию. В нашем случае никаких операторов, кроме MsgBox, не нужно. Забывчивый пользователь вставляет дискету в дисковод и снова жмет кнопку Command1. Все в порядке.

Обратите внимание на оператор Exit Sub. Если бы его не было, оператор MsgBox выполнялся бы всегда, даже при нормальной работе процедуры.

Написанный нами код слишком примитивен. Во-первых, мы не можем понять, какой оператор дал ошибку - Open или Input. Во-вторых, причин ошибки может быть несколько: отсутствие дискеты в дисководе, отсутствие файла на дискете и т.п. В первом случае мы вдобавок к оператору On Error GoTo m1 вставляем после Open, но перед Input, оператор On Error GoTo m2. В результате ошибка в строке Input будет обрабатываться операторами, начинающимися с метки m2., а ошибка в строке Open будет продолжать обрабатываться операторами, начинающимися с метки m1.

Во втором случае к нашим услугам объект **Err**, свойство которого **Number** принимает разные значения в зависимости от характера ошибки. Мы можем, например, писать такие строки:

```
If Err.Number = 71 Then
      MsgBox ("В дисководе нет дискеты.")
ElseIf ............
```

Номера и имена констант ошибок вы найдете так: **Help→Contents→ MSDN Library** (если она у вас установлена) **→Visual**

Studio documentation →Visual Basic documentation→Reference→Trappable Errors.

Глава 17. Функции. Параметры процедур и функций

Процедуры и функции с параметрами - важное средство сделать вашу программу надежнее и понятнее.

17.1. Функции. Параметры функций

Мы с вами уже сталкивались со стандартными функциями. Например, выражение ***10+Abs(-20)*** имеет значение 30, так как функция ***Abs(-20)*** обозначает абсолютную величину числа -20. Стандартная функция - это некая скрытая программа, которая принимает свои параметры, указанные в скобках, в качестве исходных данных, что-то делает с ними и в результате получает одну величину, которая и является **значением функции**. Множество других примеров функций вы найдете в 11.1, 16.1 и по всей книге.

Если вам недостаточно стандартных функций, вы можете создавать собственные **функции пользователя**:

Задача: Предположим, вам часто приходится вычислять периметры прямоугольников. Тогда вам было бы удобно иметь функцию вида ***perimetr(10,4)***, которая имела бы значение периметра прямоугольника со сторонами 10 и 4. Рассмотрим, как это делается, на примере программы вычисления суммарного периметра трех прямоугольников:

```
Private Function perimetr(dlina As Integer, shirina As Integer) As Long
   perimetr = 2 * (dlina + shirina)
End Function

Private Sub Command1_Click()
   Debug.Print    perimetr(10, 4) + perimetr(20, 30) + perimetr(3, 8)
End Sub
```

Здесь щелчком по кнопке ***Command1*** мы запускаем вычисление суммы периметров трех прямоугольников: 10*4, 20*30 и 3*8. А для того, чтобы узнать, как вычислять perimetr, компьютер заглядывает в определение функции, выделенное мной полужирным шрифтом.

Вы видите, что определение функции очень похоже на определение процедуры. Но функция в отличие от процедуры обладает некоторыми свойствами переменной величины и поэтому определение функции отличается от определения процедуры следующими двумя вещами:

- В заголовке функции после скобок с параметрами должен быть указан тип функции (у нас это ***Long***).
- Внутри определения функции ей хотя бы раз должно быть присвоено какое-нибудь значение, как если бы это была не функция, а обычная переменная (у нас этим занимается оператор ***perimetr=2*(dlina+shirina)***).

Обращение к функции также отличается от обращения к процедуре. Если обращение к процедуре - самостоятельный оператор (***Portos***, ***Aramis*** из 10.2, ***Рисуем_мой_значок 100, 200, vbBlue*** из 10.5), то обращение к функции - это обычно составная часть выражения (***10+Abs(-20)***, ***Debug.Print perimetr(10,4) + perimetr(20, 30)***).

Рассмотрим другой пример функции:

Задача: Напишите функцию Ответ, которая бы в ответ на вопрос человека "Какое сегодня число?" принимала значение правильной даты, а ответом на любой другой вопрос должно служить "Не знаю".

Вот программа:

```
Dim q As String

Private Function Ответ(Вопрос As String) As String
   If Вопрос = "Какое сегодня число?" Then Ответ = Date Else Ответ = "Не знаю"
End Function

Private Sub Command1_Click()
   q = InputBox("Задайте вопрос")
   Debug.Print Ответ(q)
End Sub
```

Здесь щелчком по кнопке ***Command1*** мы вызываем появление на экране окна InputBox с предложением задать вопрос. Введенный нами вопрос запоминается в переменной ***q***. Следующая строка (***Debug.Print Ответ(q)***) вызывает к жизни нашу функцию. При этом компьютер, выполняя тело функции, вместо параметра Вопрос работает с переменной ***q***, так как именно она была указана в обращении к функции. Получается очень удобно: когда мы пишем функцию, нам не нужно заботиться о том, какие имена переменных будут использованы при обращении к функции, мы просто даем параметру любое пришедшее в голову имя. И наоборот, когда мы пишем обращение к функции, нам не нужно заботиться о том, какие имена имеют параметры в заголовке функции.

Задание 128: Напишите функцию, вычисляющую любое число Фибоначчи. Например, ***fib(8)*** - это 8-е число Фибоначчи.
Указание: Если будете использовать массив, то дайте ему имя, отличающееся от имени функции, например, ***f***.

17.2. Локальные переменные

Материал следующих разделов этой главы сложен для восприятия и на первый взгляд непонятно, зачем он нужен. Но для уверенного плавания по океану Visual Basic он необходим. Итак, наберите побольше терпения и кислорода. Идем вглубь.

Будем называть процедуры и функции **подпрограммами**, так как они являются составными частями программы.

Переменные, которые объявлены в начале окна кода, могут быть использованы в любой подпрограмме этого окна. Говорят, что они **видны** из любой подпрограммы. Их называют **локальными переменными формы**.

Если же переменные объявлены внутри подпрограммы или являются параметрами подпрограммы, то они и использованы могут быть только внутри нее. Говорят, что они **не видны** из других подпрограмм. Их называют **локальными переменными подпрограммы**. О так называемых глобальных переменных поговорим позже (19.4).

Пример:

```
Dim a As Integer                    'a - локальная переменная формы

Private Sub Проц1()
   a = 1
End Sub

Private Sub Проц2()
   Dim b As Integer                 'b - локальная переменная процедуры
   b = 2
End Sub

Private Sub Проц3(c As Integer)     'c - локальная переменная процедуры
   Debug.Print c
End Sub

Private Sub Command1_Click()
   Проц1
   Debug.Print a
End Sub
```

Здесь ***a*** - локальная переменная формы, ***b*** и ***c*** - локальные переменные процедур. Вы видите, что a нормально используется в двух процедурах. Запустим проект. Щелкнем по кнопке ***Command1***. Печатается 1. Все в порядке. Процедуры Проц2 и Проц3 здесь не работали. Они приведены только для иллюстрации.

Теперь попробуем использовать ***b*** вне процедуры ***Проц2***. Для этого добавим к программе такую процедуру:

```
Private Sub Command2_Click()
   Проц2
   Debug.Print b                    'Ошибочный оператор. Переменная b из процедуры Проц2 отсюда не видна и не будет напечатана
End Sub
```

Щелкнем по кнопке ***Command2***. Вместо 2 печатается пустая строка. Также неудачей закончится попытка использования c.

Зачем такие ограничения и неудобства? Какой во всем этом смысл? Поговорим об этом.

Создание разных зон видимости для разных переменных является способом повышения надежности больших программ и понижения вероятности запутаться при их написании. Программы, создаваемые сегодня профессиональными программистами, очень велики - десятки и сотни тысяч строк. Таково, например, большинство игровых программ. Естественно, один человек не может достаточно быстро создать такую программу, поэтому пишется она обычно большой группой программистов. Для этого программа делится на части, и каждый программист пишет свою часть. И все равно, в каждой части присутствуют десятки и сотни подпрограмм с десятками и сотнями переменных, в которых человеку легко запутаться.

Исследуем взаимодействие подпрограмм в такой программе. Для этого рассмотрим бессмысленную "сложную" программу, не использующую локальных переменных подпрограмм. В ней автор озабочен вычислением переменной y и печатью переменной x:

```
Dim x As Integer                    'x - локальная переменная формы
Dim y As Integer                    'y - локальная переменная формы

Private Sub B()
   y = 10 * 10
   Debug.Print "Результат равен";
End Sub

Private Sub Command1_Click()
   x = 5
   B
   Debug.Print x
End Sub
```

Очевидно, программа, как и хотел ее автор, напечатает:

```
Результат равен 5
```

Все хорошо. Но при большом объеме программы возникает опасность, что автор случайно использует внутри какой-нибудь подпрограммы для ее нужд имя переменной, используемой в другой подпрограмме для других нужд, и таким образом испортит ее значение. Пусть, например, в нашей процедуре ***B*** автор опрометчиво присвоил бы значение 10*10 не переменной с именем ***y***, а переменной с именем ***x***. Тогда эта строчка программы выглядела бы так:

```
x = 10 * 10
```

Очевидно, данная программа к неудовольствию ее автора напечатала бы

```
Результат равен 100
```

Для защиты от таких ошибок автор должен внимательно следить, чтобы разные подпрограммы не использовали переменных с одинаковыми именами. Но для больших программ этот контроль очень трудоемок и неудобен. Для того, чтобы избежать его, в современных языках программирования и разработан механизм локальных переменных. Если программист знает, что его число 10*10 нигде, кроме как в процедуре ***B***, не нужно, он объявляет соответствующую переменную любым именем внутри процедуры ***B***, ничуть не заботясь, что переменные с таким же именем встречаются в других местах программы, и все нормально работает:

```
Dim x As Integer                    'x - локальная переменная формы

Private Sub B()
   Dim x As Integer                  'x - локальная переменная процедуры
   x = 10 * 10
   Debug.Print "Результат равен";
End Sub

Private Sub Command1_Click()
   x = 5
   B
   Debug.Print x
End Sub
```

Данная программа к удовольствию ее автора напечатает

```
Результат равен 5
```

Произойдет это вот по какой причине: Переменные, объявленные внутри и снаружи подпрограммы или внутри разных подпрограмм, Visual Basic считает разными переменными, даже если они имеют одинаковые имена. Переменная ***x***, объявленная снаружи подпрограммы, это совсем другая переменная, чем ***x***, объявленная в подпрограмме, и помещаются эти переменные в разных местах памяти. Поэтому и не могут друг друга испортить. Вы можете вообразить, что это переменные с разными именами $x_{локформ}$ и $x_{локпроц}$.

Переменная, объявленная внутри подпрограммы, невидима снаружи. Она локальна в этой подпрограмме. Она существует, пока работает подпрограмма, и исчезает при выходе из подпрограммы.

Переменная же, объявленная снаружи подпрограммы, видна из любой подпрограммы окна кода и каждая подпрограмма может ее испортить. Однако, когда подпрограмма натыкается на переменную ***x***, объявленную внутри самой этой подпрограммы, то она, благодаря описанному механизму, работает только с ней и не трогает переменную ***x***, объявленную снаружи.

Для тренировки рассмотрим еще один пример:

```
Dim x As Integer                    'x - локальная переменная формы
Dim z As Integer                    'z - локальная переменная формы

Private Sub B()
   Dim x As Integer                  'x - локальная переменная процедуры
   Dim y As Integer                  'y - локальная переменная процедуры
   x = 20: y = 30: z = 40
End Sub

Private Sub Command1_Click()
   x = 1: z = 2                      'x и z - локальные переменные формы
   B
   Debug.Print x, z                  'x и z - локальные переменные формы
End Sub
```

Программа напечатает

```
1        40
```

Пояснение: Оператор ***Debug.Print x, z*** находится снаружи процедуры ***B***, и поэтому локальная переменная процедуры ***x=20***, объявленная внутри ***B***, из него не видна. Зато прекрасно видна локальная переменная формы ***x=1***, которую он и печатает. Переменная же ***z*** не объявлена внутри ***B***, поэтому она является локальной переменной формы, и оператор ***z=40*** с полным правом меняет ее значение с 2 на 40.

Для полной ясности приведу порядок работы компьютера с этой программой:

В оперативной памяти Visual Basic отводит ячейки под X локформ и Z локформ.

Процедура Command1_Click начинает выполняться с присвоения значений X локформ = 1 и Z локформ = 2. Почему локформ, а не локпроц? Потому что переменные с именами X и Z в процедуре Command1_Click не объявлены, а значит волей-неволей процедуре приходится пользоваться локальными переменными формы.

Вызывается процедура B. При этом в оперативной памяти отводится место под X локпроц и У локпроц.

Присваиваются значения X локпроц = 20, У локпроц = 30 и Z локформ = 40.
Программа выходит из процедуры B. При этом исчезают переменные X локпроц = 20 и У локпроц = 30.
Компьютер печатает X локформ = 1 и Z локформ = 40.

Таким образом, вы видите, что смысл механизма локальных переменных - в увеличении надежности программирования. Я советую те переменные, которые наверняка не понадобятся вне процедуры, делать локальными в ней.

Статические переменные

Исчезновение значения локальной переменной при выходе из процедуры не всегда удобно. Например, если процедура предназначена для увеличения суммы, объявленной локальной переменной. Следующая процедура будет работать неверно, так как при каждом вызове процедуры сумма будет обнуляться:

```
Private Sub Command1_Click()
   Dim Число As Integer
   Dim Сумма As Integer
   Число = Text1.Text
   Сумма = Сумма + Число
End Sub
```

Чтобы она не обнулялась, объявим сумму **статической переменной**:

```
Private Sub Command1_Click()
   Dim Число As Integer
   Static Сумма As Integer
   Число = Text1.Text
   Сумма = Сумма + Число
End Sub
```

Теперь все в порядке.

17.3. Массивы как параметры

В качестве параметров процедур и функций можно подставлять любые выражения подходящего типа. Например, вместо

y = perimetr (10 , 4)

можно было бы написать

a = 4 : y = perimetr (10 , a)

или

a = 3 : y = perimetr (10 , a+1).

Параметры процедур и функций можно объявлять, как имеющие самые разные типы, причем не только простые, но и сложные. Если вы не знаете, как правильно объявить тип параметра в заголовке функции или процедуры, можете объявить его как Variant. Рассмотрим для иллюстрации пример с массивами.

Задача: Имеется два массива, по два числа в каждом. Напечатать сумму элементов каждого массива. Использовать функцию ***sum***, единственным параметром которой является суммируемый массив.

Программа:

```
Dim a(1 To 2) As Integer
Dim b(1 To 2) As Integer

Private Function sum(c As Variant) As Integer
   sum = c(1) + c(2)
End Function

Private Sub Command1_Click()
   a(1) = 10: a(2) = 20
   b(1) = 40: b(2) = 50
   Debug.Print sum(a), sum(b)
End Sub
```

Вычисляя функцию sum(a), Visual Basic работает с ячейками для элементов массива a, делая с ними все то, что операторы тела функции должны делать со значениями элементов массива c. Вычисляя же функцию sum(b), Visual Basic аналогично работает с ячейками для элементов массива b.

Задание 129. В школе два класса. В каждом - 5 учеников. Каждый ученик получил отметку на экзамене по физике. Определить, какой из двух классов учится ровнее (будем считать, что ровнее учится тот класс, в котором разница между самой высокой и самой низкой отметкой меньше).

Указание: Создать функции ***Минимум(c)***, ***Максимум(c)*** и ***Разница(c)***.

17.4. Передача параметров по ссылке и по значению

Передача параметров по значению - еще один способ повысить надежность программирования. Рассмотрим пример.

Вот процедура ***Квадр***, вычисляющая периметр и площадь квадрата по его стороне:

```
Dim A As Integer            'сторона
Dim P As Integer            'периметр
Dim S As Integer            'площадь

Private Sub Квадр(Сторона As Integer, Периметр As Integer, Площадь As Integer)
   Периметр = 4 * Сторона
   Площадь = Сторона ^ 2
End Sub

Private Sub Command1_Click()
   A = 10
   Квадр A, P, S
   Debug.Print "Сторона="; A; "Периметр="; P; "Площадь="; S
End Sub
```

Результат:

```
Сторона= 10 Периметр= 40 Площадь= 100
```

Здесь щелчком по кнопке вы приказываете компьютеру вычислить периметр и площадь квадрата со стороной 10. Компьютер, выполняя тело процедуры ***Квадр*** и присваивая значения переменным ***Периметр*** и ***Площадь***, тем самым присваивает значения переменным ***P*** и ***S***, то есть изменяет содержимое ячеек памяти, отведенных под эти переменные. Говорят, что в этом случае между вызывающей и вызываемой процедурами осуществляется **передача параметров по ссылке**. Чтобы подчеркнуть, что вы используете именно этот способ передачи параметров, вы можете записать заголовок процедуры так:

Private Sub Квадр (Сторона As Integer, **ByRef** Периметр As Integer, **ByRef** Площадь As Integer)

хоть это и излишне.

При этом способе вызываемая процедура получает полный контроль над переменными ***A***, ***P*** и ***S*** вызывающей процедуры и может присваивать им все, что хочет. Это удобно, но небезопасно. Ведь в этом случае переменные становятся беззащитными против ошибок в вызываемой процедуре. Так, программист может случайно, для каких-то других нужд, включить в процедуру какой-нибудь оператор, меняющий значение параметра ***Сторона***, например, ***Сторона=1***. Предположим, в этом случае процедура примет такой вид:

```
Private Sub Квадр(Сторона As Integer, Периметр As Integer, Площадь As Integer)
   Периметр = 4 * Сторона
   Площадь = Сторона ^ 2
   Сторона = 1
End Sub
```

Тогда результаты будут напечатаны неверно:

```
Сторона= 1 Периметр= 40 Площадь= 100
```

Чтобы обезопасить себя от такой ситуации, вы можете явно приказать Бэйсику, чтобы он не смел трогать такую-то переменную, отдающую свое значение параметру. Для этого достаточно заголовок процедуры написать так:

Private Sub Квадр (**ByVal** Сторона As Integer, Периметр As Integer, Площадь As Integer)

Теперь, что бы ни произошло с параметром ***Сторона***, значение переменной ***A*** меняться не будет. Убедитесь, что теперь снова все в порядке. Такой способ передачи параметров называется **передачей параметров по значению**.

Не переборщите с надежностью. Так, глупо было бы писать

Private Sub Квадр (**ByVal** Сторона As Integer, **ByVal** Периметр As Integer, **ByVal** Площадь As Integer)

так как результат в этом случае был бы такой:

```
Сторона= 10 Периметр= 0 Площадь= 0
```

Задание 130: На двух метеостанциях (***A*** и ***B***) в течение года измерялась температура. Соответственно созданы два массива чисел длиной 365. Затем оказалось, что на обеих станциях термометры были испорчены: на станции ***A*** термометр все время показывал температуру на 2 градуса выше настоящей, а на станции ***B*** - на 3 градуса ниже. Написать процедуру с двумя параметрами, которая исправляет один исходный массив и с ее помощью исправить оба массива. Один параметр - величина поправки, другой - массив температур.

17.5. Индукция. Рекурсия

Понятие *рекурсии* - сложное, но необходимое понятие для программиста.

Здесь мне никуда не уйти от классического примера о факториале. Факториалом целого положительного числа N называется произведение всех целых чисел от 1 до N. Например, факториал пяти равен 1*2*3*4*5, то есть 120. Факториал единицы считается равным 1.

Все понятно. Однако, существует еще один, совершенно ужасный способ определения, что такое факториал. Этот способ определения называется **индуктивным**. Вот он:

*"Факториал единицы равен 1. Факториал любого целого положительного числа **N**, большего единицы, равен числу **N**, умноженному на факториал числа **N-1**."*

Если вам уже все ясно, значит вы - профессор математики. Для обычных людей поясню. Возьмем какое-нибудь конкретное

N, например, 100. Тогда ужасное определение будет звучать проще: Факториал числа 100 равен числу 100, умноженному на факториал числа 99.

Ну и что? И как же отсюда узнать, чему равен какой-нибудь конкретный факториал, скажем, факториал трех? Будем рассуждать также совершенно чудовищным образом:

Смотрю в определение: Факториал трех равен 3 умножить на факториал двух. Не знаю, чему равен факториал двух. Поэтому спускаюсь на ступеньку ниже.

Смотрю в определение: Факториал двух равен 2 умножить на факториал единицы. Не знаю, сколько это. Спускаюсь еще на ступеньку.

Смотрю в определение: Факториал единицы равен 1. Вот, наконец-то - впервые узнал конкретное число. Значит можно подниматься обратно.

Поднимаюсь на одну ступеньку. Факториал двух равен 2 умножить на 1, то есть 2. Хорошо.

Поднимаюсь еще на ступеньку. Факториал трех равен 3 умножить на 2, то есть 6. Задача решена!

Рассуждая таким образом, можно вычислить факториал любого числа. Этот способ рассуждения называется **рекурсивным**.

:

Какое отношение все это имеет к компьютерам? Дело в том, что рекурсивный способ рассуждений реализован во многих языках программирования, в том числе - и в Visual Basic. Значит, этим языкам должен быть понятен и индуктивный способ написания программ.

Обозначим кратко факториал числа ***N***, как ***Factorial(N)***, и снова повторим наш индуктивный способ объяснения:

*"Если **N=1**, то **Factorial(N) = 1**.*

*Если **N>1**, то **Factorial(N)** вычисляется умножением **N** на **Factorial(N-1)**."*

В соответствии с этим объяснением напишем на Visual Basic функцию ***Factorial*** для вычисления факториала:

```
Private Function Factorial(ByVal N As Integer) As Long
   If N = 1 Then Factorial = 1
   If N > 1 Then Factorial = N * Factorial(N - 1)
End Function

Private Sub Command1_Click()
   Debug.Print Factorial(3)
End Sub
```

Что самое удивительное - функция работает! Несмотря на то, что в программе нигде не употребляется оператор цикла. Вся соль программы в том, что функция ***Factorial*** вместо этого включает в себя вызов самой себя - ***Factorial(N-1)***.

:

Что же происходит в компьютере во время выполнения программы? Механизм происходящего в точности соответствует нашему путешествию по ступенькам:

Все начинается с того, что мы щелкаем по кнопке и Visual Basic пробует выполнить строку ***Debug.Print Factorial(3)***. Для этого он вызывает функцию ***Factorial***. Выполнение подпрограммы начинается с того, что в памяти отводится место для всех параметров и локальных переменных, а значит и для нашего параметра ***N***. Затем число 3 подставляется на место параметра ***N***, то есть в память в ячейку ***N*** посылается 3. Затем выполняется тело функции. Так как 3>1, то Visual Basic пытается выполнить умножение ***3* Factorial(3-1)*** и сталкивается с необходимостью знать значение функции ***Factorial(2)***, для чего вызывает ее, то есть отправляется ее выполнять, недовыполнив ***Factorial(3)***, но предварительно запомнив, куда возвращаться.

Спускаюсь на ступеньку ниже. В соседнем месте памяти отводится место для ***N***. Это уже другое ***N***, путать их нельзя! В эту ячейку ***N*** посылается 2. Затем выполняется тело функции. Пусть вас не смущает, что Visual Basic второй раз выполняет тело функции, не закончив его выполнять в первый раз. Так как 2>1, то Visual Basic пытается выполнить умножение ***2* Factorial(2-1)*** и сталкивается с необходимостью знать значение функции ***Factorial(1)***, для чего вызывает ее.

Спускаюсь еще на ступеньку. В соседнем месте памяти отводится место еще для одного ***N***. В эту ячейку ***N*** посылается 1. Затем выполняется тело функции. Так как 1=1, то Visual Basic вычисляет ***Factorial=1***. Вот - впервые конкретное число. Затем Visual Basic пытается выполнить следующую строку ***if N>1 then Factorial = N* Factorial(N-1)***. Поскольку нельзя сказать, что 1>1, то строка не выполняется и выполнение тела функции закончено. Значит можно подниматься.

Поднимаюсь на одну ступеньку. Visual Basic возвращается внутрь тела функции (той, где ***N=2***) и успешно выполняет умножение - ***Factorial =2*1=2***.

Поднимаюсь еще на ступеньку. Visual Basic возвращается внутрь тела функции (той, где ***N=3***) и успешно выполняет умножение - ***Factorial =3*2=6***. Задача решена.

Итак, **рекурсией** в программировании называется вызов подпрограммы из тела самой подпрограммы.

Чем хорош рекурсивный стиль программирования? В нашей программе о факториале мы как бы и не программировали вовсе, а просто обяснили компьютеру, что такое факториал. Как бы перешли на новый уровень общения с компьютером: вместо программирования - постановка задачи.

Чем плох рекурсивный стиль программирования? Если мы для решения той же задачи напишем программу не с рекурсией, а с обычным циклом, то такая программа будет выполняться быстрее и потребует меньше памяти.

Задание 131: Напишите рекурсивную функцию ***fib*** для вычисления чисел Фибоначчи.

17.6. Сортировка

Здесь вы не узнаете ничего нового о Visual Basic. Будем совершенствовать технику программирования.

Пусть имеется ряд чисел: 8 2 5 4. Под **сортировкой** понимают их упорядочивание по возрастанию (2 4 5 8) или убыванию (8 5 4 2). Сортировать можно и строки (как слова в словаре).

Сортировка - очень распространенная вещь в самых разных программах, в частности - в системах управления базами данных.

Задача: Задан массив из 100 произвольных положительных чисел. Отсортировать его по возрастанию.

Идея решения: Если мы не можем сходу запрограммировать задачу, нужно подробно представить себе, в каком порядке мы решали бы ее вручную, без компьютера. Как бы мы сами сортировали 100 чисел? Мы бы запаслись пустым листом бумаги из 100 клеток. Затем нашли бы в исходном массиве максимальное число и записали его в самую правую клетку, а в исходном массиве на его месте записали бы число, меньшее самого маленького в массиве (в нашем случае подойдет 0). Затем нашли бы в изменившемся исходном массиве новое максимальное число и записали его на второе справа место, а на его место в исходном массиве - 0. И так далее.

Вот программа для 10 чисел:

```
Const N = 10                                          'N - размер массива
Dim massiv_ishodn(1 To N) As Integer                  'Это исходный массив
Dim massiv_rezult(1 To N) As Integer                  'Это наш пустой лист бумаги

    'Вспомогательная функция для поиска максимума в массиве m(1 To N). Она выдает значение
    'максимального элемента (maximum) и заодно номер этого элемента (Nomer_max):
Private Function maximum(m, N As Integer, Nomer_max As Integer) As Integer
  Dim i As Integer, max As Integer
  max = m(1): Nomer_max = 1                           'max -  "временный" максимум
  For i = 2 To N
    If max < m(i) Then
      max = m(i)
      Nomer_max = i
    End If
    maximum = max
  Next
End Function

    'Основная процедура сортировки исходного вектора mass_ish размера N в результирующий - mass_rez:
Private Sub sortirovka(mass_ish, N As Integer, mass_rez)
  Dim Nom_max As Integer
  For i = 1 To N
    mass_rez(N + 1 - i) = maximum(mass_ish, N, Nom_max)     'Пишем "в правую клетку"
    mass_ish(Nom_max) = 0                             'Ноль - на старое место
  Next
End Sub

Private Sub Command1_Click()
  massiv_ishodn(1) = 41                               'Задаем значения элементов исходного массива
  massiv_ishodn(2) = 8
  massiv_ishodn(3) = 17
  massiv_ishodn(4) = 82
  massiv_ishodn(5) = 20
  massiv_ishodn(6) = 2
  massiv_ishodn(7) = 30
  massiv_ishodn(8) = 12
  massiv_ishodn(9) = 6
  massiv_ishodn(10) = 9

  sortirovka massiv_ishodn, N, massiv_rezult          'Сортируем массив

  For i = 1 To N
    Debug.Print massiv_rezult(i);                     'Распечатываем отсортированный массив
  Next
End Sub
```

Функция ***maximum***, кроме того, что сама имеет значение максимального элемента массива, выдает еще порядковый номер максимального элемента - ***Nomer_max***. Это называется **побочным эффектом функции**.

Методов сортировки много. Приведенный метод - самый примитивный. Мало того, что нам пришлось расходовать память

на второй массив, для выполнения сортировки массива из 100 элементов понадобилось бы около 100*100=10000 операций сравнения элементов массива между собой.

Существуют методы гораздо более эффективные. Приведу один из них - **метод пузырька**. Представьте себе тонкую вертикальную трубку с водой. Запустим снизу пузырек воздуха. Он поднимется до самого верха и остановится. Затем пустим еще один. Он поднимется наверх и остановится сразу же под первым. Затем запустим третий и так далее все сто пузырьков.

А теперь представим, что это не трубка, а наш исходный массив, а вместо пузырьков поднимаются максимальные элементы.

Вот алгоритм: Сравним первый элемент массива со вторым, и если второй больше, то ничего не делаем, а если первый больше, то меняем местами первый и второй элементы. В этом вся соль метода. Затем повторяем это со вторым и третьим элементами - если третий больше, то ничего не делаем, а если второй больше, то меняем местами второй и третий элементы. Затем повторяем все это с третьим и четвертым элементами и так далее. Где-то по пути мы встретим максимальный элемент, и он, как пузырек, поднимется у нас до самого верха.

Теперь, когда мы знаем, что элемент номер 100 у нас самый большой, нам предстоит решить задачу сортировки для массива из остальных 99 элементов. Метод тот же. Запускаем второй пузырек и так далее.

Метод пузырька не требует второго массива, да и сравнений здесь в два раза меньше.

Вот программа:

```
Const N = 10                          'N - размер массива
Dim massiv(1 To N) As Integer         'Это  массив

    'Сортировка массива mass размером Razmer:
Private Sub puziryok(mass, Razmer As Integer)
  Dim i As Integer
  For m = Razmer To 2 Step -1         'Всего пузырьков - 9
    For i = 1 To m - 1                'i увеличивается - пузырек ползет вверх
      If mass(i) > mass(i + 1) Then   'Стоит ли обмениваться значениями
        c = mass(i)                   'Три оператора для обмена значений двух элементов с помощью
        mass(i) = mass(i + 1)         'транзитного элемента c
        mass(i + 1) = c
      End If
    Next i
  Next m
End Sub

Private Sub Command1_Click()
  massiv(1) = 41                      'Задаем значения элементов массива
  massiv(2) = 8
  massiv(3) = 17
  massiv(4) = 82
  massiv(5) = 20
  massiv(6) = 2
  massiv(7) = 30
  massiv(8) = 12
  massiv(9) = 6
  massiv(10) = 9

  puziryok massiv, N                  'Сортируем массив

  For i = 1 To N
    Debug.Print massiv(i);            'Распечатываем отсортированный массив
  Next
End Sub
```

В заключение скажу, что существуют методы гораздо более эффективные, чем метод пузырька.

Задание 132: Отсортируйте по возрастанию двумерный массив. Я имею в виду - нужно сделать так, чтобы:

a(1,1) <= a(1,2) <= ... <= a(1,N) <=
<= a(2,1) <= a(2,2) <= ... <= a(2,N) <=
<= a(3,1) <= a(3,2) <= ...

17.7. Объекты, как параметры процедур

Параметры процедур вполне могут быть объектного типа. Рассмотрим пример. Пусть вам нравится шрифт Times размера 20, стиля - курсив, синего цвета и вы хотите без хлопот настраивать указанным образом любой подходящий элемент управления на форме. Вы пишете следующую процедуру:

```
Private Sub Настройка_шрифта_для (Элемент_упр  As  Control)
  Элемент_упр.Font = "Times"
  Элемент_упр.FontSize = 20
  Элемент_упр.FontItalic = True
  Элемент_упр.ForeColor = vbBlue
End Sub
```

Здесь Control - объектный тип элементов управления. Теперь вы можете настраивать шрифт элементов управления очень

просто:

```
Private Sub Command1_Click()
   Настройка_шрифта_для   Label1
   Настройка_шрифта_для   Text1
End Sub
```

Глава 18. Проект, который выглядит солидно

В этой главе мы создадим заготовку "солидного" пректа. Солидного в том смысле, что он будет обладать элементами, наиболее часто встречающимися в большинстве солидных приложений Windows, таких как Microsoft Word, Visual Basic и др. Это диалоговые окна открытия и сохранения файла, выбора цвета, панели инструментов. Для их создания нам понадобятся элементы управления CommonDialog, Toolbar, ImageList. В качестве темы для заготовки проекта я выбрал графический редактор. Несмотря на то, что заготовка будет сверхпримитивна, вам будет понятно, как ее при желании улучшить и приблизить к настоящим графическим редакторам, а также снабдить элементами, которых у настоящих графических редакторов нет.

18.1. Из чего бывает "сделано" приложение Windows

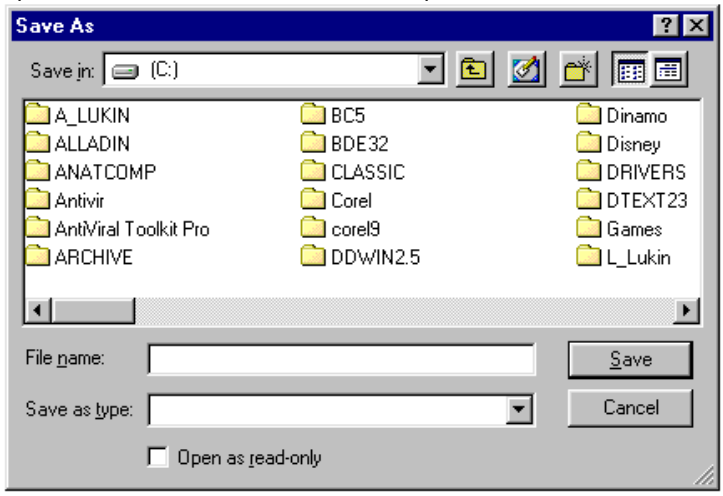
Посмотрим внимательнее, "из каких составных частей собрано" большинство солидных приложений Windows. Конечно, у каждого приложения есть элементы, присущие только ему, они-то и определяют работу приложения в соответствии с его назначением. Так Microsoft Word работает с текстом, Paint – с рисунками а Visual Basic выполняет программы. Но многие составные части присущи большинству приложений, основные из них мы и рассмотрим. При этом пометим для себя, какие элементы Visual Basic создают упомянутые части, умеем ли мы их создавать и работать с ними.

Составная часть приложения Windows	Элемент Visual Basic	Умеем?
Главное окно приложения	Форма	Да
Несколько окон внутри главного окна приложения	MDI Form	Нет
Меню	Menu Editor	Да
Полосы прокрутки	HScrollBar, VScrollBar	Да
Панели инструментов	Toolbar, CoolBar, ImageList	Нет, но сейчас научимся
Диалоговое окно открытия файла	CommonDialog	Нет, но сейчас научимся
Диалоговое окно сохранения файла	CommonDialog	Нет, но сейчас научимся
Диалоговое окно выбора цвета	CommonDialog	Нет, но сейчас научимся
Диалоговое окно выбора шрифта	CommonDialog	Нет
Диалоговое окно выбора принтера для печати	CommonDialog	Нет
Окно помощи Help	CommonDialog	Нет

Обратите внимание, какую богатую палитру возможностей обеспечивает элемент управления CommonDialog. А сам Visual Basic - это магазин кубиков, из которых можно собрать любое приложение Windows.

18.2. Элемент управления CommonDialog

До сих пор я знакомил вас с довольно простыми элементами управления, такими как кнопки, текстовые поля и т.п. Достаточно ли таких простых элементов, чтобы запрограммировать любые приложения Windows? В принципе, да. Возьмем, например, диалоговое окно сохранения файла. Оно похоже в большинстве приложений Windows.



Вы видите, что оно состоит из кнопок, флажка, раскрывающихся списков и других простых элементов, большинство из которых мы проходили. Все эти элементы собраны на форме с заголовком Save As. Таким образом, если мы захотим организовать сохранение файла, мы должны будем в проекте создать еще одну форму, которая и будет диалоговым окном (как созда-

вать в проекте дополнительные формы, я расскажу в 19.1). Затем мы поместим на эту форму нужные элементы управления и запрограммируем работу каждого из них. В общем, работа довольно большая, но выполнимая.

Разработчики Visual Basic постарались облегчить жизнь программистам и создали элемент управления **CommonDialog**, который *уже* имеет вид нужного диалогового окна. Простые элементы управления внутри диалогового окна *уже* запрограммированы надлежащим образом. Все, что остается программисту - это написать немного кода для настройки окна.

Common Dialog - настоящий многоликий Янус. Он может по желанию программиста перестраиваться на выполнение самых разных задач, перечисленных в нижеприведенной таблице. То, какую именно задачу он будет решать, определяется тем, какой из методов этого элемента будет выполнен:

Вставьте Common Dialog на панель Toolbox из **Projects→Components**. Там вы его найдете под именем Microsoft Common Dialog Control 6.0. Затем поместите его на форме. Его имя - Common Dialog1. В режиме работы он не виден и возникает в нужном обличье только при выполнении одного из своих методов:

Задача, решаемая элементом управления Common Dialog1	Метод
Открытие файла	CommonDialog1.ShowOpen
Сохранение файла	CommonDialog1.ShowSave
Выбор цвета	CommonDialog1.ShowColor
Выбор шрифта	CommonDialog1.ShowFont
Выбор принтера для печати	CommonDialog1.ShowPrinter
Окно помощи Help	CommonDialog1.ShowHelp

Пример открытия и сохранения файлов с помощью элемента Common Dialog

Чтобы лучше представить себе механику работы элемента Common Dialog, рассмотрим пример его использования для считывания и записи информации в произвольные текстовые файлы. Задача: Нажатием на кнопку Command1 поместить на экран диалоговое окно открытия файла, выбрать в нем произвольный текстовый файл, считать из него первую строку, затем при желании нажатием на кнопку Command2 преобразовать эту информацию (дописать в конец строки восклицательный знак), затем нажатием на кнопку Command3 поместить на экран диалоговое окно сохранения файла, выбрать в нем произвольный текстовый файл и информацию записать в него. Вот программа:

```
Dim s As String                    'Переменная для хранения считанной и преобразованной информации

Private Sub Command1_Click()
   CommonDialog1.ShowOpen          'Показать диалоговое окно открытия файла
   Файл = CommonDialog1.FileName   'Это имя файла выбрано из диалогового окна
   Open Файл For Input As #1       'Открываем выбранный файл для чтения
   Line Input #1, s                'Считывание первой строки из выбранного файла
   Debug.Print s
   Close #1
End Sub

Private Sub Command2_Click()
   s = s & "!"                     'Преобразование информации - добавление в конец строки восклицательного знака
End Sub

Private Sub Command3_Click()
   CommonDialog1.ShowSave          'Показать диалоговое окно сохранения файла
   Файл = CommonDialog1.FileName   'Это имя файла выбрано из диалогового окна
   Open Файл For Output As #1      'Открываем выбранный файл для записи
   Print #1, s                     'Запись информации в выбранный файл
   Debug.Print s
   Close #1
End Sub
```

Пояснения: На операторе CommonDialog1.ShowOpen появляется диалоговое окно открытия файла (аналогичное диалоговому окну сохранения файла, картинка которого показана выше), программа останавливается и ждет, когда мы выберем в окне какой-нибудь текстовый файл. После выбора окно пропадает, имя выбранного файла вместе с адресом становятся значением свойства FileName элемента CommonDialog, программа продолжает работу и это значение используется нами в операторе Open.

Аналогично работает и сохранение.

В приложениях Windows, таких как Word и Paint, мы привыкли, что открываемый файл становится нам виден в каком-нибудь окне. Здесь же ничего подобного нет, содержимого файла мы не видим. Для этого нужны дополнительные средства. На них мы не останавливаемся.

В диалоговом окне правильно работают многие кнопки и другие элементы. Однако нам нужно позаботиться кое о каких подробностях. Так, если мы хотим видеть в окне только текстовые файлы, нам нужно запрограммировать раскрывающийся список типов файлов. В противном случае мы можем нечаянно открыть графический файл, как текстовый, с неожиданными результатами. Также нужно позаботиться о том, чтобы при нажатии на кнопку Cancel не возникала ошибка. А возникает она по той причине, что оператор Open не знает, какой файл ему открывать. На этом я останавливаться не буду.

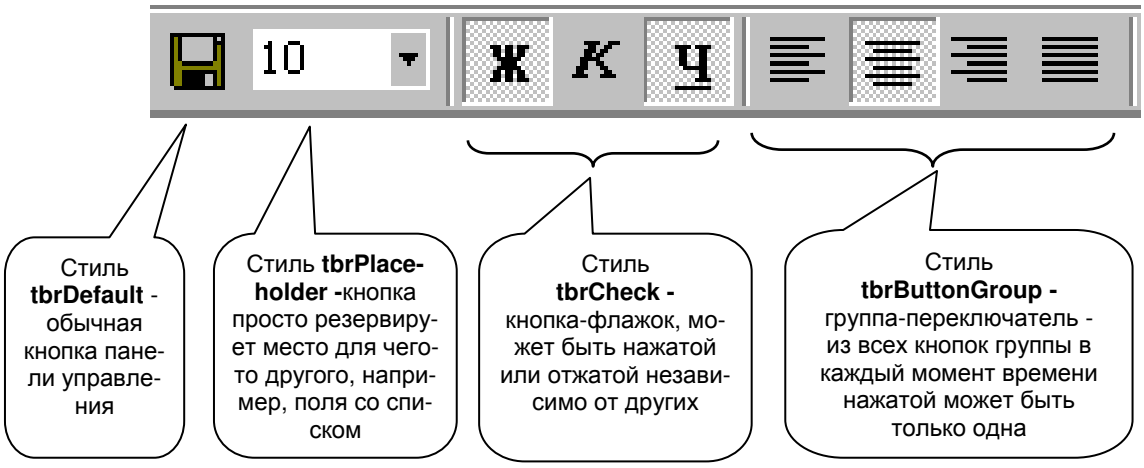
Вот более "правильная" запись этой же программы, использующая процедуры с параметрами:

```
Dim s As String                    'Переменная для хранения считанной и преобразованной информации

Private Sub Command1_Click()
   CommonDialog1.ShowOpen          'Показать диалоговое окно открытия файла
```

```
    Файл = CommonDialog1.FileName    'Это имя файла - параметр для процедуры Открой
    Открой Файл
End Sub

Private Sub Открой(Файл)
    Open Файл For Input As #1
    Line Input #1, s                 'Считывание первой строки из файла
    Debug.Print s
    Close #1
End Sub

Private Sub Command2_Click()
    s = s & "!"                      'Преобразование информации - добавление в конец строки восклицательного знака
End Sub

Private Sub Command3_Click()
    CommonDialog1.ShowSave           'Показать диалоговое окно сохранения файла
    Файл = CommonDialog1.FileName    'Это имя файла - параметр для процедуры Сохрани
    Сохрани Файл, s                  'Сохрани в Файл строку s
End Sub

Private Sub Сохрани(Файл, s As String)
    Open Файл For Output As #1
    Print #1, s                      'Запись информации в файл
    Debug.Print s
    Close #1
End Sub
```

18.3. Панель инструментов Toolbar

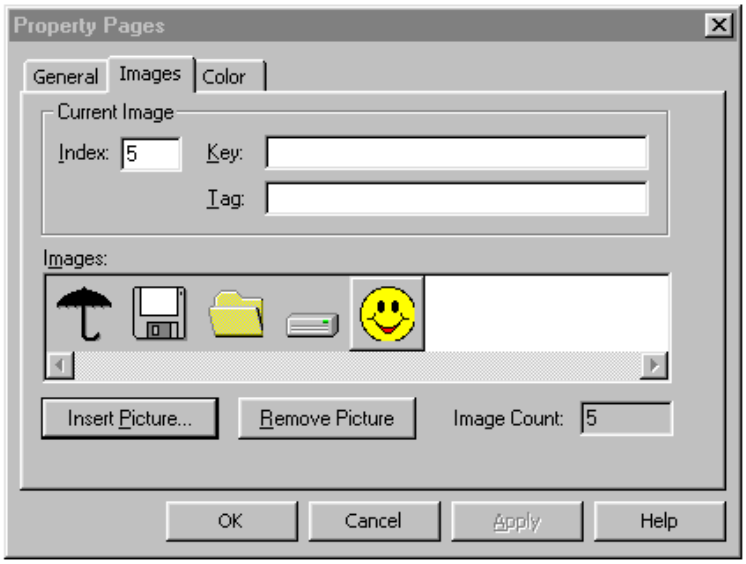
Все вы видели панели инструментов. Они имеются в большинстве солидных приложений Windows. Панель инструментов Visual Basic вы можете видеть на картинке в 1.3. Панель инструментов представляет набор кнопок и других элементов, предназначенных для быстрого выполнения наиболее часто встречающихся действий, таких как открытие, сохранение файлов, выбор размера шрифта и т.п. Кнопки на панели инструментов бывают разных стилей. Некоторые самые распространенные стили показаны на рисунке, где изображен участок панели управления Microsoft Word 97:



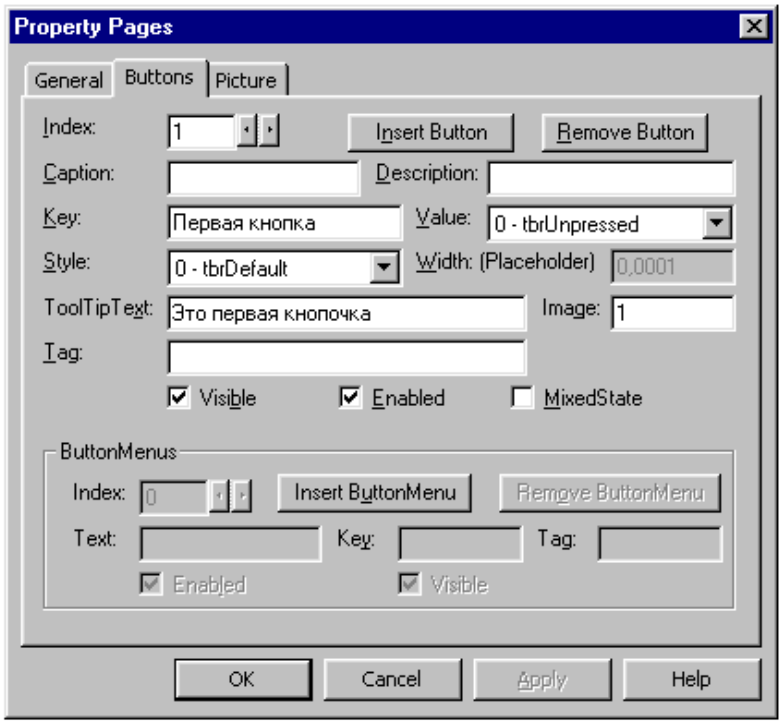
Задача: Создать панель инструментов из нескольких кнопок. По нажатии одной кнопки Visual Basic должен делать что-нибудь одно, например, выполнять оператор Debug.Print "Нажата первая кнопка"). По нажатии другой кнопки Visual Basic должен делать что-нибудь другое, например, выполнять оператор Debug.Print "Нажата вторая кнопка"). И так далее.

