Никита Культин, Лариса Цой

bhv®

Small Basic

+ДИСТРИБУТИВ

Основы теории программирования

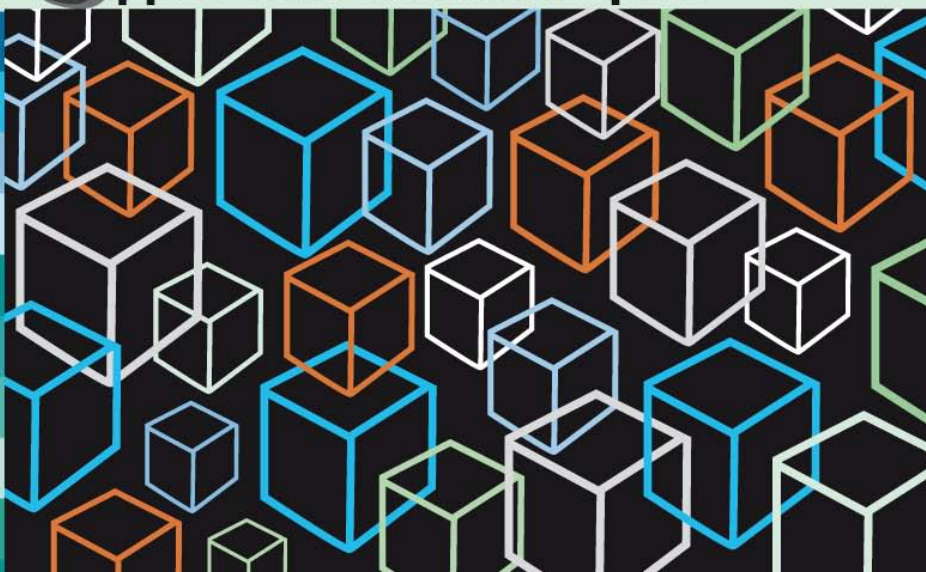
Правила составления программы

Инструкции выбора и циклов

Работа с массивами

Программирование графики

ДЛЯ НАЧИНАЮЩИХ



Никита Культин
Лариса Цой

Small Basic

для начинающих

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.06
ББК 32.973.26-018.2
К90

Культин, Н.

К90 Small Basic для начинающих / Н. Культин, Л. Цой. — СПб.: БХВ-Петербург, 2011. — 256 с.: ил. + Дистрибутив (на DVD)

ISBN 978-5-9775-0664-9

В доступной форме изложены основы теории программирования, приведено описание современного языка программирования для начинающих — Microsoft Small Basic и рассмотрен процесс создания программы от составления алгоритма до отладки. Показано, как записать инструкции программы, использовать инструкции выбора и циклов, ввести исходные данные и вывести результат работы программы на экран, работать с массивами, файлами, графикой и др. На DVD содержится дистрибутив среды разработки Small Basic, примеры, рассмотренные в книге, и справочная информация.

Для начинающих программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии и оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 08.11.10.
Формат 60×90$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 16.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0664-9

Оглавление

Предисловие

Small Basic — что это?

В последнее время возрос интерес к программированию. Это связано с развитием и внедрением в повседневную жизнь информационно-коммуникационных технологий. Если кто-то имеет дело с компьютером, то рано или поздно у него возникает желание, а иногда и необходимость, программировать.

Среди пользователей персональных компьютеров в настоящее время наиболее популярна операционная система Windows, и естественно, что тот, кто хочет программировать, стремится писать программы "под Windows".

И здесь возникает проблема: современные системы программирования, такие как Microsoft Visual Basic, Delphi и, тем более, Microsoft Visual C++, ориентированы на профессиональную разработку и предполагают наличие у пользователя знаний и начального опыта в области программирования. Другими словами, они не подходят для целей обучения программированию. Как быть? Конечно, можно освоить базовые концепции, изучив язык программирования Pascal. Однако в этом случае придется выполнять упражнения и решать задачи в явно устаревшей, ориентированной на работу в операционной системе DOS среде разработки Turbo Pascal, столкнуться с массой проблем при ее использовании в современных операционных системах семейства Windows (начиная с проблемы переключения на русский алфавит при наборе текста программы).

Очевидно, осознав проблему отсутствия современной среды разработки, ориентированной на начинающих, Microsoft предложила свое решение — Microsoft Small Basic.