Ваши действия:

1. Поместить в Toolbox группу элементов управления Microsoft Windows Common Controls 6.0. Вы их найдете в **Projects→Components**. Из этих элементов вам понадобятся два: Toolbar и ImageList.
2. Разместить элементы Toolbar и ImageList на форме. При этом Toolbar сразу "прилипнет" к верхнему краю формы, как ей в общем-то и положено. Если вам это не нравится, то в пункте 6 вы это поправите. ImageList не будет виден в режиме работы. Его назначение вспомогательное - хранить в виде удобного списка картиночки, которые вы желаете видеть на кнопках панели инструментов.
3. Вставить в ImageList понравившиеся вам картиночки, например из принадлежащей Visual Basic папки Graphics. Для этого щелчком правой клавиши мыши по ImageList и выбором опции Properties вызываем на экран Страницы свойств (Property Pages) этого элемента. Там заходим в закладку Images, нажимаем кнопку Insert Picture, находим в появившемся окне нужную нам папку с картинками и выбираем картинку → **Open**. Замечаем, что выбранная картинка появляется в поле Images, а в поле Index появляется 1. Снова Insert Picture и аналогичным образом вставляем вторую картинку. Ее индекс - 2. И так далее. Закончив, нажимаем OK.



4. Вставить в Toolbar кнопки нужных вам стилей с картинками, взятыми из ImageList. Для этого щелчком правой клавиши мыши по Toolbar и выбором опции Properties вызываем на экран Страницы свойств (Property Pages) этого элемента. Там в закладке General в раскрывающемся списке ImageList объясняем компьютеру, в каком из нескольких возможных элементов управления ImageList хранятся картинки для кнопок. Затем заходим в закладку Buttons, нажимаем кнопку Insert Button, замечаем, что в поле Index появляется 1. Здесь нам необходимо выбрать стиль кнопки (Style) и ввести данные в два поля - Image и Key (ключ). В поле Image введем индекс картинки из ImageList, которую мы хотим поместить на кнопку. В поле Key введем любое слово, по которому в дальнейшем, как по имени, будем обращаться к этой кнопке. Также полезно заполнить поле ToolTipText, в этом случае при подведении мыши к кнопке будет всплывать подсказка. Закончив с первой кнопкой, снова нажимаем на Insert Button и повторяем всю процедуру для второй кнопки. И так далее. Закончив, нажимаем OK.



5. Запрограммировать действия кнопок. Для этого сделаем двойной щелчок по панели инструментов. Появится заготовка процедуры, в которой при помощи оператора Select Case программируем действие всех кнопок панели:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
  Select Case Button.Key
    Case "Первая кнопка"
      Debug.Print "Нажата первая кнопка"
    Case "Вторая кнопка"
      Debug.Print "Нажата вторая кнопка"
  End Select
End Sub
```

6. Настроить внешний вид и поведение панели инструментов (если хотите, конечно). Это можно сделать в страницах свойств и в окне свойств панели. Особое внимание обратите на свойство Align, от которого зависит положение панели инструментов на форме.

Кроме панели инструментов Toolbar существует еще интересная, с богатыми возможностями панель инструментов Coolbar, на которой мы останавливаться не будем.

Панели инструментов предоставляют возможность в режиме работы изменять свой внешний вид и состав, с чем вы, возможно, сталкивались, работая с такими приложениями Windows, как Microsoft Word.

18.4. Проект - "Графический редактор"

Проиллюстрируем применение элемента управления CommonDialog и панели инструментов на примере создания простейшего графического редактора. Пусть наш редактор умеет только следующие вещи:

- Рисовать мышкой, как карандашом.
- При помощи двух кнопок на панели инструментов выбирать одну из двух возможных толщин рисуемой линии.
- Выбирать цвет линии.
- Загружать рисунки из любого графического файла формата BMP.
- Сохранять рисунки в любом графическом файле формата BMP.

Вот внешний вид нашего графического редактора:

Вы видите здесь загруженную с диска картинку, поверх которой линиями разной толщины и цвета нарисованы самолеты и текст. Под кнопкой выбора цвета вы видите панель инструментов из двух кнопок для выбора толщины линии.

Вот программа нашего графического редактора (посмотрите, какая короткая!):

```
Dim Цвет As Long                    'Цвет карандаша

Private Sub Form_Load()             'Начальные установки редактора:
   AutoRedraw = True
   BackColor = vbWhite              'Цвет листа для рисования - белый
   Цвет = vbBlack                   'Поначалу цвет карандаша - черный
   DrawWidth = 3                    'Поначалу линия тонкая
End Sub

Private Sub Command1_Click()        'Загружаем графический файл
   CommonDialog1.ShowOpen
   Form1.Picture = LoadPicture(CommonDialog1.FileName)
End Sub

Private Sub Command2_Click()        'Сохраняем графический файл
   CommonDialog1.ShowSave
   SavePicture  Image, CommonDialog1.FileName
```

```
End Sub

Private Sub Command3_Click()          'Выбираем цвет карандаша
  CommonDialog1.ShowColor
  Цвет = CommonDialog1.Color
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  PSet (X, Y), Цвет                   'Мышка превращается в карандаш
End Sub

Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
  Select Case Button.Key
    Case "Тонкая линия"               'Если нажата первая кнопка на панели инструментов,
      DrawWidth = 3                   'то линия тонкая
    Case "Толстая линия"              'Если нажата вторая кнопка на панели инструментов,
      DrawWidth = 20                  'то линия толстая
  End Select
End Sub
```

Пояснения:

В процедуре открытия файла я для краткости вместо

```
CommonDialog1.ShowOpen
Файл = CommonDialog1.FileName
Form1.Picture = LoadPicture(Файл)
```

написал

```
CommonDialog1.ShowOpen
Form1.Picture = LoadPicture(CommonDialog1.FileName)
```

В программе используется новый для вас оператор

SavePicture Image, CommonDialog1.FileName

по которому картинка, нарисованная на поверхности формы, сохраняется в файл, имя которого указано в свойстве CommonDialog1.FileName. На смысле параметра Image я не буду останавливаться.

Работа процедуры выбора цвета вам должна быть понятна по аналогии с процедурой открытия файла.

При вставке кнопок в панель инструментов я дал первой из них ключ "Тонкая линия", а второй - "Толстая линия".

У нашей процедуры рисования Form_MouseMove есть два существенных недостатка:

- Она рисует, по сути, не линию, а последовательность точек, которые сливаются в линию только при медленном движении мышки. Происходит это потому, что события MouseMove вырабатываются не непрерывно, а через некоторые, пусть небольшие, промежутки времени.
- Мышь рисует всегда, когда движется, а нам было бы привычней и удобней, чтобы она делала это лишь при нажатой клавише мыши.

Вот как дополнится наша программа для устранения этих недостатков:

```
Dim Клавиша_мыши_нажата As Boolean      'Удерживается ли нажатой клавиша мыши
                    'Координаты мыши при предыдущем (перед текущим) наступлении события MouseMove:
Dim X_предыдущее As Long
Dim Y_предыдущее As Long

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  If Клавиша_мыши_нажата Then
                    'Рисуем отрезок прямой от предыдущего положения мыши до текущего:
    Line (X, Y)-(X_предыдущее, Y_предыдущее), Цвет
                    'Запоминаем текущее положение мыши для будущего срабатывания MouseMove, когда оно будет уже предыдущим:
    X_предыдущее = X
    Y_предыдущее = Y
  End If
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Клавиша_мыши_нажата = True
                'В начальный момент рисования предыдущее и текущее положения мыши совпадают:
  X_предыдущее = X
  Y_предыдущее = Y
End Sub

Private Sub Form_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Клавиша_мыши_нажата = False
End Sub
```

Пояснения:

Логично считать, что клавиша мыши удерживается нажатой, когда из двух событий - MouseDown и MouseUp последним по времени наступило MouseDown.

Вместо точек теперь рисуются отрезки прямых, соединяющие между собой соседние точки. Получается ломаная линия, неотличимая от плавной, гладкой кривой.

Если вы хотите улучшить наш графический редактор

Если вы хотите улучшить наш графический редактор, то:

- Сделайте кнопку очистки листа.
- Создайте палитру цветов, как в графическом редакторе Paint. Палитра цветов - та же панель инструментов, только кнопки в ней покрашены в разные цвета.
- Создайте еще одну панель инструментов с ластиком, отрезком прямой, прямоугольником, овалом (эллипсом).
- Интересно сделать пульверизатор (распылитель). Здесь вам понадобятся случайные величины.
- Создайте процедуры для рисования крестиков, треугольников и других фигур, которых нет в стандартных графических редакторах, и включите кнопки для их рисования в панель инструментов.
- Не советую создавать операции с фрагментами изображения: лупу, перенос, копирование и т.п. Для этого вы еще мало знаете.

Глава 19. Проекты из нескольких форм и модулей

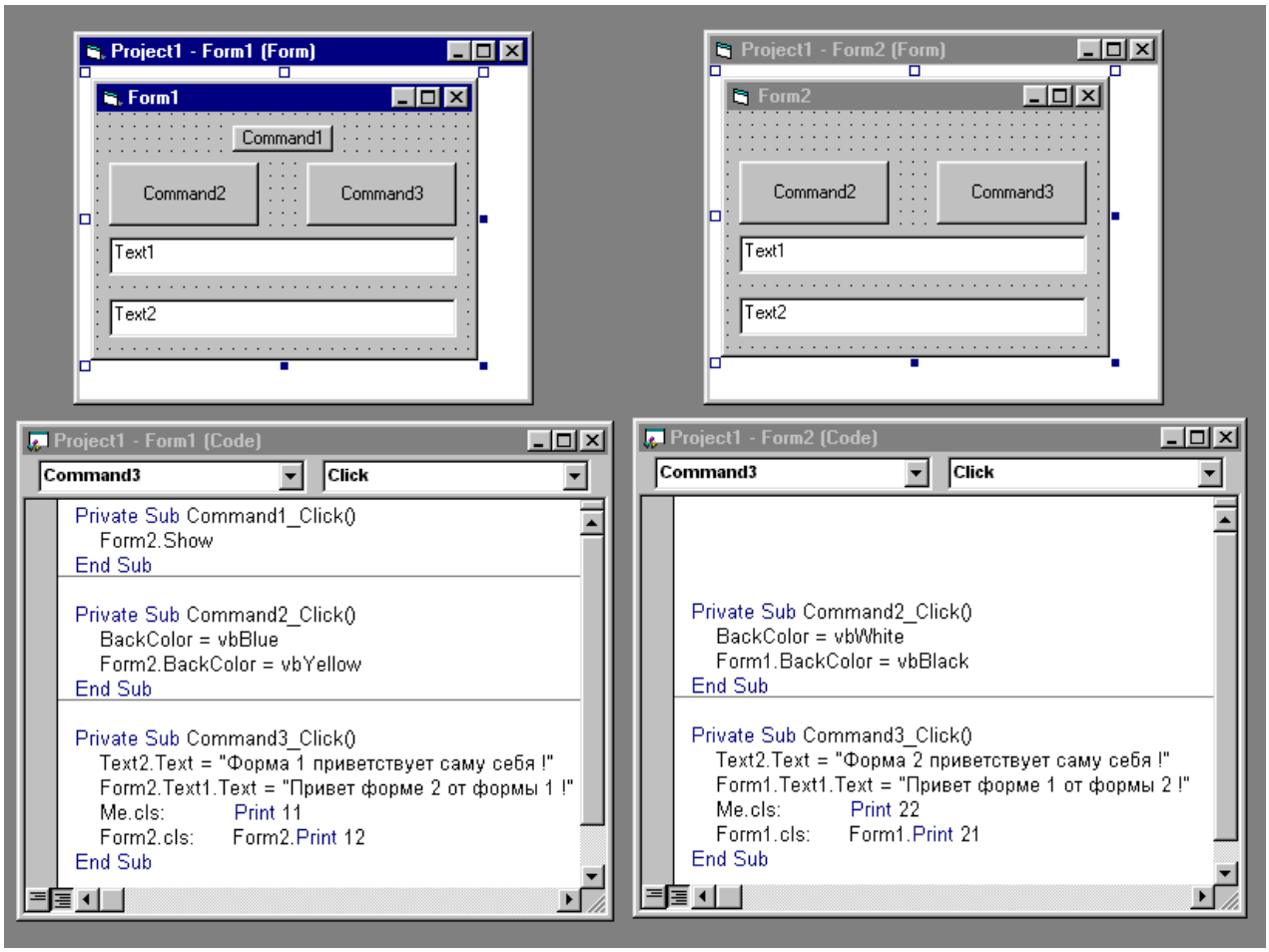
Эта глава - последний бастион, который нужно взять перед штурмом крепости по имени "Объекты". Но она и сама по себе нужна и важна.

19.1. Работа с несколькими формами

Все проекты, которые мы до этого создавали, работали в одном окне. Окном этим служила форма, на которой мы и собирали проект. Однако, вы вполне можете создать проект, который работает в двух или нескольких окнах. Например, вы создали игру в шахматы. Игра идет, как обычно, в одном окне. Пока компьютер обдумывает свой ход, вы со скуки щелчком по кнопке в окне игры запускаете новое окно, которое предлагает вам поиграть в созданный вами же тетрис. Когда компьютер сделает ход, вы можете поставить тетрис на паузу и продолжить игру в шахматы. Эту игру тетрис вы создавали в том же самом проекте, но собирали ее на другой форме этого проекта.

Посмотрим, как создать в проекте еще одну форму и как наладить простейшее взаимодействие между двумя формами.

Создадим новый проект. При этом, как обычно, будет создана форма Form1. Чтобы добавить в проект еще одну форму, нужно выбрать **Project→Add Form→New→Form→Open**. Откроется форма Form2, а при ней свое окно кода. Таким образом, вы видите, что Visual Basic предоставляет удобную возможность программировать все происходящее в новой форме в окне кода, принадлежащем именно этой форме. Поместим для удобства рядом, как это показано на рисунке, обе формы со своими окнами кода. Разместим на формах показанные элементы управления, а в окна кода запишем программный текст.



Запустим проект. Мы видим, как и положено, Form1, а вот Form2 не появляется. Чтобы она появилась, нужно выполнить оператор Form2.Show. Щелкнем по Command1 - форма Form2 появилась.

Пощелкайте по кнопкам Command2 на обеих формах. Теперь посмотрите на содержимое процедур Command2_Click в обоих окнах кода. Вы видите, что первая строка обеих процедур красит своего хозяина, вторая - соседа. Таким образом, в каждом окне кода можно, как и обычно, писать операторы, приказывающие что-то сделать в форме-хозяине данного окна, а можно с таким же успехом писать операторы, приказывающие что-то сделать в любой другой форме проекта. Чтобы Visual Basic отли-

чал первый случай от второго, во втором случае перед именем свойства, принадлежащего другой форме, нужно писать имя этой формы с точкой, в первом же случае - не обязательно. Не обязательно, но можно. Так, в левом окне вместо BackColor = vbBlue можно было бы написать Form1.BackColor = vbBlue или даже так - Me.BackColor = vbBlue. Слово **Me** является указанием на хозяина.

Пощелкайте по кнопкам Command3 на обеих формах. Теперь посмотрите на содержимое процедур Command3_Click в обоих окнах кода. Вы видите, что первая строка обеих процедур выводит текст в текстовое окно Text2 своего хозяина, вторая - в текстовое окно Text1 соседа. Таким образом, соглашение из предыдущего абзаца об имени формы с точкой применимо не только к свойствам формы, но и к объектам, принадлежащим форме, и, как мы сейчас увидим, методам.

3-я строка обеих процедур при помощи методов cls и Print очищает от графики поверхность формы-хозяина и печатает на ней число (11 на левой, 22 на правой). Слово Me пришлось написать, чтобы Visual Basic не посчитал cls: меткой. 4-я строка обеих процедур очищает от графики поверхность формы-соседа и печатает на ней число (21 на левой, 12 на правой).

Чуть позже вы увидите, что из одной формы можно обращаться не только к свойствам, объектам и методам, принадлежащим другой форме, но и к переменным, процедурам и другим элементам. Правило везде одно:

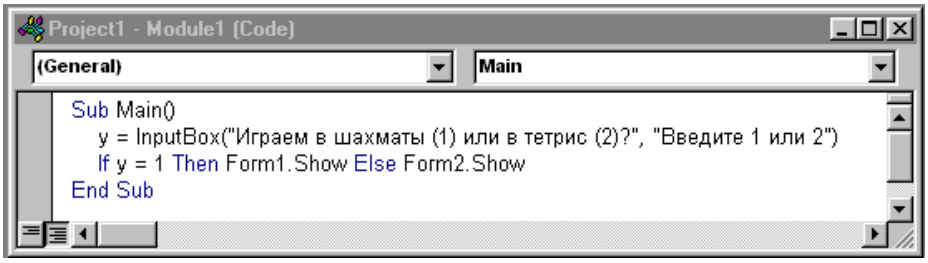
Перед именем элемента, принадлежащего другой форме, нужно ставить имя формы-хозяина с точкой.

19.2. Модули кода

Пусть вы создали проект из двух форм: в одной игра в шахматы, в другой игра в тетрис. Каждая из форм при запуске проекта довольно долго загружается. Поэтому вы хотите, чтобы перед загрузкой форм компьютер спросил у вас, какую форму нужно загружать. Но мы привыкли, что при запуске проекта на экране какая-нибудь форма появляется обязательно и первым делом. Мы можем изменить этот порядок и сделать так, чтобы никакая форма не загружалась.

Добавим в проект так называемый **модуль кода** (его еще называют программным или стандартным или просто модулем): **Project→Add Module→New→Module→Open**. Вы видите, что в проект добавилось окно кода модуля Module1. А ничего похожего на форму не появилось. Ни формы, ни кнопок, ни меток, никаких других элементов управления в модуле кода нет и быть не может. Если форма без кода - тело без души, то модуль - душа без тела. Ну что ж, в предыдущем разделе мы научились вселять душу в чужое тело, значит и душа нашего модуля на что-нибудь сгодится.

Проект можно настроить так, чтобы при запуске ни одна форма не загружалась, а выполнялась процедура со специальным названием **Main**, находящаяся в модуле кода. Запишем в окно кода нашего модуля четыре строки:



А вот как настраивать проект: **Project → Project1 Properties → General → Startup Object →** здесь Visual Basic предложит вам выбор, с чего начинать запуск проекта - с загрузки формы Form1, с загрузки формы Form2 или с выполнения процедуры Main. Вы, конечно, выбираете **→Sub Main→OK**.

Запустив этот проект, вы не видите никакой формы, вместо этого перед вами InputBox с приглашением ввести 1 или 2. Введя 1, вы тем самым загружаете Form1, а введя 2 - Form2.

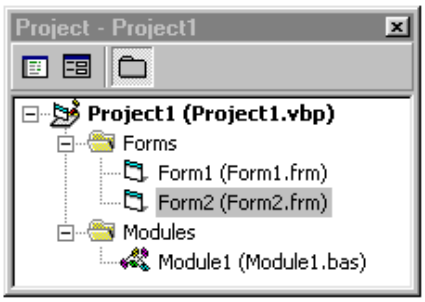
Термином **модуль** часто называют не только модуль кода, но и форму, и неизвестный пока нам модуль класса и некоторые другие сохраняемые в собственных файлах главные компоненты проекта. Я тоже буду так делать, когда это не будет вызывать путаницы.

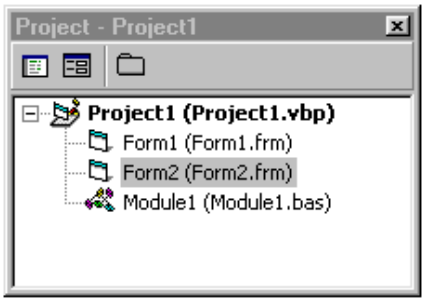
О другом, более часто встречающемся применении модуля кода см. в Глава 20.

19.3. Структура проекта. Окно Project Explorer.

Работа с несколькими модулями

А теперь посмотрите, как выглядит проект предыдущего раздела в окне Project Explorer.

Он имеет такой 

или такой 

вид в зависимости от того, нажата или нет правая из трех кнопок на панели окна. В обоих случаях отображается одна и та же информация, а именно тот факт, что проект Project1 состоит из трех модулей: формы Form1, формы Form2 и модуля кода Module1. Только в первом случае для нашего удобства модули одного типа объединены в воображаемые папки.

Каждый из модулей сохраняется на диске в своем файле. Имя файла вы видите в скобках. Щелкая в окне Project Explorer по нужному модулю, а затем по одной из двух кнопок слева на панели окна, мы можем удобно переключаться между различными окнами модулей.

Вы можете удалять модули из проекта, щелкая правой клавишей мыши в окне Project Explorer по ненужному модулю, а затем в открывшемся контекстном меню выбирая Remove. Так, если у вас простой вычислительный проект, вы вообще можете удалить все формы, обойдясь одним только модулем кода. Для ввода информации в компьютер вам достаточно будет использовать InputBox, для вывода - MsgBox, а для вычислений - процедуры в модуле кода.

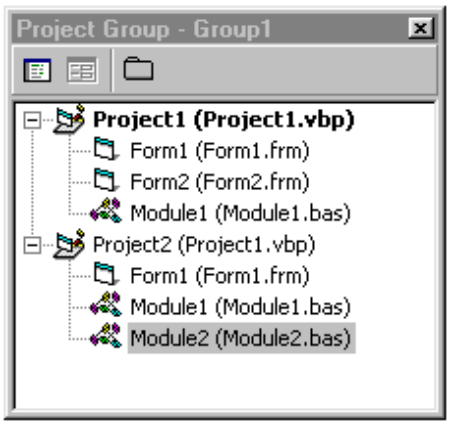
Работа с несколькими проектами

При работе с Visual Basic довольно часто встречаются ситуации, когда необходимо открыть в среде Visual Basic одновременно два или несколько проектов. Причина может быть разная, самая тривиальная - необходимость удобного копирования отдельных частей программного текста из одного проекта в другой.

Откройте сначала обычным образом один проект. Чтобы открыть еще и другой, вам нельзя использовать, как вы привыкли, **File → Open Project** (открыть проект), так как при этом уже открытый проект закроется. Нужно использовать **File → Add Project** (добавить проект). Откроется диалоговое окно, глядя в которое вы должны решить, хотите ли вы добавить в среду разработки новый проект (закладка **New**) или же один из уже существующих (закладка **Existing**). Затем выбираете нужный проект и → **Open**. Теперь на экране вы можете видеть одновременно все формы и окна кода обоих проектов.

Если вы добавили новый проект, то при сохранении он попытается сохранить свои файлы в ту же папку, где сохранен первый проект. Не стоит этого разрешать, а если уж разрешили, то проследите, чтобы имена файлов в обоих проектах различались, иначе файлы нового проекта затрут одноименные файлы первого.

Вот как выглядит Project Explorer в среде Visual Basic с двумя проектами:



Какой из двух проектов будет запускаться, когда вы привычно нажмете кнопку Start на панели инструментов? Вы сами можете это задать, щелкнув правой клавишей мыши в окне Project Explorer по названию нужного проекта, а затем в открывшемся контекстном меню выбрав Set as Start Up. Аналогичным образом можно удалить проект из среды (не с диска), выбрав в этом же контекстном меню Remove Project.

19.4. Зоны видимости

Зоны видимости переменных

А теперь поговорим о важном средстве обеспечения удобства и надежности программирования на Visual Basic - о механизме задания зон видимости переменных, процедур и других элементов Visual Basic. В 17.2 мы уже сталкивались с этим механизмом, когда переменные, объявленные внутри процедуры, являлись локальными в процедуре, то есть невидимыми снаружи процедуры. Поэтому их нельзя было использовать в других процедурах модуля. Если вы подзабыли этот раздел, то сейчас перечтите.

Перечли? Хорошо. Для переменных в Visual Basic определены 3 зоны видимости:

1	Локальные переменные процедуры	Видны только внутри процедуры, в которой они объявлены	Объявляются оператором Dim внутри процедуры
2	Локальные переменные модуля	Видны везде внутри модуля, в котором они объявлены. Из других модулей не видны	Объявляются оператором Dim или Private в верхней части модуля, снаружи процедур
3	Глобальные (общедоступные) переменные проекта	Видны отовсюду из всех модулей проекта	Объявляются оператором Public в верхней части модуля, снаружи процедур

Как видите, первая зона - самая узкая, третья - самая широкая. Пример окна кода:

```
Public a As Integer          'Глобальная переменная
Private b As Integer         'Локальная переменная модуля

Private Sub Command1_Click()
    Private c As Integer     'Локальная переменная процедуры
  . . . . . . .
End Sub
```

С первыми двумя зонами видимости вы знакомы. С третьей зоной сейчас познакомимся. Вот как можно из одного модуля обращаться к глобальным переменным, объявленным в другом модуле. Пусть в нашем проекте созданы две формы: Form1 и Form2. Вот окно кода формы 1:

```
Public a As Integer
Private Sub Command1_Click()
   Form2.Show
   a = 10
End Sub
```

Оператор **Public** объявляет переменную a как **глобальную** (или общедоступную), видимую изо всех модулей. Щелкнув по кнопке формы 1, вы присваиваете переменной a значение 10.

Вот окно кода формы 2:

```
Private Sub Command1_Click()
   Debug.Print Form1.a
End Sub
```

Щелкнув по кнопке формы 2, вы печатаете 10 - правильное значение переменной a.

Как видите, для того, чтобы обратиться к переменной, объявленной в другом модуле, необходимо указать хозяина переменной. Visual Basic привык, что у чужих элементов хозяин должен быть указан, если же хозяин не указан, значит элемент "свой". Попробуем вместо Debug.Print Form1.a написать просто Debug.Print a. Поскольку хозяин не указан, Visual Basic поймет, что переменная a "своя", то есть принадлежит форме 2. (Это ничего, что она не объявлена, Visual Basic все равно считает ее существующей.) А поскольку "своей" переменной a ничего не присвоено, то ничего и не будет напечатано. В пошаговом режиме вы увидите, что если в форме 1 значение a равно 10, то в форме 2 значение a равно **Empty** (a=Empty). По английски это значит "пусто", то есть переменной не было присвоено никакого значения.

Итак, если мы хотим, чтобы переменная была видна во всем проекте, мы объявляем ее оператором Public. Такая переменная называется глобальной. Если же мы хотим, чтобы переменная была видна только в своем модуле (была локальной в модуле), мы объявляем ее оператором Dim. Вместо оператора Dim принято использовать оператор **Private**. По действию они неотличимы, но английский смысл слова Private (частная собственность, вход запрещен) лучше подходит к случаю, поэтому программисты в основном используют его.

Зоны видимости процедур

Процедуры могут быть или глобальными или локальными

Для процедур в Visual Basic определены 2 зоны видимости:

1	Локальные процедуры модуля	Видны везде внутри модуля, в котором они объявлены. Из других модулей не видны	Объявляются оператором Private
2	Глобальные (общедоступные) процедуры проекта	Видны из всех модулей проекта	Объявляются оператором Public

Вот пример использования глобальной процедуры П2.

Окно кода формы 1:

```
Private Sub Command1_Click()
   Form2.П2
End Sub
```

Окно кода формы 2:

```
Public Sub П2()
   Debug.Print "Выполнилась процедура П2"
End Sub
```

Щелчком по кнопке формы 1 мы печатаем текст "Выполнилась процедура П2".

Зоны видимости констант и типов

Для констант в Visual Basic так же, как и для переменных, определены 3 зоны видимости:

1	Локальные константы про-	Видны только внутри процедуры, в кото-	Объявляются оператором Const внутри про-

	цедуры	рой они объявлены	цедуры
2	Локальные константы модуля	Видны везде внутри модуля, в котором они объявлены. Из других модулей не видны	Объявляются оператором Const в верхней части модуля, снаружи процедур
3	Глобальные (общедоступные) константы проекта	Видны изо всех модулей проекта	Объявляются оператором Public Const в верхней части модуля кода (и только в нем).

Перечислимые типы могут задаваться только на уровне модуля, а не процедуры. Слова Private и Public по отношению к ним имеют обычный смысл. Например, в окне кода формы 1 вы можете определить тип:

```
Public Enum tip
  a
  b
End Enum
```

Тогда в окне кода формы 2 вы можете объявить переменную:

```
Dim s As tip
```

Кстати, в определении перечислимого типа можно убрать слово Public. Visual Basic по умолчанию считает перечислимые типы глобальными.

Пользовательские типы тоже могут задаваться только на уровне модуля. Слова Private и Public по отношению к ним также имеют обычный смысл. Но не во всех модулях разрешено объявлять глобальные пользовательские типы.

19.5. Затенение

Переменные разных модулей или разных процедур вполне могут иметь одинаковые имена. Спрашивается, как Visual Basic определяет, какая из видимых одноименных переменных имеется в виду в каждом конкретном случае? Здесь вступает в действие **эффект затенения**: из нескольких одноименных переменных всегда имеется в виду более локальная переменная, то есть та, чья зона видимости меньше. То есть переменные, локальные в процедуре, имеют предпочтение перед переменными, локальными в модуле, а те - перед глобальными переменными. В этом есть глубокий смысл. Программист, объявляющий переменные в своей процедуре, может не заботиться о том, что где-то в модуле есть переменные с тем же именем. А программист, объявляющий переменные в своем модуле, может не заботиться о том, что переменные с тем же именем есть где-то в проекте.

Пример: Имеется проект из двух форм. Окно кода формы 1:

```
Public a As Integer
Private Sub Command1_Click()
  Form2.Show
  a = 1
End Sub
```

Окно кода формы 2:

```
Public a As Integer
Private b As Integer

Private Sub Command1_Click()
  a = 2
  Debug.Print Form1.a      'Печатается 1
  Debug.Print a            'Печатается 2
  b = 3
  Debug.Print b            'Печатается 3
End Sub

Private Sub Command2_Click()
  Dim a As Integer
  Dim b As Integer
  a = 4
  Debug.Print a            'Печатается 4
  b = 5
  Debug.Print b            'Печатается 5
End Sub
```

Все, что я сказал о зонах видимости и о затенении переменных, в той или иной степени относится и к константам, и к свойствам, и к процедурам, и к другим элементам Visual Basic. Я не буду вдаваться в подробности, но дам один совет, который позволит вам избежать путаницы и конфликтов: *давайте разным элементам разные имена*. Отсюда вытекает желательность давать элементам разной природы разные префиксы, о чем в следующем разделе.

Теперь несколько слов о пользе модуля кода. Удобство его в том, что он не является "хозяином". Глобальные переменные, константы, типы, процедуры, определенные в нем, являются всеобщим достоянием и могут употребляться в любых других модулях, не требуя указания "хозяина". Модуль кода и используется как вместилище таких глобальных элементов, нужных в других модулях.

19.6. Префиксы имен

Пусть в вашем проекте вы придумали кнопке имя Сумма, текстовому полю имя Сумма и переменной тоже имя Сумма. Так делать, конечно, нельзя: все имена перепутаются. Но и отказываться от одинаковых имен тоже никак не хочется, потому что они на ваш взгляд наиболее удачно передают смысл своих обладателей. Для того, чтобы не оказаться в такой ситуации, профессиональные программисты используют **префиксы** - приставки к именам. У всех элементов одной природы префикс одинаков, у элементов разной природы он разный. В нашем случае кнопка будет иметь имя cmdСумма, текстовое поле - txtСумма, переменная типа Double - dblСумма.

Вот какие префиксы рекомендует Microsoft для элементов управления:

Check box	chk
Combo box, drop-down list box	cbo
Command button	cmd
Common dialog	dlg
Directory list box	dir
Drive list box	drv
File list box	fil
Form	frm
Frame	fra
Horizontal scroll bar	hsb
Image	img
ImageList	ils
Label	lbl
Line	lin

List box	lst
Menu	mnu
Picture box	pic
ProgressBar	prg
RichTextBox	rtf
Shape	shp
Slider	sld
Text box	txt
Timer	tmr
Toolbar	tlb
TreeView	tre
UpDown	upd
Vertical scroll bar	vsb

А такие - для переменных:

Boolean	bln
Byte	byt
Collection object	col
Currency	cur
Date (Time)	dtm
Double	dbl
Error	err
Integer	int

Long	lng
Object	obj
Single	sng
String	str
User-defined type	udt
Variant	vnt

Несмотря на то, что от добавления префиксов имена становятся корявыми, мы имеем два серьезных выигрыша: имена не перепутаются и по имени элемента сразу же можно сказать, какого он типа. Если же вы хотите стать профессиональным программистом, то тем более от префиксов вам никуда не уйти, так как глаз должен привыкать читать чужие программы, да и свои программы вы захотите показать профессионалу, который к префиксам привык.

А теперь о чувстве меры. Некоторые имена, будь то имя переменной, формы или другого элемента, в программе используются очень часто. Если им давать "по науке" длинные имена с префиксами, то текст вашей программы будет очень громоздким. Поэтому никто на вас не обидится, если вы в цикле вместо

```
intПеременная_цикла = intПеременная_цикла + 1
y(intПеременная_цикла) = 2 * a(intПеременная_цикла)
```

напишете просто

```
i = i + 1
y(i) = 2 * a(i)
```

19.7. К чему все эти сложности?

Давайте еще раз зададимся вопросом: зачем было выдумывать все эти глобальные и локальные переменные, зоны видимости, эффект затенения и прочее? Не проще ли было все переменные и другие элементы сделать равноправными, тогда, глядишь, и никаких зон, эффектов и прочих сложностей не понадобилось бы? Давайте попробуем так и сделать. Для простоты будем рассуждать только о переменных.

Итак, сделаем все переменные равноправными. И пусть безо всяких сложностей все переменные будут видны во всем проекте, другими словами, по нашей терминологии, пусть все они будут глобальными. Тогда большие проекты, создаваемые командой программистов, будет крайне тяжело отлаживать, так как придется следить за тем, чтобы нигде не встретилось одинаковых имен, что и очень неудобно, кстати. Кроме этого, при отладке любой процедуры придется следить за тем, как бы случайно не испортить значение той или иной глобальной переменной, от которой зависит работа других процедур. Во многом потеряют смысл процедуры с параметрами, так как вся их прелесть в том как раз и состоит, чтобы "развязать" создателя процедуры с проектом, чтобы сделать из процедуры "вещь в себе", надежно защищенную от "помех" со стороны проекта.

Итак, сделать все переменные глобальными нельзя. Тогда, может быть, сделать их всех локальными? Тоже не получится, так как это крайне затруднит обмен информацией между процедурами и модулями. А без этого обмена проект рассыпется на отдельные процедуры, как карточный домик. Вот и получается, что переменные нужны разные - и глобальные и локальные, и никуда от этого не денешься. Надо разрешать и одноименные переменные. А раз так, то нужны и зоны видимости и эффект затенения.

Встает вопрос: какую зону видимости нам выбрать для каждой конкретной переменной? Объявлять ли ее глобальной, локальной в модуле или локальной в процедуре? Здесь совет один - любую переменную делайте как можно более локальной, пусть ее зона видимости будет как можно меньше. Если ее значение нужно только в одной процедуре и больше нигде, делайте ее локальной в этой процедуре. Если ее значение нужно в нескольких процедурах одного модуля, а в других модулях нет, то делайте ее локальной в этом модуле. И только если значение переменной нужно в нескольких модулях, делайте ее глобальной. Такой подход обеспечит вашему проекту максимальную надежность и удобство в отладке.

Второй совет - вместо процедур без параметров используйте процедуры с параметрами. Покажу на простом примере, что я имею в виду.

Задача: Переменная равна 3. Нужно увеличить ее на 2 и результат напечатать.

Дадим переменной имя intA. Вот простейшая программа для решения задачи:

```
Private intA As Integer
Private Sub Command1_Click()
   intA = 3
   intA = intA + 2
   Debug.Print intA
End Sub
```

Однако простейшая программа нас не устраивает. Нам нужно показать всю прелесть параметров. Для этого мы будем действовать так же, как действовал Том Сойер в книге "Приключения Гекльберри Финна", когда ему нужно было освободить негра Джима из ветхого сарайчика, где того держали взаперти. Вместо того, чтобы просто отпереть дверь имевшимся у него ключом, Том придумал массу ненужных вещей вроде веревочной лестницы, отпиленной ножки кровати и т.п. И все для того, чтобы побег из заточения был обставлен "как положено".

Вот и мы сейчас усложним проект, чтобы все было "как положено". Прежде всего разобьем его на две процедуры П1 и П2, первая из которых увеличивает переменную на 2, а вторая распечатывает результат.

```
Private intA As Integer

Private Sub Command1_Click()
   intA = 3
   П1
   П2
End Sub

Private Sub П1()
   intA = intA + 2
End Sub

Private Sub П2()
   Debug.Print intA
End Sub
```

Программа работает. Но этого мало. Будем усложнять дальше. Добавим к нашим процедурам параметры:

```
Private intA As Integer

Private Sub Command1_Click()
   intA = 3
   П1 intA
   П2 intA
End Sub

Private Sub П1(intAA As Integer)
   intAA = intAA + 2
End Sub

Private Sub П2(intAAA As Integer)
   Debug.Print intAAA
End Sub
```

В чем смысл этого ужасного усложнения, этого превращения мухи в слона? Конечно, для нашего простейшего примера нет никакого смысла усложнять программу. Но когда дело дойдет до сложных проектов и процедур, здесь смысл прямой.

Зачем делить проект на процедуры, я уже рассказывал. А вот зачем понадобились параметры.

Если ваша процедура сложная и делает что-нибудь полезное, то вы вполне можете захотеть использовать ее и в других проектах. Но в другом проекте переменные скорее всего имеют другие имена, например, вместо intA там используется intB. В этом случае вам придется переписывать текст процедуры (в нашем конкретном случае везде заменить intA на intB). А переписывать не хочется. В общем, процедура теряет свою универсальность. Чтобы и переписывать не надо было и универсальность не потерять, применяются параметры. Обратите внимание, что нашим процедурам с параметрами абсолютно все равно, что переменная имеет имя intA. Нигде внутри процедуры оно не встречается, поэтому процедура уверенно делает свое дело, каково бы имя ни было. Да и программисту как-то понятней, чем занимается процедура, если он видит в ее заголовке список параметров, да еще и с подробными комментариями. Это лучше, чем глядеть в тексте процедуры на переменную intA и вспоминать, что это, черт возьми, за переменная и в чем ее смысл. В общем, совет такой - с глобальными переменными и локальными переменными модуля работайте по возможности через параметры. Что значит "по возможности"? А то, что неко-

торые такие переменные буквально пронизывают все процедуры проекта и организовывать для них в каждой процедуре параметры - та овчинка, которая не стоит выделки. Обычно таких переменных бывает немного и их легко просто запомнить.

Глава 20. Объекты пользователя

Ну вот и пришло время создания своих собственных объектов. Поговорим о них.

Введение процедур в программирование резко повысило надежность создаваемых больших программ и их обозримость, сделало сам процесс программирования более удобным. Следующим шагом на пути совершенствования программирования стало введение объектов. **Объект** – это синтез данных (в частности переменных величин) и процедур, которые эти данные обрабатывают.

Объекты в программировании напоминают объекты реального мира. Например, чтобы описать стенные часы, мы должны описать как совокупность их составных частей (радиусы шестеренок, длину маятника и прочее – в общем, «данные») так и совокупность процессов взаимодействия этих частей (как качается маятник, как шестеренка цепляет шестеренку и так далее – в общем «процедуры и функции»).

Любой элемент управления Visual Basic - объект. Свойства элемента управления - это данные, методы - процедуры.

Почти все приложения Windows и сама операционная система Windows запрограммированы на основе объектного подхода. Типичный пример объекта в Windows – окно. Чтобы заставить окно на экране вести себя, как положено, программисту нужно описать его размер, цвет, толщину рамки и прочее (данные) плюс процессы перетаскивания его по экрану, изменения размера и прочее (процедуры и функции).

20.1. Инкапсуляция - "Объект в футляре"

Нам будет легче проникнуть в суть объектов в программировании, если мы предварительно поговорим об объектах реального мира, так как объекты в программировании очень их напоминают. Возьмем, например, игрушечный автомобиль, управляемый на расстоянии с пульта. Чтобы в нем разобраться, достаточно знать две вещи:

- Как он устроен (назовем это *данные*). Если объект - животное, то этим занимается анатомия.
- Как он работает (назовем это *действия*). Если объект - животное, то этим занимается физиология.

У игрушечного автомобиля данных множество. Например:

- Номер автомобиля
- Громкость звукового сигнала
- Цвет кузова
- Скорость движения в данный момент
- Высота кресел
- Величина электрического тока в двигателе в данный момент
- Толщина гайки в таком-то месте корпуса

И так далее и тому подобное.

Действий тоже достаточно. Например:

- Поворот по команде с пульта управления
- Торможение по команде с пульта управления
- Подпрыгивание автомобиля на маленьком камушке
- Изменение скорости вращения оси двигателя при изменении в нем электрического тока
- Изменение света фар при изменении в них электрического тока
- Возникновение жужжания двигателя при трении шестеренок друг об друга

И так далее и тому подобное.

Как видите, данные и действия бывают крупные и мелкие, важные и менее важные. Но нас интересует другое подразделение, а именно:

- на те данные и действия, что видны снаружи автомобиля или доступны для изменения и управления со стороны внешних объектов, например, прохожих или мальчика, держащего в руках пульт управления (такие данные и действия близки понятию *глобальных* переменных и процедур)
- и на те данные и действия, что не видны или не доступны (а эти близки понятию *локальных*)

Проведем подразделение построже и поподробнее. Данные будем делить на те, что видны снаружи (это первые 4), и те, что не видны (последние 3). Данные, видимые снаружи, назовем **свойствами** объекта.

Действия будем делить на те, которыми можно управлять снаружи (первые 2 действия), и те, что вызываются внутренней механикой автомобиля (остальные). Действия, вызываемые снаружи, назовем **методами** объекта.

Теперь будем подразделять свойства. Мы их разделим на две категории:

- Те, что можно произвольно менять снаружи (вообразим, что любой наблюдающий за нашим автомобилем может, вооружившись ведром краски, в любой момент произвольно менять цвет автомобиля. В реальной жизни так не бывает, но в программировании это сделать очень легко). Назовем их свойствами *для чтения-записи*.
- Те, что снаружи менять нельзя (например, громкость звукового сигнала). Назовем их свойствами *только для чтения*.

Конструктор нашего игрушечного автомобиля при его создании стремится к тому, чтобы автомобиль был надежен, защи-

щен и просто управлялся. Для этого он должен придерживаться двух принципов:

- Количество методов должно быть минимально необходимым. Старт, торможение, поворот налево и направо. Этого достаточно. Если разрешить управлять с пульта жужжанием двигателя и тонким процессом изменения скорости вращения колес при изменении тока в двигателе, то недолго и сжечь двигатель, нажав на пульте не ту комбинацию кнопок.
- Количество свойств для чтения-записи должно быть минимальным. Действительно, вообразим крайний случай: все данные мы сделали свойствами, да еще и для чтения-записи. Тогда любой, кому не лень, сможет в любой момент менять, например, величину тока в двигателе, что немедленно приведет к катастрофе. Или, например, вообразите, что вы едете в автомобиле, а любой прохожий может протянуть руку и покрутить баранку вашего автомобиля.

А теперь о том, нужно ли все данные делать свойствами, то есть делать их видимыми отовсюду. Дело вкуса. Покуда вы не делаете их свойствами для чтения-записи, это не влияет на надежность объекта. Сами решайте, нужно ли всем желающим видеть высоту сидений и величину тока. Конечно, когда речь идет о реальном игрушечном автомобиле, все получается как-то само собой: что снаружи - видно, что изнутри - не видно и ничего тут не поделаешь. Но у программиста полная свобода любое данное сделать или не сделать свойством, в том числе и для чтения-записи, и любое действие сделать или не сделать методом.

При создании объектов программисты стараются придерживаться принципа **инкапсуляции**, который заключается в следующем:

Данные и действия объекта представляют собой единое целое, образующее всю механику объекта, и хранятся они в одной "упаковке". Чуть позже вы увидите, что эта упаковка - отдельный модуль. Они должны быть как можно меньше видимы снаружи. Хорошо инкапсулированный объект представляет собой некий "черный ящик", эдакого "человека в футляре". Вся работа идет внутри. Внутренние данные меняются при помощи внутренних действий. Никто снаружи не может вмешаться в эту внутреннюю работу. Наружу показывается лишь тот минимум (*интерфейс*), который необходим для связи с окружающим миром.

Влиять на работу объекта можно только тремя способами:

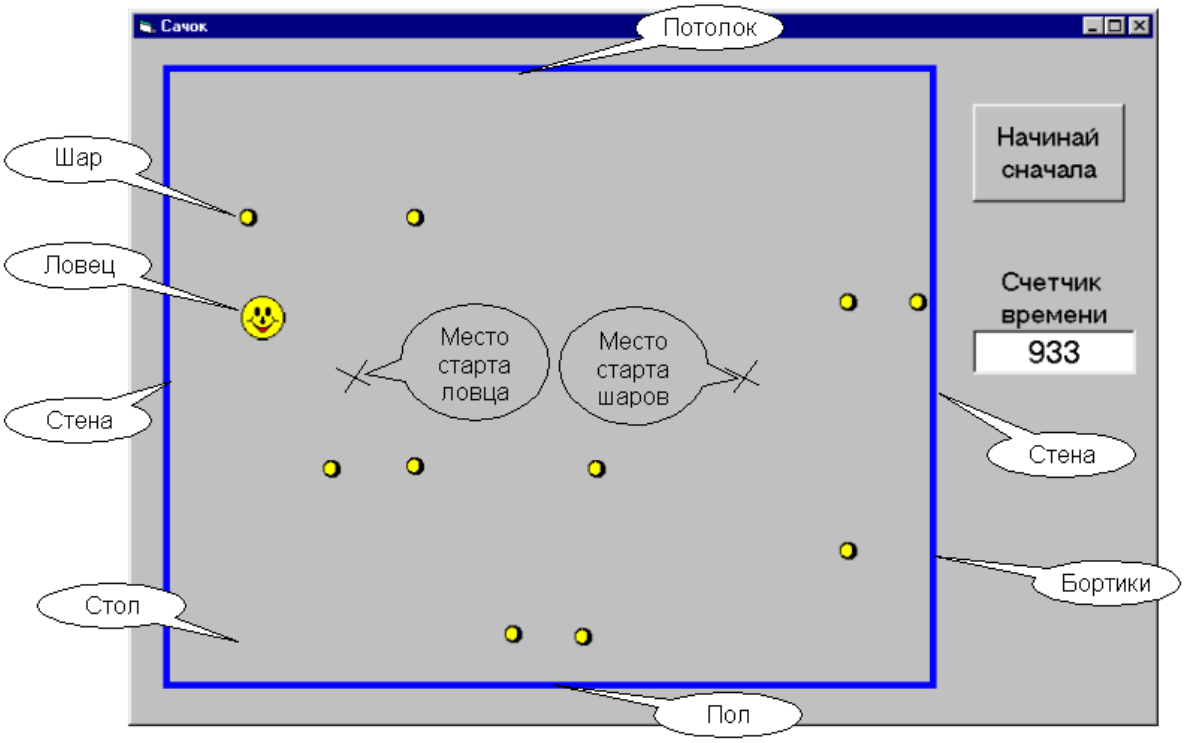
- Методами
- Изменяя значения свойств для чтения-записи
- Изменяя свойства окружающего мира, например, положив на пути автомобиля камешек.

Инкапсуляция - то, что объединяет объекты в программировании с объектами реального мира. Возьмите летящий самолет. Вы не можете снаружи ни видеть работу его двигателя, ни как-то повлиять на нее. Вы вообще никак не можете повлиять на самолет ни в чем.

Все вышесказанное является введением в идеологию объектного программирования. Как мне кажется, это введение поможет вам легче разобраться в механике объектов, создаваемых вами на компьютере. И теперь пришла пора такие объекты создавать. Лучше всего это делать на примере решения реальной задачи. Вот она.

20.2. Игра "Сачок". Постановка задачи

Давайте создадим такую игру:



Как только проект запущен, из места старта на столе разлетаются во все стороны со случайными скоростями и в случайных направлениях 10 шариков. Они ведут себя как биллиардные шары на столе. Ударившись о бортик, любой шарик отскакивает по закону отражения. Примечание: Для удобства программирования я назвал по-разному 4 бортика: пол, потолок, стены.

Трения нет - и шарики могут бесконечно кататься по столу. Для простоты я не стал программировать соударение шариков между собой, хотя это вполне можно было сделать. Поэтому шарики при столкновении просто пролетают друг сквозь друга. На поле присутствует Ловец (которого раньше я хотел назвать "Сачок", причем в обоих смыслах). Ловцом управляет с клавиатуры игрок. Ловец может двигаться с постоянной скоростью в 4 направлениях: вверх, вниз, влево, вправо, подчиняясь соответствующим клавишам клавиатуры, а также стоять (клавиша Ctrl). Каждый раз, когда ловец соприкасается с шариком, шарик исчезает (он пойман). Задача игрока - побыстрее поймать все 10 шариков. Счетчик времени (импульсов таймера) замирает в момент поимки последнего шарика, чтобы вы могли запомнить результат. При нажатии на кнопку "Начинай сначала" игра начинается сначала. Вот и все.

Что здесь будет объектом? Несмотря на то, что у нас нет опыта, интуиция подскажет нам. Конечно же, шары и ловец.

20.3. Таймер и общая механика работы проекта

Теперь давайте продумаем общую механику работы проекта. Она, в общем, напоминает механику работы проектов "Будильник" и "Гонки". Если вы подзабыли эту механику, обязательно перечтите 11.3 и 12.3. Там я писал, что важную роль в моделировании на компьютере согласованной работы разных механизмов играет таймер, который своими импульсами синхронизирует работу этих механизмов (объектов). Конкретно, очередной импульс таймера "будит" по очереди все эти объекты, каждый из которых выполняет весь цикл своей жизнедеятельности, то есть все свои действия, которые положено выполнять при приходе импульса, после чего "засыпает" до следующего импульса. На следующем импульсе таймера все повторяется снова. Поскольку импульсы следуют друг за другом с большой частотой, создается впечатление непрерывной и одновременной работы объектов (несмотря на то, что они все "бодрствуют" по очереди).

Посмотрим, как применить эту идеологию к нашему проекту. В игре 11 объектов: ловец и 10 шаров. Имеется 1 таймер. Он непрерывно на протяжении всей игры посылает импульсы с частотой 18 импульсов в секунду. Вот игра началась. От таймера пошел 1-й импульс. Оживает 1-й шар, "осознает себя, как объект", заглядывает к себе внутрь, запускает всю свою механику и выполняет все, что ему положено в данной обстановке. В начале игры это означает, что он сдвигается на некоторый шаг от точки старта. Выполнив все свои дела, он "засыпает" и "просыпается" 2-й шар, который тоже делает все, что ему положено (сдвигается), и "засыпает" и т.д. Когда засыпает 10-й шар, просыпается ловец и тоже выполняет все, что диктует ему его механизм, в частности смотрит, не нажата ли клавиша на клавиатуре, и в зависимости от этого сдвигается или остается на месте. После чего засыпает. Все. Все объекты сделали свои дела и спят до прихода 2-го импульса от таймера. Вот пришел 2-й импульс. Просыпается 1-й шар, все делает (сдвигается еще на шаг в том же направлении) и засыпает, просыпается 2-й шар и так далее.

Обратите внимание, что в каждый момент времени не спит, работает и движется только один объект. Остальные неподвижны и спят. Однако, поскольку импульсы от таймера следуют друг за другом с большой частотой, создается впечатление непрерывной и одновременной работы объектов: шары летают, ловец ловит.

Обратите внимание, что таймер ничего не говорит объектам о том, *что* они должны делать, они знают это сами, так как это запрограммировано в их процедурах. Таймер всего лишь будит их через равные промежутки времени.

Кроме шаров и ловца импульсам таймера у нас в проекте подчиняется счетчик времени на форме.

20.4. Этап проектирования

Хватит теории. Садимся за компьютер. Создайте новый проект. Для краткости дайте форме имя f. Разместите на форме элемент управления Shape - это будут бортики нашего стола. Размер стола не имеет значения, потому что мы так запрограммируем шары и ловца, что они будут чувствовать бортики. Разместите на форме кнопку cmd_Начинай_сначала и текстовое поле txtСчетчик_времени

Украсьте как-нибудь проект. Только не надо ничем заливать прямоугольник стола и не надо помещать на форму фотографию - шары будут тормозить.

Создайте модуль кода. В нем нам будет удобно объявлять глобальные элементы нашего проекта.

20.5. Порядок создания объектов

Создание объектов пользователя в Visual Basic проходит в два этапа:

1. Создается собственный **пользовательский класс объектов**. Аналогия: пусть ваш объект - алюминиевая тарелка. Тогда класс объектов Тарелка есть штамп, которым мы можем наштамповать сколько угодно объектов-тарелок. Вся работа по созданию механизма будущего объекта проходит именно на этом этапе.
2. По готовому классу объектов создается ("штампуется") сам **объект** или, как его еще называют, **экземпляр класса**. Для этого достаточно двух строчек кода. Экземпляров с разными именами можно создать сколько угодно. Можно создать и массив объектов (именно это мы впоследствии и сделаем, создав массив из 10 объектов-шаров).

Итак, согласно вышесказанному мы создадим два класса объектов с именами clsЛовец и clsШар. Затем по классу clsЛовец создадим объект Ловец, а по классу clsШар создадим массив объектов Шар(1 To 10).

20.6. Создаем ловца

Разобьем создание проекта тоже на две ступени. На первой ступени мы создадим ловца, а шаров не будет. Проект будет функционировать нормально, только без шаров. На второй ступени проект будет готов полностью.

Приступим к созданию класса clsЛовец. Каждый пользовательский класс объектов создается в собственном **модуле класса**. Ведь форма - это модуль, модуль кода - модуль, и класс тоже требует своего модуля, который и будет той "упаковкой", в которой безопасно будут работать его данные и процедуры.

Значит, создаем модуль класса. Делается это просто: **Project → Add Class Module→ New →Class Module→ Open**. И вот перед вами чистое окно кода модуля класса. Впечатление совсем такое же, как при создании модуля кода. В этом окне вы будете объявлять переменные объекта и писать процедуры и функции, из которых и состоит вся механика объекта. Никаких элементов управления и формы у модуля класса нет.

В окне свойств модуля класса дайте имя своему классу - clsЛовец. Сохранимся. Мы видим, что Visual Basic сохраняет модуль класса в отдельный файл. Придадим файлу имя Ловец.cls. Все готово. Можно бы писать код. Но это делать еще рано. Сначала нужно очень точно продумать поведение нашего ловца, все, что он должен уметь делать. Выпишем все его умения:

A. По приходе импульса от таймера он должен:
- Сдвинуться на некоторый шаг вверх, вниз, влево, вправо, подчиняясь соответствующим клавишам клавиатуры, или стоять (клавиша Ctrl).
- Проверить, не наткнулся ли он на бортик, и если да, то остановиться.

B. При нажатии кнопки "Начинай сначала" возвращаться в исходное положение и там останавливаться.

Все. Вы спросите, а как же умение ловить шары, ради которого ловец и создан? Я отвечу - В нашем случае проще запрограммировать "исчезалки", чем "ловилки". Поясню. Наткнувшись на шар, ловец не будет предпринимать никаких действий по его "поимке". Наоборот, шар, наткнувшись на ловца, потрудится исчезнуть. Со стороны - никакой разницы.

Запрограммируем все действия, перечисленные в пункте *A*, в одной процедуре, и назовем ее Действие. Запрограммируем все действия, перечисленные в пункте *B*, в другой процедуре, и назовем ее Начальная_установка.

Объект пользователя - мозг без тела

Я уже говорил о разнице между модулями формы и кода. Модуль формы снабжен элементами, которые мы собственными глазами видим на экране: это сама прямоугольная форма и элементы управления, расположенные на ней. У модуля кода ничего визуального нет, это просто набор переменных, констант, типов, процедур и функций. Модуль класса в этом повторяет модуль кода. Он может своими процедурами обеспечивать сколь угодно сложное поведение и движение объекта, но все это где-то там, глубоко в памяти компьютера, а самого объекта и движения вы никогда на экране не увидите, так как в модуле класса, как и в модуле кода нет встроенных средств визуализации. Что же делать? Приходится модулю класса пользоваться чужими средствами, конкретно - элементами управления формы.

Поместите на форму два элемента управления Image, одному дайте имя imgЛовец, другому - imgШар (в дальнейшем из imgШар мы породим массив элементов управления imgШар(1 To 10)). Они-то и будут теми актерами, которым предназначено изображать живую жизнь наших бестелесных объектов. Это просто марионетки. Умные объекты будут дергать их за ниточки, и они будут надлежащим образом двигаться по экрану. Всем окружающим будет казаться, что это движутся сами объекты. Придайте им соответствующие картинки.

У модуля класса нет ни одного элемента управления, значит и таймера тоже нет. Таймером нашего проекта будет таймер, принадлежащий форме. Поместите его на форму. Задайте ему интервал = 10. Щелкните по нему дважды - в окне кода формы появится заготовка процедуры. Напомню, что эта процедура выполняется на каждом импульсе таймера. Значит это и будет главная процедура нашего проекта, задающая ритм всем объектам и элементам управления.

Перечислю действия, которые должна выполнять эта процедура:
- Разбудить по очереди Шар(1), Шар(2), … Шар(10) и заставить каждого выполнить свою невидимую работу, в частности вычислить свое положение (координаты) на форме.
- Переместить элементы управления imgШар(1), imgШар(2), … imgШар(10) в положение, вычисленное их объектами Шар(1), Шар(2), … Шар(10).
- Разбудить ловца и заставить его выполнить свою процедуру Действие, в частности вычислить свое положение (координаты) на форме.
- Переместить элемент управления imgЛовец в положение, вычисленное объектом Ловец.
- Увеличить на 1 счетчик времени (импульсов таймера) на форме.

Перечислю действия, которые должны выполняться при нажатии на кнопку "Начинай сначала":
- Установить в 0 счетчик времени
- Установить в 0 переменную-счетчик пойманных шаров. Он нам понадобится, чтобы определить момент окончания игры - как только счетчик станет равен 10.
- Заставить ловца выполнить свою процедуру Начальная_установка, то есть прыгнуть в точку старта ловца и там остаться.
- Заставить каждый из шаров прыгнуть в точку старта шаров и подготовиться к старту.

Как создать объект по его классу

Сначала объект Ловец нужно **объявить** как экземпляр класса clsЛовец. Точно так же, как мы объявляем переменную вели-

чину а, как хранительницу экземпляра значения типа Integer. То есть, мы объявляем, из какого "штампа" будет штамповаться наша тарелка:

Public Ловец **As** clsЛовец

Почему Public? Потому что наш ловец должен быть виден и из модуля формы (хотя бы для того, чтобы знать его координаты), и из модуля шара (который тоже должен знать его координаты, чтобы вовремя исчезнуть при столкновении).

Строку объявления удобнее написать в модуле кода, чтобы не надо было указывать хозяина объекта.

А теперь объект Ловец нужно создать ("отштамповать" тарелку):

Set Ловец **=** **New** clsЛовец

Слово New означает, что создается новый представитель класса объектов clsЛовец. Начиная с этого момента объектом можно пользоваться. Объекты можно создавать в любой момент работы проекта, поэтому указанную строку можно помещать в любое место кода. Поскольку наш ловец понадобится нам с самого начала, поместим эту строку в процедуру Form_Load нашей формы.

Первая ступень проекта

Итак, вот как выглядит наш проект на первой ступени. На второй ступени к модулю кода и окну кода формы просто добавятся без малейшего изменения строки, касающиеся шаров. А модуль clsЛовец я привожу, естественно, целиком, в окончательном варианте.

Модуль кода

```
Public Const Размер_ловца = 500
Public Ловец As clsЛовец                    'Объявляем объект Ловец класса clsЛовец
```

Модуль формы

```
Private Sub Form_Load()                     'Эта процедура выполняется один раз при запуске проекта
   imgЛовец.Height = Размер_ловца
   imgЛовец.Width = Размер_ловца

   Set Ловец = New clsЛовец                 'Порождаем объект Ловец
   Начальная_установка
   KeyPreview = True                        'Чтобы форма реагировала на клавиатуру
   txtСчетчик_времени.Locked = True         'Чтобы в процессе игры нельзя было вручную менять показания счетчика
End Sub

Private Sub cmd_Начинай_сначала_Click() 'Что происходит при нажатии кнопки НАЧИНАЙ СНАЧАЛА
   Начальная_установка
   txtСчетчик_времени.SetFocus              'Чтобы фокус ушел с кнопки, иначе первое нажатие на стрелки клавиатуры не вызывает движения ловца
End Sub

Private Sub Начальная_установка()
   txtСчетчик_времени.Text = 0              'Обнуляем счетчик времени
   Ловец.Начальная_установка                'Ловец встает в исходную позицию и настраивается на новую игру
End Sub

Private Sub Timer1_Timer()                  'Главная_процедура игры, выполняется один раз на каждом импульсе иаймера
   Ловец.Действие                           'Сначала объект Ловец вычисляет свои координаты,
   imgЛовец.Left = Ловец.x                  'Затем изображение Ловца сдвигается на вычисленные координаты
   imgЛовец.Top = Ловец.y
   txtСчетчик_времени.Text = txtСчетчик_времени.Text + 1    'Действует счетчик времени на форме, увеличивая свои показания на 1
End Sub
                    'Обработка события - нажатия клавиши на клавиатуре для управления ловцом:
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
   Select Case KeyCode
      Case vbKeyUp:          Ловец.Руль = вверх
      Case vbKeyLeft:        Ловец.Руль = влево
      Case vbKeyDown:        Ловец.Руль = вниз
      Case vbKeyRight:       Ловец.Руль = вправо
      Case vbKeyControl:     Ловец.Руль = стоп
   End Select
End Sub
```

Модуль clsЛовец

```
Public x As Long                            'Координаты ловца
Public y As Long
Private dx As Long                          'Шаг ловца по горизонтали и вертикали между двумя импульсами таймера
Private dy As Long

Public Enum типРуль                         'Направление движения ловца или состояние неподвижности
   вверх
   влево
   вниз
```

```
    вправо
    стоп
End Enum
Public Руль As типРуль

Private Sub Class_Initialize()                    'Процедура, выполняющаяся при рождении ловца
    dx = 100
    dy = 100
End Sub

Public Sub Начальная_установка()                  'Ловец встает в исходную позицию и останавливается
    Руль = стоп
        'Ставим ловца в исходную позицию:
    x = f.shpБортик.Left + f.shpБортик.Width * 1 / 4  'Он отстоит по горизонтали на четверть ширины стола от левого его края
    y = f.shpБортик.Top + f.shpБортик.Height / 2      'Он по вертикали расположен посредине стола
End Sub

Public Sub Действие()                             'Главная процедура ловца, выполняется один раз на каждом импульсе таймера
    Выбираем_куда_ехать_и_делаем_шаг              'Ловец слушается клавиатуру
    If Ловец_у_пола_или_потолка Or Ловец_у_стен Then Руль = стоп          'Чтобы ловец случайно не улетел за пределы стола
End Sub

Private Sub Выбираем_куда_ехать_и_делаем_шаг()
    Select Case Руль
        Case вверх:          y = y - dy
        Case вниз:           y = y + dy
        Case влево:          x = x - dx
        Case вправо:         x = x + dx
        Case стоп:           'Поскольку никуда идти не надо, постольку ничего не делаем
    End Select
End Sub

Private Function Ловец_у_пола_или_потолка() As Boolean
                                  'ЕСЛИ ловец находится у потолка ИЛИ у пола , ТО:
    If y < f.shpБортик.Top Or y + Размер_ловца > f.shpБортик.Top + f.shpБортик.Height Then
        Ловец_у_пола_или_потолка = True
    Else
        Ловец_у_пола_или_потолка = False
    End If
End Function

Private Function Ловец_у_стен() As Boolean
                                  'ЕСЛИ ловец находится  у левой стены ИЛИ у правой, ТО:
    If x < f.shpБортик.Left Or x + Размер_ловца > f.shpБортик.Left + f.shpБортик.Width Then
        Ловец_у_стен = True
    Else
        Ловец_у_стен = False
    End If
End Function
```

Запустите проект. Проверьте, правильно ли движется ловец. У бортика он должен сам останавливаться. Его можно передвигать и за пределами стола, но с трудом.

Обратите внимание, что разные модули используют одноименные процедуры. Например, Начальная_установка. Проблемы в этом нет, так как если процедура задана, как Private, то из других модулей она просто не видна, а если даже Public, то перед своим именем она будет требовать имени хозяина.

А теперь пояснения. Работу процедур я буду разбирать в хронологическом порядке, то есть в том, в котором они вызываются после запуска проекта. Об инкапсуляции поговорим попозже.

Первой при запуске проекта выполняется процедура Form_Load. Начинается она с того, что высота и ширина элемента управления imgЛовец (см. текст программы) становятся равными константе Размер_ловца. Затем порождается объект Ловец. При этом генерируется событие - инициализация объекта - и в соответствии с этим выполняется процедура Class_Initialize в модуле класса. Там я только присваиваю значения шагу ловца. После инициализации управление возвращается в процедуру Form_Load, а там запускается процедура Начальная_установка, которая выделена в отдельную процедуру, так как выполняется не только при загрузке формы, но и при щелчке по кнопке "Начинай сначала".

Выше я уже перечислил действия, которые нужно выполнить при начальной установке. Вы видите, что в текстовом поле на форме появляется 0, затем из процедуры Начальная_установка формы запускается процедура Начальная_установка ловца. Здесь Руль устанавливается в положение стоп. Затем вычисляются координаты точки старта ловца. Возможно, у вас вызовут трудность формулы, встречающиеся в коде. Не поленитесь разобраться в них. Например, f.shpБортик.Left означает горизонтальную координату левого края элемента управления shpБортик на форме f. Как видите, объект Ловец использует для своей работы информацию о внешнем мире.

После выполния процедуры Начальная_установка ловца управление возвращается в процедуру Начальная_установка формы f. Не найдя там больше ничего, Visual Basic возвращается в Form_Load на строку KeyPreview = True. После выполнения этой и

следующей строк Form_Load завершает работу и на игровой площадке наступает покой до прихода первого импульса от таймера.

Если вас затруднили все эти пассажи с возвратами управления, запустите пошаговый режим. Это прекрасный способ убедиться в правильности кода.

А теперь давайте остановимся и подумаем, что успело произойти. Если вы дошли до этого момента в пошаговом режиме, то обнаружили, что ловец, вопреки ожиданиям, не занял место на старте, несмотря на то, что координаты x и y.вычислены правильно. Это что - упущение проекта? Нет. Просто я не стал обременять процедуру Начальная_установка заботами о перемещении imgЛовец. Дело в том, что через 1/18 долю секунды все равно придет импульс от таймера и заработает процедура Timer1_Timer. Там-то и запрограммировано перемещение изображения. Никто и не заметит, что ловец встал на место не сразу.

И вот импульс грянул. Заработала процедура Timer1_Timer и первым делом запустила процедуру Ловец.Действие. Давайте пока не будем в нее заглядывать, вообразим, что там ничего существенного не произошло, и пойдем дальше. Следующие две строки как раз и перемещают изображение ловца в точку старта. Вместо 0 в счетчике на форме появляется 1. На этом процедура Timer1_Timer заканчивает свою работу. Все замирает до тех пор, пока через 1/18 долю секунды не придет следующий импульс от таймера.

Предположим, вы за это время еще не успели прикоснуться к клавиатуре. Вот импульс грянул. Заработала процедура Timer1_Timer и запустила процедуру Ловец.Действие. Рассмотрим, как она работает. Ее тело состоит из двух строк, вызывающих процедуры и функции с интуитивно ясными именами. Первой выполняется процедура Выбираем_куда_ехать_и_делаем_шаг. Руль у нас после начальной установки находится в положении стоп и ничто его оттуда не вывело, значит оператор Select Case не меняет ни x ни y.

Хорошо. Дальше выполняется следующая строка. Вызывается функция Ловец_у_пола_или_потолка. Вы можете сами убедиться, что поскольку ловец пока далеко от бортиков, то эта функция принимает значение False. Аналогично, значение False принимает и функция Ловец_у_стен. На этом вторая строка процедуры Ловец.Действие завершается, а с ней и вся процедура. Visual Basic возвращается в процедуру Timer1_Timer. Поскольку ни x ни y не менялись, то следующие две строки процедуры оставляют изображение ловца на месте.

Ждем следующего импульса. Пусть до момента его прихода мы успели нажать на клавиатуре стрелку вправо, желая направить ловца направо. Немедленно сработала процедура Form_KeyDown в модуле формы и руль встал в положение вправо. Вот пришел импульс. Опять процедура Timer1_Timer направляет нас в процедуру Ловец.Действие, та - в процедуру Выбираем_куда_ехать_и_делаем_шаг. Поскольку руль повернут направо, вычисляется новое значение x, которое на dx больше предыдущего. Возвратившись наконец в процедуру Timer1_Timer, Visual Basic согласно новому значению x смещает imgЛовец чуть вправо.

Ждем следующего импульса и так далее. Вот и вся механика ловца.

Теперь поговорим о глобальном и локальном. В проекте я старался максимально придерживаться принципа инкапсуляции, все что можно я делал Private. Поглядите повнимательнее в код ловца. Вы видите 4 процедуры и 1 функцию. Именно они и обеспечивают эту механику. Никакая процедура и функция снаружи модуля в этой механике не участвуют. Ловец самодостаточен. Снаружи осуществляется только запуск двух методов объекта - Начальная_установка и Действие. Именно поэтому они сделаны Public, иначе их снаружи и запустить было бы нельзя. Но вся механика этих методов, опять же, находится внутри ловца, так что никакого нарушения суверенитета нет. Остальные процедуры и функции сделаны Private, они снаружи и не видны и недоступны.

Всю мыслительную работу ловец выполняет сам. Было бы логично, если бы он сам и передвигал imgЛовец, то есть, если бы строки

```
imgЛовец.Left = Ловец.x
imgЛовец.Top = Ловец.y
```

находились бы не в модуле формы, а в модуле класса. Здесь это можно было сделать просто, но в том, что касается шаров, пришлось бы пойти на некоторое усложнение программы, и я на это не пошел ни там, ни там.

Теперь о переменных. Две переменные, dx и dy, объявлены Private, поэтому они не видны снаружи. И не надо, естественно. А вот переменные x и y объявлены Public, от этого они стали свойствами. Принцип инкапсуляции плачет. Зачем нужно было это делать? Для чего они нужны снаружи? Только для смещения imgЛовец. Если бы объект Ловец смещал свое изображение сам, то и не надо было бы делать x и y свойствами. Ну ладно. Сделал так сделал. А уж если так, то надо бы сделать их, по крайней мере, только для чтения. Я займусь этим в 20.8.

Вот и все о первой ступени проекта.

20.7. Создаем шар. Завершаем проект

Модуль класса clsЛовец остается прежним. Приведу в окончательном виде все остальное, а именно: модуль класса clsШар, модуль кода и модуль (окно кода) формы.

Модуль кода

```
Public Const Число_шаров = 10
Public Const Размер_шара = 200
Public Const Размер_ловца = 500
Public Const Дальность = 200                'Это расстояние, на котором ловец достает шар
Public Ловец As clsЛовец                    'Объявляем объект Ловец класса clsЛовец
Public Шар(1 To Число_шаров) As clsШар      'Объявляем массив объектов Шар класса clsШар
```

Модуль формы

```
Public intЧисло_пойманных_шаров As Integer
```

```
Private Sub Form_Load()                    'Эта процедура выполняется один раз при запуске проекта
   Dim i As Integer
   Randomize                               'Шары должны разлетаться со случайной скоростью и в случайном направлении
                'Настраиваем размеры изображений шара и ловца:
   imgШар(1).Height = Размер_шара
   imgШар(1).Width = Размер_шара
   imgЛовец.Height = Размер_ловца
   imgЛовец.Width = Размер_ловца
                'Порождаем массив изображений шара:
   For i = 2 To Число_шаров
      Load imgШар(i)
      imgШар(i).Visible = True
   Next i
                'Порождаем объект Ловец и массив объектов-шаров:
   Set Ловец = New clsЛовец
   For i = 1 To Число_шаров
      Set Шар(i) = New clsШар
   Next i

   Начальная_установка
   KeyPreview = True                       'Чтобы форма реагировала на клавиатуру
   txtСчетчик_времени.Locked = True        'Чтобы в процессе игры нельзя было вручную менять показания счетчика
End Sub

Private Sub cmd_Начинай_сначала_Click()    'Что происходит при нажатии кнопки НАЧИНАЙ СНАЧАЛА
   Начальная_установка
   txtСчетчик_времени.SetFocus             'Чтобы фокус ушел с кнопки, иначе первое нажатие на стрелки клавиатуры не вызывает движения ловца
End Sub

Private Sub Начальная_установка()          'Все объекты встают в исходную позицию и настраиваются на новую игру
   Dim i As Integer
   txtСчетчик_времени.Text = 0             'Обнуляем счетчик времени
   intЧисло_пойманных_шаров = 0            'Обнуляем число пойманных шаров

   Ловец.Начальная_установка               'Ловец встает в исходную позицию и настраивается на новую игру
   For i = 1 To Число_шаров                'Все шары встают в исходную позицию и настраиваются на новую игру
      Шар(i).Начальная_установка
   Next i
End Sub

Private Sub Timer1_Timer()                 'Главная_процедура игры, выполняется один раз на каждом импульсе таймера
   Dim i As Integer
                'Действуют все шары:
   For i = 1 To Число_шаров
      Шар(i).Действие                      'Сначала объект Шар вычисляет свои координаты,
      imgШар(i).Left = Шар(i).x            'Затем изображение шара сдвигается на вычисленные координаты
      imgШар(i).Top = Шар(i).y
   Next i
                'Действует ловец:
   Ловец.Действие                          'Сначала объект Ловец вычисляет свои координаты,
   imgЛовец.Left = Ловец.x                 'Затем изображение Ловца сдвигается на вычисленные координаты
   imgЛовец.Top = Ловец.y
                'Действует счетчик времени на форме, увеличивая свои показания на 1:
   If intЧисло_пойманных_шаров <> Число_шаров Then txtСчетчик_времени.Text = txtСчетчик_времени.Text + 1
End Sub
                'Обработка события - нажатия клавиши на клавиатуре для управления ловцом:
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
   Select Case KeyCode
      Case vbKeyUp        Ловец.Руль = вверх
      Case vbKeyLeft      Ловец.Руль = влево
      Case vbKeyDown      Ловец.Руль = вниз
      Case vbKeyRight:    Ловец.Руль = вправо
      Case vbKeyControl: Ловец.Руль = стоп
   End Select
End Sub
```

Модуль clsШар

```
Public x As Long                           'Координаты шара
Public y As Long
Private dx As Long                         'Шаг шара по горизонтали и вертикали между двумя импульсами таймера
Private dy As Long
Private Макс_шаг As Long                   'Максимально возможное значение шага шара
```

```
Private Sub Class_Initialize()                    'Процедура, выполняющаяся при рождении шара
   Макс_шаг = 100
End Sub

Public Sub Начальная_установка()                  'Шар встает в исходную позицию и настраивается на новую игру
         'Ставим шар на исходную позицию:
   x = f.shpБортик.Left + f.shpБортик.Width * 3 / 4   'Она отстоит по горизонтали на четверть ширины поля от правого его края
   y = f.shpБортик.Top + f.shpБортик.Height / 2       'Она по вертикали расположена посредине поля
         'Вычисление шага:
   dx = Макс_шаг * (1 - 2 * Rnd)                  'Шаг по горизонтали случаен и не превосходит Макс_шаг
   dy = Макс_шаг * (1 - 2 * Rnd)                  'Шаг по вертикали случаен и не превосходит Макс_шаг
End Sub

Public Sub Действие()                             'Главная_процедура шара, выполняется один раз на каждом импульсе таймера
   If Поймали Then Выход_шара_из_игры             'Сначала шар определяет, не поймали ли его,
   Отскакивать_или_нет                            'затем, если бортик рядом, реагирует на него,
   Шаг                                            'и наконец делает шаг
End Sub

Private Sub Отскакивать_или_нет()                 'Реакция на бортик
   If Шар_у_пола_или_потолка Then
      dy = -dy                                    'Отскок от пола или потолка
   ElseIf Шар_у_стен Then
      dx = -dx                                    'Отскок от стен
   End If
End Sub

Private Sub Шаг()
   x = x + dx
   y = y + dy
End Sub

Private Function Шар_у_пола_или_потолка() As Boolean
   If y < f.shpБортик.Top Or y + Размер_шара > f.shpБортик.Top + f.shpБортик.Height Then
      Шар_у_пола_или_потолка = True
   Else
      Шар_у_пола_или_потолка = False
   End If
End Function

Private Function Шар_у_стен() As Boolean
   If x < f.shpБортик.Left Or x + Размер_шара > f.shpБортик.Left + f.shpБортик.Width Then
      Шар_у_стен = True
   Else
      Шар_у_стен = False
   End If
End Function

Private Function Поймали() As Boolean
         'ЕСЛИ расстояние по горизонтали между центрами шара и ловца меньше Дальности
         'И если расстояние по вертикали между центрами шара и ловца меньше Дальности, ТО
   If Abs(x - Ловец.x - ((Размер_ловца - Размер_шара) / 2)) < Дальность _
      And Abs(y - Ловец.y - ((Размер_ловца - Размер_шара) / 2)) < Дальность _
   Then
      Поймали = True
   Else
      Поймали = False
   End If
End Function

Private Sub Выход_шара_из_игры()
   x = -10000: y = -10000: dx = 0: dy = 0         'Убрать шар подальше с глаз долой и чтоб не двигался
   f.intЧисло_пойманных_шаров = f.intЧисло_пойманных_шаров + 1
End Sub
```

Запустите проект. Проверьте, правильно ли он работает. Поиграйте значениями шагов ловца и шара, их размерами, числом шаров и другими величинами, подберите наиболее удобные для себя.

Пояснения. Обратите внимание, что в обоих классах много одноименных переменных и процедур. Как я уже говорил чуть выше, никакой путаницы здесь произойти не может. Иметь одинаковые имена для элементов одинакового смысла удобно и правильно.

В модуле формы разберитесь самостоятельно. Я думаю, что комментариев достаточно. Поговорим подробнее о модуле шара. Совершенно аналогично модулю ловца здесь имеется два метода - Начальная_установка и Действие. Метод Действие главный, он определяет, что должен делать шар в каждое мгновение своего полета. Он должен знать, поймали его или нет, и пора ли отскакивать от бортика. Этому и посвящены первые две из трех строк процедуры Действие. Эти две строки вычисляют

нужным образом dx и dy, а третья строка - Шаг - изменяет в соответствии с этими значениями координаты x и y. Все, больше ничего во время движения шара делать не нужно.

Теперь посмотрим, что происходит при нажатии на кнопку "Начинай сначала". Выполняется процедура cmd_Начинай_сначала_Click и после пары прыжков по процедурам Visual Basic передает управление процедуре Начальная_установка каждого шара. Здесь трудности у вас может вызвать вычисление dx и dy. Поскольку значение Rnd есть случайное число в диапазоне от 0 до 1, то легко видеть, что как dx, так и dy будут случайными числами в диапазоне от -100 до 100. Этого достаточно, чтобы шар полетел со случайной скоростью в случайном направлении.

Теперь насчет отскока. Возможно, тем, кто не очень силен в математике и физике, покажется удивительным, что для отскока от горизонтальной преграды достаточно выполнить оператор dy = -dy, то есть поменять вертикальную составляющую шага на ее противоположное значение. "Но это действительно так!" Аналогично, достаточно выполнить оператор dx = -dx для отскока от вертикальной преграды. Чтобы лучше понять этот факт, запустите пошаговый режим при Макс_шаг=1000 и Число_шаров=1. При этом проследите внимательно за dx и dy, x и y.

Недоработки проекта

Замеченные мной недоработки проекта вызваны нежеланием усложнять и увеличивать в объеме его код. Вот они:

- Иногда шар, вместо того, чтобы отскочить от борта, начинает двигаться скачками вдоль него. Возьмем к примеру левый бортик. Почти наверняка это связано с неточностью обработки вещественных чисел в операторе x=x+dx, в результате чего после отскока опять выполняется условие x < f.shpБортик.Left и шарик опять выполняет оператор dx = -dx. Рискну посоветовать (сам не проверял): в момент отскока незначительно увеличьте абсоютное значение шага: dx = -(dx+0.001).
- Я не знаю, как поведет себя шар, попавший точно в угол. По идее, он должен там застрять. Но у меня так ни разу не было.
- Постоянными нажатиями на клавиши направления мы можем передвигать ловца за пределами стола. Ни к чему это. Надо бы запретить.

20.8. Еще об объектах

У вас могло создаться превратное впечатление, что объект - это обязательно что-то движущееся, причем в темпе, задаваемом таймером. Совсем нет. Вспомните элементы управления - список, флажок. Это все тоже объекты. Вы можете создать класс пользователя, представляющий собой набор процедур и функций для сложной обработки информации в текстовом поле. И таймера здесь никакого не нужно.

Вы знаете, что у элементов управления есть свойства, методы и события. У объекта пользователя мы изучили только свойства и методы. А события? Они тоже есть, вернее, вы можете запрограммировать их. Но останавливаться на этом я не буду.

Форма как объект

Ревниво оберегая и инкапсулируя объекты пользователя, мы совсем забыли о форме. Ведь это тоже объект. И не годится оставлять его незащищенным извне, то есть со стороны модулей классов и кода. Надо инкапсулировать. Все, что можно, делать Private.

Кстати, мы ведь можем во время работы проекта "штамповать" нашу форму, как "штамповали" объекты по их классу. Делается это так. Добавьте в форму нашей игры кнопку Command1 и запишите в окно кода формы такую процедуру:

```
Private Sub Command1_Click()
   Dim f1 As New f     'Объявляется новая форма f1 как копия формы f
   f1.Show             'Форма f1 загружается и показывается на экране
End Sub
```

Запустите проект и в один из моментов игры нажмите кнопку Command1. Вы увидите, что игра идет уже на двух столах, причем одна и та же игра. Подвигайте ловца, понажимайте на кнопки. Поразмыслите сами, какие это сулит возможности в будущем.

Свойства только для чтения

Свойства только для чтения, как я уже говорил чуть раньше, нужны для того, чтобы обезопасить наш объект от вмешательства извне. Давайте сделаем свойством только для чтения свойство x ловца. Идея здесь такая. Объявим x, как Private. Проект сразу перестает работать, так как модулю формы для смещения изображения ловца нужно знать его координаты. Придумаем нашему свойству x "псевдоним", под которым он будет виден снаружи - Хл. Вся "фишка" в том, чтобы написать в классе clsЛовец глобальную функцию с именем Хл, все тело которой состоит из единственного оператора Хл=x:

```
Public Function Хл() As Long
   Хл = x
End Function
```

Теперь заменим везде снаружи обращение Ловец.x на Ловец.Хл. Поскольку обращение к переменной и функции без параметров синтаксически неотличимо, то снаружи могут сколько угодно думать, что обращаются к переменной Хл, когда на самом деле это функция. Проект работает, проверьте. А где же здесь желанная цель "только для чтения"? А она достигнута. Потому что присваивать значение функции мы можем только в ее теле, а уж никак не из других модулей.

Теперь сотрите эту функцию. Создадим ее другим, общепринятым путем. Visual Basic предлагает удобный интерфейс для

создания процедур, функций, событий и свойств: **Tools→Add Procedure→** поставьте переключатели в положение Property (так как мы хотим создать свойство) и Public, дайте имя свойству - Хл **→OK**. Перед нами появятся две заготовки:

```
Public Property Get Хл() As Variant

End Property

Public Property Let Хл(ByVal vNewValue As Variant)

End Property
```

Слова **Property Get** означают "Получи (узнай) значение свойства", а слова **Property Let** означают "Присвой значение свойству". Нижнюю заготовку стираем, так как она предлагает записать код, позволяющий менять значение Хл снаружи. А верхнюю можно использовать вместо старой функции, заменив для порядка Variant на Long:

```
Public Property Get Хл() As Long
   Хл = x
End Property
```

Здесь открывается простор для контроля со стороны объекта над желающими снаружи видеть свойство и даже для введения их в заблуждение, так как в теле свойства мы можем писать какой угодно код, например:

```
Public Property Get Хл() As Long
   Хл = x + 1000
   If x > 5000 Then MsgBox ("Дальше замучаю сообщениями")
End Property
```

Наследование, полиморфизм

Кроме инкапсуляции у классов объектов есть еще две замечательные черты: наследование и полиморфизм. Они важны в тех проектах, где имеются не один-два класса объектов, а целые их системы, иерархии. Понятие о наследовании и полиморфизме я дам вам на уровне аналогий.

Вообразим, что мы создали простой класс Автомобиль, наделив его всего лишь двигателем, рулем и 4 колесами. И никаких подробностей. Полюбовавшись работой получившейся самоходной тележки, мы решили развивать проект дальше и создать еще три класса, более богатых и подробных: Амфибия, Грузовик и Автобус. У каждого из новых классов, как и у Автомобиля, есть двигатель, руль и 4 колеса, но вдобавок к ним и много подробностей, например, у Амфибии гребной винт, а у Грузовика - кузов.

Как создавать эти 3 класса? Можно так же, как мы создавали clsШар, когда мы вручную записали весь код, причем скопировав часть кода из clsЛовец. В нашем случае нам можно было бы скопировать весь код Автомобиля в каждый из трех модулей, добавив затем в них для каждого свои процедуры и функции. Но есть гораздо более удобный и "правильный" способ - это **наследование**. Мы просто объявляем, что новый класс Грузовик является наследником класса Автомобиль. При этом Грузовик неявно (невидимо) приобретает весь код своего родителя - Автомобиля. Ничего не записав в код Грузовика, мы уже можем пользоваться им как Автомобилем. Чтобы снабдить Грузовик дополнительными чертами, мы пишем ему новые процедуры и функции. Аналогично поступаем с Амфибией и Автобусом.

Самое интересное то, что если мы изменяем что-то в коде родителя (Автомобиль), то это изменение тут же сказывается на наследниках. Например, если мы в Автомобиле заменили двигатель внутреннего сгорания на электрический, то эта замена немедленно произойдет и в Амфибии, и в Грузовике, и в Автобусе. Это очень ценное и удобное качество.

Полиморфизм в частном случае - это выполнение разных действий процедурами с одинаковыми именами. Так, метод Начальная_установка шар и ловец выполняют по-разному. При наследовании полиморфизм проявляется тогда, когда мы у наследника как-то изменяем процедуру родителя. Скажем, если для Автомобиля процедура Остановка это просто остановка, то для Автобуса это еще и объявление по громкоговорителю названия остановки.

В заключение должен сказать, что Visual Basic версий 6.0 и более ранних не поддерживает настоящие полиморфизм и наследование. Впервые их поддержка осуществлена в Visual Basic.NET.

Глава 21. Visual Basic и Интернет

В этой главе мы рассмотрим, как знание Visual Basic поможет вам добавить в вашу Web-страничку такие элементы, которые подвластны только программистам. Вы обнаружите, что ваше знание не только полезно, но и опасно для окружающих. Эта глава дает понятие о работе с Web-страницей, но ни в коем случае не систематические знания в этой области. Основываясь на примерах, приведенных в главе, вы сможете легко сделать что-нибудь интересное на своей страничке. Однако, чтобы стать экспертом, читайте более толстые книжки.

21.1. Понятие об Интернет, Web-станицах и языке HTML

Интернет - это сто миллионов компьютеров на земном шаре, соединенные между собой телефонными линиями и другими каналами связи. Упрощенно структуру Интернет можно представить следующим образом. Забудем пока о миллионах. В мире существуют многие тысячи (не миллионы) мощных компьютеров, которые соединены между собой более-менее скоростными линиями связи и никогда не выключаются. Называют их **узлами** или **Web-серверами**. Они-то и составляют "спинной хребет" Интернета. Схема их соединения на карте Земли напоминает паутину дорог, соединяющих города. Города - узлы, дороги - линии связи, такая аналогия. Если вы хотите подключить ваш домашний компьютер к Интернет, то можете это сделать, только подключившись к какому-нибудь узлу, чаще всего ближайшему. Узлом (Web-сервером) владеет фирма, которая называется **провайдером**, она берет с вас деньги и разрешает подключаться к своему Web-серверу. Чаще всего это подключение идет по низкоскоростной телефонной линии. К этому же серверу подключены и тысячи других желающих из ваших мест. Вот таким образом и получается сеть из ста миллионов компьютеров.

Вы знаете, что для большинства владельцев компьютеров путешествие по Интернет - это бесконечное перелистывание огромного числа занимательных, полезных, скучных, грязных, глупых, добрых Web-страничек. Любая из этих страничек круглосуточно доступна каждому подключившемуся к Интернет. В настоящее время любой подключившийся к Интернет имеет также возможность простыми средствами создать свою собственную Web-страницу и сделать ее видимой в Интернет всему миру. Страничку вы не спеша создаете на своем компьютере, затем соединяетесь со своим (можно и с некоторыми другими) сервером и щелкнув несколько раз мышкой, помещаете ее на сервере. Иначе, если она будет помещена не на сервере, а на вашем компьютере, то когда ваш компьютер выключен, она никому не будет доступна. На жестком диске сервера ваша и тысячи других Web-страниц будут неограниченно долго храниться.

Итак, все странички в Интернет размещены на серверах.

Давайте представим, что происходит, когда вы хотите увидеть на экране своего компьютера страничку, размещенную на сервере где-нибудь в Америке. Вы посылаете адрес странички на свой сервер, тот по линии связи соединяется с сервером в Америке и нужная страничка отправляется к вам в компьютер.

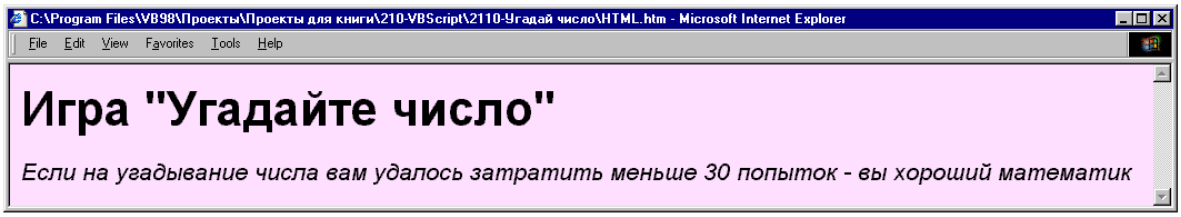
Все вы видели Web-странички, они напоминают страницы красочных журналов, на них много текста на красочном фоне, есть фотографии, простейшая анимация. Тот, кто знаком с красочной графикой, знает, что она требует для своего представления большого объема информации. Такое количество слишком долго будет передаваться по линиям связи. Как же решается эта проблема? Приведу аналогию. Два художника живут в разных городах. Один звонит другому и говорит, что хочет посмотреть на его новую картину. Тому посылать картину по почте долго и дорого, поэтому вместо картины он присылает письмо такого содержания: "Возьми холст и краски. В нижнем левом углу холста нарисуй златокудрую нимфу, фотография которой вложена в письмо. В правом верхнем углу нарисуй грозовую тучу. …" И так далее. Первый художник, выполнив все, что сказано в письме, будет иметь перед собой картину, очень похожую на оригинал.

Итак, в Интернете по линиям связи передается не сама страница, а описание того, как ее рисовать, что и каким шрифтом писать плюс очень экономно закодированные фотографии с этой страницы и некоторые другие элементы. В вашем компьютере эту информацию поджидает программа, которая играет роль художника, рисующего картину по ее описанию в письме. Эта программа называется **броузером**. Броузер не только рисует на экране страничку по описанию, но и позволяет удобно листать странички и делать почти все, что нужно для работы в Интернет. В мире наиболее распространены два броузера: Internet Explorer и Netscape Navigator.

Описание Web-страницы выполняется на специальном языке, который называется **HTML**. Язык этот прост и понятие о нем дается в следующем разделе.

21.2. Создание Web-страницы

В этой главе мы с вами для примера будем создавать Web-страницу с игрой "Угадай число". Начнем с малого. Пусть пока ваша страничка должна выглядеть так:



Для этого вам достаточно в Notepad (это "Блокнот" - простейший текстовый редактор Windows) создать документ такого содержания:

```
<html>
<body bgcolor="#FFDFFF">
<h1>Игра "Угадайте число"</h1>
<i>Если на угадывание числа вам удалось затратить меньше 30 попыток - вы хороший математик</i>
</html>
```

Пояснения:

- Любой документ на языке HTML должен начинаться с **тега** <html> и заканчиваться тегом </html>.
- Строка <body bgcolor="#FFDFFF"> приказывает броузеру задать цвет фона страницы в 16-й системе счисления (см. 9.6).
- Тег <h1> обозначает "самый крупный заголовок", поэтому на нашей страничке первая строка получилась крупным шрифтом. Тег <h2> означал бы заголовок поменьше и так далее. Вообще, в большинстве случаев теги встречаются парами - *открывающий* и *закрывающий* теги, причем закрывающий отличается от открывающего косой чертой. Пара тегов рассказывает броузеру о том, что нужно делать с элементом информации, который она охватывает.
- Тег <i> обозначает "курсив", поэтому на нашей страничке вторая строка получилась курсивом.

Этот HTML-документ и является описанием нашей Web-страницы. После создания HTML-документа его необходимо сохранить с расширением htm или html. Теперь, чтобы увидеть получившуюся Web-страничку живьем, вам достаточно открыть этот документ в броузере, имеющемся на вашем компьютере.

Вообще, для создания Web-страницы совсем не обязательно вручную создавать HTML-документ. Вы можете сконструировать Web-страничку в редакторе Microsoft Word или в специальной программе FrontPage Express, не написав ни одного тега, подобно тому, как в Visual Basic мы конструируем проект в режиме проектирования. HTML-документ при этом создается автоматически. Рассматривая страничку в броузере Internet Explorer, вы всегда можете увидеть и редактировать породивший его HTML-документ при помощи **View→Source**.

Я предлагаю вам такой порядок доработки Web-страницы. Вы дописываете в HTML-документ очередную строку, сохраняете его, затем в броузере нажимаете кнопку обновления страницы (Refresh). Страница приобретает вид в соответствии с последними изменениями в HTML-документе. Затем вы снова изменяете HTML-документ и так далее.

21.3. Сценарий на Web-странице

Итак, мы решили развлечь читателя нашей Web-страницы. Мы предложим ему поиграть на нашей странице в игру "Угадай число". Задание на создание такой игры в Visual Basic вы уже получали раньше (16.2). Напомню условие. Компьютер загадывает число из диапазона от 1 до миллиарда. Человек должен его отгадать. Причем за наименьшее число попыток. При каждой попытке компьютер выводит номер попытки и подсказку - "*мало*" или "*много*". Сохраняться, как того требовало задание 127, мы для простоты не будем.

Сначала запрограммируем игру не на Web-странице, а как мы привыкли - в Visual Basic. Разместим на форме:

- Кнопку cmdTry с надписью "Попытка"
- Текстовое поле txtNumber - для того, чтобы человек вводил туда очередное число
- Текстовое поле txtMessage - для слов "Много", "Мало" и "Вы угадали"
- Текстовое поле txtNumberTry - для отображения количества попыток

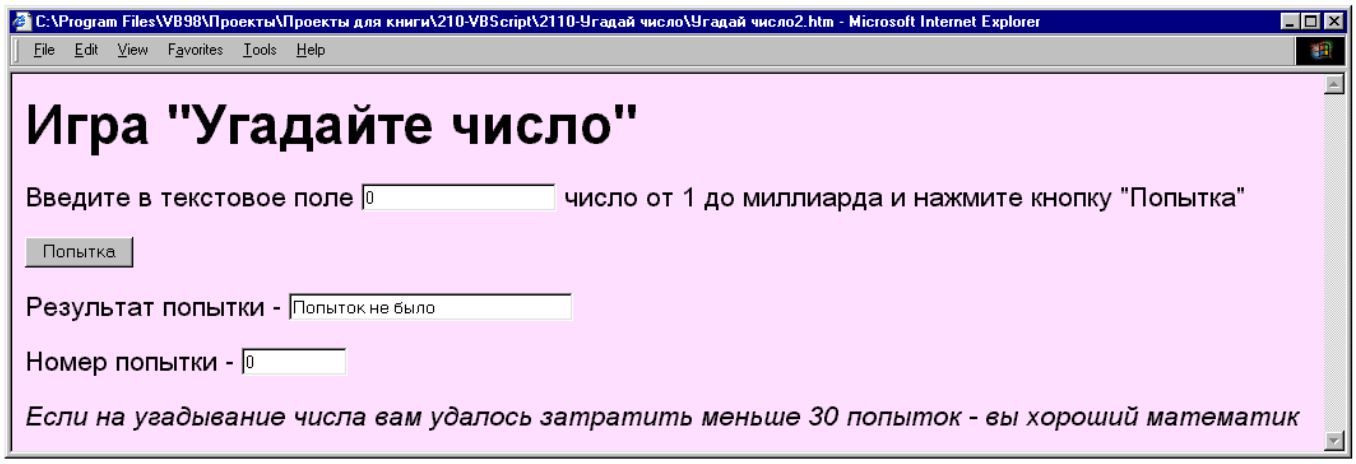
Вот программа:

```
Dim A As Long                    'это число - очередная попытка человека
Dim SecretNumber   As Long       'это загаданное число

Private Sub Form_Load()          'Начальные установки:
   Randomize
   SecretNumber = Round(1000000000 * Rnd)
   txtNumber.Text = 0
   txtMessage.Text = "Попыток не было"
   txtNumberTry.Text = 0
End Sub
                                 'Процедура обработки нажатия на кнопку:
Sub cmdTry_Click()
   A = Val(txtNumber.Text)       'превращение строки в число
   If A > SecretNumber Then
      txtMessage.Text = "Много"
   ElseIf A < SecretNumber Then
```

```
        txtMessage.Text = "Мало"
      Else
        txtMessage.Text = "Вы угадали"
      End If
      txtNumberTry.Text = txtNumberTry.Text + 1  'увеличиваем счетчик числа попыток
    End Sub
```

Программа проста и не требует пояснений. Полужирным шрифтом я выделил фрагменты текста, которые подвергнутся изменениям при программировании для Web-страницы.

Теперь приступим к программированию нашей игры на Web-странице. Вот внешний вид страницы, который мы хотим получить:



Программа для этой игры должна быть включена в HTML-документ страницы. Там она будет называться **сценарием**. Пишется она не совсем на Visual Basic, а на так называемом языке **VBScript**. Это сильно упрощенный и немного измененный Visual Basic. Поэтому простой перенос в HTML-документ программы из Visual Basic не принесет успеха. Самое интересное то, что для программирования на VBScript вам совершенно не нужно иметь на компьютере Visual Basic. На том компьютере, где будет читаться ваша страница, он тоже не нужен. Необходимое условие одно - тот, кто будет читать вашу страницу, должен делать это при помощи броузера Internet Explorer. Не думаю, чтобы это было сильным ограничением, поскольку Windows включает в себя Internet Explorer. Во всяком случае, в России подавляющее большинство пользователей Windows используют именно этот броузер.

Internet Explorer в процессе рисования вашей страницы на экране читает ее HTML-документ и наткнувшись в его тексте на сценарий, выполняет его точно так же, как Visual Basic выполняет программу при нажатии на Start. Вот такой замечательный продукт этот Internet Explorer. Мастер на все руки.

Сценарий выполняется заново каждый раз, когда вы загружаете страницу в Internet Explorer.

Вот HTML-документ нашей страницы со включенным в него сценарием:

```
<html>

<body bgcolor="#FFDFFF">
<h1>Игра "Угадайте число"</h1>
Введите в текстовое поле   <input type="text" size="20" name="txtNumber">
число от 1 до миллиарда и нажмите кнопку "Попытка"<p>
<input type="button" name="cmdTry" value="Попытка"><p>
Результат попытки - <input type="text" size="30" name="txtMessage"><p>
Номер попытки - <input type="text" size="10" name="txtNumberTry" ><p>
<i>Если на угадывание числа вам удалось затратить меньше 30 попыток - вы хороший математик</i>

<script language="VBScript"><!--                    'СЦЕНАРИЙ
  dim A                              'это число - очередная попытка человека
  dim SecretNumber                   'это загаданное число

  Randomize
  SecretNumber = Round (1000000000 * Rnd)
                                     'Начальные значения текстовых полей:
  txtNumber.Value=0
  txtMessage.Value = "Попыток не было"
  txtNumberTry.Value=0
                                     'Процедура обработки нажатия на кнопку:
Sub cmdTry_OnClick()
  A = int(txtNumber.Value)           'превращение строки в число
  if A>SecretNumber then
     txtMessage.Value = "Много"
  elseif A<SecretNumber then
     txtMessage.Value = "Мало"
  else
     txtMessage.Value = "Вы угадали"
```

```
    end if
    txtNumberTry.Value= txtNumberTry.Value + 1
End Sub
--></script>

</html>
```

Пояснения: Сценарий располагается между строками

<script language="VBScript"><!--

и

--></script>

Сравните текст сценария с текстом программы на Visual Basic. Они очень похожи. Полужирным шрифтом я выделил фрагменты сценария, которые отличают его от соответствующих фрагментов программы на Visual Basic. Так, новостью является запрет указания типа в операторах Dim. В этом случае переменные имеют тип Variant. Вместо свойства Text используется свойство Value. Вместо Click - OnClick. вместо Val - Int.

Для работы сценария необходимо было разместить на странице следующие элементы:

- Кнопку cmdTry с надписью "Попытка"
- Текстовое поле txtNumber - для того, чтобы человек вводил туда очередное число
- Текстовое поле txtMessage - для слов "Много", "Мало" и "Вы угадали"
- Текстовое поле txtNumberTry - для отображения количества попыток

Но откуда взять эти элементы, если мы Visual Basic даже не запускали? Оказывается, размещаются они на странице средствами языка HTML. Язык VBScript для этого тоже не нужен.

Давайте не спеша читать HTML-документ сверху вниз. Вот первая незнакомая строка:

```
Введите в текстовое поле   <input type="text" size="20" name="txtNumber">
число от 1 до миллиарда и нажмите кнопку "Попытка"<p>
```

Посмотрите, какая строка Web-страницы ей соответствует. Выражение

<input type="text" size="20" name="txtNumber">

как раз и размещает в этой строке Web-страницы текстовое поле. То, что это должно быть именно текстовое поле, а не, скажем, кнопка, задает выражение input type="text", размер по горизонтали 20 задается выражением size="20", а имя поля задается выражением name="txtNumber".

В этой строке есть незнакомый нам тег - <p>. Это просто перевод строки на Web-странице.

Аналогично изложенному, следующая строка HTML-документа

```
<input type="button" name="cmdTry" value="Попытка"><p>
```

размещает в следующей строке Web-страницы кнопку (благодаря выражению input type="button"). Надпись на кнопке задается выражением value="Попытка".

Следующие три строки поясняются аналогично.

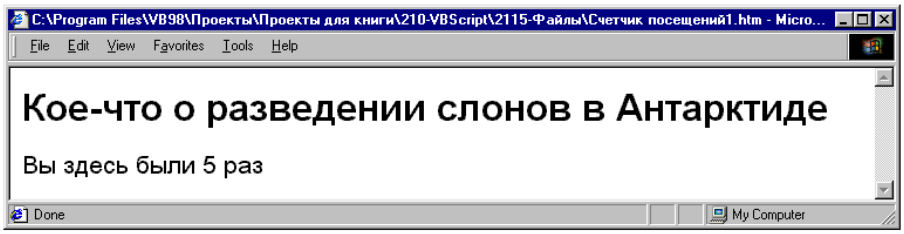
21.4. Доступ к локальному диску

Будем называть *локальным диском* жесткий диск чужого компьютера, на котором читается ваша Web-страница (если она читается на вашем же компьютере, то тогда локальный диск - это жесткий диск вашего компьютера). Доступом к локальному диску будем называть возможность при помощи сценария вашей Web-страницы читать, стирать или записывать информацию в файлы локального диска (примерно так, как мы это делали в 16.2 с файлами на диске нашего компьютера), а также осуществлять все другие операции с файлами и папками.

Спрашивается, хорошо это или плохо - при помощи вашей Web-страницы иметь доступ к диску чужого компьютера? Это примерно то же самое, что спросить, хорошо это или плохо - прийти к незнакомому человеку в гости и пользуясь его доверчивостью иметь доступ ко всем вещам и секретным документам в его квартире. Если вы честный человек, то ничего не украдете и подглядывать не будете. Зачем вам тогда доступ? Получается, что доступ - это плохо? Не всегда. Он часто бывает нужен в общении между знакомыми людьми, которые доверяют друг другу и которым было бы удобно считывать информацию с локальных дисков друг друга.. Или возьмите ситуацию сохранения в играх. Пользователь, играющий на вашей страничке в "Угадай число" и желающий после 20-й попытки сохраниться, должен иметь возможность это сделать. Сделать же это проще всего на локальном диске. Но для этого нужен доступ к нему со стороны сценария игры. Выходит, что без доступа все-таки не обойтись.

Создатели VBScript предоставили программистам доступ к локальным дискам, но они прекрасно понимали опасность такого доступа, поэтому вся работа с файловой системой в VBScript организована совсем по-другому, чем в Visual Basic. Вместо непосредственной работы с файлами, как мы это делали с файлами на диске нашего компьютера, организованы специальные объекты, представляющие файловую систему и обеспечивающие более безопасную работу.

Проиллюстрирую работу VBScript с локальным диском на одном-единственном примере. Пусть ваша страничка посвящена разведению слонов в Антарктиде и она настолько интересна, что побывавший на ней снова и снова туда возвращается. Вставим на страничку сценарий, единственная цель которого - напомнить пользователю, сколько раз он был на этой странице. Вот внешний вид странички после открытия (сведения об успехах разведения слонов будут приведены в следующем издании книги):



Идея сценария такова. При первом открытии страницы на данном компьютере сценарий выдает на страницу сообщение "Вы на этой страничке ни разу не были", создает на локальном диске в корне диска C: файл INFORMAT.TXT и записывает в него значение счетчика посещений - число 0.

При каждом открытии страницы сценарий ищет в корне диска C: файл INFORMAT.TXT и если находит, то считывает с него значение счетчика, увеличивает его на 1 и отображает на странице в виде "Вы здесь были 5 раз". Если же файл не найден, сценарий делает вывод, что на этом компьютере страница еще не открывалась, и делает то, что я описал в предыдущем абзаце.

Вот HTML-документ нашей страницы со включенным в него сценарием:

```
<html>
<h2>Кое-что о разведении слонов в Антарктиде</h2>
<script language="VBScript"><!--

    Dim objFs        'Объект - Файловая система локального диска
    Dim objFile      'Объект - Файл
    Dim sAdres       'Переменная - Адрес файла на диске
    Dim intSchetchik 'Переменная - счетчик посещений страницы

    sAdres="c:\INFORMAT.TXT"
    Set objFs = CreateObject ("Scripting.FileSystemObject")
    If objFs.FileExists (sAdres) Then
        Set objFile = objFs.OpenTextFile(sAdres, 1)
        intSchetchik = objFile.ReadLine
        intSchetchik = intSchetchik + 1
        Document.Write "Вы здесь были " & intSchetchik & " раз"
    Else
        Document.Write "Вы на этой страничке ни разу не были"
        intSchetchik = 0
        Set objFile = objFs.CreateTextFile (sAdres)
    End If
    objFile.Close
    Set objFile = objFs.OpenTextFile(sAdres, 2)
    objFile.WriteLine (intSchetchik)
    objFile.Close
    Set ObjFs = Nothing
--></script>
</html>
```

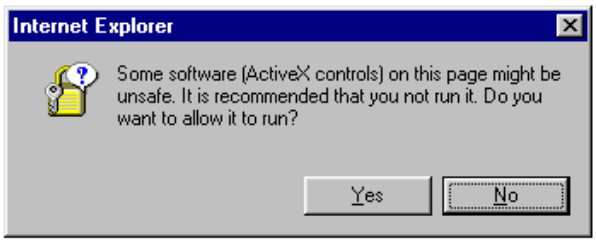
Пояснения: Прочтите строки объявлений. Далее рассмотрим строку

sAdres = "c:\INFORMAT.TXT"

Она задает адрес и имя текстового файла на локальном диске, в котором сценарий будет хранить счетчик посещений. Строка

Set objFs = CreateObject ("Scripting.FileSystemObject")

создает экземпляр объекта Файловая система. С этого мгновения вступают в действие меры безопасности. На экране компьютера, читающего вашу Web-страницу, возникает сообщение:



которое предупреждает пользователя, что программы на этой страничке могут быть опасными и не рекомендует разрешать их выполнение. У пользователя еще есть возможность нажать на No. Мой совет прост: Если эта страничка не принадлежит вашему лучшему другу - жмите No. Теперь рассмотрим строку

If objFs.FileExists (sAdres) Then

Здесь используется метод FileExists объекта objFs, который определяет, существует ли файл по указанному адресу sAdres. Смысл строки такой: Если файл c:\INFORMAT.TXT существует, то …

Строка

Set objFile = objFs.OpenTextFile(sAdres, 1)

открывает объект - текстовый файл для чтения (потому что 1). Строка

intSchetchik = objFile.ReadLine

считывает из него строку и присваивает счетчику. Следующая строка увеличивает счетчик на 1, а строка

Document.Write "Вы здесь были " & intSchetchik & " раз"

записывает на страничку указанный текст.

Из дальнейших строк поясню следующие:

Set objFile = objFs.CreateTextFile (sAdres)

создает на диске файл по указанному адресу.

objFile.Close

закрывает файл, независимо от того, какая ветвь оператора If выполнялась - Then или Else.

Set objFile = objFs.OpenTextFile(sAdres, 2)

открывает файл для записи (потому что 2).

objFile.WriteLine (intSchetchik)

записывает в файл значение счетчика.

Set ObjFs = Nothing

освобождает память компьютера от объекта Файловая система.

21.5. Собственный броузер

Вы можете внутри своего проекта создать собственный броузер, который даст вам возможность просматривать Web-страницы в окне, открытом прямо на форме. Ваши действия: **Project → Components → Microsoft Internet Controls → OK**. Затем поместите на форму элемент управления **WebBrowser** размером побольше. Теперь вам достаточно выполнить строку

WebBrowser1.Navigate "http://www.yahoo.com/"

И если вы в данный млмент подсоединены к Интернету, в окне вашего броузера появится страничка с указанным в кавычках адресом:

Здесь **Navigate** - метод объекта WebBrowser1.

В отличие от фирменных броузеров у вашего броузера нет ни кнопок, ни списков, ни других инструментов, облегчающих навигацию по Интернет. Вы сами прекрасно сможете организовать все, что вам нужно, используя в проекте кнопки, списки и другие стандартные элементы управления Visual Basic. В этом вам помогут следующие элементы объекта WebBrowser1:

- Свойство **Busy**. Оно равно True, если броузер занят - ищет или скачивает страницы.
- Событие **DocumentComplete**. Наступает, когда страница или фрейм страницы загружены в окно броузера.
- Метод **Stop**. Пользуйтесь им, чтобы прервать слишком медленную загрузку Web-страницы.

.

Кроме описанных возможностей работы в Интернет, Visual Basic позволяет организовать работу с электронной почтой (E-mail), запускать Internet Explorer, не выходя из проекта, и т.д.

Если вы хотите совершенствоваться в создании Web-страничек, то сначала освойте стандартные средства HTML или Word или FrontPage, не используя программирования. И только затем снова беритесь за программирование.

Глава 22. Visual Basic и базы данных

Если вы спросите профессионального программиста, зачем нужен Visual Basic, он ответит - В большинстве случаев для программирования баз данных. Да, действительно, ведь компьютер - это инструмент для обработки информации, а информация в деловом мире хранится в основном в базах данных, поэтому важная задача языка программирования, если он хочет быть нужным - обеспечить быструю, удобную и надежную работу с ними.

Здесь я познакомлю вас с тем, как Visual Basic работает с базами данных. Это только знакомство. Вы сможете выполнять несколько основных операций с базами данных и для общего представления о предмете этого достаточно, но для профессиональной работы вам нужно почитать книжки потолще, тем более, что почти каждая толстая книжка по Visual Basic уделяет базам данных чуть ли не половину своей толщины.

22.1. Понятие о базах данных

Что такое база данных, я пояснил в 14.2 на примере базы данных компьютерных игр. В подавляющем большинстве случаев база данных - это одна или несколько прямоугольных таблиц, таких, например, как эта таблица, посвященная боксерам:

Фамилия И.О.	Дата рождения	Спортивное общество	Разряд по боксу
Агамалов С.В.	12.4.78	Трудовые резервы	3
Парменов Л.В.	31.11.82	Динамо	3
Васин А.Н.	16.10.71	ЦСКА	3
Попов А.А.	14.2.79	Спартак	2
Яньков В.Ю.	27.1.84	Спартак	2
Иноземцев И.М.	3.3.80	Восток	1

Столбцы таблицы называются **полями**, строки - **записями**. Поля могут быть текстовыми (Спортивное общество), числовыми (Разряд по боксу), типа даты и времени (Дата рождения), булевскими, содержать объекты (картинки, звук, видео). Количество записей в реальных базах данных достигает многих тысяч. Обратите внимание, что база данных, рассмотренная нами в 14.2, к концу раздела приобретает более сложную структуру, чем просто одна прямоугольная таблица, но я ограничусь этим самым простым и распространенным случаем.

Для того, чтобы человек мог удобно работать с базами данных, написаны специальные программы - **системы управления базами данных** (**СУБД**). Основное, что нужно человеку от СУБД при работе с готовой базой данных - это сортировка, поиск и фильтрация. Например, наша база боксеров **отсортирована** по разряду. Пользователь должен иметь возможность легко сортировать базу данных по любому полю. Если в базе данных больше ста записей, становится актуальной проблема быстрого **поиска** нужной записи. Например, в базе данных о 20000 преступников нужно найти запись о Вольдемаре Сидорове, 1972 года рождения, по кличке Бармалей, чтобы посмотреть, есть ли у него шрам на левой щеке. Когда же мы ищем преступника по приметам, то из 20000 записей мы хотим вывести на экран только те несколько, что соответствуют кареглазым блондинам ростом от 175 до 180 см с татуировкой на правой руке. Этот процесс называется **фильтрацией**.

Кроме этих основных задач порядочная СУБД позволяет пользователю изменять содержимое записей, дополнять базу новыми записями, стирать ненужные, распечатывать в удобном виде нужную информацию из базы данных, позволяет при помощи программирования автоматизировать наиболее трудоемкие операции с базами данных и приспосабливать базы данных к конкретным нуждам пользователя.

СУБД, наиболее близкая к Visual Basic - это Microsoft Access, входящая в пакет Microsoft Office Professional. В Access вы можете и безо всякого программирования прекрасно работать с реальными базами данных. Ну а программирование работы в Access осуществляется на специальном языке Visual Basic for Applications, который является "диалектом" языка Visual Basic применительно к пакету Microsoft Office.

Если вы решили заняться базами данных, то у вас два пути:

- Быстренько освоить на уровне пользователя Access, и тогда ваша работа с базами данных будет удобной и приятной, а необходимость в программировании вы почувствуете не скоро. Когда же почувствуете, то будете программировать на практически известном вам Visual Basic for Applications.
- Осваивать работу с базами данных в среде Visual Basic. Это значительно труднее, но зато вы почти сразу же начнете программировать и сможете создавать проекты, в которых работа с базами данных объединена с совершенно другими областями обработки информации (например, с играми).

Название книги обязывает меня выбрать второй путь.

22.2. Создаем заготовку базы данных при помощи Visual Data Manager

Пусть мы хотим создать базу данных книг, выпущенных мифическим издательством "Контакт" в мифические времена. База данных должна иметь такой вид:

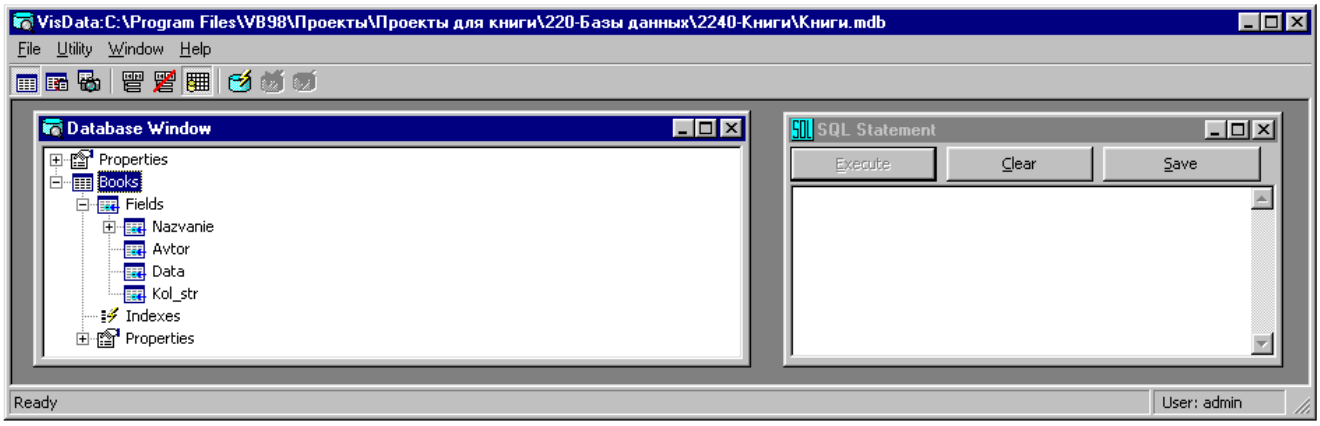
Название книги	Автор	Дата выпуска	Количество страниц
Понедельник начинается в субботу	Стругацкие	3.5.1965	187
Затерянный мир	Конан Дойль	15.11.1920	210
За миллиард лет до конца света	Стругацкие	14.7.1974	118
Белые ночи	Достоевский	30.9.1848	55
Туманность Андромеды	Ефремов	12.9.1957	348
Сорочинская ярмарка	Гоголь	31.12.1831	26

Наша задача - научиться осуществлять с этой базой все основные необходимые операции: создание, заполнение, изменение, сортировку, фильтрацию.

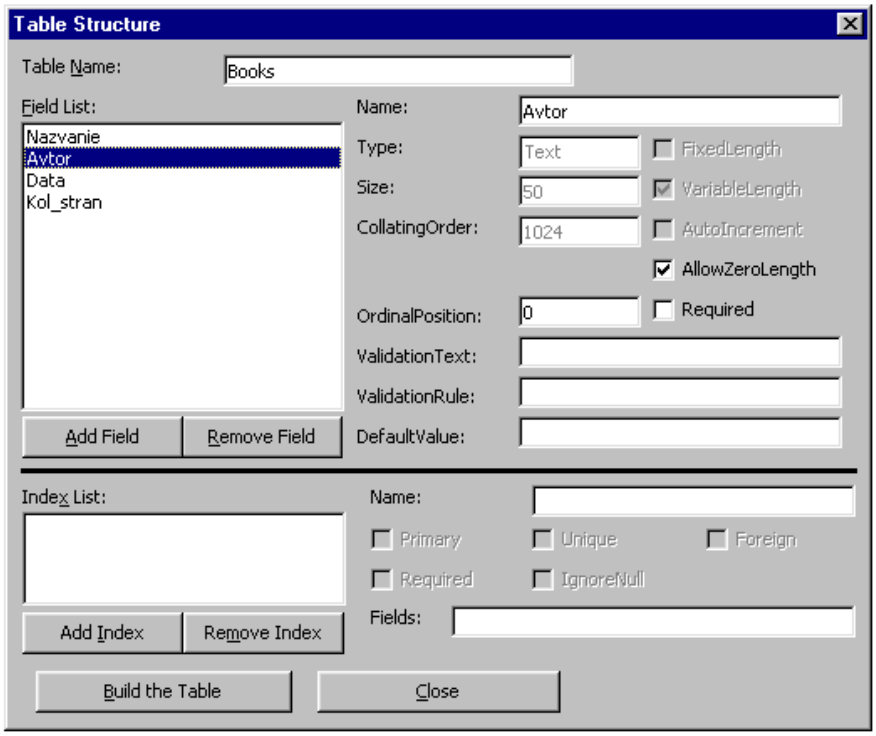
Конечно же, мы не будем пользоваться доморощенным программированием, как в 14.2 В Visual Basic имеются специальные инструменты для более быстрого и удобного выполнения всех этих операций. Первым мы используем **Visual Data Manager**, расположенный в меню Add-Ins. Он представляет собой по сути простенькую СУБД, работающую автономно от среды Visual Basic. Но я его использую только для создания структуры базы данных.

Начнем создавать нашу базу, но сначала придумаем имя для таблицы - "Books" (отдельное имя для таблицы необходимо потому, что база данных может содержать несколько таблиц). И для полей тоже придумаем имена: Nazvanie, Avtor, Data, Kol_stran.

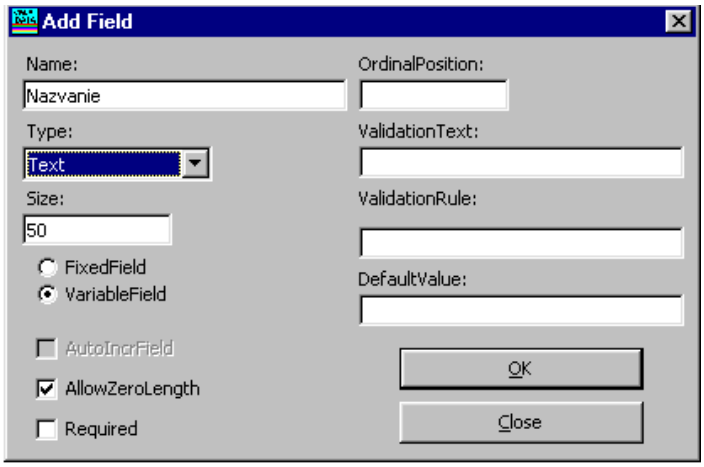
Add-Ins → Visual Data Manager→ File→ New→ Microsoft Access (мы выбрали Microsoft Access, это значит, что с этой базой данных можно будет в будущем работать не только из Visual Data Manager, но и в Access) **→ Version 7.0 MDB**. Перед вами откроется окно сохранения, которое предложит сохранить будущую базу в файле с расширением MDB. Дадим файлу имя "Книги" и сохранимся. Окно Visual Data Manager приобретет следующий вид (на картинке в левом окне показана структура таблицы, которую она приобретет позже, а пока там почти пусто):



Файл создан. Пора создавать таблицу. Щелкнем правой клавишей мышки внутри левого окна и в открывшемся контекстном меню выберем New Table. Перед нами откроется пустое окно структуры таблицы (на рисунке оно показано уже заполненным информацией о полях таблицы, причем слева виден список полей, а справа - информация о выделенном поле (Avtor) нашей таблицы):



Дадим имя таблице. Затем щелкнем по кнопке Add Field, перед нами откроется пустое окно для добавления поля (на рисунке оно показано уже заполненным информацией о поле Nazvanie нашей таблицы):



Вам достаточно только дать имя полю и выбрать под меткой Type его тип. Остальное - тонкости. Нажимая OK, введем информацию о всех полях, затем закроем окно добавления поля (кнопка Close).

Поле можно удалить, выделив его в окне структуры таблицы и нажав кнопку Remove Field.

Закончив ввод информации о всех полях, нажимаем кнопку Build the Table и на этом создание структуры таблицы закончено. Все остальное мы будем делать другими инструментами.

22.3. Работа с базами данных. Элементы управления Data и DBGrid. Язык SQL.

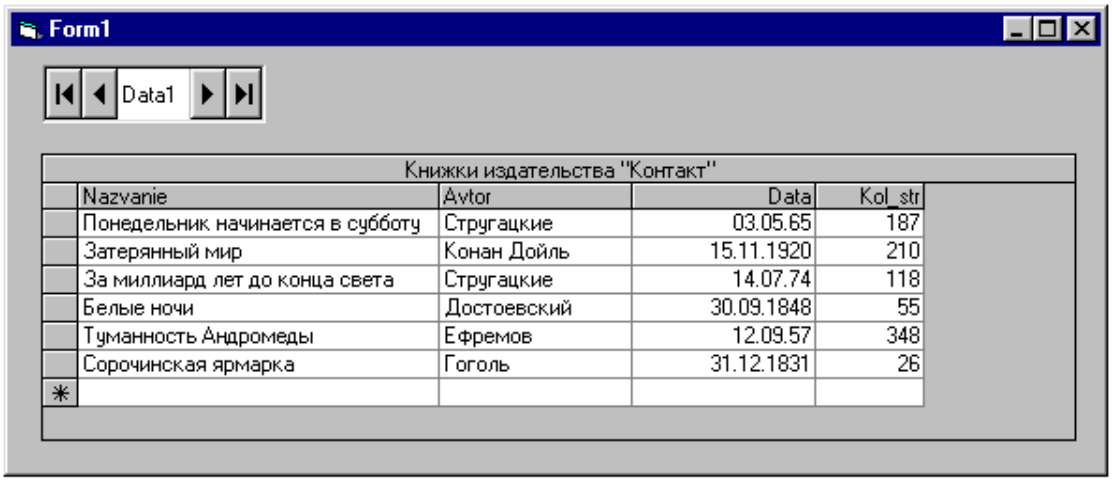
Для работы с базами данных непосредственно изнутри проекта Visual Basic удобно применять элемент управления Data, расположенный в Toolbox. Разместим его на форме. Пусть его имя будет Data1. Прежде всего объясним ему, с каким файлом базы данных он должен работать. Для этого придадим его свойству DatabaseName значение адреса созданного в предыдущем разделе файла. Затем проследим, чтобы свойство Connect имело значение Access, EOFAction - Add New, установим в качестве значения свойства **RecordSource** (источник записей для работы элемента Data1) имя таблицы нашей базы - Books. Эти свойства можно устанавливать и в режиме работы. Например,

Data1.RecordSource = "Books"

С этого момента вы можете запускать проект и работать с базой данных программным путем. Однако, не очень-то удобно это делать, не видя во время работы саму таблицу базы данных, а Data сам по себе ее не показывает. Способов увидеть ее несколько. Самый лучший - использование элемента управления DBGrid. Находится он так: **Project → Components→ Microsoft Data Bound Grid Control 5.0**. Разместите его на форме. Пусть его имя будет DBGrid1. Работать он будет в паре с Data, таблицу которого и будет делать видимой в режиме работы. Установим его свойства:

Прежде всего, поскольку в проекте могут присутствовать несколько элементов Data, нужно указать, с каким именно из них будет работать наш DBGrid, другими словами, к кому он будет **привязан**. Для этого установим значение его свойства DataSource в Data1. Установим также в True следующие свойства DBGrid1: AllowAddNew (разрешить вручную добавлять в таблицу новые записи), AllowDelete (разрешить вручную удалять записи из таблицы), AllowUpdate (разрешить вручную изменять данные в таблице). Можете придать какое-нибудь значение свойству Caption.

Приступим к заполнению таблицы данными о книгах. Для этого запустим проект. На экране появится пустая таблица. После того, как вы вручную заполните эту таблицу, проект будет иметь следующий вид:



В верхней части вы видите элемент Data1, от лицезрения которого толку мало, поэтому его вполне можно сделать невидимым.

Данные, которые вы вводите в поле, проверяются на принадлежность к типу поля. Обратите внимание, что Visual Basic "укоротил" те даты, которые считает "и так понятными".

Чтобы удалить запись, выделите ее щелчком по серому прямоугольнику слева от записи и нажмите на клавиатуре Delete. Можно расширять и сужать строчки и столбцы, перетаскивая границу между серыми прямоугольниками.

Когда вы завершите выполнение проекта, таблица будет автоматически сохранена в файл.

К Data можно привязывать не только DBGrid, но и другие элементы, например текстовые поля. Разместите на форме четыре текстовых поля и свяжите их с Data1. Также каждое из этих полей свяжите при помощи свойства DataField со своим полем таблицы. Запустите проект. Пощелкайте мышкой по разным записям в DBGrid. Вы видите, что значения текстовых полей автоматически становятся равными значениям полей текущей записи таблицы. И наоборот, стоит нам изменить содержимое текстового поля, как меняется соответствующее содержимое таблицы.

До сих пор мы меняли содержимое базы данных вручную. Посмотрим, как это можно делать программным способом. Таблица разделена на клетки-ячейки. Существует понятие *текущей ячейки* (current cell), с которой происходит в настоящий момент работа. Она задается номером записи (**DBGrid1.Row**) и номером поля (**DBGrid1.Col**). Свойство **DBGrid1.Text** означает содержимое текущей ячейки. Самая верхняя запись имеет номер 0, а не 1. То же относится и к самому левому полю. Например, при работе с нашей базой данных фрагмент

```
DBGrid1.Row = 3
DBGrid1.Col = 1
Debug.Print DBGrid1.Text
```

напечатает слово Достоевский.

Вот пример процедуры, которая заполняет числами 2, 3, 4 поле Kol_str в записях со 2-й по 4-ю:

```
Private Sub Command1_Click()
  DBGrid1.Col = 3
  For i = 2 To 4
    DBGrid1.Row = i
    DBGrid1.Text = i
  Next
End Sub
```

А вот строка, увеличивающая на одни сутки дату выпуска:

DBGrid1.Text = DateAdd("d", 1, DBGrid1.Text)

SQL

А теперь посмотрим, как при помощи простейших операторов выполняются такие сложные вещи, как сортировка и фильтрация. Для этого будем использовать популярный универсальный язык общения с базами данных **SQL**. Вот процедура, сортирующая нашу базу данных по дате выпуска книги:

```
Private Sub Command4_Click()
  Data1.RecordSource = "SELECT  Avtor, Nazvanie, Data FROM Books ORDER BY Data"
  Data1.Refresh
End Sub
```

Выполнив эту процедуру, вы увидите на экране такую таблицу:

Avtor	Nazvanie	Data
Гоголь	Сорочинская ярмарка	31.12.1831
Достоевский	Белые ночи	30.9.1848
Конан Дойль	Затерянный мир	15.11.1920
Ефремов	Туманность Андромеды	12.9.57
Стругацкие	Понедельник начинается в субботу	3.5.65
Стругацкие	За миллиард лет до конца света	14.7.74

Пояснения: Выше мы писали строку

Data1.RecordSource = "Books"

имея в виду, что источником данных для работы элемента Data1 будет целиком таблица Books. А вот первая строка нашей процедуры выбирает для работы элемента Data1 только три поля: Avtor, Nazvanie, Data. Действительно, в переводе с английского команда

SELECT Avtor, Nazvanie, Data **FROM** Books **ORDER BY** Data

означает

ВЫБРАТЬ Avtor, Nazvanie, Data ИЗ Books УПОРЯДОЧИТЬ ПО Data

То есть работа будет вестись не со всеми полями, а только с тремя перечисленными, причем записи будут упорядочены по возрастанию даты выпуска книги.

Если вместо ORDER BY Data написать ORDER BY Data **DESC**, то записи будут упорядочены по убыванию даты.

Строка

Data1.Refresh

просто приводит предыдущую команду SELECT в действие.

Следующая процедура осуществляет фильтрацию.

```
Private Sub Command6_Click()
   Data1.RecordSource = "SELECT * FROM Books WHERE Kol_str>100"
   Data1.Refresh
End Sub
```

Пояснения: Звездочка в выражении

SELECT * FROM Books

означает, что мы выбираем все поля таблицы Books.

Выражение

WHERE Kol_str >100

переводится

ГДЕ количество страниц >100

и приказывает вывести на экран только те книжки, что толще 100 страниц.

Следующая таблица иллюстрирует некоторые другие возможности SQL:

Команда	Смысл
SELECT * FROM Books WHERE Avtor = 'Стругацкие' **AND** Data > #21/11/1970#	На экран выводятся книжки Стругацких, выпущенные позже 21 ноября 1970 года. Таким образом, в выражении могут содержаться логические операции. Обратите внимание, что 'Стругацкие' берутся в одинарные кавычки, а не двойные, так как вся команда SELECT сама является ни чем иным, как строкой в операторе присвоения, и вторые двойные кавычки создали бы путаницу.
SELECT **TOP** 3 * FROM Books ORDER BY Data DESC	На экран выводятся 3 последние книжки, выпущенные издательством. Выражение ORDER BY Data DESC сортирует записи по убыванию даты, а выражение TOP 3 показывает на экране только верхние 3 записи из отсортированных.
SELECT * FROM Books **WHERE** Kol_str **BETWEEN** 100 **AND** 200	На экран выводятся книжки с числом страниц МЕЖДУ 100 И 200.

Глава 23. До свидания

Ну вот мы и заканчиваем. Самое время посмотреть, чего мы не сделали и что надо сделать.

23.1. Нерассмотренные возможности Visual Basic

Как я уже говорил, Visual Basic огромен и в одну книжку не умещается. Я сейчас перечислю те солидные части Visual Basic, которые я из-за их сложности или объема оставил за бортом книжки. Ведь мир - это не только то, что у нас за спиной, но и впереди тоже.

ActiveX

Если вам не хватает элементов управления, предоставляемых Бэйсиком, вы можете создать собственные. Скажем, вам нужен элемент управления, который вы можете поместить на Toolbox и состоящий, подобно элементу Common Dialog, из кнопки, метки, списка и других объектов, которые взаимодействуют запрограммированным персонально вами образом. Ваши действия: вы создаете проект, но при его создании в диалоговом окне New Project выбираете не Standard EXE, а ActiveX Control. После этого вы конструируете и программируете ваш проект с прицелом на то, что он превратится в нужный вам элемент управления. Процесс этот более сложный, чем при создании обычного проекта. Вы имеете возможность создавать свойства, события и методы будущего элемента управления, который будет называться **элемент управления ActiveX**.

Использовать его можно и на Web-страницах.

Windows API

Visual Basic обладает множеством стандартных функций, пользуясь которыми программист выполняет те или иные действия над элементвми проектируемого приложения. Операционная система Windows тоже обладает множеством функций, пользуясь которыми программист может выполнять действия над элементвми Windows. Это множество функций Windows называется **Windows API**. Многие функции Windows API позволяют выполнять действия, недоступные стандартным функциям Visual Basic, например, перезагрузку компьютера или установку фона рабочего стола. Visual Basic предоставляет возможность прямо из проекта пользоваться функциями Windows API, однако делать это не так просто, как использовать функции Visual Basic. Нужно хорошо представлять работу нужной вам функции Windows API и смысл ее параметров.

Многодокументный интерфейс - MDI

В Microsoft Word вы можете одновременно работать с несколькими документами и одновременно видеть их на экране, каждый - в своем окне. Все эти окна не обладают полной свободой на экране, а находятся внутри главного окна Word. Каждое окно на этапе проектирования было отдельной формой. В 19.1 я показал вам, как создавать проект с несколькими формами. Однако, более естественным средством для создания многооконного приложения является использование так называемого многодокументного интерфейса -**MDI**.

Создайте проект. Как всегда, он будет включать в себя обычную форму Form1. Затем - **Project → Add MDI Form**. В проект добавится еще одна форма, но необычная, это MDIForm1 - будущее главное окно приложения. Затем сделаем так, чтобы форма Form1 стала окном внутри этого главного окна. Для этого установим свойство MDIChild формы Form1 в True. Запустите проект. Последите за поведением формы и за ее реакцией на перетаскивание и щелчки по кнопкам в правом верхнем углу.

OLE

Пусть ваше приложение предназначено для получения по электронной почте неких исходных данных и составления на их основе качественных документов. Обеспечить пользователю возможность удобно создавать качественные документы - задача, которая потребует большой работы программиста. С другой стороны, уже имеются готовые приложения Windows типа Microsoft Word, которые прекрасно справляются с этой задачей. Спрашивается, можно ли, не выходя из своего приложения, воспользоваться возможностями Word? Можно. Эту возможность обеспечивает технология **OLE**. В 3.7 я уже описал, как вставить в свой проект документ Word. Но чтобы по настоящему работать с ним, нужно приложить еще много усилий. Аналогично можно вставить в приложение и электронную таблицу Microsoft Excel. Чтобы посмотреть, как по-настоящему работает технология OLE, зайдите в Word, найдите на панели инструментов кнопку "Добавить таблицу Excel " и щелкните по ней.

23.2. Миг между прошлым и будущим

Вот и все. На этом изложение программирования на Visual Basic я заканчиваю. Того, что вы знаете, вполне достаточно для программирования большинства задач из любой сферы человеческой деятельности. Дальнейшее изучение Visual Basic позволит вам составлять более эффективные программы, имеющие новые возможности в смысле использования объектов, работы с базами данных, Интернетом и т.д.

Дорогой друг! Вы вполне созрели для того, чтобы выполнить задание на звание *«Программист-любитель II ранга»*. На выбор – одно из трех:

- Игра в морской бой
- Игра в крестики-нолики на бесконечном поле
- Игра "Танковый бой".

Во всех трех программах игра должна вестись между человеком и компьютером.

Правила морского боя без компьютера общеизвестны. Вы с приятелем тайком друг от друга рисуете на листочках в клетку квадратные "моря" размером 10 на 10 клеток, обозначаете строки буквами, а столбцы цифрами и расставляете свои корабли. Стреляете по очереди, называя координаты квадратика, куда производится выстрел. Правила крестиков-ноликов на бесконечном поле такие же, как и у обычных крестиков-ноликов на поле 3 на 3, с тем отличием, что в линию нужно выстроить не 3, а 5 ноликов или крестиков. Между прочим, очень приятная игра. Конечно, запрограммировать игру на бесконечном поле довольно трудно, поэтому рекомендую ограничиться полем 20 на 20. Правила танкового боя приведу чуть ниже.

Требования к первым двум играм:

- Компьютер должен обнаруживать незаконное расположение кораблей и незаконные ходы в крестики-нолики.
- Компьютер должен вести счет партий и отображать его на экране
- Компьютер должен обеспечить возможность сохранения игры и загрузки сохраненной игры
- Удобный интерфейс. В частности, человек должен иметь возможность легко расставлять корабли, ставить нолики или крестики (например, мышкой или при помощи клавиш передвижения курсора и клавиши пробела)
- Неплохо сделать меню с такими, примерно, пунктами: *сохранить игру, загрузить игру, выход из игры, помощь*. В случае помощи достаточно по требованию игрока показать правила игры
- Для того, чтобы не было игр-близнецов, ходы компьютера не должны быть железно заданы. Например, свои корабли компьютер должен располагать от игры к игре с разумной долей случайности, чтобы человек не мог легко догадаться, где будут стоять корабли в следующей игре. То же относится к выстрелам и ходам в крестики-нолики.
- Рекомендации по выбору уровня сложности стратегии: В *морском бое* вы вполне сможете сделать стратегию компьютера очень сильной, чтобы человеку было трудно у него выиграть. А вот в *крестиках-ноликах* стратегию компьютера сделать очень сильной затруднительно. Достаточно, если компьютер будет обнаруживать простейшие угрозы человека: четверки с одним свободным концом и тройки с двумя свободными концами - и сам стремиться к их созданию.
- Ни в той, ни в другой игре нет особой нужды использовать объекты пользователя.

А вот правила танкового боя.

Посмотрите на рисунок. Танки передвигаются по коридорам. Ваши танки показаны светлыми треугольничками и стрелкой, танки противника - черными стрелками. Светлая стрелка - ваш личный танк, вы управляете им мышкой или с помощью клавиатуры. Остальными вашими танками и танками противника управляет компьютер. Танки стреляют, стараясь попасть в противника и не попасть в своего. По коридорам танки бродят случайно.

Если ваш личный танк врежется в любой другой танк, то он погибнет, остальные танки при столкновении не погибают.

В бою побеждает сторона, уничтожившая все танки противника. При этом неважно, погиб ваш личный танк или нет.

Необязательные правила: Вы можете выбрать любой из трех типов игры:

1 - Простая игра. В случае победы в бою вы переходите на следующий уровень. Всего в игре 12 уровней, различающихся числом снарядов у вашего танка и числом рядов на поле боя. Число снарядов у остальных танков очень большое. Сохраняться в простой игре нельзя.

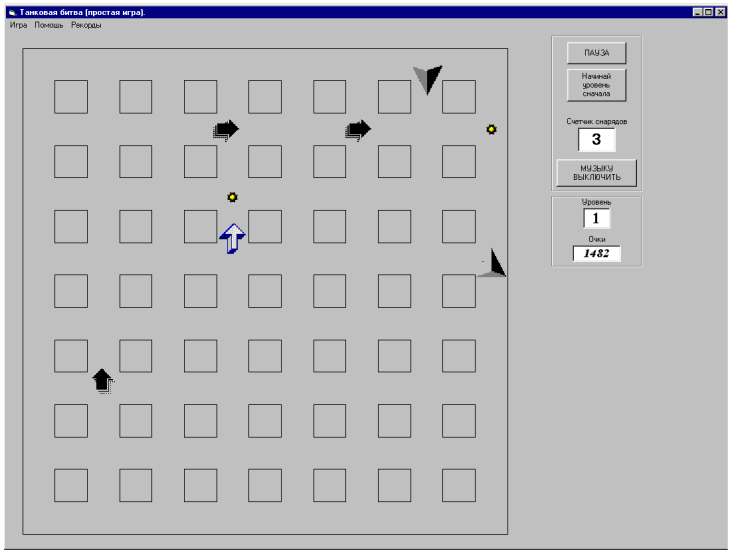
2 - Произвольная игра. Здесь нет уровней. Число снарядов и число рядов вы настраиваете сами.

3 - Трудная игра. Уровни те же, что и в простой игре, но для перехода на следующий уровень мало просто победить в бою, надо победить убедительно.

При трудной игре ваши результаты сохраняются автоматически. При входе в трудную игру вы должны ввести свое имя и пароль. Пароль хорош тем, что никто другой не сможет войти в игру под вашим именем и таким образом воспользоваться вашими достижениями. Плох пароль тем, что если вы его забудете, то вам придется начинать игру с 1 уровня и под другим именем.

Отчаявшись пройти уровень в трудной игре, вы можете перейти на тот же уровень в простой.

Когда я программировал эту игру, я не заботился о красоте и занимательности. Это - хотя и большая, но всего лишь учебная задача на использование объектов. Улучшайте игру и ее правила, как вам заблагорассудится.



Советы по программированию. Первые две игры не требуют объектов, третья требует. Правда, в морском бое вполне

можно создать массив объектов "Корабль". Хоть корабли и неподвижны, но так, возможно, будет проще анализировать игру. Не знаю, не пробовал.

В первых двух играх информацию о ходе игры удобно представлять в двумерных массивах, в третьей игре это не нужно.

В первых двух играх программы получатся проще, если после каждого хода человека компьютер будет анализировать игровое поле с самого начала, так, как если бы он видел его впервые и не знал, какой из ходов человека был последним. Очень трудно в крестиках-ноликах заставить компьютер размышлять о том, почему человек поставил нолик именно в данную клетку и что за каверзу он там задумал. Гораздо легче просмотреть все поле и поискать там четверки с одним свободным концом и тройки с двумя.

В танковой битве объектами будут:

- Танк под управлением компьютера
- Танк под управлением человека
- Снаряд

Для простоты запретите танку стрелять, пока его снаряд от предыдущего выстрела все еще летит. Тогда размер массива снарядов не превысит размер массива танков.

Любая из этих задач достаточно сложна и потребует многих дней напряженной работы. В книге таких сложных задач я не программировал. Если вам кажется, что вы не сможете удовлетворить всем перечисленным в задании требованиям, потому что «мы этого не проходили», то я вам заявляю – все, что нужно, мы проходили! Нужно только как следует подумать над самым важным: над представлением информации об игре и над алгоритмами, по которым компьютер будет делать ходы.

У вас есть все шансы сделать так, что любая программа может приобрести популярность во всем мире, так как при выполнении всех требований задания и при некоторой фантазии с вашей стороны у нее будет достаточно привлекательный вид.

Желаю успеха!

Приложение 1. Необходимые сведения о компьютере и программе

В этой части излагаются на уровне "для начигающего" следующие вещи:

- Что такое программа. Что такое цикл, ветвление, процедура, и какая от них польза.
- Принцип действия компьютера и его устройств: оперативной памяти, принтера, винчестера и других.
- Взаимодействие устройств во время работы компьютера.
- Принципы кодирования информации в разных устройствах компьютера.

Если что-то из вышеупомянутого вам знакомо, вы можете это пропустить или прочитать "по диагонали".

В этой части мы *не будем* программировать на Visual Basic. А будем знакомиться с перечисленными выше вещами, без которых сознательное программирование невозможно.

Если вы ничего не знаете о компьютере и программе, начинайте читать книгу именно с этой части..

Глава 24. Первое представление о компьютере и программе

24.1. Что такое компьютер. Первое представление о программе.

Зададимся целью ответить на вопрос: Почему компьютер такой умный, откуда в нем умение делать такие удивительные вещи, как, например, играть в шахматы на уровне гроссмейстера, разговаривать человеческим голосом, предсказывать погоду и т.д. и т.п.? Ну что же ответить на этот вопрос? Прежде всего, нужно сказать, что когда-то компьютеры ничего такого делать не умели. И их приходилось учить. Как учат компьютер? Примерно так же, как учат людей, рассказывая им, как делать то-то и то-то. Пусть, например, вы живете на 17 этаже многоэтажного дома и к вам в гости приехал человек, никогда не бывавший в городе. Предположим, вы хотите научить его спускаться во двор на прогулку. Для этого вы даете ему такую инструкцию, состоящую из шести команд:

1. Выйти из квартиры
2. Подойти к двери лифта
3. Нажать на кнопку
4. Когда дверь откроется, войти
5. Нажать на кнопку с цифрой 1
6. Когда лифт спустится и дверь откроется, выйти во двор

Если ваш гость умеет ходить и нажимать на кнопки, то помня эту инструкцию, он отныне сможет самостоятельно спускаться во двор.

А как же научить сделать что-нибудь не человека, а компьютер? Например, вы хотите, чтобы компьютер нарисовал на экране монитора синюю тележку. Для этого вы даете ему на специальном, понятном для него языке (например, на Visual Basic) инструкцию примерно такого содержания:

1. Нарисовать в таком-то месте экрана одно колесо.
2. Нарисовать в таком-то месте экрана другое колесо.
3. Нарисовать в таком-то месте экрана корпус тележки.
4. Покрасить корпус в синий цвет.

Если компьютер умеет рисовать колеса, корпуса и красить их, то он поймет эту инструкцию и выполнит ее, в результате чего тележка будет нарисована. Если не умеет, то ему нужна инструкция, как рисовать колеса, корпуса и т.д.

Инструкция для компьютера по выполнению задания, написанная на специальном, предназначенном для него языке, называется *программой*,

если же она написана на обычном русском или другом человеческом языке в расчете на то, чтобы ее понял не компьютер, а человек, то она называется **алгоритмом**. Таким образом, мы только что написали алгоритм из четырех **команд**[*].

Поскольку у многих компьютеров нет ушей-микрофона (а если и есть, то компьютер неважно разбирает устную речь), программу вы ему не рассказываете вслух, а печатаете ее текст на клавиатуре (по-другому говоря - **вводите** с клавиатуры), откуда она тут же сама собой попадает внутрь компьютера. Отныне компьютер по первому вашему приказу сможет эту тележку рисовать.

Программа для рисования тележки очень простая и короткая. Если же вы хотите научить ваш компьютер делать что-нибудь более сложное, например, играть в шашки, то программу для этого должны будете придумать тоже, конечно, очень сложную и длинную. В этой программе будут встречаться команды такого примерно смысла: если противник сходил так-то, ходи так-то; если твоя шашка попала на последнюю горизонталь, обращай ее в дамку; если шашку противника можно брать, то бери и т.д. Как только вы напишете такую программу и введете ее в компьютер, он сразу же сможет играть в шашки, причем ровно настолько хорошо, насколько хороша ваша программа.

Итак, вы должны запомнить, что

[*] Имейте в виду, что я дал частное определение программы и алгоритма. В общем случае они определяются, как набор правил для получения нужного результата.

для того, чтобы компьютер хоть что-нибудь умел, он должен иметь внутри себя программу этого умения

И наоборот, если компьютер что-нибудь умеет, это значит, что кто-то когда-то придумал программу этого умения и ввел ее в компьютер. Следовательно, если ваш компьютер умеет играть в игру «Quake», это значит, что внутри него находится программа этой игры, которую кто-то туда ввел. Разучится ваш компьютер играть в «Quake» только тогда, когда вы удалите программу этой игры из компьютера (или нечаянно, или чтобы освободить в компьютере место для других программ).

Таким образом, мы можем определить *компьютер*, как устройство, предназначенное для выполнения широкого круга заданий и вообще для обработки самой разной информации по программе.

Вернемся к игре в шашки. Вот, например, ваш компьютер в шашки играть умеет. Как теперь научить играть в шашки другие компьютеры? Можно, конечно, в каждый компьютер ввести упомянутую программу с клавиатуры. Но это долго и утомительно, да и опечаток понаделаешь. Есть способы быстро и безошибочно переносить программы с одного компьютера на другой. Самый популярный из них, но устаревающий - использование **дискеты** - маленькой круглой покрытой магнитным веществом пластиковой пластинки в квадратном пластмассовом или бумажном футляре, при помощи которой программы переносятся с одного компьютера на другой точно так же, как при помощи магнитофонной кассеты с одного магнитофона на другой переносятся песни. Эпоха дискет кончается, через несколько лет они будут вытеснены гораздо более быстрыми и вместительными лазерными дисками CD-R, CD-RW и другими подобными.

Когда новенький компьютер выходит с завода, он почти ничего не умеет. Покупатель этого компьютера, чтобы научить его тому, что ему нужно, покупает дискеты или компакт-диски с программами нужных ему умений и переписывает с них эти программы в свой компьютер. Если нужная программа не существует в природе или просто диск нигде достать не удалось, то программу приходится придумывать самому и вводить с клавиатуры.

Распространен еще один путь, при помощи которого программы могут попасть в ваш компьютер: Если ваш компьютер связан линиями связи с другими компьютерами (напимер, при помощи так называемого **модема** или другими способами), то вы можете «перекачивать» программы с других компьютеров на ваш по линии связи.

24.2. Как человек общается с компьютером

Как я уже говорил, информацию, которую человек хочет сообщить компьютеру, он обычно вводит с клавиатуры, дискеты, компакт-диска или по линии связи. Есть и другие способы ввода информации, но о них позже.

Если же, наоборот, компьютер хочет сообщить человеку какую-то информацию, то он обычно показывает ее на экране монитора. Такой информацией могут быть числа, слова, тексты, картинки, мультипликация, видео. По желанию человека компьютер может печатать информацию на бумаге при помощи печатающего устройства, которое называется **принтером**. Кроме этого компьютер может исполнять музыкальные мелодии и даже разговаривать. Все это он умеет делать, напоминаем, не сам по себе, от рождения, а только по написанным человеком программам.

Что же обычно делает человек, сидя за компьютером? Все зависит от того, что ему от компьютера нужно. А нужно ему в большинстве случаев вот что:

- Переписать программу с дискеты в компьютер или наоборот - с компьютера на дискету. В этом случае человек просто вставляет дискету в компьютер и нажимает на клавиатуре несколько определенных клавиш.
- Ввести придуманную им программу в компьютер с клавиатуры. В этом случае человеку достаточно набрать весь текст программы на клавиатуре.
- Выполнить программу, уже имеющуюся внутри компьютера. Конечно, это нужно человеку чаще всего. Внутри компьютера (если только он не вчера куплен) обычно уже имеется очень много программ, поэтому человек сначала выбирает, какая программа ему нужна. Так, если ему нужна программа, показывающая мультфильмы, то он нажатием нескольких клавиш на клавиатуре выбирает именно ее и приказывает компьютеру ее выполнить (другими словами - **запускает программу** на выполнение). Компьютер выполняет программу, послушно делая все то, что в программе приказано, в результате чего на экране идет мультик.
- Отвечать на вопросы компьютера. Это происходит только тогда, когда в программе содержится команда компьютеру задать человеку какой-нибудь вопрос. Как видите, без программы компьютер не только ничего не делает, но даже и вопросов не задает. Обычно программа приказывает задать такой вопрос, без ответа на который компьютер не может дальше выполнять задание, данное ему человеком. Например, если компьютер вычисляет траекторию полета к Марсу, то где-то в начале счета он может задать вам вопрос, который вы увидите на мониторе: "Каков стартовый вес ракеты в тоннах?" В ответ вам нужно набрать на клавиатуре правильное число, скажем 2500. Если же вы с компьютером играете, то вам часто приходится отвечать на вопросы типа "Будете ли вы продолжать игру?" и т.п.
- Очень часто в процессе работы с компьютером вам приходится отдавать ему приказы. Например, если вы играете с компьютером в воздушный бой, то нажатием на одну какую-то клавишу вы приказываете самолету выпустить ракету, нажатием на другую - совершить посадку и т.п.

Будем называть человека, профессия которого состоит главным образом в написании программ, **программистом**, а человека, который в основном сидит за компьютером и пользуется готовыми программами, - **пользователем**.

Глава 25. Программа и программирование

25.1. Список команд. Командный и программный режимы

А теперь подробнее рассмотрим, что такое программа. Чтобы лучше это понять, давайте на время забудем о компьютерах. Предположим, в вашем распоряжении находится не компьютер, а настоящий робот. Робот этот умеет понимать и выполнять команды из следующего списка (только их и никаких других):

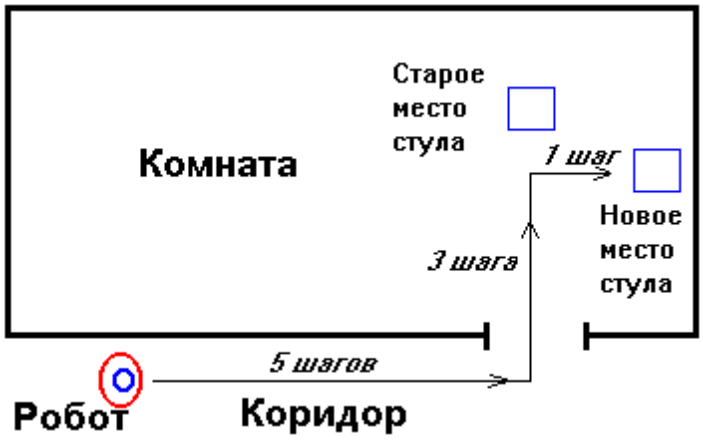
Список команд робота:

ШАГ ВПЕРЕД
НАЛЕВО
НАПРАВО
ВОЗЬМИ ПРЕДМЕТ
ОПУСТИ ПРЕДМЕТ
ПОВТОРИ несколько РАЗ выполнение одной из этих команд

Кроме этого, у робота есть две кнопки:

СЛУШАЙ ПРОГРАММУ
ВЫПОЛНЯЙ ПРОГРАММУ

Запомните, что робот не умеет делать ничего, кроме того, что упомянуто в списке его команд.
Пусть ваш робот стоит в коридоре и вам нужно, чтобы он переставил стул в комнате на новое место.



Но в списке команд робота нет такой команды "Переставить стул в комнате".Что же делать? Можно идти рядом с роботом и в нужные моменты времени приказывать ему: ШАГ ВПЕРЕД, ШАГ ВПЕРЕД,..., НАЛЕВО,..., ВОЗЬМИ ПРЕДМЕТ... и так далее. В результате стул будет переставлен. Этот режим управления роботом (как, впрочем, и компьютером) называется **командным режимом**. Однако, совсем не обязательно сопровождать робота на каждом шагу. Пусть вы заранее измерили все необходимые расстояния. Тогда достаточно в тот момент, когда робот находится в исходной позиции, сообщить ему инструкцию по выполнению задания, то есть задать точный порядок его действий, приводящих к перестановке стула, а затем приказать выполнить ее. Конечно, инструкция должна состоять только из команд, которые робот понимает и умеет выполнять. Вы уже знаете, что называется такая инструкция программой. Вот она:

Программа для робота	Пояснения для нас с вами
1. ПОВТОРИ 5 РАЗ ШАГ ВПЕРЕД	Робот идет по коридору до дверей
2. НАЛЕВО	Робот поворачивается лицом к дверям
3. ПОВТОРИ 3 РАЗА ШАГ ВПЕРЕД	Робот подходит к стулу
4. ВОЗЬМИ ПРЕДМЕТ	Робот берет стул
5. НАПРАВО	Робот поворачивается к новому месту стула
6. ШАГ ВПЕРЕД	Робот подносит стул к новому месту
7. ОПУСТИ ПРЕДМЕТ	Робот ставит стул на новое место

Очевидно, работая по этой программе, робот правильно переставит стул.
Итак, если вы решили не сопровождать робота на каждом шагу, а заставить его работать по программе, вы совершаете следующие действия:

Последовательность работы человека с роботом

1. Придумываете программу, что не всегда легко, так как нужно хотя бы знать расположение мебели, количество шагов до дверей и т.п.
2. Подходите к роботу, стоящему в исходном положении, и нажимаете кнопку СЛУШАЙ ПРОГРАММУ
3. Сообщаете ему программу
4. Нажимаете кнопку ВЫПОЛНЯЙ ПРОГРАММУ

После этого робот работает по программе, то есть выполняет одну за другой команды, из которых составлена программа, в том порядке, в котором он их услышал, в результате чего задание оказывается выполненым, а вы, пока программа выполняется, можете и отдохнуть, чего не могли позволить себе в командном режиме .

Этот режим управления роботом называется **программным режимом.**

25.2. Что важно знать о программе

Чем хороша программа? Ее великое значение в том, что она заставляет робота делать вещи гораздо более сложные, чем те, которые перечислены в списке его команд. По программе робот делает то, что без программы делать не умеет.

Спрашивается, можно ли написать программу для гораздо более сложной задачи, например для перестановки всей мебели на этаже? Разумеется, можно, только программа для этого будет достаточно длинной.

Что требует от нас программа? Она требует абсолютной точности при ее составлении. Если, например, мы в первой команде самую чуточку ошибемся и скажем ПОВТОРИ 6 РАЗ ШАГ ВПЕРЕД (вместо 5 раз), робот проскочит мимо двери, на второй команде повернется и, добросовестно выполняя программу, на третьей команде проломит стену. Если мы перепутаем местами команды 6 и 7, то робот сначала поставит стул, а потом об него же и споткнется.

Запомните, что после того, как вы нажали кнопку ВЫПОЛНЯЙ ПРОГРАММУ и робот начал ее выполнять, вы не можете программу изменить, пока он не закончил работу. Даже если вы увидите, что робот по вашей программе делает что-то не то, бесполезно кричать на него, хватать его за руку и т.п. Он на вас не обратит внимания. Максимум, что вы можете, это - подбежать к нему и выключить его. После этого вы должны отвести его в исходное положение и только затем можете сообщить ему измененную программу.

Сообщать роботу команды мы обязаны только теми словами, которые приведены в списке команд, потому что робот понимает только их. Если мы вместо команды ВОЗЬМИ ПРЕДМЕТ дадим команду БЕРИ ПРЕДМЕТ, робот нас не поймет и команду не выполнит.

Задание 133: Напишите программу, по которой робот сходит в комнату за стулом и вернется с ним в коридор в исходное положение.

25.3. Понятие о процедуре. Может ли робот поумнеть?

Умен ли наш робот? Судя по его реакции на ошибки в программе, весьма глуп. Умный робот не стал бы проламывать стенку. Однако, наш робот не виноват в своей глупости. Ведь его умственные возможности исчерпываются списком его команд. А список этот очень бедный. Чем он беден?

*Первое***.** В этом списке нет сложных команд, таких как "Наведи порядок в комнате", "Перенеси мебель к другой стенке" и даже такой сравнительно простой, как "Переставь стул". Давайте исправим это положение. Пусть мы имеем возможность дополнять список команд робота нужными нам командами. Для этого робот должен уметь запоминать программы. Например, вы хотите, чтобы робот выполнял неизвестную ему команду на перестановку стула. Для этого вы придумываете программу перестановки стула (мы ее уже придумали в 25.1), затем придумываете, как будет звучать сама новая команда, например ПЕРЕСТАВЬ СТУЛ, и наконец сообщаете роботу программу, приказываете навсегда ее запомнить и говорите ему, что отныне он должен ее выполнять по команде ПЕРЕСТАВЬ СТУЛ. Такая программа называется **процедурой**, а новая команда ПЕРЕСТАВЬ СТУЛ – **обращением к процедуре** или **вызовом процедуры**.

Итак, мы дополнили список команд робота новой командой. Можно ли считать, что робот поумнел? Конечно. Но не очень. И вот почему. Пусть стул находится от дверей не в трех шагах, а в двух. Тогда наш робот, выполняя процедуру ПЕРЕСТАВЬ СТУЛ, как она написана в 25.1, споткнется об него, а это, конечно, не говорит о его уме. Чтобы переставить стул, где бы он ни был в комнате, робот должен сначала его найти, но команды на поиск нет в списке его команд, а составить из команд этого списка процедуру поиска невозможно. Мы начинаем видеть, чем еще беден список команд робота:

*Второе***.** Он беден не только количеством команд, но и их содержанием. Все его команды касаются только ходьбы и переноса предметов. Фактически, наш робот очень мало умеет, он может только бездумно расхаживать и таскать с места на место мебель. Он не может включить телевизор, так как не имеет хотя бы команды нажатия на кнопку. Он не может сходить в магазин, так как не умеет считать деньги. Он не может поздороваться с вами, так как не имеет команды что-нибудь произнести. Можете ли вы что-нибудь с этим поделать? Ничего не можете, потому что таким его сделали на заводе. Вам нужен робот с гораздо более разнообразными командами. Кроме этого, хорошо было бы иметь команды для очень мелких движений, например "согнуть средний палец на правой руке". Тогда из таких команд можно было бы составить процедуры вроде "пожми руку" или "нажми кнопку" или "включи телевизор". Но воображаемый робот с такими умениями был бы очень сложным и очень дорого бы стоил.

Вопрос: Что сделает наш поумневший робот, получив, находясь в исходном положении, такую бессмысленную программу:

Программа для робота	Пояснения для нас с вами
1.ПЕРЕСТАВЬ СТУЛ	Обращение к процедуре ПЕРЕСТАВЬ СТУЛ
2.НАЛЕВО	Робот поворачивается налево
3.ПЕРЕСТАВЬ СТУЛ	Обращение к процедуре ПЕРЕСТАВЬ СТУЛ

Ответ: Выполняя первую команду (обращение к процедуре ПЕРЕСТАВЬ СТУЛ), робот благополучно переставит стул. Выполняя вторую команду, он повернется лицом к дальней стенке. Выполняя третью команду (обращение к процедуре ПЕРЕСТАВЬ СТУЛ), он на первой же команде этой процедуры (ПОВТОРИ 5 РАЗ ШАГ ВПЕРЕД) врежется в дальнюю стенку.

25.4. Программа для компьютера на машинном языке

Теперь, когда вы понимаете, какую важную роль играет список команд, которые может выполнять робот, настало время вернуться обратно к компьютерам. Программа для компьютера тоже состоит из отдельных команд. Я уже говорил, что человек, который пишет программу для компьютера, называется программистом. Естественно, когда программист пишет программу, ему совершенно необходимо знать список команд, которые может выполнять компьютер. Мы еще поговорим подробнее об этом списке. Но сначала подумаем, а что вообще может компьютер. Вспомним все его умения, о которых мы слышали по телевизору или читали в журналах. Большинство из них сводится в конце концов к тому, что компьютер думает, а затем что-то изображает на экране монитора (числа, тексты, картинки, мультики) или же исполняет какую-нибудь музыкальную мелодию или обменивается информацией с дисками. Программы для всех этих умений состоят из команд компьютера.

Взглянем на список команд новенького компьютера, только что покинувшего заводские стены. Внутри него нет почти никаких программ, поэтому умеет он выполнять только команды, заложенные в него на заводе. Каждая из этих команд заставляет компьютер выполнить какое-то одно *простейшее очень маленькое действие*, по своей незначительности подобное сгибанию пальца у робота. Отдавать эти команды компьютеру нужно на специальном языке, понятном компьютеру - **машинном языке**. Поскольку изучение машинного языка нам сейчас не нужно, я приведу только смысл некоторых мельчайших задач, выполняемых командами машинного языка (на русском языке).

Примеры задач, выполняемых командами машинного языка:

Сложить два числа.
Определить, какое из двух чисел больше.
Следующие задачи уже слишком трудны для одной команды машинного языка и под силу только совокупности таких команд:
Изобразить на экране в заданном месте светящуюся точку заданного цвета.
Изобразить на экране заданную букву или цифру.
Включить звук заданной высоты.
Выключить звук.
Запомнить, какую клавишу нажал человек на клавиатуре.

В машинном языке еще много команд, и все они такие же "мелкие". Спрашивается, как же при помощи таких слабеньких команд заставить компьютер сделать хоть что-нибудь путное, скажем, написать слово "ЭВМ" или нарисовать кружочек? Я думаю, вы уже догадались, что нужно сделать - нужно написать программу и сделать ее процедурой. Вот, например, алгоритм программы, изображающей на экране слово "ЭВМ":

1. Изобразить на экране букву "Э"
2. Изобразить на экране букву "В"
3. Изобразить на экране букву "М"

А вот алгоритм программы, вычисляющей выражение (5-7)/(10+40):

1. Вычти 7 из 5
2. Прибавь 40 к 10
3. Раздели первый результат на второй
4. Покажи результат деления на экране монитора

Это ничего, что результат получился отрицательный и дробный. Компьютеры непринужденно справляются с такими числами.

А как же нарисовать кружочек, если компьютер может нарисовать только точку? Если вы посмотрите на экран монитора в увеличительное стекло, то заметите, что изображение любого предмета состоит из маленьких светящихся точек (**пикселов**), которые расположены так близко друг к другу, что сливаются в единое изображение. Примерно то же самое вы видите через лупу на фотографии в газете. Вполне можно написать программу, которая рисует рядышком одну за другой множество точек так, чтобы они образовали окружность. Рисунок, поясняющий принцип получения изображения на экране, приведен 26.4.

25.5. Языки программирования

В чем недостаток команд машинного языка? В том, что действия, вызываемые этими командами, очень мелки. Поэтому программа выполнения даже очень простого задания будет состоять из большого числа команд. Это все равно, что строить дом не из кирпичей, а из косточек домино, - построить можно, но слишком долго и утомительно (зато орнамент из кирпичей на этом доме получится плохой, грубый, а из косточек домино – гораздо более богатый и тонкий).

Поскольку этот недостаток машинного языка был давным-давно понятен всем программистам, то они составили из команд машинного языка процедуры для выполнения наиболее популярных маленьких заданий, таких как:

- Нарисовать кружочек заданного размера в заданном месте экрана
- Нарисовать прямоугольник заданного размера и формы в заданном месте экрана
- Нарисовать отрезок прямой
- Покрасить заданным цветом определенную область экрана
- Воспроизвести мелодию по заданным нотам
- Написать на экране заданное слово, заданный текст
- Запомнить слово или текст, введенные с клавиатуры
- Вычислить математическую формулу

Как видите, действия, вызываемые этими процедурами, гораздо более крупные, чем у команд машинного языка. Поэтому эти процедуры более удобны для написания программ, хотя бы для таких, как программа, рисующая синюю тележку с надписью "Игрушки". Для ее написания достаточно согласиться с тем, что колесо - это кружочек, а корпус - прямоугольник.

Конечно, хотелось бы иметь все подобные процедуры внутри компьютера. Поэтому давным-давно существуют дискеты и компакт-диски, на которых записаны целые "сборники" таких процедур. И каждый желающий может взять диск, переписать его содержимое в компьютер и пользоваться им.

Процедуры на таком диске записаны не разобщенно, а в комплексе, как составные части особой большой программы. Если мы перепишем эту большую программу в компьютер и запустим ее на выполнение, то она позволит человеку, во-первых, писать по определенным правилам собственные программы из упомянутых процедур, а во-вторых, сделает этот процесс удобным, то есть будет обнаруживать многие ошибки в ваших программах, позволит быстро запускать их на выполнение, исправлять, переписывать на диск и т.д.

Называют такую комплексную программу сложно и по-разному, например, "Среда и компилятор языка программирования высокого уровня C++". Основное для нас в этом названии - понятие "язык программирования" или будем говорить проще - "язык". Итак, вывод: Чтобы компьютер позволил человеку программировать, нужна специальная программа, которая называется *язык программирования*. Но если язык, то какой? Попробуем вникнуть. У людей есть русский, английский, китайский языки. Что такое любой из этих языков общения людей? Грубо говоря, это набор букв, слов, знаков препинания и правил, по которым все эти элементы нужно выстроить в цепочку, чтобы получить правильное предложение. Язык программирования – примерно то же самое. Важнейшая часть языка программирования – набор правил, по которым различные объекты (в том числе и обращения к упомянутым процедурам*) нужно выстроить в цепочку, чтобы получить правильную программу.

Вот некоторые наиболее популярные языки программирования:

Visual Basic	Вижуал Бэйсик	Замечательный язык как для начинающих, так и для профессиональных программистов
Object Pascal (Delphi)	Обджект Паскаль в среде Дельфи	Универсальный язык, позволяющий прекрасно программировать самые разные задачи
Visual C++	Вижуал Си++	Сложный, мощный язык для профессиональных программистов
Java	Ява (Джава)	Мощный модный язык, применяемый пока в основном в Интернете
Assembler	Ассемблер	Сложный, мощный язык, с самыми мелкими командами, близкими к командам машинного языка
Logo	Лого	Язык, рассчитанный на детей, позволяющий просто и занимательно рисовать картинки и программировать простейшие игры
LISP, Prolog	Лисп, Пролог	Языки для создания искусственного интеллекта, роботов

Языков программирования, как и человеческих языков, придумано много. Зачем? Причина - в разнообразии потребностей программистов, в разных уровнях их квалификации и во многом другом. Так, начинающим вряд ли стоит предлагать Ассемблер, а профессионалу не нужен Лого. Часто разные языки ориентированы на разные предметные области. Например, язык Пролог позволяет удобно описывать логические взаимосвязи в окружающем нас мире, Лого позволяет удобно рисовать фигуры и снабжен для этого соответствующим набором процедур, а вот решать сложные математические задачи с его помощью лучше и не пытаться.

Программистам пока еще не удалось создать язык, удовлетворяющий всех, да и неизвестно, возможно ли вообще его создать, и надо ли.

Во всех человеческих языках есть слова «ходить», «есть», «спать», обозначающие понятия, общие для всех людей. Точно так же большинство языков программирования позволяет выполнять общепринятые процедуры, такие, например, как вывод информации на экран. Только записываются обращения к этим процедурам в разных языках по-разному. Прикажем, например, компьютеру к трем прибавить два и результат показать на экране монитора. Вот как эта процедура вызывается на языке Лого:

```
покажи  3 + 2
```

А вот как она вызывается на Паскале:

```
Write (3+2)
```

А вот как на Visual Basic:

```
Print 3 + 2
```

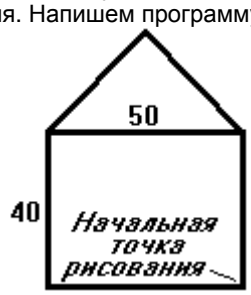
В языках программирования приказы, которые отдают на данном языке, называют не только обращениями к процедурам, но и **командами** (язык Лого и др.), и **операторами** (языки Бэйсик, Паскаль и др.). Между понятиями «обращение к процедуре» и «оператор» существует значительная разница, о которой вы узнаете позже, сейчас вам важно знать только одно – команда Лого, обращение к процедуре и оператор являются довольно сложными приказами. Не нужно их путать с командами машин-

* Строго говоря, процедуры не являются составной частью языка, однако, вы должны знать, что держа в руках компакт-диск с надписью “Visual Basic” или «C++» или какой-либо другой язык, вы держите в руках целый комплекс программ, который содержит и большое количество этих самых процедур и средства для удобной разработки ваших программ с их использованием.

ного языка, так как они гораздо «крупнее». Так команда языка Лого покажи 3 + 2 фактически является обращением к процедуре из нескольких команд машинного языка, которые сначала приказывают компьютеру вычислить сумму, а потом показать ее на экране. Нет команд более мелких, чем команды машинного языка, поэтому любая команда, оператор или процедура на любом другом языке (кроме Ассемблера) сводится в конце концов к выполнению набора команд машинного языка.

25.6. Пример настоящей программы для компьютера на языке Лого

Давайте для иллюстрации напишем настоящую программу на настоящем языке программирования. Для этого выберем язык Лого. Он предназначен в основном для рисования. Напишем программу для рисования домика, вот такого:



Начнем с того, что у нас в руках находится дискета с языком Лого. Вставим ее в компьютер. После нескольких нажатий на клавиши посредине экрана возникает вот такая маленькая черепашка:

С этого момента компьютер готов принимать нашу программу и выполнять ее. Занимательность и простота работы с Лого заключается в том, что многие его команды являются командами для черепашки нарисовать на экране те или иные разноцветные линии, что-нибудь покрасить и т.п. Передвигается черепашка по экрану маленькими шагами. Размер экрана по горизонтали и вертикали - несколько сотен шагов.

Из всего длинного списка команд Лого нам для рисования домика понадобятся только две. Приведем примеры их записи с пояснением:

ВПЕРЕД 28 — По этой команде черепашка продвинется вперед на 28 шагов, оставляя за собой тонкий след, то есть фактически она нарисует отрезок прямой длиной в 28 шагов.

НАЛЕВО 60 — По этой команде черепашка повернется на месте налево на 60 градусов.

А теперь введем в компьютер программу:

Программа	Пояснения
ВПЕРЕД 40	Черепашка идет вверх и рисует правую стенку дома
НАЛЕВО 90	Собираемся рисовать не крышу, а потолок
ВПЕРЕД 50	Черепашка рисует потолок
НАЛЕВО 90	Собираемся рисовать левую стенку дома
ВПЕРЕД 40	Черепашка рисует левую стенку дома
НАЛЕВО 90	Собираемся рисовать пол
ВПЕРЕД 50	Черепашка рисует пол
НАЛЕВО 90	Готовимся добраться до крыши по правой стене
ВПЕРЕД 40	Черепашка забирается на крышу по правой стене
НАЛЕВО 45	Собираемся рисовать правый скат крыши
ВПЕРЕД 36	Черепашка рисует правый скат крыши
НАЛЕВО 90	Собираемся рисовать левый скат крыши
ВПЕРЕД 36	Черепашка рисует левый скат крыши

После этого по нажатии на клавишу клавиатуры черепашка выполняет программу, в результате чего на экране возникает домик.

Как и программа для нашего воображаемого робота, любая программа для компьютера требует абсолютной точности записи. Нельзя допускать орфографических ошибок - НАЛЕВА, нельзя записывать команды по-другому - ВЛЕВО. Компьютер в этом случае просто откажется выполнять программу. Но, что самое худшее, если мы, соблюдая эти формальные правила, все же по невнимательности допустим смысловую ошибку в программе, компьютер программу выполнять не откажется и, выполнив ее, получит неправильный результат. Например, если в программе пятую сверху команду (для рисования левой стены) мы запишем так - ВПЕРЕД 60 (а не ВПЕРЕД 40), то домик будет выглядеть так:

Так же, как и в случае с роботом, если мы в процессе выполнения программы увидим, что черепашка рисует что-то не то, у нас не будет возможности на ходу исправить программу. Нам или придется ждать, когда она дорисует все до конца или нажатием на клавиши все стереть с экрана и привести черепашку в исходное состояние. После этого программу можно исправлять.

Это я говорил о программном режиме. Лого допускает и командный режим, когда черепашка выполняет команду сразу же, как получит ее с клавиатуры.

Может ли черепашка поумнеть? Да. Объясните черепашке, что составленная программа есть процедура с именем ДОМИК – и отныне вам достаточно будет отдать команду ДОМИК – и черепашка его нарисует.

25.7. Последовательность работы программиста на компьютере

Запишем, в каком порядке проходит работа программиста на компьютере. Она практически копирует порядок работы с воображаемым роботом (см.25.1):

0. Сначала программист получает задачу для компьютера, например, нарисовать домик или вычислить траекторию полета на Марс.
1. Затем он думает, как из команд (операторов), которые ему предоставляет в распоряжение язык программирования, написать программу. На обдумывание и написание программы уходит от нескольких минут до нескольких лет в зависимости от сложности задачи. Почти всегда программист, чтобы не запутаться, пишет сначала для себя алгоритм программы, а потом уже саму программу.
2. Наконец программа написана. Теперь программист включает компьютер и нажатием нескольких клавиш приказывает ему приготовиться к приему программы.
3. Программист набирает всю программу (если она, конечно, не очень длинная) от первой до последней буквы на клавиатуре. При этом программа автоматически по проводу, соединяющему клавиатуру с компьютером, поступает в компьютер и запоминается в его памяти. Программа попала в память компьютера, но это не значит, что компьютер программу “узнал” и “понял”, как узнает и понимает человек. Человек, прочитав какую-нибудь программу, воспринимает ее целиком и хотя бы примерно представляет ее назначение, структуру и т.п. Компьютер же никогда программу у себя из памяти целиком не читает и никогда не понимает ее общего смысла и назначения (но это не значит, что он не сможет ее выполнить).
4. Программист нажатием на пару клавиш приказывает компьютеру выполнить программу. Компьютер после некоторой подготовки (куда входит, как главный элемент, перевод программы, написанной на языке программирования, на машинный язык) читает у себя в памяти первую команду программы и выполняет ее, затем читает вторую команду и выполняет, затем третью и т.д. до тех пор, пока не дойдет до конца программы. В результате, если программа составлена правильно, на экране оказывается нарисованным домик или на принтере печатаются результаты расчета траектории полета к Марсу. Но и выполнив правильную программу, компьютер не понял ее смысла и не поумнел.
5. Все программисты допускают в программах ошибки. В результате почти никогда компьютер по только-что написанной программе не делает того, что нужно. Увидев на экране косой домик, программист начинает чесать в затылке и искать в программе команду, виновную в печальном исходе. Найдя ошибочную команду, он исправляет программу и вновь запускает ее на выполнение. Однако результаты снова обычно бывают плачевные. Это происходит потому, что часто программа содержит сразу несколько ошибок. Исправив очередную ошибку, программист снова запускает программу и т.д. Этот захватывающий процесс называется **отладкой**. Отладка заканчивается, когда программист удовлетворен результатом работы программы. Хотя, если программа сложная, это не всегда означает, что в ней ошибок больше нет!. Просто некоторые ошибки влияют на результат почти незаметно, как, например, ошибка ВПЕРЕД 37 вместо ВПЕРЕД 36 в последней команде из программы предыдущего раздела.

25.8. Основные приемы программирования

Сведение сложного к простому. Цикл. Итак, чтобы заставить компьютер что-то сделать, нужно написать программу. И тут невольно возникает вопрос - неужели возможно, записывая одна за другой довольно примитивные команды языка программирования, написать программы для всех тех замечательных умений компьютера, некоторые из которых я уже упоминал? Возьмем, например, игру в воздушный бой. Ведь самолетик по экрану должен двигаться. Но в списках команд большинства языков программирования нет команды движения. Или возьмем траекторию полета к Марсу. Для ее вычисления нужно решать сложнейшие дифференциальные уравнения высшей математики. Но процедуры языков Фортран, Бэйсик, Паскаль, которые используются для этих целей, не могут ничего, что выходит за рамки школьного курса. И непонятно, в конце концов, как научить компьютер разговаривать, если в нашем распоряжении только команда извлечения из компьютера простого звука

заданной высоты.

На все эти вопросы ответ один: если вы хорошо разбираетесь в поставленной задаче, то вы обязательно сможете **разложить ее на много маленьких задач**, каждая из которых вполне поддается программированию, после чего из многих получившихся программок можно собрать одну большую программу, которая и решает задачу. Это все равно, как разгромить войско противника по частям. Действительно, когда у вас была задача нарисовать тележку, вы разбили ее на задачи рисования колес и корпуса. Если бы и эти задачи вам показались сложными, вы бы разбили их дальше и т.д.

Разберем для иллюстрации несколько основных идей и приемов сведения сложного к простому.

Вот воздушный бой. Здесь нужно задаться вопросом - а что такое движение? Рассмотрим иллюзию движения, возникающую на экране кинотеатра. Если вы держали в руках кинопленку фильма, изображающего, скажем, движение автомобиля, то должны были обратить внимание, что она представляет собой последовательность неподвижных слайдов (кадров), на каждом следующем из которых автомобиль находится чуть-чуть в другом месте, чем на предыдущем. Показывая эти кадры один за другим с большой скоростью, мы создаем иллюзию движения автомобиля. Точно так же поступают с созданием движения на экране компьютера. Запишем алгоритм движения по экрану слева направо обыкновенного кружочка:

1.	Зададим (в уме компьютера) позицию кружочка в левой части экрана. Перейдем к команде 2.
2.	Нарисуем в заданной позиции кружочек. Перейдем к команде 3.
3.	Сотрем его. Перейдем к команде 4.
4.	Изменим (в уме компьютера) позицию кружочка на миллиметр правее. Перейдем к команде 5.
5.	Перейдем к команде 2.

Каждая из приведенных команд алгоритма легко программируется на большинстве языков. В том числе и на Visual Basic. Кстати, любой компьютер, выполнив очередную команду, автоматически переходит к выполнению следующей (так что нам не обязательно было писать, скажем, во второй команде "Перейдем к команде 3."). Однако, если мы захотим, то можем заставить компьютер изменить этот порядок, что мы и сделали в команде 5, благодаря чему компьютер стал многократно выполнять последовательность команд 2-3-4-5. Такая многократно выполняемая последовательность называется **циклом**. Цикл - основное средство заставить компьютер сделать много при помощи короткой программы.

В коротком промежутке времени после выполнения команды 2 кружок будет появляться на экране и на команде 3 исчезать, но этого достаточно, чтобы человеческий глаз его заметил. Благодаря команде 4 кружок будет мелькать каждый раз в новом месте, а поскольку смена “кадров” будет очень быстрой, нам будет казаться, что происходит плавное движение кружка.

Теперь перейдем к задаче о траектории. Существует наука - вычислительная математика - которая утверждает, что решение многих самых сложных и страшных математических уравнений можно свести к многократному выполнению четырех действий арифметики, и показывает, как это делать. Грубо говоря, вместо решения одного сложного уравнения она предлагает выполнить пять миллионов простых сложений. Это как раз то, что нужно компьютеру. Пять миллионов сложений он выполнит за доли секунды.

И наконец о том, как научить компьютер разговаривать. Акустикам хорошо известно, что любой звук человеческого голоса можно заменить наложением многих простых звуков различной высоты. Наложением сложным и в разные моменты времени разным, однако вполне поддающимся программированию.

Собственные процедуры.. Большую помощь в упрощении программирования сложных задач оказывает возможность, предоставляемая программисту большинством языков, составлять собственные процедуры из команд и операторов языка. Составляется процедура точно так же, как в 25.1 для робота составлялась процедура ПЕРЕСТАВЬ СТУЛ. Пусть, например, вы хотите, чтобы компьютер изобразил на мониторе три поезда, каждый из которых состоит из одного паровоза и четырех вагонов. Сначала мы пишем программу для рисования вагона (прямоугольник и четыре кружочка). Сейчас мы не будем обсуждать, как правильно расположить на экране друг относительно друга прямоугольник, кружочки, вагоны и поезда. Вам остается поверить, что делается это довольно легко. Вот алгоритм программы из 5 команд для рисования вагона:

1.	Нарисовать прямоугольник.
2.	Нарисовать кружочек.
3.	Нарисовать кружочек.
4.	Нарисовать кружочек.
5.	Нарисовать кружочек.

Для того, чтобы нарисовать четыре вагона, вам понадобилась бы программа из 20 команд. Писать их утомительно, поэтому программисты поступают по-другому. Программу рисования вагона они называют процедурой, придумывают ей имя, скажем, ВАГОН и сообщают его компьютеру. Аналогично процедуре ВАГОН они составляют процедуру ПАРОВОЗ (которую я не буду приводить). Из обращений к этим готовым процедурам они составляют процедуру ПОЕЗД, алгоритм которой будет выглядеть так:

1.	Выполни процедуру ПАРОВОЗ
2.	Выполни процедуру ВАГОН
3.	Выполни процедуру ВАГОН
4.	Выполни процедуру ВАГОН
5.	Выполни процедуру ВАГОН

И наконец, они пишут программу для рисования трех поездов, которая оказывается совсем короткой:

1.	Выполни процедуру ПОЕЗД
2.	Выполни процедуру ПОЕЗД
3.	Выполни процедуру ПОЕЗД

При отсутствии процедур пришлось бы писать программу из нескольких десятков команд. Выигрыш в объеме работы и в понятности записи программы очевиден.

Ветвление (выбор). У начинающего программиста интерес должен вызвать такой вопрос: как компьютер принимает

решения, как он выбирает, какое действие из нескольких возможных нужно выполнить в данный момент? Возьмем тот же воздушный бой. Предположим, при нажатии на клавишу R самолет летит вверх, при нажатии на клавишу V - вниз. Как компьютер чувствует нажатие на клавиши, откуда он знает, что нужно делать при нажатии на каждую из них? Ведь в другой игре, например, "Детский сад", при нажатии на клавишу R дети бегут гулять, а при нажатии на клавишу V - обедать. Естественно, этот выбор делает программа для игры, сам по себе компьютер ничего выбрать не может. В программе игры в воздушный бой заранее пишутся процедуры для полета вверх и для полета вниз, а выбор между ними делает специальная команда выбора, имеющаяся в каждом языке программирования. Вот алгоритм выбора между полетом вверх и вниз:

1. Определи, нажата ли какая-нибудь клавиша.
2. Если не нажата, то переходи к команде 5.
3. Если нажата клавиша R, то выполняй процедуру ВВЕРХ.
4. Если нажата клавиша V, то выполняй процедуру ВНИЗ.
5. Продолжай полет.

Здесь команды 2,3,4 - команды выбора. В общем случае команда выбора содержит условие, от которого зависит, будет ли выполняться какая-нибудь команда или группа команд. Это условие может быть самым разным: нажата или нет любая клавиша, нажата или нет конкретная клавиша, больше ли одно число другого, правда ли, что с клавиатуры введено такое-то слово и т.д.

Напишем для примера примитивный алгоритм, позволяющий имитировать вежливое общение компьютера с человеком при включении компьютера:

1. Покажи на мониторе текст "Здравствуйте, я - компьютер, а вас как зовут?"
2. Жди ответа с клавиатуры.
3. Если на клавиатуре человек набрал "Петя" или "Вася", то покажи на мониторе текст "Рад встретиться со старым другом!", иначе покажи на мониторе текст "Рад познакомиться!"
4. Покажи на мониторе текст "Чем сегодня будем заниматься - программировать или играть?"
5. Жди ответа с клавиатуры.
6. Если . . . . . .
   . . . . . .

Выбор называют **ветвлением** по аналогии с разветвляющимся деревом (когда мы залезаем на дерево, мы время от времени делаем выбор, по какой из нескольких веток лезть дальше).

На этом мы завершим умозрительное рассмотрение основных идей программирования. Конкретное их воплощение на языке Visual Basic вы увидите в книжке. А сейчас пришла пора посмотреть на внутреннее устройство компьютера.

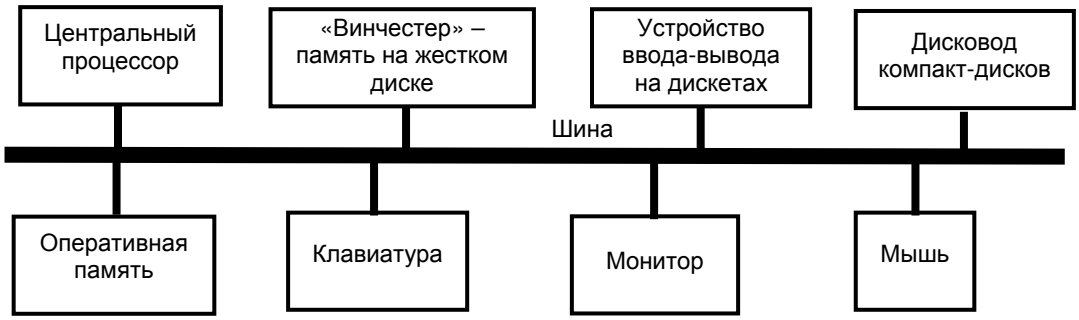
Глава 26. Устройство и работа компьютера

Для того, чтобы водить автомобиль, совсем не обязательно знать, отчего крутятся колеса. Но чтобы быть хорошим водителем, устройство и взаимодействие отдельных частей автомобиля представлять все же необходимо. То же самое можно сказать и о компьютере: можно на нем работать, используя готовые программы, и даже самому программировать, не зная внутреннего устройства компьютера. Однако квалифицированным пользователем и программистом не станешь, не изучив устройство и взаимодействие отдельных его частей.

26.1. Как устроен и работает компьютер

Компьютеры бывают разные. Вы работаете на так называемом **персональном компьютере**. Вы, безусловно, можете показать, где у компьютера монитор (дисплей), клавиатура, мышь. Но не все знают, что большая коробка, та самая, в которой размещены дисководы для дискет и компакт-дисков, называется **системным блоком**.

Чтобы легче понять взаимодействие различных частей персонального компьютера, представим себе его устройство схематически:



Схематическое устройство персонального компьютера

На схеме изображено только восемь самых необходимых устройств, хотя в компьютере их может быть гораздо больше. Внутри системного блока находятся: *центральный процессор* (или просто процессор), *оперативная память* (или просто память), *шина*, а также жесткий диск (*винчестер*) и дисководы для дискет и компакт-дисков. Кроме того, туда входит много вспомогательных электронных схем и устройств.

Во время работы компьютера все устройства обмениваются между собой информацией. Например, программа с дискеты или клавиатуры отправляется в память. Чтобы информация могла попадать из одного устройства в другое, все устройства соединены между собой общей шиной, которая представляет собой ряд электрических проводников, по которым эта информация передается в виде электрических сигналов.

Процессор

Рассмотрим, чем занимается каждое устройство компьютера в отдельности. Прежде всего выделим два устройства, образующие "мозг" компьютера - это процессор и оперативная память. Именно эти два устройства осуществляют главную обработку информации, они решают все задачи, вычисляют траектории, обдумывают шахматные ходы и т.д. Только не надо забывать, что все это они делают, слепо выполняя команды программы, а это значит, что весь интеллект компьютера сосредоточен не в них, а в программе, умен не сам компьютер, а умна находящаяся в нем программа. Поэтому часто вместо того, чтобы сказать "компьютер решил задачу", говорят "программа решила задачу".

Процессор можно назвать "начальником" над остальными устройствами компьютера, так как он во время выполнения программы руководит работой всех устройств. Именно он "понимает смысл" каждой команды программы и выполняет ее сам или приказывает выполнить ее другим устройствам. Однако, процессор почти ничего не помнит. Вернее, его память хороша, но очень мала - он может запомнить всего несколько чисел или букв.

А вот оперативная память специально предназначена для того, чтобы быстро запоминать и быстро вспоминать большие объемы информации, больше ничего она делать не умеет. Короче, процессор и память - как слепой и глухой - по отдельности беспомощны, а вместе вполне способны существовать.

Быстродействие процессора во многом определяется его **тактовой частотой**, которая показывает, как часто «бьется его сердце». У современных компьютеров тактовая частота составляет сотни и тысячи мегагерц (миллионов герц, то есть миллионов тактов в секунду). Когда вы слышите слова «Селерон 800», это значит, что имеется в виду процессор «Селерон» с тактовой частотой 800 мегагерц. За 1 такт компьютер выполняет одну или несколько коротких команд программы.

Порядок обмена информацией между устройствами компьютера

Рассмотрим порядок, в котором обычно обмениваются информацией устройства компьютера во время выполнения программы. Пусть мы только-что придумали программу для перемножения двух чисел, одно из которых находится на дискете, а другое должно вводиться с клавиатуры. Вот алгоритм программы:

1.	Ввести число с дискеты
2.	Ввести число с клавиатуры
3.	Перемножить эти числа
4.	Показать результат на мониторе

Пусть мы придумали эту программу на языке Visual Basic и теперь хотим ее выполнить на компьютере. Для этого Visual Basic должен уже иметься на жестком диске нашего компьютера. Мы включаем компьютер, он несколько секунд готовится к работе, после чего мы нажимаем несколько клавиш. Visual Basic с жесткого диска через шину переписывается (загружается) в память. После этого компьютер дает нам понять, что готов принимать от нас программу на языке Visual Basic, и мы эту программу набираем на клавиатуре.

Все компьютеры могут выполнять программу только тогда, когда она находится в оперативной памяти.

В соответствии с этим требованием наша программа автоматически по мере ввода с клавиатуры отправляется в память и там запоминается. Примерно в это же время мы вставляем в дисковод дискету с одним из двух чисел. Как только вся программа введена, мы нажатием клавиши приказываем компьютеру ее выполнить. И вот что в общих чертах происходит дальше. (Имейте в виду, что следующие два абзаца довольно нудные, в них мы, пожалуй, слишком глубоко забираемся в компьютер. Но без глубин нет вершин.)

Напомним, что на Бэйсике программа состоит из операторов, в нашем случае (допустим для простоты и для соответствия с алгоритмом) - из четырех операторов, хотя на самом деле их будет немного больше. Visual Basic выполняет эти операторы один за другим, по порядку. Вы также знаете, что при выполнении оператор языка программирования заменяется выполнением набора машинных команд. Мы, опять же, чтобы не затемнять изложение подробностями, допустим пока, что в нашей программе каждому оператору соответствует одна команда машинного языка.

Итак, как только программа была запущена на выполнение, процессор прежде всего приказывает памяти послать ему по шине первую команду программы. После того, как эта команда (ввод числа с дискеты) пришла в процессор, он "осознает" ее и отдает приказы устройствам компьютера на ее выполнение в соответствии с тем, как он ее осознал. В нашем случае он отдает приказ дисководу прочесть с дискеты число (пусть это было число 3) и направить его по шине в оперативную память, а оперативной памяти приказывает принять это число и запомнить. Как только число запомнилось, процессор считает команду выполненной и приказывает памяти послать ему вторую команду (которой оказался ввод числа с клавиатуры). Осознав ее, он приказывает компьютеру остановиться и ждать, когда человек введет с клавиатуры какое-нибудь число. Как только человек набрал число на клавиатуре (пусть он набрал -0.25), клавиатура докладывает об этом процессору и тот приказывает ей направить число в память или, возможно, запоминает его сам. После этого он принимает из памяти третью команду (умножение). Внутри процессора имеется арифметическое устройство - своеобразный автоматический карманный калькулятор, способный выполнять четыре действия арифметики. Пусть второе число уже находится в процессоре, тогда он приказывает памяти послать ему по шине первое число, перемножает оба числа, после чего запоминает результат сам или отправляет его в память (предположим, он выбрал память). Наконец, он получает из памяти последнюю команду, согласно которой приказывает памяти же отправить результат (-0.75) на монитор, а тому - принять результат и изобразить его на экране. На этом выполнение программы заканчивается, компьютер останавливается и ждет от человека ввода новой программы или исправления старой.

Итак, мы видим, что работа процессора состоит в том, чтобы считывать из памяти по порядку команды программы, осознавать их, после чего выполнять их самому или приказывать выполнить другим устройствам.

Работа оперативной памяти состоит в том, чтобы хранить программу во время ее выполнения, а также принимать от любых устройств, запоминать и отправлять в любые устройства любую информацию, с которой работает программа. Такая информация, в отличие от программы, называется **данными**. В нашем случае данными являются числа 3 и -0.25 (это **исходные данные** решения задачи), а также -0.75 (это данное является **результатом**). Программа - это предписание того, что нужно делать с исходными данными, чтобы получить результат, а данные - это информация, над которой производит действия программа и которая зачастую в программе не содержатся. Так. в нашей программе нигде не заданы значения перемножаемых чисел. Оба они находятся совсем в другом месте - одно на дискете, другое вводится с клавиатуры.

Если программа предназначена для сложения 10000 чисел, записанных на дискете, то данными будут эти 10000 чисел. Если программа предназначена для подсчета количества слов в тексте рассказа, вводимого с клавиатуры, то данными будет этот текст. Если программа предназначена для распечатки на принтере изображения с экрана дисплея, то данными будет изображение. Если программа предназначена для распознавания речи, вводимой в компьютер с микрофона, то данными будет звук. В подавляющем большинстве случаев данные во время их обработки хранятся в оперативной памяти.

Взаимодействие различных устройств компьютера можно уподобить взаимодействию нескольких заводов, стоящих вдоль скоростного шоссе (шины) и производящих друг для друга различную продукцию (информацию). При этом память - это не завод, а большой перевалочный склад. А на заводах собственных складов нет или они маленькие. Пусть сегодня один завод произвел для другого большое количество деталей, которое другой завод будет использовать в течение целого месяца. Целиком все детали этот второй завод сегодня забирать не будет, потому что ему складывать их некуда. Первому заводу их тоже негде хранить. Тогда первый завод везет все детали на перевалочный склад, откуда второй завод будет их понемножку забирать по мере надобности.

Назначение других устройств компьютера, кроме процессора и памяти, рассмотрим в 26.4.

26.2. Устройство и размеры оперативной памяти

Представьте себе тетрадный листок в клеточку. В каждую клетку вы имеете право записать карандашом какую-нибудь букву или цифру или знак препинания или вообще любой символ, который можно найти на клавиатуре. А можете и стереть ластиком и записать другой символ. Много ли букв можно записать на листе? Ровно столько, сколько на нем клеток.

Оперативная память компьютера устроена аналогично этому листу. Только размер ее гораздо меньше, чем у тетрадного листа, а клеточек гораздо больше, и каждая клеточка называется **байтом**. Для запоминания слова КОШКА понадобится 5 байтов. На странице вашего учебника около 1000 букв и других символов (включая запятые, точки и пробелы), значит, для запо-

минания страницы текста нужно 1000 байтов. Вы можете сами подсчитать, сколько страниц текста может запомнить современный компьютер, если я скажу, что его память не бывает меньше миллиона байтов. (Если запоминаются не текст, а числа, то место в памяти отводится немножко по другим правилам.)

Оперативная память компьютера электронная. Информация хранится в ней в виде электрических импульсов или потенциалов в миниатюрных электронных схемах и передается из одного места в другое со скоростью близкой к скорости света. Запись, стирание, считывание информации из нее осуществляются по приказам процессора в соответствии с программой меньше чем за стомиллионную долю секунды.

26.3. Взаимодействие программ в памяти

Важно помнить, что компьютер работает по программе не только тогда, когда выполняет нашу программу умножения из 26.1, но и до и после этого. Так уж он устроен. Спрашивается, по какой же программе он работает, когда не выполняет нашу. Рассмотрим упрощенно, что происходит в кратко описанный мной в 26.1 период между моментом включения компьютера и моментом начала выполнения нашей программы.

Внутри компьютера в специальном **постоянном запоминающем устройстве** находится программа самопроверки компьютера. Как только вы включаете компьютер, он всегда начинает выполнять именно ее. Если в результате ее выполнения компьютер решит, что его здоровье в порядке, он продолжает работу и обязательно переписывает в память с винчестера (о котором подробнее - позже) основную часть так называемой **операционной системы** (ОС) - комплекса служебных программ, предназначенного для того (скажем пока), чтобы обеспечить человеку и созданным им программам нормальную работу на компьютере. На вашем компьютере ОС – это Windows. Переписав ОС, компьютер сразу же переходит к ее выполнению и в процессе выполнения останавливается на той команде ОС, которая приказывает ему ждать указаний от человека, что ему делать дальше. Вы решаете, например, что вам нужно работать с Visual Basic, и несколькими щелчками мыши приказываете компьютеру запустить Visual Basic в работу. После этого процессор переходит к выполнению следующих команд ОС, которые “осознают” ваше указание и выполняют его, в результате чего переписывают (**загружают**) большую комплексную программу, которой является Visual Basic, с жесткого диска в память и запускают эту программу на выполнение.

Важно понимать, что запуск на выполнение целой программы - Visual Basic явился результатом выполнения очередной команды другой программы - ОС (говорят - ОС **вызывает** Бэйсик или **управление передается** Бэйсику). От того, что начал выполняться Visual Basic, ОС не ушла в небытие, она осталась в памяти, притаилась и ждет, когда Visual Basic, как и положено каждой порядочной программе, решив поставленные человеком задачи, закончит свою работу и уступит место. В этот момент ОС как ни в чем не бывало продолжит свою работу с команды, которая следует сразу же за той, что запускала Visual Basic (говорят - **управление возвращается** к ОС). Выполнив несколько следующих своих команд и поделав маленькие свои дела, ОС снова натыкается на свою команду, которая приказывает компьютеру ждать указаний от человека, что ему делать дальше. На этот раз человек может пожелать поиграть в какую-нибудь игру. ОС переписывает с винчестера в память и затем вызывает программу этой игры. После окончания игры управление снова возвращается ОС и т.д. Так и проходит с утра до вечера работа на компьютере: после выполнения очередного желания человека ОС получает управление, выполняет некоторую подготовительную работу (чистит память и т.п.) и снова ждет от человека новых пожеланий.

А теперь рассмотрим подробнее период между запуском программы-Visual Basic и завершением ее работы. Visual Basic берет пример с ОС. Получив управление, он выполняет некоторые подготовительные действия и останавливается на той своей команде, которая ожидает ввода программы. Вы вводите с клавиатуры свою программу умножения, а Visual Basic отправляет ее в память. Затем Visual Basic останавливается на другой своей команде, ждущей пожеланий человека. Здесь вы можете пожелать исправлять программу, запустить ее на выполнение, сохранить ее на диске и т.д. Предположим, вы приказываете выполнять программу. Тогда следующие команды Visual Basic, проанализировав ваш приказ, выполняют вашу программу, то есть происходит примерно то, что я подробно описал в 26.1.

Обратите внимание на то, сколько программ находится в этот момент в оперативной памяти. Во-первых, это ОС, которая ждет, когда вам надоест работать на Бэйсике. Во-вторых, это Visual Basic, который выполняет вашу программу, а выполнив, будет ждать от вас дальнейших приказов. И в-третьих, это сама ваша программа умножения. Это обычная практика работы всех компьютеров: в памяти может одновременно находиться от нескольких программ до нескольких десятков. Во многих из них есть команды, которые передают управление другим программам, а затем получают его обратно. Такая передача управления происходит постоянно и зачастую автоматически, без ведома человека. Представьте себе детей, играющих в мяч и перекидывающих его друг другу. Дети - программы, мяч - компьютер, вернее - власть над компьютером. Каждый ребенок может делать с мячом, что хочет - бросить другому ребенку, проткнуть гвоздем (не связывайтесь с подозрительными программами!).

Начинающий программист может ничего этого и не знать. Ему достаточно знать те несколько клавиш, которые он должен нажать, и приказов, которые он должен отдать, чтобы добраться до Visual Basic и производить там элементарные действия - ввод программы, ее исправление, запуск и т.п.

26.4. Внешние устройства компьютера

Как я уже говорил, процессор и оперативная память образуют "мозг" компьютера. Остальные устройства по отношению к ним являются **внешними** или **периферийными**. Мы рассмотрим назначение самых популярных из них, для чего не очень четко разобьем их на три класса:

- **Устройства ввода** в компьютер информации, поступающей от человека, других компьютеров и аппаратов.
- **Устройства вывода** из компьютера информации, предназначенной для человека, других компьютеров и аппаратов.
- **Внешняя память**, то есть устройства памяти, дополняющие оперативную память.

Разбиение получилось не очень четким потому, что некоторые устройства можно отнести сразу к нескольким классам.

Устройства ввода

Первые три устройства предназначены для ввода информации в компьютер непосредственно от пальцев человека.

1. Клавиатура.

2. Мышь.

3. Джойстик. Предназначен примерно для того же, что и мышь. Часто представляет собой коробочку с торчащим из нее рычагом. Коробочка стоит на месте, а рычаг вы можете наклонять в любую сторону, управляя движением объекта на экране.

4. Сканер. Если вы литературовед, то вас вполне могут интересовать вопросы такого типа: "Сколько раз встречается имя "Наполеон" в романе Льва Толстого "Война и мир"?". Поскольку сам роман очень большой, то хотелось бы ответ на этот вопрос поручить компьютеру. Однако тут возникает трудность: чтобы компьютер мог решить эту задачу, текст романа должен оказаться у него в памяти. Для этого вы должны весь текст набрать на клавиатуре - работа на несколько месяцев. Есть способ быстро ввести печатный текст с листа в компьютер. Для этого используется сканер - прибор, главной составной частью которого является специальное считывающее устройство, передающее изображение листа бумаги с текстом в компьютер.

Итак, изображение текста находится в памяти компьютера. Однако, это не значит, что компьютер "знает", что записано у него в памяти. Ведь компьютер сам не умеет по изображению буквы различить, что это за буква. Нужна специальная программа, которая различает между собой печатные буквы различных шрифтов.

Вы спросите, а как же компьютер различает буквы, вводимые с клавиатуры"? Ответ: А он различает совсем не буквы, а нажимаемые клавиши.

Сканером можно вводить с листа не только текст, но и картинки. Но для распознавания картинок нужны программы еще более сложные, чем для распознавания текста. Так, в мире пока не существует программы, которая бы могла по фотографии отличить собаку от кошки.

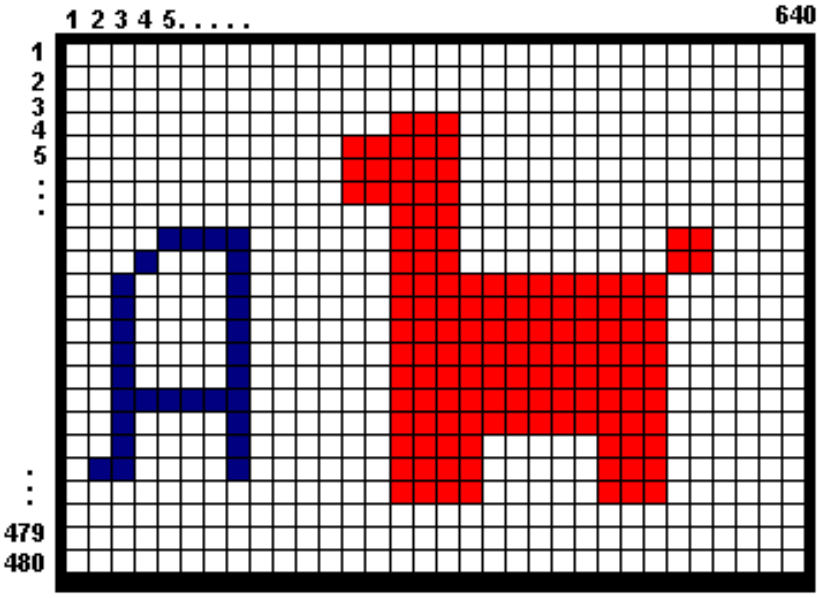
5. Микрофон. Предназначен для того, чтобы вы могли что-нибудь приказать компьютеру или записать свой голос на диск.

6. Ввод с дискеты. Вы уже знаете, что с дискеты можно вводить (**загружать**) в компьютер программы и другую информацию. Более подробно с дискетами вы познакомитесь чуть ниже.

7. Ввод с компакт-диска. С компакт-дисков тоже загружаются в память программы и другая информация. Более подробно с компакт-дисками вы познакомитесь чуть позже.

Устройства вывода

1. Монитор (дисплей). Компьютер не умеет рисовать на экране монитора ничего, кроме светящихся точек (пикселов). Однако, расположив несколько светящихся точек в ряд вплотную друг к другу, получим линию, а сплошь заполнив ими некоторую область экрана, получим изображение фигуры. Пикселы расположены на экране стройными рядами. Каждый пиксел по указанию программы может быть потухшим или гореть заданным цветом (см. рисунок).



Принцип создания изображения на экране монитора

На рисунке вы видите, что на экране умещается 640 столбцов и 480 строк пикселов . Общее количество пикселов получается равным 640x480=307200. Минимальный различимый размер пиксела зависит от качества монитора. Чем этот размер меньше, тем тоньше и правдоподобнее можно изобразить рисунки на экране. Общим количеством столбцов и строк управляете вы сами через специальную электронную схему, находящуюся в компьютере **– видеоадаптер** (**видеокарту**). Если вы заставите видеоадаптер разбить экран на слишком маленькое количество пикселов, они будут слишком большими и изображение будет грубым, неподробным. Если вы заставите видеоадаптер разбить экран на слишком большое количество пикселов, так что их размер станет меньше минимального различимого, обеспечиваемого монитором, они перестанут быть различимыми и качество изображения не улучшится. Нужна золотая середина. Количество цветов, на которое вы можете настроить видеоадаптер, тоже разное - от 256 до миллионов.

2.Принтер. Если мы хотим, чтобы числа, текст, рисунки, полученные компьютером, оказались не на экране, а на листе бумаги, то пользуемся готовой программой печати их на принтере. Принтер - это, попросту говоря, автоматическая пишущая машинка, подключенная к компьютеру и печатающая текст, рисунки и фотографии по командам программы. Изображение на листе получается из отдельных точек примерно по такому же принципу, что и изображение на экране. По физическому способу получения точек на листе принтеры делятся в основном на матричные, струйные и лазерные. **Матричный принтер** получает черные точки на листе ударами маленьких штырьков (игл) по красящей ленте, которая оставляет след на бумаге. Матричные принтеры самые дешевые и цветными быть не могут, они уже устаревают. **Струйный принтер** впрыскивает на лист

мельчайшие капельки разноцветных чернил из специальных шприцев (сопел), поэтому изображение на листе может быть цветным. **Лазерный принтер** при помощи лазерного луча снимает в нужных точках электрический заряд со специального барабана, после чего тот входит в контакт с красящим порошком. Порошок пристает к барабану только там, где его притягивает электрический заряд, в результате чего на барабане получается изображение. Затем барабан прокатывается по листу бумаги и отдает изображение ему. Лазерные принтеры бывают и цветные.

3.Плоттер - это подсоединенное к компьютеру автоматическое чертежное устройство, которое чертит пером на бумаге чертежи, графики и другие изображения под управлением программы.

4.Звук. Если вы наблюдали, как работает персональный компьютер, то обратили внимание, что во время работы он издает звуки. Это могут быть отдельные редкие попискивания, простенькие мелодии. На старых компьютерах их издает устройство, которым снабжаются все компьютеры и которое называется **PC Speaker**. На современных компьютерах их издает качественное звуковое устройство, которое называется **звуковая карта**. С ней вы сможете услышать целый симфонический оркестр и внятную человеческую речь.

5.Вывод на дискету. Вы знаете, что на дискеты можно записывать из компьютера программы и другую информацию. Более подробно с дискетами вы познакомитесь чуть ниже.

Внешняя память

Мы рассмотрим 5 самых распространенных типов внешней памяти.

1. Винчестер (жесткий диск). У оперативной памяти есть два существенных недостатка: **1)** Когда вы выключаете компьютер, все содержимое оперативной памяти стирается. Электронная оперативная память не может что-то помнить, если к ней постоянно не подведен электрический ток. **2)** Оперативная память сравнительно дорога, много ее не купишь, поэтому на большинстве персональных компьютеров сегодня установлено от 4 до 500 миллионов байтов (сокращенно и приблизительно 1 миллион байтов называют мегабайтом) оперативной памяти. Однако некоторые программы, например, игровые, настолько велики, что требуют для своего запоминания больше 100 мегабайтов. Это значит, что в компьютеры с маленькой памятью они просто не уместятся, а следовательно, не могут быть запущены.

Для преодоления этих двух недостатков большинство современных персональных компьютеров снабжаются "**винчестером**" - устройством памяти на жестких магнитных дисках. Запись информации в нем производится на быстро вращающийся диск, покрытый магнитным веществом (см. рисунок). Принцип записи и считывания тот же, что и при записи и воспроизведении песенки на магнитофоне. Цена одного мегабайта памяти винчестера гораздо меньше, чем цена одного мегабайта оперативной памяти, поэтому сегодня на большинстве персональных компьютеров она имеет размер от 100 до 50000 мегабайтов.

Схема работы памяти на магнитном диске

За дешевизну расплачиваемся быстродействием. Чтобы считать информацию с диска, необходимо подвести магнитную головку к тому месту диска, где записана нужная информация, что занимает сотую долю секунды. По сравнению с оперативной памятью жесткий диск - черепаха.

Винчестер используется для хранения самых необходимых и часто используемых больших программ: операционных систем, сред программирования и т.д. Теперь не беда, что большая программа не умещается в оперативной памяти, в ней в каждый момент времени находится только та часть программы, которую необходимо выполнять именно в данный момент, а остальная часть программы находится на диске и ждет своей очереди, чтобы быть загруженной в память и выполняться.

Кроме этого, вы используете винчестер, выключая вечером компьютер, чтобы переписать на него нужное вам содержимое оперативной памяти, например, создаваемую вами программу, которую вы сегодня только наполовину ввели в память, или полезные вам в будущем результаты сегодняшней работы других программ.

2. Дискета. Часто одним персональным компьютером пользуются по очереди несколько человек. Небрежный пользователь случайным нажатием на клавиши может стереть с винчестера нужную информацию, свою или чужую. В этой ситуациии помогает дискета. Это небольшой гибкий магнитный диск, упакованный в квадратный пластиковый или бумажный пакет. В системном блоке компьютера расположено устройство для считывания и записи информации на дискету - **дисковод**. У дискет есть огромное преимущество перед винчестером - они съемные, а это значит, что важную информацию вы всегда можете переписать с винчестера на дискету, вынуть дискету из дисковода и унести домой, где с нее никто ничего не сотрет. Если с вашей информацией на винчестере что-нибудь случилось, вы приносите из дома дискету, вставляете ее в дисковод и переписываете с нее информацию обратно на винчестер. Вместимость одной дискеты - порядка полутора мегабайтов.

Дискеты имеют еще одно важное применение - с их помощью вы можете переносить понравившиеся вам программы и другую информацию с одного компьютера на другой.

У дискет гораздо ниже быстродействие, чем у винчестера, так как у них гораздо ниже скорость вращения, к тому же после каждого считывания или записи на дискету она прекращает вращаться и для следующего считывания или записи ее приходится раскручивать. Диск же винчестера вращается непрерывно на протяжении всей работы компьютера.

Дискеты менее долговечны, чем винчестер, так как магнитная головка во время работы скользит по поверхности дискеты и постепенно стирает ее магнитный материал, к тому же внутрь дискеты попадает пыль, приводящая к повреждению поверхности дискеты. В винчестере же магнитная головка не соприкасается с поверхностью диска, скользя над ней на воздушной подушке. Пыль внутрь винчестера тоже не попадает, так как он герметически закрыт.

3. Компакт-диски (CD-ROM). Компакт-диски в основном штампуются на заводе, как грампластинки. Информация на них хранится в виде микроскопических бугорков и бороздок на зеркальной поверхности металлической пленки под стекловидной поверхностью диска и считывается лазерным лучом.

Компакт-диски сменяемы, надежны, долговечны, вместительны (порядка 700 мегабайтов). На них обычно находятся большие коммерческие программы, изображения, аудиозаписи, видеофильмы для просмотра на компьютере. Конечно, главный их недостаток - то, что на них ничего нельзя записать (однако, в последнее время стали продаваться лазерные диски с возможностью записи и соответствующие дисководы).

4. Компакт-диски с возможностью записи (CD-R) и перезаписи (CD-RW). В последнее время цена компакт-дисков с возможностью записи и перезаписи упала до цены дискет. Когда подешевеют и дисководы для работы с такими дисками, эпоха дискет закончится, так как вместимость таких дисков равна вместимости обычных компакт-дисков, а скорость записи-считывания гораздо больше, чем у дискет.

Есть и другие типы дисков для хранения информации (например, DVD), но они пока применяются реже.

5. Флэш-память. Это перспективный вид переносимой памяти. Не требует питания. Не имеет движущихся частей. Имеет вид маленькой коробочки. Пока дорога. Используется в основном для хранения фотографий в цифровых фотоаппаратах и для переноса этих фотографий в компьютер и принтер.

Связь компьютеров между собой. Модем. Сети

Для переноса информации с одного компьютера на другой не обязательно использовать дискеты. Если два компьютера расположены рядом на соседних столах, то в простейшем случае их достаточно соединить коротким проводом, и при помощи простенькой программы любая информация с винчестера одного компьютера по проводу небыстро переписывается на винчестер другого. Такое соединение позволяет и играть на двух компьютерах в одну игру.

Теперь поговорим о соединении *нескольких* компьютеров. Группу компьютеров, постоянно соединенных друг с другом каким-нибудь способом для обмена информацией, называют **компьютерной сетью**.

Когда эти компьютеры расположены в пределах одного здания, их соединяют специальным кабелем и при помощи мощной операционной системы они могут удобно и с высокой скоростью обмениваться между собой любой информацией. Такая компьютерная сеть называется **локальной**.

Если соединенные компьютеры находятся в разных концах города или даже земного шара, то говорят, что они образуют **глобальную** компьютерную сеть. Правда, протягивать специальные кабели на большие расстояния дорого. Поэтому для дальней связи компьютеров часто используют обычную телефонную сеть. Люди, чтобы переговариваться по телефонной сети, используют телефоны; компьютеры же вместо телефонов используют специальные устройства - **модемы**. Самый известный пример всемирной глобальной компьютерной сети – **Internet**. Телефонная сеть изначально не была предназначена для передачи компьютерной информации, поэтому Интернет при связи по телефонному проводу работает гораздо медленнее локальной сети.

26.5. Кодирование информации в компьютере

Поговорим о том, как физически представлена информация в компьютере. Что значит «физически»? Вот на листе учебника буквы физически представлены типографской краской, в человеческом мозге информация физически представлена электрическими импульсами, которые передаются из одной нервной клетки мозга в другую. В компьютере принят тот же способ представления, что и в мозге - из одного устройства компьютера в другое и внутри устройств информация передается электрическими импульсами. Посмотрим поподробнее, как электрические импульсы несут информацию в компьютере.

Прежде всего заметим, что информация в компьютере - это или программы или данные, с которыми эти программы работают.

Из чего состоит программа? Программа на языке программирования состоит из команд, записанных при помощи букв, цифр, знаков математических действий, знаков препинания и других символов*. Будем понимать под **символом** любой знак (букву, цифру, знак математического действия, знак препинания и др.), который понимает компьютер. Многие символы вы можете видеть на клавиатуре.

Из чего состоят данные? Если это числовые или текстовые данные, то они тоже состоят из символов†. О графических данных (изображениях) и звуке поговорим чуть ниже.

Таким образом, значительная часть информации в компьютере состоит из символов. Посмотрим, как в компьютере представлены символы. Для этого вспомним, как кодируются символы в азбуке Морзе, активно использовавшейся не так давно для передачи сообщений на расстояние. Каждый символ (буква, цифра) представлен в ней цепочкой точек и тире. Например, буква А представлена, как . -, буква Ч - как - - - . . В компьютере каждый символ тоже кодируется, но по-другому – цепочкой из восьми единиц и ноликов. Например, буква А представлена, как 10000000, буква Ч - как 10010111, а цифра 7, как 00110111.

Кстати, вот полезная задачка для будущего программиста: Сколько всего символов можно закодировать цепочкой из восьми единиц и ноликов? Подумайте на досуге.

Пока мы с вами говорили о символах и их кодировании безотносительно к тому, какими физическими процессами они представлены в компьютере. Мы были на так называемом «логическом» уровне. Теперь перейдем на физический уровень. Пусть память передает на принтер букву Ч. В этом случае она посылает по шине в течение, скажем, восьми микросекунд, серию из восьми электрических импульсов и промежутков между импульсами:

Первая микросекунда	-	импульс
Вторая микросекунда	-	промежуток
Третья микросекунда	-	промежуток
Четвертая микросекунда	-	импульс
Пятая микросекунда	-	промежуток
Шестая микросекунда	-	импульс
Седьмая микросекунда	-	импульс
Восьмая микросекунда	-	импульс

Как видите, последовательность импульсов и промежутков в серии соответствует последовательности единиц и ноликов в

* Программа на машинном языке представлена по-другому.

† Опять же, числа в компьютере далеко не всегда состоят из символов-десятичных цифр. Когда компьютер производит над числами арифметические и другие операции, числа представлены совсем по-другому.

коде буквы Ч. Величина импульса не играет никакой роли, все импульсы в микросхемах компьютера имеют обычно одну и ту же величину, скажем 2 вольта.

Таким же примерно образом обмениваются группами из 8 импульсов все устройства компьютера. В памяти эти группы живут в "замороженном" виде. В каждом байте оперативной памяти или памяти на диске умещается ровно одна такая группа, поэтому говорят, что устройства обмениваются байтами информации.

В оперативной памяти единичка представляется наличием электрического потенциала в определенной точке электронной микросхемы, а нолик - его отсутствием. А поскольку таких точек в памяти многие миллионы, то столько же там и единиц с ноликами. В памяти на магнитных дисках единичка представляется наличием намагниченности в определенной точке диска, а нолик - его отсутствием или намагниченностью в другом направлении. В компакт-дисках единичка - это бороздка или бугорок в определенной точке диска, а нолик - его отсутствие, то есть участочек с зеркальной поверхностью.

Перейдем снова на логический уровень. Когда кодируется изображение, то кодируется информация о каждом пикселе изображения (в виде группы единиц и ноликов). Например,

Код 111 -	пиксел горит белым цветом
Код 100 -	пиксел горит синим цветом
Код 010 -	пиксел горит красным цветом
Код 001 -	пиксел горит зеленым цветом
. . . . .	
Код 000 -	пиксел не горит (черный)

Если программа предназначена для распечатки изображения с экрана монитора на цветном принтере, то она просто посылает на принтер по очереди коды информации о каждом пикселе изображения.

При кодировании звука используются разные способы, но факт то, что результатом кодировки являются все те же группы единиц и ноликов.

В заключение отмечу две неточности в моем изложении материала этого пункта. Я говорил, что единички в разных устройствах компьютера представляются наличием потенциала или намагниченности или бороздок и т.д., а вот нолики - их отсутствием. На самом деле в отдельных устройствах может быть и наоборот - единички это отсутствие, а нолики - наличие. Это не принципиально.

Второе: коды чисел в компьютере часто не являются совокупностью кодов цифр, эти числа образующих. Так, число 88 часто не представляется цепочкой 00111000 00111000, а для кодирования чисел используется другой, более экономный способ.

Вывод – любая информация в компьютере закодирована в виде цепочек, состоящих из единиц и нулей, и в таком закодированном виде передается внутри устройств и между устройствами компьютера. Обычно длина цепочки равна 8 и тогда такая цепочка называется **байтом**, а каждый из восьми ноликов или единичек называется **битом**. Таким образом, 1 байт = 8 битов.

_____ – _____

Мне кажется, тех сведений, которые вы получили в этой части, достаточно для того, чтобы приступить к *сознательному* программированию на Visual Basic.

Приложение 2. Работа в Windows. Ввод текста

Это приложение - для тех, у кого совсем нет опыта работы за клавиатурой и с мышкой на компьютере под управлением Windows. Здесь я научу вас азам, которые необходимы и достаточны для начинающего программиста, чтобы работать с Visual Basic.

Работа в Windows

Включение и выключение компьютера. Первые шаги

Итак, вы впервые или почти впервые сели за компьютер. Не беда. Книжка у вас в руках. Все, что вам нужно знать, это то, что компьютер исправен, у него есть дисковод CD-ROM и что на нем установлена и нормально работает операционная система Windows 95 или Windows NT 4.0 или более поздние версии Windows.

Включите компьютер. Некоторое время по экрану будет бежать всякая информация, наконец картинка установится. Пространство экрана в Windows, которое вы сейчас видите, называется **рабочим столом**. На нем вы должны сейчас видеть как минимум два значка и панель задач. **Значки** называются "Мой компьютер" и "Корзина". **Панель задач** - это серая (обычно) "доска", "приклеенная" к одному из краев экрана (почти всегда - к нижнему). На одном конце панели задач должна находиться кнопка "Пуск". Если панель задач не видна (что иногда бывает), значит она спрятана (для экономии места) за одним из краев экрана. Попробуйте достать ее оттуда мышиным курсором, копнув им край экрана, как лопатой. Если она снова убежит за край - не беда, вы всегда при необходимости выкопаете ее оттуда. Если вы никогда не держали мышку в руках, прочитайте сначала чуть ниже "Работу с окнами Windows".

Щелкните кнопку "Пуск" на панели задач. (Когда я говорю "Щелкните", я всегда подразумеваю "Щелкните левой клавишей мыши".) Перед вами появится **стартовое меню** Windows - список приказов, которые вы можете отдать компьютеру, или, что точнее, - список программ, установленных на данном компьютере. Чтобы выполнить программу, щелкнете по пункту меню мышкой.

Стартовое меню ступенчатое, то есть некоторые пункты меню открывают вложенное в них другое меню. Такие пункты помечены черной треугольной стрелочкой. Так в ресторанном меню пункт меню "Десерт" содержит такие пункты: "Мороженое", "Клубника со сливками" и т.д.

Когда вам захочется выключить компьютер, ни в коем случае сразу же не нажимайте кнопку выключения компьютера на системном блоке. Ваши действия: Кнопка "Пуск" на стартовом меню → пункт "Завершение работы компьютера" → выставьте флажок (точку) против пункта "Выключить компьютер" или щелчком по черной треугольной стрелке добейтесь, чтобы в окне появились именно эти слова → **Да**. Через некоторое время компьютер или выключится сам или на экране появится надпись, разрешающая отключить питание.

Работа с окнами Windows

Когда вы работаете в Windows, Visual Basic или других приложениях Windows, все, что вы видите на экране, вы видите в окнах. Не зря же слово Windows переводится с английского, как "окна". Окно имеет вид прямоугольника в тоненькой рамочке, внутри которого находится нужная вам информация. Окна - весьма удобная вещь и работа с ними ведется так же естественно, как с документами на рабочем столе.

Открытие окон из значков. Значки - это, упрощенно говоря, закрытые окна. Пока они закрыты, толку от них нет. Чтобы их открыть, нужно проделать одно из трех действий (для примера попробуйте открыть окно из значка "Мой компьютер"):

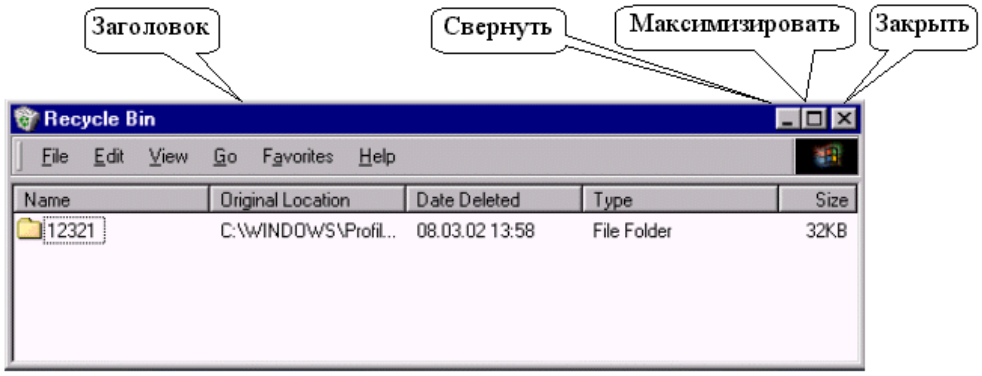
- Поставьте острие стрелки мышиного курсора (в дальнейшем я просто буду говорить "Поставьте мышку") на значок. (Только не на название под значком, а на сам значок.) Теперь, держа мышь совершенно неподвижно (как прибитую к коврику!), сделайте **двойной щелчок**, то есть, щелкнув пальцем по левой клавише мыши, тут же, через какую-то долю секунды, не сдвинув мышь ни на йоту, щелкните еще раз. Окно откроется. У начинающего с первого раза двойной

щелчок не получится ни за что. Этот способ самый быстрый, но требует некоторой тренировки, поэтому для начала предлагаю три других способа, полегче:

- Держа мышь неподвижно, щелкните мышкой по значку (не по названию под значком, а по самому значку). Если щелкнули успешно, значок должен потемнеть. Теперь нажмите на клавиатуре клавишу Enter. Окно откроется.
- Держа мышь неподвижно, щелкните по значку правой клавишей мыши. **Правая клавиша мыши** - очень удобная клавиша. Если вы хотите что-нибудь сделать со значком или с каким-нибудь другим объектом на экране, но забыли, что именно и на какие кнопки нужно при этом нажимать, щелкните по этому объекту правой клавишей мыши - из объекта выскочит так называемое **контекстное меню** - список самых популярных действий, которые можно проделать с этим объектом. Вам остается только щелкнуть мышкой по нужному. В нашем случае выбирайте пункт "Открыть". Окно откроется.
- Попросите опытного человека настроить Windows на больший промежуток времени между щелчками двойного щелчка.

Итак, **окно** - это прямоугольная рамочка, внутри которой видна та или иная информация, будь то текст, картинка, видео, содержимое папки, игра или тот же Visual Basic. Когда вы открываете значок папки, то открывается окно с содержимым этой папки. Когда вы открываете значок файла, то открывается окно с программой, тип которой определяется типом файла. Так, если ваш файл - это текстовый документ, то открывается окно с программой, показывающей в этом окне содержание этого документа. Если ваш файл - запускающий файл какой-нибудь программы, то открыть этот файл - значит запустить программу. Так, файл Cat.exe запускает игру "Кошки". (О том, что такое файл и папка, смотрите в следующем разделе).

Давайте для примера откроем окно корзины, щелкнув как положено по значку "Корзина" на рабочем столе.



Посмотрим, что можно делать с окном. Прежде всего, окно можно мышкой перемещать ("таскать") по экрану. Для этого нужно поместить острие мышиного курсора на полосу заголовка окна, нажать левую клавишу мыши и, не отпуская ее, перемещать курсор по экрану (это называется "**тащить**" мышь). Окно "потащится" за мышью.

Вы можете изменять размеры окна, перетаскивая мышью его границы. Для этого нужно поместить острие мышиного курсора точно на одну из четырех границ окна. В этом случае курсор приобретет форму двойной стрелки (↔). В этот момент начинайте тащить мышь. Граница "потащится" за мышью. Удобно таскать и углы окон.

Три кнопки. В правом верхнем углу окна имеются три кнопки. Щелкните по кнопке "*Закрыть*". Окно исчезнет с экрана. Для того, чтобы снова его открыть, нужно будет снова щелкать по его значку.

Щелкните по кнопке "*Свернуть*". Окно свернется, то есть уйдет с экрана на панель задач, где будет существовать в виде прямоугольного значка. Достаточно одинарного щелчка по этому значку на панели задач (не двойного, что удобно), чтобы окно снова развернулось. Обратите внимание, что значок на панели задач при этом не исчезнет. Пощелкайте по нему.

Щелкните по средней из трех кнопок - кнопке "Максимизировать", которая имеет вид квадратика. Окно развернется на весь экран. Обратите внимание, что средняя кнопка теперь изменила вид, она стала такой - . Щелкнем по ней. Окно уменьшилось до прежних размеров.

Окно, входящее в состав среды Visual Basic, при максимизации развертывается не на весь экран, а лишь внутри главного окна Visual Basic. При этом его три кнопки как бы "отделяются" от него и располагаются под тремя кнопками главного окна.

Работа с несколькими окнами. Когда на тесном экране несколько окон, они загораживают друг друга и мешают друг другу. Откройте, например, два окна щелчками по значкам "Мой компьютер" и "Корзина". Увеличьте в размерах окно "Корзина" и надвиньте его на окно "Мой компьютер" так, чтобы оно полностью его загородило. Как теперь добраться до окна "Мой компьютер"? У каждого окна есть свой значок на панели задач. И быстрее всего - щелкнуть по значку "Мой компьютер" на панели задач.

Мой совет: Когда на рабочем столе много окон, переключайтесь между ними щелчками по их значкам на панели задач.

Файлы и папки

Общие понятия

Здесь мы разберем структуру хранения информации на компьютерных дисках, будь то жесткий диск, дискета или компакт-диск. Компьютерный диск - вместительное хранилище, на котором вы храните самые разные вещи: и программы, и тексты, и

картинки, и музыку, и видеофильмы. Каждая из этих вещей, хранящихся на диске, называется **файлом.** Многие файлы, содержащие текст, принято называть **документами**.

У каждого файла должно быть **имя,** например, Sveta25. Файлы появляются на диске двумя способами: или вы их переписываете на диск из другого места и тогда у них уже есть имя, или вы сами создаете файл и тогда имя ему придумываете.

На диске хранятся тысячи файлов. Просмотр содержимого диска - операция, которую вы выполняете очень часто, и поэтому многие программы, включая Visual Basic, позволяют вам это делать. При этом вы всегда видите на мониторе список имен файлов.

Обычно вам нужно найти среди них какой-то один, чтобы начать с ним работать. Но когда в списке тысяча файлов, найти среди них какой-то один довольно затруднительно. Нужно навести среди файлов какой-то порядок, ввести какую-то систему. Можно упорядочить имена файлов по алфавиту или как-то по-другому. Но это не решает проблему. Что же делать?

Вообразите, что вы собираете марки. Альбома у вас нет и вы храните марки в пакетиках. У вас есть огромный пакет, на котором написано "Марки". Внутри него у вас находятся пакеты помельче с надписями "Российские марки", "Африканские марки" и т.д. В пакете "Российские марки" у вас находятся пакетики "Марки о спорте", "Марки о космосе" и т.д.

Для чего вам все эти пакетики? Для того, чтобы легче было искать нужную марку. На диске принята та же система. Только вместо слова "пакетик" мы говорим "**папка**" (раньше был в ходу другой термин - **каталог**). А вместо слова "марка" мы говорим "файл". Каждая папка тоже имеет имя. Типичная ситуация: На диске в ряду других папок есть папка "Программы", папка "Рисунки", папка "Переписка" и т.д. Внутри папки "Переписка " находятся две папки: "Деловая переписка" и "Личная переписка". Внутри папки "Личная переписка " находятся папки: "Переписка с Васей", "Переписка с Асей" и т.д. Внутри папки "Переписка с Асей " находятся файлы-документы, каждый из которых представляет собой письмо, например, файл с именем "Письмо Асе в Алушту в мае 2000 года".

Папки, как и файлы, вы или создаете на диске сами или переписываете откуда-нибудь вместе со всем их содержимым.

Имена файлов и папок

Имена файлам и папкам можно придумывать произвольные, но с некоторыми ограничениями. Имена не должны содержать символов **\ / : * ? " < > |** Начинающим я не рекомендую использовать в именах точки и вообще ничего, кроме букв, цифр и пробелов.

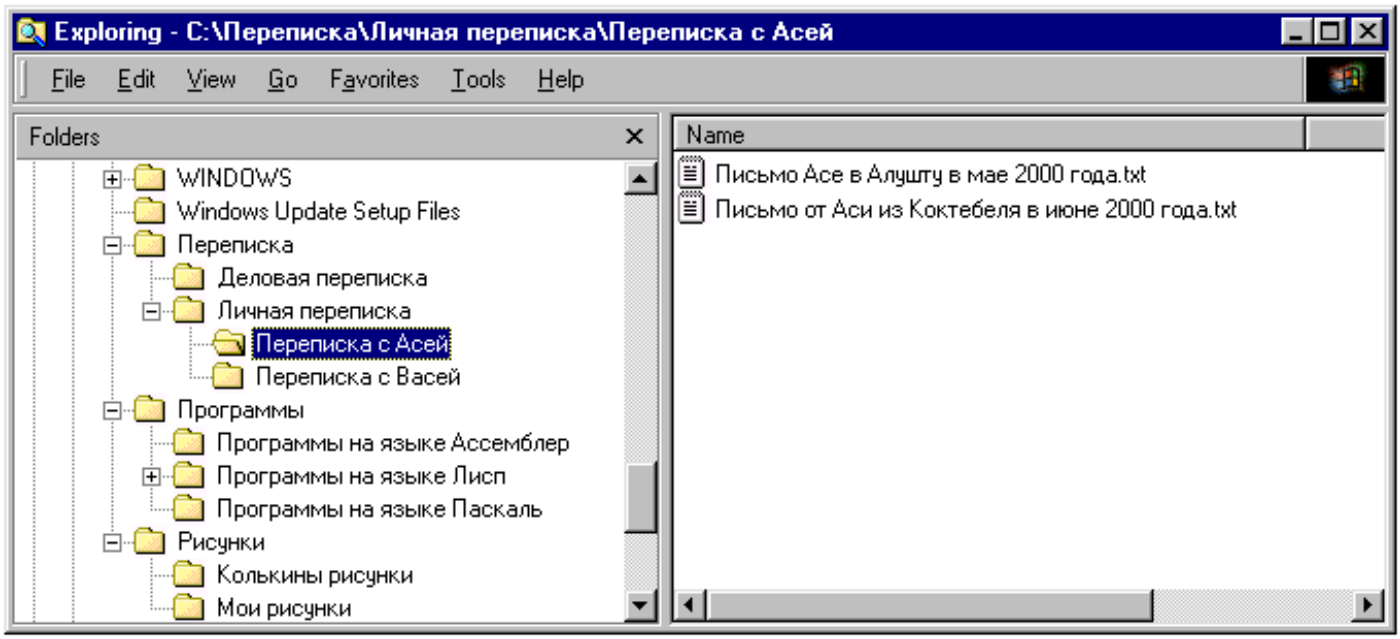
Ограничения зависят также и от самой программы. Многие старые программы были созданы для работы под управлением старой операционной системы MS-DOS и поэтому строги к именам. Они требуют, чтобы имя файла или каталога было не длиннее 8 символов и состояло из латинских букв, цифр, еще кое-каких символов и не содержало пробелов.

К имени файла вы можете справа приписать добавку, состоящую из точки и (справа от нее) нескольких символов, обычно не более трех латинских букв. Эта «фамилия» называется **расширением**. Например, файл, в котором вы описываете, как Ира печет булки, вы могли бы назвать Bulki.Ira. Но начинающим я не рекомендую самим приписывать расширения к файлу.

Часто расширение автоматически и незаметно для вас приписывается к имени файла программой (Паскаль, Visual Basic, …), в которой вы работаете. Так, если вы в Visual Basic создали проект и решили записать его на диск под именем Train, то на самом деле файл с этим проектом на диске будет иметь имя Train.vbp. По расширению программа (Паскаль, Visual Basic, …) узнает «свои» файлы, а опытные пользователи узнают, в какой программе файл был создан. И наконец, расширения у файла может и не быть.

Проводник

Находясь в Windows, вы можете в любой момент посмотреть структуру папок и файлов любого диска вашего компьютера. Сделать это можно по-разному. Фирма Microsoft рекомендует для этого (и многого другого) использовать Проводник - стандартную программу, входящую в состав Windows. Чтобы ее запустить, выберите в стартовом меню Windows пункт "Программы", а в нем - пункт "Проводник". На рисунке вы видите в работе окно проводника, в котором отображена часть папок жесткого диска.



Окно проводника разбито на две панели - левую и правую. Разберемся в *левой панели*.

Отвлечемся пока от белых квадратиков с плюсами и минусами. Желтые прямоугольные значки ( ) - это обозначения папок на диске. По взаимному расположению папок можно понять, внутрь какой папки входит данная папка - это первая из вышерасположенных папок, которая находится левее данной. Например, папка "Переписка с Васей" находится внутри папки "Личная переписка", а та - внутри папки "Переписка". Помогают понять структуру папок и линии, выходящие из данной папки и идущие вниз - потом направо - в папки, находящиеся внутри нее. В совокупности эти линии образуют лежащее **дерево**, ствол - слева, самые тоненькие ветки - справа.

Если папка содержит внутри себя хотя бы одну папку, слева от нее вы видите квадратик с плюсом или минусом. Щелчком по квадратику вы можете менять плюс на минус и наоборот. Поменяв минус на плюс, вы скрываете из вида папки, находящиеся внутри данной (хотя бы для того, чтобы не утомлять глаза).

Если вы щелкните в левой панели по значку папки, имя папки потемнеет, а ее содержимое вы увидите в *правой панели*. У нас в правой панели вы видите два файла из папки "Переписка с Асей". Файлы обозначаются значками самой разной формы, но всегда отличающейся от формы папок. В левой панели файлы не видны, видны только папки.

У правого края левой панели вы видите вертикальную **полосу прокрутки** с двумя кнопками в виде черных треугольников. В каждый момент времени в левой панели мы видим не все папки, но щелкая по этим кнопкам мы введем в поле зрения любую часть дерева.

Кроме просмотра папок проводник позволяет делать массу других вещей. Часть из них (например, копирование) вы можете проделать с помощью правой клавиши мыши, как это описано в 3.5

Логические диски. Адрес файла (путь, дорожка к файлу)

Многие программы (в том числе Windows и Visual Basic) позволяют вам создавать, удалять и переименовывать файлы и папки, копировать и переносить их из любой папки в любую другую и обратно, с жесткого диска на дискету и обратно.

В процессе общения с этими программами вам приходится объяснять им, где, в какой папке находится такой-то файл или папка, и самим понимать их объяснения. Например, вам нужно понимать, что значит запись

C:\Переписка\Личная переписка\Переписка с Асей\Письмо Асе в Алушту в мае 2000 года.txt

Для этого сначала разберем, что такое логические диски.

Пусть на вашем компьютере есть жесткий диск, дисковод для дискет и дисковод для компакт-дисков. Компьютер именует все эти дисководы буквами латинского алфавита. Дисковод для дискет должен иметь имя A или B. Жесткий диск почти всегда имеет имя C. Однако, у многих жестких дисков имеется странность, доставшаяся им, как аппендицит, от старых версий операционных систем. Эта странность состоит в том, что винчестер делится на несколько независимых участков. Каждый участок называется **логическим диском**. Эти логические диски получают имена C, D, E и т. д. Операционная система предлагает нам пользоваться этими логическими дисками, как независимыми винчестерами. Что ж, в принципе, пользователю все равно, он может даже и не знать, что у него на компьютере не несколько жестких дисков, а один. Компакт-диск тоже получает одну из букв, следующую по алфавиту.

Итак, как же понимать вышеприведенную запись? Она означает, что файл с именем "Письмо Асе в Алушту в мае 2000 года.txt" находится в папке "Переписка с Асей", которая находится в папке "Личная переписка", которая находится в папке "Переписка", которая находится на жестком диске. Эта запись называется **путем** или **дорожкой** к файлу. Я предпочитаю называть ее **адресом**. Обратите внимание, что после имени логического диска нужно ставить двоеточие.

Эта запись довольно длинная и скучная. К счастью, довольно часто компьютер помнит, в какой папке вы работаете, считает ее "текущей" и в этом случае вам достаточно указать ему только имя файла.

Как вводить программу в компьютер или работа с текстом в текстовом редакторе

Ввод вашей программы в компьютер производится при помощи специальной программы, которая называется текстовым редактором и входит в состав Visual Basic. Программа на Visual Basic - это обычный текст, поэтому вам достаточно знать работу в текстовом редакторе с обычным текстом. Приемы работы в текстовом редакторе Visual Basic практически ничем не отличаются от приемов работы в других текстовых редакторах.

Те предложения, которые мы сейчас для тренировки будем писать (типа "Маша ела кашу"), представляют с точки зрения Visual Basic невообразимую чушь и поэтому после ввода каждой строки он будет нещадно ругаться. Чтобы он от нас отстал, отключим опцию проверки грамматики при вводе текста в окно кода. Для этого в пункте Tools главного меню Visual Basic найдем пункт Options, а там в закладке Editor снимем флажок Auto Syntax Check. Только не забудьте при переходе к вводу реальных программ снова поставить этот флажок.

Работа с одной строкой текста

Когда мы начинаем работать в текстовом редакторе Visual Basic, перед нами обычно - пустое окно кода. Пустое пространство окна - наш лист, и на нем мы будем писать. На картинке вы видите окно кода, уже частично заполненное текстом.

Сейчас я проведу вас "за руку" от начала и до конца. Однако и в этом случае вы можете натолкнуться на неожиданности. Помощь вы найдете в дальнейших строках материала вплоть до «Работы с несколькими строками». Рекомендую сейчас быстренько пробежать глазами этот материал.

Ввод строки. Пусть мы хотим ввести строчку "юный пионер Коля любит Bubble Gum". Первая клавиша, по которой мы должны щелкнуть, - русская буква "ю". Но где она появится на экране? - В том месте, где сейчас мигает **текстовый курсор** - маленькая вертикальная черточка. А в начале работы редактора он должен мигать в левом верхнем углу пустого окна. Если не мигает, щелкните мышкой по окну кода.

Не путайте текстовый курсор с мышиным. Мышиный свободно перемещается по экрану вслед за движениями мыши по коврику и не мигает, у текстового такой свободы нет и он мигает.

Итак, щелкаем по клавише "ю" - и на экране на месте текстового курсора возникает буква "ю", а сам текстовый курсор перемещается чуть вправо, там потом возникнет следующая буква. *Где бы текстовый курсор ни находился, следующая буква возникает на его месте*. Если получилась не буква "ю", а точка, значит прочитайте чуть ниже про английские и русские буквы. Если буква получилась заглавной, почитайте про заглавные буквы. Вот какая получается картина - "ю/". По клавише именно щелкаем, «клюем» ее, а не нажимаем, потому что компьютер настроен так, что если мы задерживаем палец на клавише дольше некоторой доли секунды, он считает, что нажатий было не одно, а два, еще подольше – три, и так далее, а это значит, что на экране мы увидим "ююююююю/". Поэтому в дальнейшем изложении, когда я говорю «Нажмите клавишу», я буду иметь в виду «Щелкните по клавише».

Получив на экране "ю", щелкнем по "н". На экране видим "юн/". И так далее, пока на экране мы не получим "юный/".

Если мы в процессе работы случайно нажали на клавишу не с той буквой, то щелкнем по клавише BackSpace. Она стирает последнюю введенную букву. Эта клавиша имеет еще маркировку BS или ⇦.

После ввода слова "юный" нужно ввести пробел перед следующим словом. Для этого щелкните по самой длинной горизонтальной клавише. Затем аналогично вводите слово "пионер" и пробел.

Заглавные буквы. Чтобы буква "к" в слове "Коля" получилась заглавной, нужно нажать на клавишу Shift и держать ее нажатой. Смело держите ее нажатой сколько угодно - ничего плохого от этого не произойдет. Затем, удерживая ее нажатой, щелкните по букве "к" - она получится заглавной. Теперь отпустите Shift. Можно работать дальше со строчными буквами. Иногда клавиша Shift имеет маркировку ⇧.

Ненароком в процессе работы вы можете нажать клавишу CapsLock и не заметить этого. После этого при нажатии на любую буквенную клавишу вы получите заглавную букву вместо строчной. При этом в правом верхнем углу клавиатуры горит индикатор CapsLock. Еще раз нажмите на CapsLock, индикатор потухнет и все вернется на свои места.

Английские и русские буквы. Вот вы дошли до английского слова "Bubble Gum". Как его набрать? Вы уже заметили, что на большинстве клавиш букв две - сверху английская (латинская), снизу русская. Предположим, до сих пор у вас при нажатии на клавишу всегда выходила русская буква. Говорят, что вы работали в **русском регистре**. Если вы нажмете на пару служебных клавиш, то теперь при нажатии на любую буквенную клавишу у вас будут получаться английские буквы, пока вы снова не нажмете на эти самые служебные клавиши, чтобы вернуться к русским буквам. Вот эти клавиши:

- *Левая* Alt - Shift — На клавиатуре имеется две клавиши Alt. Имеется в виду, что удерживая нажатой левую клавишу Alt, вам нужно щелкнуть по одной из клавиш Shift.
- Ctrl - Shift — Удерживая нажатой клавишу Ctrl, вам нужно щелкнуть по клавише Shift.

Если вдруг эти клавиши не подействуют, то найдите в правой части панели задач индикатор с обозначением **En** или **Ru** и щелкните по нему, после чего в открывшемся меню выберите мышкой нужный язык.

Знаки препинания. Вы набрали все предложение. Теперь вам самое время поставить точку, а мне поговорить о знаках препинания и других полезных символах. Над буквами вы увидите горизонтальный ряд клавиш с цифрами от 0 до 9. На эти же клавиши нанесены и многие символы. Вы их сможете получить при нажатой клавише Shift. Если кроме цифры на клавишу нанесены два значка, то левый из них получается при работе в английском регистре, а правый - в русском. Кроме этого, работая в английском регистре, вы можете получить остальные нужные символы на русских буквах Х, Ъ, Ж, Э, Б, Ю и еще на паре клавиш. Работая в русском регистре, удобно точку и запятую получать на клавише со знаком вопроса рядом с правой клавишей Shift, причем запятая получится при нажатой клавише Shift. Можно также воспользоваться для этого клавишами Б и Ю

при нажатой клавише Alt.

Удаление букв из текста. Вот вы напечатали всю строку "юный пионер Коля любит Bubble Gum.". Теперь попробуем удалить слово "пионер". Для этого нам нужно уметь перемещать курсор. Для перемещения курсора служат **клавиши перемещения курсора** - четыре клавиши внизу клавиатуры: ← → ↑ ↓. Попробуйте, как они работают. Особой свободы вы не почувствуете. Текстовый курсор передвигается только в пределах введенного текста и еще на строку вниз. (Не обращайте внимания, если Visual Basic изменит вам цвет шрифта).

Перемещать текстовый курсор можно и мышкой - просто щелкните мышкой в дозволенных пределах - текстовый курсор перепрыгнет в место щелчка. Щелкайте аккуратно, удерживая мышь совершенно неподвижной, иначе у вас вместо прыжка текстового курсора может получится выделение черным цветом того или иного фрагмента текста. Не пугайтесь, просто щелкните мышкой еще раз. Получается, что мышиный курсор работает "проводником" для текстового.

Поставьте текстовый курсор на пробел между словом "пионер" и словом "Коля" вплотную к букве р. Мы уже начинаем привыкать, что если нажать какую-нибудь клавишу, то что-то произойдет именно в том месте, где текстовый курсор. Чтобы стереть по очереди все буквы слова "пионер", несколько раз нажмите на клавишу BackSpace. Обратите внимание, что буквы слова "пионер" слева от курсора исчезают по одной, текст справа от курсора смыкается налево, так что пустого пространства на месте слова "пионер" не остается. У вас должно получиться "юный Коля любит Bubble Gum.".

Для стирания символов существует еще одна клавиша - Delete. Иногда она имеет маркировку Del. Давайте сотрем слово "любит". Поставим курсор на пробел между словом "Коля" и словом "любит". Нажмем несколько раз на Delete. Слово любит "стерлось", а текст справа от курсора снова сомкнулся налево.

Таким образом, клавиша BackSpace стирает символ слева от курсора а клавиша Delete – справа. В обоих случаях текст справа от курсора смыкается налево к курсору.

Вставка букв в текст. Теперь у нас на экране строка "юный Коля Bubble Gum.". Давайте вставим перед словом "Коля" слово "бойскаут". Для этого поставим курсор на пробел перед словом "Коля" вплотную к букве "К". После этого напечатаем слово "бойскаут" и пробел. Мы увидим, что буквы слова "бойскаут" появляются на месте курсора, "Коля" вместе с "Bubble Gum" подвинулись направо и мы достигли поставленной цели. Теперь у нас на экране строка "юный бойскаут Коля Bubble Gum.".

Этот способ работы текстового редактора, когда вставляемый текст отодвигает вправо остальной текст, называется **режимом вставки**. Если вы нажмете на клавишу Insert, иногда маркируемую Ins, то перейдете в **режим замещения**, когда текст не будет отодвигаться, а "бойскаут" сотрет "Колю". В этом режиме курсор увеличивает свою толщину и целиком покрывает букву, которой предстоит быть замещенной. Чтобы вернуться в режим вставки, еще раз нажмите на Insert.

А теперь вставьте в подходящее место предложения слово «ненавидит».

Работа с несколькими строками

Ваша задача – ввести такой текст из нескольких строк:

В небе
Облака из серой ваты
Сыровато-сероваты,
Не беда - ведь я привык.

В луже
Эта вата намокает
И волнуясь пробегает
Под водою мой двойник.

Нужную реакцию на могущие возникнуть неожиданности вы можете найти в дальнейшем материале вплоть до конца этого раздела. А пока начнем по порядку.

Ввод нескольких строк. Как сделать так, чтобы, введя слова «В небе», следующие слова начать с новой строки? Для этого можно нажать клавишу ↓. Во многих текстовых редакторах клавиша перемещения курсора ↓ в этой ситуации не помогает. Приходится в тот момент, когда курсор занимает самую правую позицию на строке, нажимать клавишу Enter, по-другому Return, по-другому «Клавиша ввода». Курсор перепрыгивает в начало следующей строки. Введя вторую строку, снова перейдите в начало следующей и так далее.

А теперь введите все восемь строк задания.

Перемещение курсора по экрану. При помощи четырех клавиш перемещения курсора ← → ↑ ↓ потренируйтесь перемещать курсор куда только можно. Вы скоро обнаружите, что курсор можно свободно перемещать только там, где имеется текст. Ни правее, ни больше чем на строчку ниже введенного текста курсор переместить не удается. Поначалу вам это может показаться непривычно и неприятно, и вы захотите расширить поле действия курсора. Удовлетворить вашу прихоть довольно легко.

Подведя курсор в правый край самой нижней строки, нажмите на клавишу ввода несколько раз. У вас ниже текста образовалось несколько невидимых пустых строк, по которым вверх-вниз может свободно ходить курсор.

Я назвал это прихотью, так как при вводе текста это никогда не бывает нужно. Но то, что вы сейчас проделали, вам полезно для свободной ориентации на листе.

Собственно работа с несколькими строками. А теперь вам полезно выполнить несколько заданий.

Чтобы вставить пустые строки между строчкой «Не беда - ведь я привык.» и строчкой «В луже», поставьте курсор в конец первой из этих строк или в начало второй и несколько раз нажмите клавишу ввода.

А как теперь убрать эти пустые строки? Поставьте курсор в начало самой верхней из пустых строк и несколько раз нажмите Delete.

Как разделить строку на две части? Например, вместо «Не беда - ведь я привык.» нужно получить

Не беда –

ведь я привык.

Поставьте курсор перед буквой "в" и нажмите клавишу ввода.

А как слить эти две строки? Поставьте курсор в правый конец верхней из этих строк и нажмите Delete один или несколько раз, пока строки не сольются.

Невидимые символы. Все эти правила могут показаться запутанными и не имеющими внутренней логики. А логика есть. И если вы ее поймете, то и правил запоминать не нужно. Вот она:

Нажатие на клавишу ввода вызывает появление на экране в том месте, где был перед нажатием курсор, специального невидимого символа, точно так же, как нажатие на клавишу пробела вызывает появление невидимого символа - пустого места. Обозначим для удобства символ клавиши ввода - π.

Рассмотрим с новой точки зрения действие различных клавиш:

- Нажатие на любую буквенную клавишу или пробел вызывает вставку в текст на место курсора соответствующей буквы или пробела, а вся правая часть текста сдвигается вправо.
- Нажатие на клавишу ввода вызывает перемещение вниз на одну строку всего текста, находящегося правее и ниже курсора, причем правая часть текста в строке, где был курсор, перемещается не только вниз, но и в начало следующей строки. Отсюда видно, что в текстовом редакторе Visual Basic строка кончается обязательно символом π. Это не относится к тем текстовым редакторам, которые переводят строку автоматически. Кроме как в конце строки, символ π нигде встречаться не может.
- Клавиша Delete стирает любой символ справа от курсора, будь то буква, пробел или π. Стирание символа уничтожает не только сам символ, но и его действие. Поэтому, стерев π, мы выполняем действие, обратное действию клавиши ввода, то есть нижние строки поднимаются, а ближайшая нижняя сливается с текущей.
- Аналогично действует клавиша BackSpace.

Окно кода - маленькое окно на большой лист с текстом

Когда вы вводите большой текст, то в конце концов доходите до нижнего края окна кода. Продолжайте работать как ни в чем не бывало. Перейдя в очередной раз на следующую строку, вы обнаруживаете, что весь текст в окне ушел немного вверх, так что верхняя его часть исчезла из вида. То же самое происходит, когда слишком далеко продолжаешь строку вправо - текст уходит влево.

Впечатление такое, что имеется большой неподвижный лист с текстом, а окно кода является небольшим подвижным окном, через которое вы можете видеть этот лист. Движением окна можно управлять клавишами перемещения курсора или щелкая мышкой по кнопкам полос прокрутки, которые возникают у правого и нижнего края окна кода. Можно также таскать бегунки полос прокрутки.

Копирование перемещение, удаление фрагментов текста

Часто в программах попадаются одинаковые или почти одинаковые фрагменты. Чем вводить их каждый раз заново, лучше скопировать.

Выделение фрагмента: Поставьте курсор мыши в тот момент, когда он имеет форму вертикальной палочки (I), чуть слева от копируемого фрагмента. Затем нажмите левую клавишу мыши и, не отпуская ее, ведите мышь на конец копируемого фрагмента. Фрагмент будет выделен темным цветом. Если в процессе ваша рука дрогнет, то на экране будут происходить страшные вещи. Сохраняйте хладнокровие и ни за что не отпускайте клавишу мыши. Когда вы доведете ее до последней буквы фрагмента, все уляжется. Можете отпускать мышь. Выделенный же фрагмент Visual Basic умеет удалять, перемещать и копировать в другое место. Следующим образом:

Копирование: Поставьте курсор мыши в тот момент, когда он имеет свою обычную форму наклоненной налево стрелки (↖), острием стрелки на темный копируемый фрагмент. Щелкните правой клавишей мыши. В появившемся контекстном меню выберите пункт Copy. . Visual Basic запомнил фрагмент в специальном месте памяти, которое называется **буфер обмена**. Теперь щелкните правой клавишей мыши в том месте окна, куда вы хотите скопировать фрагмент. Убедившись, что текстовый курсор мигает в нужном месте, в появившемся контекстном меню выберите пункт Paste. То, что было в буфере обмена, переносится в нужное место.

Перемещение: Вместо Copy выбирайте Cut. Остальное - аналогично перемещению. Более быстрый способ: просто перетащите мышкой фрагмент в нужное место.

Если у вас на экране несколько окон, то вы точно так же можете копировать и переносить фрагменты из одного окна в другое. Особенно полезно «передирать» целые программы из окна помощи в свое окно кода и запускать получившийся проект.

Удаление: Выделив фрагмент, выберите в контекстном меню Delete или щелкните клавишу Delete на компьютере.

Вместо щелчков по правой клавише мыши вы можете выбрать пункты меню Edit (Copy, Cut, Paste, Delete) или щелкайте по соответствующим кнопкам на панели инструментов.

Волшебные кнопки отмены и возврата

Иногда так бывает, что в результате неосторожного нажатия на какую-нибудь кнопку в окне редактора ВСЕ СТАНОВИТСЯ ОЧЕНЬ ПЛОХО! То есть, или текст программы безнадежно испорчен, или большой фрагмент этого текста вообще пропал неизвестно куда, или еще что-нибудь. И вы не знаете, как помочь этому горю. В этом случае просто щелкните один или несколько раз по кнопке отмены на панели инструментов. Все вернется на свои места. Для интереса щелкайте по этой клавише, пока щелкается. Вы увидите, что все ваши действия в редакторе отменяются одно за другим. Чтобы вернуть их, пощелкайте по кнопке возврата .

Решение заданий

1.

```
Private Sub Квадрат_Click()
   Результат.Text = Val(Число1.Text) * Val(Число1.Text)
End Sub
```

2.

```
Private Sub СБРОС_Click()
   Число1.Text = ""
   Число2.Text = ""
   Результат.Text = ""
End Sub
```

5.

```
Private Sub Кл_вычитания_Click()
   Результат.Text = Val(Число1.Text) - Val(Число2.Text)
   Кл_вычитания.Left = 2000
   Кл_вычитания.Caption = "Ой!"
End Sub

Private Sub СБРОС_Click()
   Число1.Text = ""
   Число2.Text = ""
   Результат.Text = ""
   Кл_вычитания.Left = 3400
   Кл_вычитания.Caption = "-"
End Sub
```

6.

0

7.

Будет напечатано число 211.

8.

- 1001
- -100
- 15 -10

9.

82

10.

```
Dim a As Long
Dim b As Long
Private Sub Command1_Click()
a = 9000000
b = 1000
b = b + a
Debug.Print b
End Sub
```

11.

```
'Задача вычисления средней скорости
```

```
Dim Скорость1 As Double          'Скорость автомобиля на первом участке пути
Dim Время1 As Double             'Время прохождения первого участка
Dim Путь1 As Double              'Длина первого участка
Dim Скорость2 As Double          'Скорость автомобиля на втором участке пути
Dim Время2 As Double             'Время прохождения второго участка
Dim Путь2 As Double              'Длина второго участка
Dim Средняя_скорость As Double 'Средняя скорость автомобиля

Private Sub Command1_Click()
        'Задание исходных данных
  Скорость1 = 80
  Время1 = 3
  Скорость2 = 90
  Время2 = 2
        'Вычисление результата
  Путь1 = Скорость1 * Время1
  Путь2 = Скорость2 * Время2
  Средняя_скорость = (Путь1 + Путь2) / (Время1 + Время2)
        'Отображение результата
  Debug.Print Средняя_скорость
End Sub
```

12.

```
  'Задача: В самом углу прямоугольного двора стоит прямоугольный дом.
  'Подсчитать площадь дома, свободную площадь двора и длину забора.
        'Объявляем переменные величины
Dim Длина_двора As Integer
Dim Ширина_двора  As Integer
Dim Площадь_двора  As Integer
Dim Периметр_двора  As Integer
Dim Длина_дома As Integer
Dim Ширина_дома  As Integer
Dim Площадь_дома  As Integer
Dim Полпериметра_дома  As Integer
Dim Свободная_площадь_двора  As Integer
Dim Длина_забора  As Integer

Private Sub Command1_Click()
        'Ввод исходных данных
  Длина_двора = InputBox("Введите длину двора")
  Ширина_двора = InputBox("Введите ширину двора")
  Длина_дома = InputBox("Введите длину дома")
  Ширина_дома = InputBox("Введите ширину дома")
        'Вычисление результатов
  Площадь_двора = Длина_двора * Ширина_двора
  Площадь_дома = Длина_дома * Ширина_дома
  Периметр_двора = 2 * (Длина_двора + Ширина_двора)
  Полпериметра_дома = Длина_дома + Ширина_дома
  Свободная_площадь_двора = Площадь_двора - Площадь_дома
  Длина_забора = Периметр_двора - Полпериметра_дома
        'Отображение результатов
  Text1.Text = Площадь_дома
  Text2.Text = Свободная_площадь_двора
  Text3.Text = Длина_забора
End Sub
```

13.

```
'Задача вычисления длины окружности и площади круга
Dim R As Double                  'Радиус
Dim L As Double                  'Длина окружности
Dim S As Double                  'Площадь круга
Dim Pi As Double                 'Число "пи", равное 3,14

Private Sub Command1_Click()
        'Задание исходных данных
  R = Text1.Text                 'Величину радиуса берем из текстового поля
  Pi = 3.1416
```

```
        'Вычисление результатов
  L = 2 * Pi * R
  S = Pi * R ^ 2
        'Отображение результатов с 5 знаками после запятой
  Print "Длина окружности ="; Format(L, "0.00000")
  Print "Площадь круга ="; Format(S, "0.00000")
End Sub
```

14.

```
Dim nazvanie1 As String          'Название первой планеты
Dim nazvanie2 As String          'Название второй планеты
Dim r1 As Double                 'Радиус орбиты первой планеты
Dim r2 As Double                 'Радиус орбиты второй планеты
Dim v1 As Double                 'Скорость первой планеты
Dim v2 As Double                 'Скорость второй планеты
Dim t1 As Double                 'Продолжительность года первой планеты
Dim t2 As Double                 'Продолжительность года второй планеты
Dim Pi As Double                 'Число "пи", равное 3,14

Private Sub Command1_Click()
        'Задание исходных данных
  nazvanie1 = InputBox("Введите название первой планеты")
  r1 = InputBox("Введите радиус орбиты первой планеты (в миллионах километров)")
  v1 = InputBox("Введите скорость первой планеты (в миллионах километров в сутки)")
  nazvanie2 = InputBox("Введите название второй планеты")
  r2 = InputBox("Введите радиус орбиты второй планеты (в миллионах километров)")
  v2 = InputBox("Введите скорость второй планеты (в миллионах километров в сутки)")
  Pi = 3.1416
        'Вычисление результатов
  t1 = 2 * Pi * r1 / v1     'год = время 1 оборота = длина орбиты / скорость,
  t2 = 2 * Pi * r2 / v2     'а длина орбиты равна   два пи * радиус
        'Отображение результатов в двух вариантах:
  Print "Продолжительность года на планете "; nazvanie1; " - "; Format(t1, "0"); _
                    " суток, а на планете "; nazvanie2; " - "; Format(t2, "0"); " суток"
  Text1.Text = "Продолжительность года на планете " + nazvanie1 + " - " + Format(t1, "0") _
                    + " суток, а на планете " + nazvanie2 + " - " + Format(t2, "0") + " суток"
End Sub
```

15.

8

16.

29

17.

6

18.

```
Dim a As Double
Dim b As Double
Private Sub Command1_Click()
  a = InputBox("Введите первое число")
  b = InputBox("Введите второе число")
  If a > b Then Debug.Print a + b Else Debug.Print a * b
  Debug.Print "ЗАДАЧА РЕШЕНА"
End Sub
```

19.

```
Dim a As Double, b As Double, c As Double
Private Sub Command1_Click()
  a = InputBox("Введите первый отрезок")
  b = InputBox("Введите второй отрезок")
  c = InputBox("Введите третий отрезок")
  If  a < b + c  Then  Debug.Print "Достаточно мал"   Else  Debug.Print "Слишком велик"
End Sub
```

20.

```
Dim N As Integer, Число_голов As Integer, Число_глаз As Integer
Private Sub Command1_Click()
   N = InputBox("Введите возраст дракона")
   If   N < 100   Then   Число_голов = 3 * N    Else   Число_голов = 300 + 2 * (N - 100)
   Число_глаз = 2 * Число_голов
   Debug.Print Число_голов, Число_глаз
End Sub
```

21.

```
Private Sub Command1_Click()
   If Command1.Top < 300 Then Command1.Top = Command1.Top + 200
End Sub
```

22.

```
Dim k As Integer
Private Sub Command1_Click()
   Command1.Left = (Form1.Width - 100) * Rnd
   Command1.Top = (Form1.Height - 500) * Rnd
   k = k + 1
   Debug.Print k
End Sub
```

23.

```
Dim   Загаданное_число  As Integer,   Отгаданное_число  As Integer
Private Sub Command1_Click()
   Загаданное_число = Int(2 * Rnd)
   Отгаданное_число = InputBox("Загадано число - 0 или 1. Отгадайте!")
   If Загаданное_число = Отгаданное_число   Then  Debug.Print "Угадал"    Else  Debug.Print "Не угадал"
End Sub
```

24.

```
Private Sub Command1_Click()
   Имя = InputBox("Как вас зовут?")
   If Имя = "Коля" Then
      MsgBox ("Привет!")
   ElseIf Имя = "Вася" Then
      Form1.BackColor = vbGreen
      MsgBox ("Здорово!")
   ElseIf Имя = "John" Then
      MsgBox ("Hi!")
   Else
      MsgBox ("Здравствуйте!")
   End If
End Sub
```

25.

```
Dim imya As String
Dim vozrast As Integer
Private Sub Command1_Click()
   Print "Здравствуй, я компьютер, а тебя как зовут?"
   imya = InputBox("Жду ответа")
   Print "Очень приятно, "; imya; ". Сколько тебе лет?"
   vozrast = InputBox("Жду ответа")
   Print "Ого! Целых"; vozrast; "лет! Ты уже совсем взрослый!"
   If vozrast > 17 Then
      InputBox ("В каком институте ты учишься?")
      Print "Хороший институт"
   Else
      InputBox ("В какой школе ты учишься?")
      Print "Неплохая школа"
   End If
   Print "До следующей встречи!"
End Sub
```

26.

```
Dim a As Double, b As Double, c As Double
```

```
Private Sub Command1_Click()
   a = InputBox("Введите первый отрезок")
   b = InputBox("Введите второй отрезок")
   c = InputBox("Введите третий отрезок")
   If a > b + c Then
      Debug.Print "Треугольника не получится"
   ElseIf b > a + c Then
      Debug.Print "Треугольника не получится"
   ElseIf c > a + b Then
      Debug.Print "Треугольника не получится"
   Else
      Debug.Print "Треугольник получится"
   End If
End Sub
```

27.

Замысловатой принцессе нравятся черноглазые, кроме тех, чей рост находится в пределах от 180 до 184.

28.

```
Private Sub Command1_Click()
   a = InputBox("Введите дальность выстрела")
   If a > 28 And a < 30 Then
      MsgBox ("ПОПАЛ")
   ElseIf a >= 30 Then
      MsgBox ("ПЕРЕЛЕТ")
    ElseIf a >= 0 And a <= 28 Then
      MsgBox ("НЕДОЛЕТ")
   Else
      MsgBox ("НЕ БЕЙ ПО СВОИМ")
   End If
End Sub
```

29.

```
Dim a As String          'Приветствие человека
Dim b  As String         'Ответ компьютера
Private Sub Command1_Click()
   a = InputBox("Компьютер Вас слушает")
   If a = "Привет" Or a = "Здравствуйте" Or a = "Салют" Then
      b = a
   ElseIf a = "Добрый день" Or a = "Приветик" Then
      b = "Салют"
   ElseIf a = "Здравия желаю" Then
      b = "Вольно"
   Else
      b = "Я вас не понимаю"
   End If
   MsgBox (b)
End Sub
```

30.

```
Dim Буква As String
Private Sub Command1_Click()
   Буква = InputBox("Введите строчную букву русского алфавита")
   Select Case Буква
      Case "а", "и", "о", "у", "ы", "э"
         Print "гласный"
      Case "б", "з", "в", "г", "д", "ж", "й", "л", "м", "н", "р"
         Print "согласный звонкий"
      Case "п", "с", "ф", "к", "т", "ш", "х", "ц", "ч", "щ"
         Print "согласный глухой"
      Case "е", "ё", "ю", "я", "ъ", "ь"
         Print "какой-нибудь другой, не знаю"
      Case Else
         Print "Это не строчная буква русского алфавита"
   End Select
End Sub
```

32.

Считаем зайцев
10 зайцев
10 зайцев
11 зайцев
13 зайцев
16 зайцев
20 зайцев
25 зайцев

33.

```
5  Debug.Print "A";
   GoTo 5
```

34.

```
a = 10000
5  Debug.Print a
   a = a - 1
   GoTo 5
```

35.

```
   a = 100
5  Debug.Print Format(a, "0.00000000")
   a = a / 2
   GoTo 5
```

36.

Процедура движения налево отличается от процедуры движения направо одной строкой:

```
m1:     x = x - 0.01            'Компьютер уменьшает горизонтальную координату
```

Процедура движения вниз:

```
Private Sub Command3_Click()
        y = Image1.Top       'Компьютер узнает, откуда начинать движение
m1:     y = y + 0.01         'Компьютер увеличивает вертикальную координату
        Image1.Top = y       'Изображение встает на место, указанное верт. координатой
        GoTo m1
End Sub
```

Процедура движения вверх отличается от процедуры движения вниз одной строкой:

```
m1:     y = y - 0.01            'Компьютер уменьшает вертикальную координату
```

37 В.

```
Private Sub Command1_Click()
            'Печатаем  1  2  3  4  . . . 100:
      a = 1
m1:  Debug.Print a;
      a = a + 1
      If a <= 100 Then GoTo m1

            'Печатаем  99  98  97  96  . . . 1:
      a = 99
m2:  Debug.Print a;
      a = a - 1
      If a >= 1 Then GoTo m2
End Sub
```

38.

```
Dim a As Double
Private Sub Command1_Click()
      a = 0
m:    Debug.Print Format(a, "0.000"), Format(a ^ 2, "0.000000")
      a = a + 0.001
      If a <= 1.00001 Then GoTo m
End Sub
```

Почему я вместо If a<=1 *написал* If a<=1.00001*? Причина в незначительных погрешностях, которые допускает компьютер при действиях с десятичными дробями (о чем я писал в 4.5). На моем компьютере при многократном прибавлении 0.001 значение* a *на некотором этапе перестало быть точным. Конкретнее, у меня получилось вот что:*

0,682 + 0,001 = 0,683000000000001

Вследствие этого, при дальнейшем нарастании a *последнее сложение было таким:*

0,999000000000001 + 0,001 = 1,000000000000001

Легко видеть, что в этом случае для a=1 *задание не было бы выполнено, так как компьютер вышел бы из цикла раньше срока.*

39.

```
Private Sub Command1_Click()
      x = 2700
m1:  y = x / 4 + 20
      z = 2 * y + 0.23
      If y * z < 1 / x Then GoTo m2
      Debug.Print Format(x, "0.000000"), Format(y, "0.000000"), Format(z, "0.000000")
      x = x / 3
      GoTo m1
m2:
End Sub
```

40.

```
         x = 300
m1:      x = x + 0.01
         Image1.Left = x
         If x <= 2000 Then GoTo m1
```

41.

```
Private Sub Command2_Click()
            'Ставим объект в начальную точку:
         x = 300
         Image1.Left = x
         y = 1000
         Image1.Top = y
            'Движемся направо:
m1:      x = x + 0.01
         Image1.Left = x
         If x <= 2000 Then GoTo m1
            'Движемся вниз:
m2:      y = y + 0.01
         Image1.Top = y
         If y <= 1500 Then GoTo m2
End Sub
```

42.

```
Dim Slovo As String
Dim i As Integer
Private Sub Command1_Click()
   i = 1
   Do
      Slovo = InputBox("Введите слово")
      Debug.Print i; Slovo; "!"
      i = i + 1
   Loop Until Slovo = "Хватит"
   Debug.Print "Хватит так хватит"
End Sub
```

43.

```
Dim a As Double
Private Sub Command1_Click()
   a = 0
   Do
      Debug.Print Format(a, "0.000"), Format(a ^ 2, "0.000000")
      a = a + 0.001
   Loop While a <= 1.00001
End Sub
```

44.

```
Private Sub Command2_Click()
   x = 300
   Image1.Left = x
   y = 1000
   Image1.Top = y
            'Движемся направо:
   Do
      x = x + 0.01
      Image1.Left = x
   Loop While x <= 2000
            'Движемся вниз:
   Do
      y = y + 0.01
      Image1.Top = y
   Loop Until y > 1500
End Sub
```

45.

```
   v = 20:    t = 0:       h = 100:        s = 0
   Do
      s = v * t
      h = 100 - 9.81 * t ^ 2 / 2
      Debug.Print Format(t, "0.0"), s, Format(h, "0.000")
      t = t + 0.2
   Loop Until h < 0
```

46.

```
Private Sub Command1_Click()
   Debug.Print "Прямой счет:";
   For i = -5 To 5
      Debug.Print i;
   Next
   Debug.Print "Обратный счет:";
   For i = 5 To -5 Step -1
      Debug.Print i;
   Next
   Debug.Print "Конец счета"
End Sub
```

47.

```
   N = InputBox("Сколько всего кубиков?")
   For i = 1 To N
      a = InputBox("Введите сторону кубика")
      V = a ^ 3          'Объем кубика
      Debug.Print "Сторона кубика ="; a, "Объем кубика ="; V
   Next i
```

48.

Компьютер спросит размеры только одного зала и три раза напечатает его площадь и объем:

Площадь пола= 300 Объем зала= 1200
Площадь пола= 300 Объем зала= 1200
Площадь пола= 300 Объем зала= 1200

49.

Компьютер напечатает результаты только для последнего зала:

Площадь пола= 50 Объем зала= 150

50.

1) *Компьютер напечатает результат, на 10 превышающий правильный*
2) *Компьютер напечатает результат, в 2 раза превышающий правильный*
3) *Компьютер напечатал бы 200 нарастающих значений счетчика*
4) *Компьютер напечатает 1, если последнее число положительное, и 0 - если неположительное*
5) *Компьютер запросит только одно число и напечатает 200, если оно положительное, и 0 - если неположительное*

51.

```
c_полож = 0                    'Обнуляем счетчик положительных чисел
c_отриц = 0                    'Обнуляем счетчик отрицательных чисел
c_больше_10 = 0                'Обнуляем счетчик чисел, превышающих 10
N = InputBox("Сколько всего чисел?")
For i = 1 To N
   a = InputBox("Введите очередное число")
   If a > 0 Then c_полож = c_полож + 1
   If a < 0 Then c_отриц = c_отриц + 1
   If a > 10 Then c_больше_10 = c_больше_10 + 1
Next i
Debug.Print "Из них положительных -"; c_полож; ", отрицательных -"; c_отриц; _
            ", чисел, превышающих десятку -"; c_больше_10
```

52.

```
Dim a As Double, b As Double
Private Sub Command4_Click()
   c = 0                    'Обнуляем счетчик пар
   Do
      a = InputBox("Введите первое число пары")
      b = InputBox("Введите второе число пары")
      If a = 0 And b = 0 Then Exit Do
      If a + b = 13 Then c = c + 1
   Loop
   Debug.Print c
End Sub
```

53.

1) 18
2) 10
3) 5 и 8
4) 3
5) 10
6) 3
7) 5

54.

```
s = 0                                   'Обнуляем сумматор площади пола
For i = 1 To 40
   Dlina = InputBox("Введите длину")
   Shirina = InputBox("Введите ширину")
   s = s + Dlina * Shirina              'Наращиваем сумматор площади пола
Next i
Debug.Print "Общая площадь пола="; s
```

55.

```
N = InputBox("Сколько учеников в классе?")
s = 0                          'Обнуляем сумматор баллов
For i = 1 To N
   Балл = InputBox("Введите оценку по физике")
   s = s + Балл                'Наращиваем сумматор баллов
Next i
Debug.Print "Средний балл по физике ="; Format(s / N, "0.000")
```

56.

```
N = InputBox("Сколько сомножителей?")
proizv = 1       'Сумматор обнуляем, а накопитель произведения приравниваем 1. Почему?
For i = 1 To N
   Число = InputBox("Введите очередной сомножитель")
   proizv = proizv * Число              'Наращиваем произведение
Next i
Debug.Print "Произведение равно"; proizv
```

57.

1)

```
For k = 3 To 8
   For l = 0 To 7
      Debug.Print k; l
   Next l
Next k
```

2)

```
For k = 1 To 3
   For l = 1 To 3
      For m = 1 To 3
         For n = 1 To 3
            Debug.Print k; l; m; n
         Next n
      Next m
   Next l
Next k
```

3)

```
i = 0                    'Обнуляем счетчик
For k = 1 To 3
   For l = 1 To 3
      For m = 1 To 3
         For n = 1 To 3
            i = i + 1
         Next n
      Next m
   Next l
Next k
Debug.Print   i
```

4)

```
i = 0                    'Обнуляем счетчик
For k = 1 To 3
   For l = 1 To 3
      For m = 1 To 3
         For n = 1 To 3
            If  k <= l  And  l <= m  And  m <= n  Then   i = i + 1 :   Debug.Print k; l; m; n
         Next n
      Next m
   Next l
Next k
Debug.Print   i
```

58.

```
N = InputBox("Сколько чисел?")
Min = InputBox("Введите число")
Номер_мин_числа = 1
For i = 2 To N
   chislo = InputBox("Введите число")
   If chislo < Min Then Min = chislo: Номер_мин_числа = i
Next i
Debug.Print Min, Номер_мин_числа
```

59.

```
Dim N As Integer, Min As Integer, Max As Integer, Рост As Integer
Private Sub Command1_Click()
   N = InputBox("Сколько одноклассников?")
   Min = 500   'Заведомо невозможно огромный рост
   Max = 0      'Заведомо ничтожный рост
   For i = 1 To N
      Рост = InputBox("Введите рост")
      If Рост < Min Then Min = Рост
      If Рост > Max Then Max = Рост
   Next i
   If Max - Min > 40 Then Debug.Print "Правда" Else Debug.Print "Неправда"
End Sub
```

60.

```
'На форме Form1 ближе к краю размещены два маленьких объекта-"кнопки"  Image1 и Image2
'с уже загруженными в них картинками, а также большой объект Image3.

Private Sub Image1_Click()              'ЧТО ДОЛЖНО ПРОИЗОЙТИ ПРИ ЩЕЛЧКЕ МЫШКОЙ ПО "КНОПКЕ" Image1:
   Image3.Stretch = False                          'Это чтобы большая "рамка"  Image3 приняла форму и размеры картины
   Image3.Visible = False                          'А это чтобы большая картина не мелькала при преобразованиях Image3
   Image3.Picture = Image1.Picture                  'Копируем картинку с "кнопки" в большую "рамку"
   Image1.BorderStyle = 1                          'А это чтобы мы видели, какую картинку уже смотрели
   Form_Factor = Form1.Width / Form1.Height          'Это продолговатость формы
   Image_Factor = Image3.Width / Image3.Height        'Это продолговатость "рамки" Image3, принявшей картинку
   If Image_Factor > Form_Factor Then                 'Если картинка продолговатей, чем форма, ТО ...
      Image3.Width = 0.9 * Form1.Width                'картинка, конечно, должна быть чуть поуже формы (на 1/10)
      Image3.Left = 0.05 * Form1.Width                'а это для симметричности по горизонтали (на 1/20 от левого края)
      Image3.Height = Image3.Width / Image_Factor      'А это чтобы не исказились пропорции картинки
      Image3.Top = (Form1.Height - Image3.Height) / 2  'А это для симметричности по вертикали
   Else                                                                                    'ИНАЧЕ ...
      Image3.Height = 0.9 * Form1.Height              'Картинка, конечно, должна быть чуть покороче формы (на 1/10)
      Image3.Top = 0.05 * Form1.Height                 'А это для симметричности по вертикали (на 1/20 от верхнего края)
      Image3.Width = Image3.Height * Image_Factor     'А это чтобы не исказились пропорции картинки
      Image3.Left = (Form1.Width - Image3.Width) / 2   'А это для симметричности по горизонтали
   End If
   Image3.Stretch = True    'А это для того, чтобы картина приняла размеры "рамки" после ее успешных преобразований
   Image3.Visible = True     'А вот теперь можно полюбоваться картиной
End Sub

Private Sub Image2_Click()              'ЧТО ДОЛЖНО ПРОИЗОЙТИ ПРИ ЩЕЛЧКЕ МЫШКОЙ ПО "КНОПКЕ" Image2:
   Image3.Stretch = False
   Image3.Visible = False
   Image3.Picture = Image2.Picture
   Image2.BorderStyle = 1
   Form_Factor = Form1.Width / Form1.Height
   Image_Factor = Image3.Width / Image3.Height
   If Image_Factor > Form_Factor Then
      Image3.Width = 0.9 * Form1.Width
      Image3.Left = 0.05 * Form1.Width
      Image3.Height = Image3.Width / Image_Factor
      Image3.Top = (Form1.Height - Image3.Height) / 2
   Else
      Image3.Height = 0.9 * Form1.Height
      Image3.Top = 0.05 * Form1.Height
      Image3.Width = Image3.Height * Image_Factor
      Image3.Left = (Form1.Width - Image3.Width) / 2
   End If
   Image3.Stretch = True
   Image3.Visible = True
End Sub
```

61.

```
Private Sub Command1_Click()
   BackColor = vbWhite                      'красим форму в белый цвет
   Circle (3300, 1200), 400                 'голова
   DrawWidth = 5                            'увеличиваем толщину линий и точек
   PSet (3450, 1100)                        'глаз
   PSet (3150, 1100)                        'глаз
   Line (3200, 1400)-(3400, 1400)           'рот
   DrawWidth = 1                            'возвращаем обычную толщину линий и точек
   ForeColor = vbRed                        'красный цвет линий и текста
   Line (3300, 1200)-(3300, 1300)           'нос
   Line (3300, 1200)-(3050, 1300)           'нос
   Line (3300, 1300)-(3050, 1300)           'нос
   ForeColor = vbBlack                      'черный цвет линий и текста
   Circle (3300, 2200), 600                 'середина
   Line (3500, 1630)-(4550, 1830), , B      'рука
   Line (2030, 1630)-(3080, 1830), , B      'рука
   FillStyle = vbSolid                      'приказ рисовать элементы со сплошной (vbSolid) заливкой
   FillColor = vbYellow                     'желтая заливка
   Line (3000, 300)-(3600, 800), , B        'шапка
```

```
   FillColor = RGB(220, 220, 220)          'серая заливка
   Circle (3300, 3600), 800                'низ
   DrawWidth = 3                           'увеличиваем толщину линий и точек
   ForeColor = vbBlue                      'синий цвет линий и текста
   Line (2200, 1300)-(1800, 4400)          'посох
   Font = "Times"                          'название шрифта
   Font.Italic = True                      'курсив
   Font.Bold = True                        'полужирный
   Font.Size = 14                          'размер шрифта
   CurrentX = 2700                         'координаты начала печати
   CurrentY = 3300
   Print "Снеговик"
   CurrentX = 2830
   Print "Ефрем"
End Sub
```

62.

```
Dim c As Long, R As Long, G As Long, B As Long
Private Sub Command1_Click()
   x = InputBox("Введите горизонтальную координату точки")
   y = InputBox("Введите вертикальную координату точки")
   c = Point(x, y)                 'Определяем код цвета заданной точки
   R = c Mod 256                   'Количество красного
   BG = c \ 256                    'Промежуточный результат
   G = BG Mod 256                  'Количество красного
   B = BG \ 256                    'Количество красного
   Debug.Print c, R, G, B, "Проверка -"; B * 256 * 256 + G * 256 + R
      'Следующие три строки - для проверки на глазок правильности определения R,G,B:
   Circle (x, y), 200
   DrawWidth = 20
   PSet (x, y), RGB(R, G, B)
      'Определяем, какого цвета больше - R,G или B:
   If R > G And R > B Then
      Debug.Print "Красного больше"
   ElseIf G > R And G > B Then
      Debug.Print "Зеленого больше"
   ElseIf B > R And B > G Then
      Debug.Print "Синего больше"
   Else
      Debug.Print "Два самых ярких или три цвета одинаково интенсивны"
   End If
End Sub
```

63.

Программа отличается от той, что в разделе, одним числом:

```
x = x + 120
```

64.

Программа отличается от предыдущей двумя числами:

```
x = 200
   Do Until x > 8000
```

65.

Вместо 100 *пишем* 200.

66.

```
Dim x As Long, y As Long
Private Sub Command1_Click()
   x = 100
   y = 6000
   Do Until x > 9000
      PSet (x, y)
      x = x + 100
      y = y - 60
   Loop
End Sub
```

67.

```
   x = 4000:    y = 3000:    R = 100
   Do Until R > 2500
      Circle (x, y), R
      R = R + 100
   Loop
```

68.

```
Private Sub Command3_Click()
   BackColor = RGB(0, 0, 150)
   ForeColor = vbYellow
      'Компакт-диск:
   x = 4000:    y = 3000:    R = 500
   Do Until R > 2500
      Circle (x, y), R
      R = R + 20
   Loop
      'Летающая тарелка:
   x = 10000:    y = 3000:    R = 500
   Do Until R > 2500
      Circle (x, y), R, , , , 1 / 2
      R = R + 20
   Loop
End Sub
```

69.

```
   x = 4000:    y = 500:    R = 0
   Do Until R > 2500
      Circle (x, y), R, , , , 1 / 2
      R = R + 50
      y = y + 150
   Loop
```

70.

```
   x = 400:    y = 500:    R = 0
   Do Until R > 1500
      Circle (x, y), R
      R = R + 20
      y = y + 60
      x = x + 120
   Loop
```

71.

```
   y = 0                        'Разлиновывать начинаем с верхнего края формы
   Do Until y > Height          'Разлиновываем до нижнего края формы
      Line (0, y)-(Width, y)    'Линию проводим до правого края формы
      y = y + 200               'Расстояние между линиями = 200
   Loop
```

72.

```
Private Sub Command2_Click()
      'Разлиновываем горизонтальными линиями:
   y = 0                        'Разлиновывать начинаем с верхнего края формы
   Do Until y > Height          'Разлиновываем до нижнего края формы
      Line (0, y)-(Width, y)    'Линию проводим до правого края формы
      y = y + 200               'Расстояние между линиями = 200
   Loop
      'Разлиновываем вертикальными линиями:
   x = 0                        'Разлиновывать начинаем с левого края формы
   Do Until x > Width           'Разлиновываем до правого края формы
      Line (x, 0)-(x, Height) 'Линию проводим до нижнего края формы
      x = x + 200               'Расстояние между линиями = 200
   Loop
End Sub
```

73.

```
Private Sub Command3_Click()
        'Разлиновываем горизонтальными линиями:
   y = 0                          'Разлиновывать начинаем с верхнего края формы
   Do Until y > Height            'Разлиновываем до нижнего края формы
      Line (0, y)-(Width, y)      'Линию проводим до правого края формы
      y = y + 200                 'Расстояние между линиями = 200
   Loop
        'Разлиновываем косыми линиями:
   x = 0                          'Разлиновывать начинаем с левого края формы
   Do Until x > Width + 2000      'Разлиновываем до правого края формы с запасом в 2000
      Line (x, 0)-(x - 2000, Height)  'Линию проводим наискосок до нижнего края формы
      x = x + 200                 'Расстояние между линиями = 200
   Loop
End Sub
```

74.

```
   x = 100                                  'Квадраты начинаем рисовать от левого края формы
   Do Until x > 8000                        'Рисуем их до координаты 8000
      Line (x, 3000)-(x + 1000, 4000), , B  'Ширина квадрата = 1000, высота = 4000-3000
      x = x + 1500                          'Шаг рисования квадратов = 1500
   Loop
```

75.

```
Dim x As Integer, y As Integer     'Координаты левого верхнего угла каждого из 64 квадратов
Dim i As Integer       'i - номер столбца на доске (от 1 до 8 слева направо)
Dim j As Integer       'j -номер строки на доске (от 1 до 8 сверху вниз)

Private Sub Command2_Click()
   For j = 1 To 8         'Пробегаем 8 клеток по вертикали сверху вниз
      For i = 1 To 8      'Пробегаем 8 клеток по горизонтали слева направо
         x = 1000 * i
         y = 1000 * j
          'ЕСЛИ сумма номеров столбца и строки четная, то заливка квадрата синяя, ИНАЧЕ желтая:
         If (i + j) Mod 2 = 0 Then Цвет_заливки = vbBlue Else Цвет_заливки = vbYellow
         Line (x, y)-(x + 1000, y + 1000), Цвет_заливки, BF     'рисуем закрашенный квадрат,
      Next i
   Next j
End Sub
```

76.

```
Dim x As Integer, y As Integer     'Координаты центров окружностей
Private Sub Command1_Click()
   y = 1000                 'По вертикали ковер простирается от 1000 до 6000 твипов
   Do Until y >= 6000
      x = 1000              'По горизонтали ковер простирается от 1000 до 8000 твипов
      Do Until x >= 8000
         Circle (x, y), 300
         x = x + 150        'Расстояние между центрами окружностей - 150 твипов
      Loop
      y = y + 150
   Loop
End Sub
```

77.

Вместо строки

```
        Circle (x, y), 300
```

пишем строку

```
        If x > 2000 Or y < 5000 Then Circle (x, y), 300
```

78.

Вместо строки

```
        Circle (x, y), 300
```

пишем строку

```
        If (x > 2000 Or y < 5000) And Not (x > 4000 And x < 5000 And y > 3000 And y < 4000) Then Circle (x, y), 300
```

которую можно вольно перевести так:

ЕСЛИ (это не левый нижний угол) И НЕПРАВДА, что (это квадрат в центре), ТО рисуй кружок

79.

```
  Line (2000, 1000)-(6000, 5500), , BF                    'Черный прямоугольник окна
  For i = 1 To 1000
     DrawWidth = Round(2 * Rnd) + 1     'Толщина звезд = 1,2,3
     PSet (2000 + 4000 * Rnd, 1000 + 4500 * Rnd), 16777216 * Rnd 'Откуда взялись числа 4000 и 4500? Вот откуда:
                                                                  '4000=6000-2000,   4500=5500-1000
  Next
```

80.

```
  For i = 1 To 40
     Circle (Width * Rnd, Height * Rnd), 200, , , , 1 / 2
  Next
```

81.

```
Private Sub Command4_Click()
  For i = 1 To 150
     Circle (Width * Rnd, Height * Rnd), 1000 * Rnd, 16777216 * Rnd
  Next
End Sub
```

82.

```
  BackColor = vbBlack                        'Черное небо
  For i = 1 To 200000     'Большое число - чтобы долго рисовалось. Сам процесс приятен.
        'Каждый луч прожектора - отрезок от центральной точки формы (Width / 2, Height / 2)
        'до случайной (Width * Rnd, Height * Rnd):
     Line (Width / 2, Height / 2)-(Width * Rnd, Height * Rnd), 16777216 * Rnd
  Next
```

83.

```
  For i = 1 To 1000
        'Левая треть стога имеет горизонтальные координаты от 0 до 2000,
                 'значит случайная точка внутри этой части - (2000 * Rnd)
        'Правая треть стога имеет горизонтальные координаты от 4000 до 6000,
                 'значит случайная точка внутри этой части - (4000 + 2000 * Rnd)
        'Поскольку стог сделан из сена, то в его цвете преобладают красная и зеленая составляющие, а не синяя
  Line (2000 * Rnd, 6000 * Rnd)-(4000 + 2000 * Rnd, 6000 * Rnd), RGB(100 + 156 * Rnd, 100 + 156 * Rnd, 40 * Rnd)
  Next
```

84.

```
  For i = 1 To 10000
     Line (Width * Rnd, Height * Rnd)-(Width * Rnd, Height * Rnd), 16777216 * Rnd, BF
     For j = 1 To 1000000: Next
  Next
```

85.

```
Private Sub Command1_Click()    'Звездное небо с порцией из 400 звезд
  BackColor = vbBlack
  For i = 1 To 400
     DrawWidth = 1 + Round(2 * Rnd)
     PSet (Width * Rnd, Height * Rnd), 16777216 * Rnd
  Next
End Sub

Private Sub Command2_Click()   'Летающая тарелка
  Randomize
  DrawWidth = 1
    'Сначала подбираем случайный радиус внутреннего отверстия тарелки:
  r0 = 500 * Rnd
    'Теперь назначаем случайные координаты тарелки:
  x = Width * Rnd
  y = Height * Rnd
    'Теперь начинаем рисовать саму тарелку - концентрические эллипсы
    'с начальным радиусом r0 и  конечным радиусом 4 * r0:
```

```
    r = r0
    Do Until r > 4 * r0
        Circle (x, y), r, vbYellow, , , 1 / 2
        r = r + 15
    Loop
End Sub
```

86.

```
Private Sub Form_Load()
    Звук.DeviceType = "WaveAudio"
    Звук.FileName = "c:\Windows\Media\Chimes.wav"
End Sub

Private Sub Музыкальная_вставка()            'Это требуемая процедура пользователя
    Звук.Command = "Open"
    Звук.Command = "Sound"
    Звук.Command = "Close"
End Sub

Private Sub Command1_Click()
    Музыкальная_вставка
    Picture1.Picture = LoadPicture("c:\temp\Rockies.bmp")
End Sub

Private Sub Command2_Click()
    Музыкальная_вставка
    Picture1.Picture = LoadPicture("c:\temp\Porthole.bmp")
End Sub
```

87.

Я, король Франции, спрашиваю вас - кто вы такие? Вот ты - кто такой?
Я - Атос
А ты, толстяк, кто такой?
А я Портос! Я правильно говорю, Арамис?
Это так же верно, как то, что я -Арамис!
Он не врет, ваше величество! Я Портос, а он Арамис.
А ты что отмалчиваешься, усатый?
А я все думаю, ваше величество - куда девались подвески королевы?
Анна! Иди-ка сюда!!!

88.

```
Private Sub Image1_Click()
    Готовим_рамку_к_приему_фото
    Image3.Picture = Image1.Picture
    Image1.BorderStyle = 1
    Увеличиваем_рамку_и_показываем_фото
End Sub

Private Sub Image2_Click()
    Готовим_рамку_к_приему_фото
    Image3.Picture = Image2.Picture
    Image2.BorderStyle = 1
    Увеличиваем_рамку_и_показываем_фото
End Sub

Private Sub Готовим_рамку_к_приему_фото()
    Image3.Stretch = False
    Image3.Visible = False
End Sub

Private Sub Увеличиваем_рамку_и_показываем_фото()
    Form_Factor = Form1.Width / Form1.Height
    Image_Factor = Image3.Width / Image3.Height
    If Image_Factor > Form_Factor Then
        Image3.Width = 0.9 * Form1.Width
        Image3.Left = 0.05 * Form1.Width
```

```
      Image3.Height = Image3.Width / Image_Factor
      Image3.Top = (Form1.Height - Image3.Height) / 2
   Else
      Image3.Height = 0.9 * Form1.Height
      Image3.Top = 0.05 * Form1.Height
      Image3.Width = Image3.Height * Image_Factor
      Image3.Left = (Form1.Width - Image3.Width) / 2
  End If
  Image3.Stretch = True
  Image3.Visible = True
End Sub
```

90.

```
Dim Otstup As Integer   'Расстояние от края формы до центра окружностей
Dim Razmer As Integer  'Радиус самой большой окружности
Dim Tsvet As Long

Private Sub Рисуем_значок_друга()
   Otstup = 300
   Razmer = 200
   Tsvet = vbRed
   Picture1.Circle (Otstup, Otstup), Razmer * 1 / 4, Tsvet
   Picture1.Circle (Otstup, Otstup), Razmer * 2 / 4, Tsvet
   Picture1.Circle (Otstup, Otstup), Razmer * 3 / 4, Tsvet
   Picture1.Circle (Otstup, Otstup), Razmer * 4 / 4, Tsvet
End Sub

Private Sub Command3_Click()
   Picture1.Picture = LoadPicture("c:\temp\Balloons.bmp")
   Рисуем_значок_друга
   Picture1.Print , "12.08.2001"
End Sub
```

91.

```
Private Sub Form_Load()
   Звук.DeviceType = "WaveAudio"
End Sub

Private Sub Музыкальная_вставка(Звуковой_файл As String)
   Звук.FileName = Звуковой_файл
   Звук.Command = "Open"
   Звук.Command = "Sound"
   Звук.Command = "Close"
End Sub

Private Sub Command1_Click()
   Музыкальная_вставка "c:\Windows\Media\Chimes.wav"
   Picture1.Picture = LoadPicture("c:\temp\Rockies.bmp")
End Sub

Private Sub Command2_Click()
   Музыкальная_вставка "c:\Windows\Media\Tada.wav"
   Picture1.Picture = LoadPicture("c:\temp\Porthole.bmp")
End Sub
```

92.

```
Private Sub Рисуем_значок_друга(Otstup As Integer, Razmer As Integer, Tsvet As Long)
   Picture1.Circle (Otstup, Otstup), Razmer * 1 / 4, Tsvet
   Picture1.Circle (Otstup, Otstup), Razmer * 2 / 4, Tsvet
   Picture1.Circle (Otstup, Otstup), Razmer * 3 / 4, Tsvet
   Picture1.Circle (Otstup, Otstup), Razmer * 4 / 4, Tsvet
End Sub

Private Sub Command3_Click()
   Picture1.Picture = LoadPicture("c:\temp\Balloons.bmp")
   Рисуем_значок_друга 300, 200, vbRed
```

```
   Picture1.Print , "12.08.2001"
End Sub
```

93.

```
Private Sub Крестик(x As Integer, y As Integer, Размер As Integer)
      'Крестик - это 2 пересекающихся отрезка (Line)
   Line (x, y + Размер / 2)-(x, y - Размер / 2)
   Line (x + Размер / 2, y)-(x - Размер / 2, y)
End Sub

Private Sub Треугольник(x As Integer, y As Integer, Размер As Integer)
      'Треугольник - это 3 отрезка (Line) с общими концами
      'x и y - координаты левого нижнего угла треугольника
   Line (x, y)-(x + Размер, y)
   Line (x, y)-(x + Размер / 2, y - Размер)
   Line (x + Размер, y)-(x + Размер / 2, y - Размер)
End Sub

Private Sub Command1_Click()
   Крестик 4000, 2000, 400
   Треугольник 3000, 1000, 800
End Sub
```

94.

```
Dim a As Integer, b As Integer

Private Sub Рисуем_срез(Выбор_цвета As Integer, Насыщенность As Integer)
   Размер = 40              'Это длина стороны квадратика
   For j = 0 To 255         'Внешний цикл - рисует строки квадратиков по вертикали сверху вниз
      y = j * Размер        'Вертикальная координата строки квадратиков
      For i = 0 To 255      'Внутренний цикл - рисует квадратики по горизонтали слева направо
         x = i * Размер     'Горизонтальная координата квадратика
         Select Case Выбор_цвета
            Case 1
               Line (x, y)-(x + Размер, y + Размер), RGB(Насыщенность, i, j), BF  'квадратик
            Case 2
               Line (x, y)-(x + Размер, y + Размер), RGB(i, Насыщенность, j), BF  'квадратик
            Case 3
               Line (x, y)-(x + Размер, y + Размер), RGB(i, j, Насыщенность), BF  'квадратик
         End Select
      Next i
   Next j
End Sub

Private Sub Command1_Click()
   a = InputBox("Введите число 1, 2 или 3. Если фиксированный цвет красный, то 1, если зеленый - 2, синий -3")
   b = InputBox("Введите насыщенность фиксированного цвета - число от 0 до 255")
   Рисуем_срез  a,  b
End Sub
```

a *и* b - *неудачные имена, так как не говорят о смысле переменных. В будущем вы увидите, что можно было бы использовать уже применяющиеся имена* - Выбор_цвета *и* Насыщенность.

95.

```
Private Sub Command1_Click()
   Debug.Print DateAdd("ww", 52, Date)
End Sub
```

96.

```
Private Sub Command2_Click()
   Дата_рождения = InputBox("Введите дату своего рождения")
   Debug.Print DateDiff("s", Дата_рождения, Now)
End Sub
```

97.

```
Private Sub Command3_Click()
```

```
   Дата_рождения = InputBox("Введите дату своего рождения")
      'Переменная   Сколько_мне_лет   не совсем точно соответствует общепринятому смыслу.
      'Это разность между текущим годом и годом рождения.
   Сколько_мне_лет = DateDiff("yyyy", Дата_рождения, Date)
   День_рождения_в_этом_году = DateAdd("yyyy", Сколько_мне_лет, Дата_рождения)
   День_рождения_в_следующем_году = DateAdd("yyyy", Сколько_мне_лет + 1, Дата_рождения)
   If День_рождения_в_этом_году >= Date Then        'Если день рождения позже сегодняшнего числа
      Сколько_дней_осталось = День_рождения_в_этом_году - Date
   Else
      Сколько_дней_осталось = День_рождения_в_следующем_году - Date
   End If
   Debug.Print Сколько_дней_осталось
End Sub
```

98.

```
Private Sub Command4_Click()
   Текущая_дата = #1/1/1920#
   Do Until Текущая_дата > #1/1/2940#
      Дата_через_год = DateAdd("yyyy", 1, Текущая_дата)
      Число_дней_в_году = DateDiff("y", Текущая_дата, Дата_через_год)
      Год = DatePart("yyyy", Текущая_дата)
      If (Число_дней_в_году = 366) And Not (Год Mod 4 = 0) Then
         Debug.Print "Лишний високосный год -"; Год, Число_дней_в_году
      End If
      Текущая_дата = Дата_через_год
   Loop
End Sub
```

Эта программа отлавливает лишние високосные года (не кратные 4) между 1920 и 2940 годами.

99.

```
Dim k As Integer

Private Sub Form_Load()
   k = 100
End Sub

Private Sub Timer1_Timer()
   Debug.Print k
   k = k + 1
   If k > 110 Then Timer1.Enabled = False
End Sub
```

100.

```
Dim x As Integer, y As Integer, R As Integer          'Координаты и радиус колес и прямоугольника
Dim Цвет_фигуры As Long, Цвет_фона As Long

Private Sub Form_Load()
   x = 1000:    y = 1500:    R = 200
   DrawWidth = 5                                          'Толщина линии
   Цвет_окружности = vbBlack
   Цвет_фона = BackColor
End Sub

Private Sub Timer1_Timer()
   Circle (x, y), R, Цвет_фигуры                          'Рисуем одно колесо
   Circle (x + 1000, y), R, Цвет_фигуры                   'Рисуем другое колесо
   Line (x - 300, y)-(x + 1300, y - 400), Цвет_фигуры, B  'Рисуем прямоугольник
   For i = 1 To 500000:     Next                          'Пустой цикл
   Circle (x, y), R, Цвет_фона                            'Стираем одно колесо
   Circle (x + 1000, y), R, Цвет_фона                     'Стираем другое колесо
   Line (x - 300, y)-(x + 1300, y - 400), Цвет_фона, B    'Стираем прямоугольник
   x = x + 30                                             'Перемещаемся немного направо
End Sub
```

101.

```
Private Sub Timer1_Timer()
```

```
   Shape1.Top = Shape1.Top - 20
   Shape2.Top = Shape2.Top - 20
End Sub
```

102.

```
Private Sub Timer1_Timer()
   Shape1.Top = Shape1.Top + 20
   Shape2.Left = Shape2.Left + 20
End Sub
```

104.

```
Dim Шаг As Integer, x As Integer

Private Sub Form_Load()
   x = Shape1.Left
   Шаг = 50
End Sub

Private Sub Timer1_Timer()
   x = x + Шаг
   Shape1.Left = x
   If x > Width - Shape1.Width Then Шаг = -50      'Если фигура улетела за правый край формы, то лететь обратно
   If x < 0 Then Шаг = 50                          'Если фигура улетела за левый край формы, то лететь обратно
End Sub
```

105.

```
Dim x As Integer, y As Integer, dx As Integer, dy As Integer
   'dx - шаг шарика по горизонтали,
   'то есть расстояние по горизонтали между двумя последовательными
   'положениями шарика. dy - аналогично по вертикали

Private Sub Form_Load()
   Show    'Чтобы форма показалась на экране до рисования стола
   Line (450, 450)-(6200, 4600), , B   'бортики стола
   x = Image1.Left: y = Image1.Top  'Начальное положение шарика
   dx = 40: dy = 60                 'Направление движения - вправо вниз
End Sub

Private Sub Timer1_Timer()
   x = x + dx: y = y + dy                  'Двигаем шарик
   Image1.Left = x:  Image1.Top = y        'Двигаем шарик
   If x < 500 Or x > 5900 Then dx = -dx 'Ударившись о левый или правый борт,
      'шарик меняет горизонтальную составляющую скорости на противоположную
   If y < 500 Or y > 4300 Then dy = -dy 'Ударившись о верхний или нижний борт,
       'шарик меняет вертикальную составляющую скорости на противоположную

       'Если шарик в левом верхнем углу или в левом нижнем
       'или в правом верхнем или в правом нижнем, то останавливай шарик:
   If (x < 800 And y < 800) Or (x < 800 And y > 4000) _
       Or (x > 5600 And y < 800) Or (x > 5600 And y > 4000) Then Timer1.Enabled = False
End Sub
```

106.

```
Dim x As Long, y As Long, x0 As Long, y0 As Long
Dim t As Double, s As Double, h As Double, v As Double

Private Sub Form_Load()
   Timer1.Enabled = False
   Show
   AutoRedraw = True
   Line (200, 400)-(400, 4400), , B                    'башня
   Line (0, 4400)-(6400, 4400)                         'земля
   x0 = 400:                y0 = 400                   'Координаты верха башни
   v = 20:                  t = 0                      'Начальные скорость и время
   Image1.Left = x0: Image1.Top = y0                   'Начальное положение камня
End Sub

Private Sub Command1_Click()                           'Бросаем камень
   Timer1.Enabled = True
End Sub
```

```
Private Sub Timer1_Timer()
   s = 40 * v * t:                h = 40 * (100 - 9.81 * t ^ 2 / 2)
   x = x0 + Round(s):             y = y0 + (4000 - Round(h))          'Координаты камня в полете
   Image1.Left = x:               Image1.Top = y
   PSet (x, y)                                                        'След камня в полете
   t = t + 0.1
   If h < 0 Then Timer1.Enabled = False                               'Если камень упал, время останавливается
End Sub
```

108.

```
Private Sub Timer1_Timer()
        Label1.FontSize = Label1.FontSize + 1
        Label1.ForeColor = Label1.ForeColor + 10
End Sub
```

110-111.

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
   If Button = 1 Then PSet (X, Y)                    'Если левая клавиша мыши нажата, то рисуем
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   If Button = 2 Then DrawWidth = DrawWidth + 1      'Если правая клавиша мыши нажата, то увеличиваем толщину линии
End Sub
```

112.

```
       'В режиме проектирования поместим на форму прямоугольник и три круга.
       'Назовем круги Красная_лампа, Желтая_лампа, Зеленая_лампа
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
   Select Case KeyCode
      Case vbKeyR
         Красная_лампа.FillColor = vbRed
         Желтая_лампа.FillColor = vbBlack
         Зеленая_лампа.FillColor = vbBlack
      Case vbKeyY
         Красная_лампа.FillColor = vbBlack
         Желтая_лампа.FillColor = vbYellow
         Зеленая_лампа.FillColor = vbBlack
      Case vbKeyG
         Красная_лампа.FillColor = vbBlack
         Желтая_лампа.FillColor = vbBlack
         Зеленая_лампа.FillColor = vbGreen
   End Select
End Sub
```

113.

```
       'В режиме проектирования поместим на форму два Image и два таймера.
       'Назовем их Самолет, Снаряд, Таймер_самолета, Таймер_снаряда

Private Sub Form_Load()
   Таймер_снаряда.Enabled = False
End Sub

Private Sub Таймер_самолета_Timer()
   Самолет.Left = Самолет.Left - 20
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
      Таймер_снаряда.Enabled = True
End Sub

Private Sub Таймер_снаряда_Timer()
   Снаряд.Top = Снаряд.Top - 50
End Sub
```

115.

1) a(i) = a(i-1) + 4

2) a(i) = 2 * a(i-1)
3) a(i) = 2 * a(i-1) - 1

116-118.

```
Dim t(1 To 7) As Integer

Private Sub Command1_Click()
   t(1) = 8:   t(2) = 14:    t(3) = 19:  t(4) = 22:   t(5) = 25:  t(6) = 28:     t(7) = 26
      'Определим среднегодовую температуру:
   s = 0
   For i = 1 To 7:    s = s + t(i):    Next
   Debug.Print s / 7
      'Определим количество теплых дней в году:
   k = 0
   For i = 1 To 7
      If t(i) > 20 Then k = k + 1
   Next
   Debug.Print k
      'Определим, каким по порядку идет самый жаркий день
   Min = t(1): nomer = 1
   For i = 2 To 7
      If t(i) > Min Then Min = t(i): nomer = i
   Next
   Debug.Print nomer
End Sub
```

119.

```
Dim fib(1 To 70) As Currency
Private Sub Command1_Click()
   fib(1) = 1:   fib(2) = 1
   For i = 3 To 70
      fib(i) = fib(i - 2) + fib(i - 1)
      Debug.Print i, fib(i)
   Next
End Sub
```

120.

```
Dim t(1 To 3, 1 To 4) As Integer
Private Sub Command1_Click()
   t(1, 1) = -8:   t(1, 2) = -14:    t(1, 3) = -19:  t(1, 4) = -18
   t(2, 1) = 25:  t(2, 2) = 28:     t(2, 3) = 26:   t(2, 4) = 20
   t(3, 1) = 11:  t(3, 2) = 18:     t(3, 3) = 20:   t(3, 4) = 25
   Min = t(1, 1): Max = t(1, 1)
   For i = 1 To 3
      For j = 1 To 4
         If t(i, j) > Max Then Max = t(i, j)
         If t(i, j) < Min Then Min = t(i, j)
      Next j
   Next i
   Debug.Print Max - Min
End Sub
```

123.

```
Private Sub Form_Load()
   Label_Минимальная.Caption = HScroll1.Min
   Label_Максимальная.Caption = HScroll1.Max
   Label_Текущая.Caption = HScroll1.Value
End Sub

Private Sub HScroll1_Change()
   Label_Текущая.Caption = HScroll1.Value
End Sub
```

123-1.

```
Private Sub Combo1_Click()
   Combo2.Text = Combo2.List(Combo1.ListIndex)
```

```
End Sub
```

124.

Я

125.

```
Private Sub Command1_Click()                    'Шифруем слово из 6 букв
   s = "Корова"
   Debug.Print Mid(s, 1, 2) + "быр" + Mid(s, 3, 2) + "быр" + Mid(s, 5, 2) + "быр"
End Sub

Private Sub Command2_Click()                    'Шифруем произвольное слово
   s = "Консенсус"
   For i = 1 To Len(s) \ 2                      'Len(s) \ 2  -  это число полных пар букв в слове
      Debug.Print Mid(s, 2 * i - 1, 2) + "быр";  'Печатаем очередную пару букв и "быр"
   Next
            'Допечатываем последнюю нечетную букву, если она есть:
   If Len(s) Mod 2 = 1 Then Debug.Print Right(s, 1)
End Sub
```

126.

```
Dim s As String                         'Исходная строка
Dim s1 As String                        'Результирующая строка

Private Sub Command1_Click()
   s = "Консенсус"
   s1 = ""                              'Результирующую строку строим с нуля
   For i = 1 To Len(s)                  'Просматриваем исходную строку слева направо
      Старый_символ = Mid(s, i, 1)      'Выделяем очередной символ в исходной строке
      If Старый_символ = "я" Then       'Букву я кодируем в букву а:
         Новый_символ = "а"
      Else                              'остальные буквы кодируем, как задано в задаче:
         Новый_символ = Chr(Asc(Старый_символ) + 1)
      End If
      s1 = s1 + Новый_символ            'Наращиваем результирующую строку на очередной символ
   Next
   Debug.Print s1                       'Печатаем результат
End Sub
```

127.

```
Dim SecretNumber As Long                        'Загаданное компьютером число
Dim A As Long                                   'Число - попытка человека
Dim Сообщение As String
Dim Количество_попыток As Integer

Private Sub Form_Load()
   Выбор = MsgBox("Продолжим старую игру?", vbQuestion + vbYesNo)
   If Выбор = vbYes Then Загружаем_сохраненную_игру Else Настраиваем_новую_игру
End Sub

Private Sub Настраиваем_новую_игру()
   Randomize
   SecretNumber = Round(1000000000 * Rnd)       'Компьютер загадывает число
   txtNumber.Text = 0                           'Текстовое поле для ввода человеком числа
   txtMessage.Text = "Попыток не было"          'Текстовое поле для вывода компьютером сообщений
   Количество_попыток = 0
   txtNumberTry.Text = Количество_попыток       'Текстовое поле для вывода количества попыток
   Open App.Path & "\Данные.txt" For Output As #1  'Открыть для записи под номером 1 файл Данные.txt из папки проекта
   Write #1, SecretNumber                       'Запись в файл загаданного числа
End Sub

Sub cmdTry_Click()                              'Нажатие на кнопку попытки
   A = Val(txtNumber.Text)
   If A > SecretNumber Then                     'В этом операторе If вся несложная логика игры
      Сообщение = "Много"
   ElseIf A < SecretNumber Then
```

```
      Сообщение = "Мало"
   Else
      Сообщение = "Вы угадали"
   End If
   txtMessage.Text = Сообщение
   Количество_попыток = Количество_попыток + 1
   txtNumberTry.Text = Количество_попыток
   Write #1, Количество_попыток; A; Сообщение        'Запись в файл данных очередной попытки
End Sub

Private Sub Загружаем_сохраненную_игру()
   Open App.Path & "\Данные.txt" For Input As #1     'Открыть для чтения под номером 1 файл Данные.txt из папки проекта
   Input #1, SecretNumber                            'Чтение из файла загаданного числа
   Show                                              'Чтобы на форме можно было печатать историю игры
   Print "ИСТОРИЯ ИГРЫ"
   Do While Not EOF(1)                               'Выполняй, пока НЕ наступил КОНЕЦ ФАЙЛА 1
      Input #1, Количество_попыток, A, Сообщение     'Чтение из файла  данных очередной попытки
      Print Количество_попыток, A, Сообщение         'Печать на форме истории угадываний
   Loop
   Close #1                                          'Закрыть файл №1
   txtNumber.Text = A
   txtMessage.Text = Сообщение
   txtNumberTry.Text = Количество_попыток
   Open App.Path & "\Данные.txt" For Append As #1    'Открыть для дозаписи под номером 1 файл Данные.txt из папки проекта
End Sub

Private Sub Form_Terminate()
   Close #1                                          'Закрыть файл №1
End Sub
```

128.

```
   'Вариант с использованием массива:
Private Function Fibonacci(Nomer As Integer) As Currency
   Dim fib(1 To 70) As Currency
   fib(1) = 1:    fib(2) = 1
   For i = 3 To Nomer:       fib(i) = fib(i - 2) + fib(i - 1):    Next
   Fibonacci = fib(i - 1)  'Потому  i - 1, что на выходе из цикла i равно Nomer + 1
End Function

   'Вариант без использования массива:
Private Function Fibonacci1(Nomer As Integer) As Currency
   fib1 = 1:    fib2 = 1
   For i = 3 To Nomer
      fib3 = fib1 + fib2
      fib1 = fib2
      fib2 = fib3
   Next
   Fibonacci1 = fib3
End Function

Private Sub Command1_Click()
      Debug.Print Fibonacci(68), Fibonacci1(68)
End Sub
```

129.

```
Dim a(1 To 5) As Integer    'Оценки одного класса
Dim b(1 To 5) As Integer    'Оценки другого класса

Private Function Минимум(c As Variant) As Integer
   Минимум = c(1)
   For i = 2 To 5
      If c(i) < Минимум Then Минимум = c(i)
   Next
End Function

Private Function Максимум(c As Variant) As Integer
```

```
   Максимум = c(1)
   For i = 2 To 5
      If c(i) > Максимум Then Максимум = c(i)
   Next
End Function

Private Function Разница(c As Variant) As Integer
   Разница = Максимум(c) - Минимум(c)
End Function

Private Sub Command1_Click()
   a(1) = 4: a(2) = 5: a(3) = 2: a(4) = 5: a(5) = 4
   b(1) = 4: b(2) = 3: b(3) = 4: b(4) = 4: b(5) = 3
   If Разница(a) < Разница(b)       Then Debug.Print "Первый класс учится ровнее" _
                                    Else Debug.Print "Второй класс учится ровнее"
End Sub
```

130.

```
Dim A(1 To 366) As Integer          'Показания термометра на станции A
Dim B(1 To 366) As Integer          'Показания термометра на станции B

Private Sub Исправление(ByVal Поправка As Integer, ByRef c)
   For i = 1 To 4                   'Для отладки приняли, что в году 4 дня.
      c(i) = c(i) + Поправка
   Next
End Sub

Private Sub Command1_Click()
   A(1) = 24: A(2) = 25: A(3) = 28: A(4) = 25
   B(1) = 14: B(2) = 16: B(3) = 14: B(4) = 17
   Исправление -2, A
   Исправление 3, B
   For i = 1 To 4                   'Распечатываем исправленные значения температур
      Debug.Print A(i), B(i)
   Next
End Sub
```

131.

```
Private Function fib(N As Integer) As Currency
   If N = 1 Or N = 2 Then fib = 1 Else fib = fib(N - 2) + fib(N - 1) 'Изумительная лаконичность!
End Function
      'За изумительную лаконичность расплачиваемся удручающе низким быстродействием.
      'Когда N переваливает за пару десятков, результата приходится ждать.
      'Ничего подобного не было при вычислении чисел Фибоначчи простым циклом.
      'Почему так? А попробуйте подсчитать, сколько в памяти компьютера одновременно
      'находится недовыполненных экземпляров подсчета функции fib.
Private Sub Command1_Click()
      Debug.Print fib(20)
End Sub
```

132.

```
'Используем метод пузырька. У нас уже имеется соответствующая программа для одномерного массива.
'Однако, просто так, в лоб, переделать ее для двумерного массива затруднительно. Чтобы не ломать
'уже готовую структуру процедуры метода пузырька, вообразим, что мы работаем не с двумерным массивом,
'а с одномерным, получившимся считыванием двумерного строчка за строчкой. Далее. Все, что делает метод
'пузырька - это берет элемент одномерного массива с указанным номером и ставит его на новое место
'с указанным номером. Поэтому для правильной работы метода нам достаточно написать функцию "Возьми",
'которая по указанному номеру воображаемого одномерного массива вычисляет номер строки и столбца
'реального двумерного массива и берет оттуда элемент, а также процедуру "Положи", которая путем таких же
'вычислений кладет элемент не на воображаемое место, а на реальное.

Const M = 3                                          'M - число строк в массиве
Const N = 4                                          'N - число столбцов
Dim a(1 To M, 1 To N) As Integer                     'Исходный массив

Private Function Возьми(ByVal Номер As Integer) As Integer     ' "Номер" - воображаемый номер элемента
```

```
   i = (Номер + N - 1) \ N                                    'вычисляется номер строки
   j = Номер Mod N                                            'вычисляется номер столбца
   If j = 0 Then j = N
   Возьми = a(i, j)
End Function

Private Sub Положи(ByVal Номер As Integer, ByVal Элемент As Integer)
        '  "Элемент" - это то, что мы кладем на место, указанное воображаемым номером "Номер"
   i = (Номер + N - 1) \ N
   j = Номер Mod N
   If j = 0 Then j = N
   a(i, j) = Элемент
End Sub

Private Sub Сортируем()    'Сортировка массива методом пузырька. Сравните с ранее написанной процедурой
   For k = M * N To 2 Step -1
      For i = 1 To k - 1
         If Возьми(i) > Возьми(i + 1) Then
            c = Возьми(i)
            Положи i, Возьми(i + 1)
            Положи i + 1, c
         End If
      Next i
   Next k
End Sub

Private Sub Command1_Click()
   a(1, 1) = 28:   a(1, 2) = 14:     a(1, 3) = 49:    a(1, 4) = 18
   a(2, 1) = 29:   a(2, 2) = 28:     a(2, 3) = 36:    a(2, 4) = 20
   a(3, 1) = 45:   a(3, 2) = 15:     a(3, 3) = 20:    a(3, 4) = 25
   Сортируем
   For i = 1 To M      'Распечатываем массив после сортировки
      Debug.Print
      For j = 1 To N
         Debug.Print a(i, j),
      Next
   Next
End Sub
```

Список литературы

1. Н.Г.Волчёнков "Учимся программировать: Visual Basic 5", Москва, ДИАЛОГ-МИФИ, 1998.
   Эта книга - для начинающих программистов.
2. Сайлер, Споттс "Использование Visual Basic 6", Издательский дом "Вильямс", 1999.
   Эта книга - для программистов средней руки.
3. Мак-КинНи Брюс "Visual Basic - крепкий орешек"
   Эта книга - для программистов высшего класса, знатоков программирования для Windows.

Все три книги хорошие.

Предметный указатель