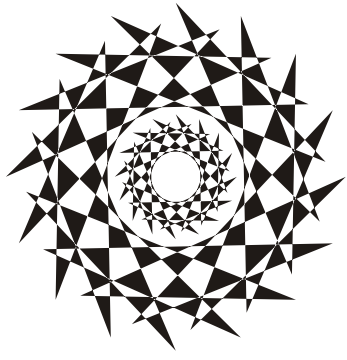
Как показал опыт, полученный в процессе написания этой книги, Microsoft Small Basic, несмотря на то, что он "маленький", вполне приличная (посмотрите программы, приведенные в *главе 10*) и, что важно, доступная для начинающих среда разработки. Она позволяет изучить базовые концепции программирования, алгоритмические структуры, инструкции управления ходом выполнения программы, циклы, научиться работать с массивами, файлами и графикой. В Microsoft Small Basic в качестве языка программирования используется диалект языка Basic, что позволяет в дальнейшем без особых проблем перейти на следующий уровень — начать работать в Microsoft Visual Basic.

Об этой книге

Книга, которую вы держите в руках, — это не описание языка программирования или среды разработки Microsoft Small Basic. Это руководство по программированию для начинающих. В нем рассмотрена вся цепочка, весь процесс создания программы: от разработки алгоритма и составления программы до отладки и переноса готовой программы на другой компьютер.

Цель этой книги — научить основам программирования: использовать операторы выбора, циклов, работать с массивами и файлами.

Чтобы научиться программировать, надо программировать, решать конкретные задачи. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Не занимайтесь просто чтением примеров, реализуйте их с помощью вашего компьютера. Не бойтесь экспериментировать — вносите изменения в программы. Чем больше вы сделаете самостоятельно, тем большему вы научитесь!



Глава 1

Microsoft Small Basic

Установка

Загрузить установочный файл Microsoft Small Basic можно со страницы Центра начинающего разработчика Microsoft (**http://msdn.microsoft.com/ru-ru/beginner/default.aspx**). После того как файл SmallBasic.msi будет загружен на компьютер, надо раскрыть папку, в которой находится файл установщика (в окне **Загрузка завершена** нажать кнопку **Открыть папку**), и сделать двойной щелчок мышью на его значке. В результате будет активизирован процесс установки. Сначала на экране появятся окна **Приветствие** и **Лицензионное соглашение**, затем — окно **Выбор папки для установки**, где можно указать каталог, в который будет установлен Small Basic (по умолчанию он устанавливается в папку C:\Program Files\Microsoft\Small Basic).

Запуск

Чтобы запустить Small Basic, надо сделать щелчок на кнопке **Пуск**, выбрать **Все программы**, раскрыть папку **Small Basic** и сделать щелчок в строке **Microsoft Small Basic** (рис. 1.1).

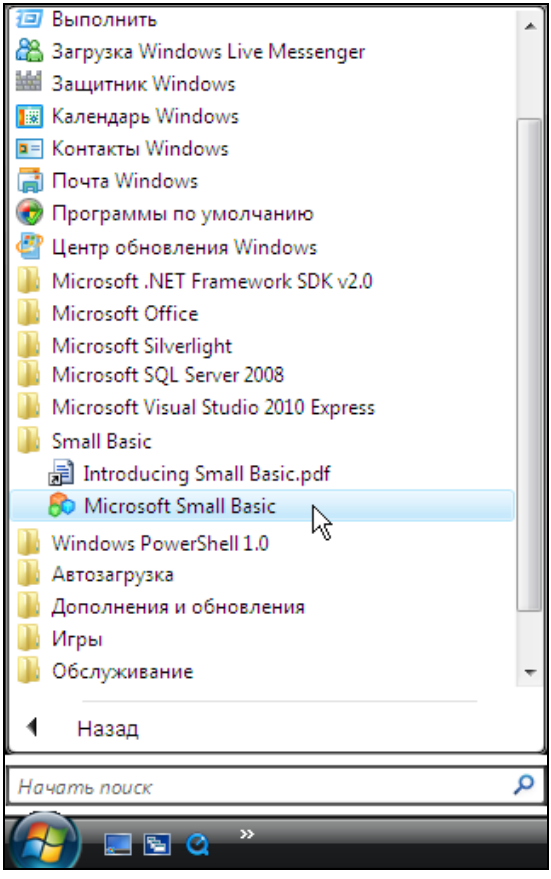


Рис. 1.1. Запуск Small Basic

Совет

Сделайте щелчок на кнопке **Пуск**, выберите **Все программы**, раскройте папку **Small Basic**, в строке **Microsoft Small Basic** сделайте щелчок *правой* кнопкой мыши и в появившемся меню выберите команду **Закрепить в меню "Пуск"**. Команда запуска Small Basic появится в меню **Пуск**.

Первый взгляд

Главное окно Small Basic сразу после запуска среды разработки выглядит так, как показано на рис. 1.2.

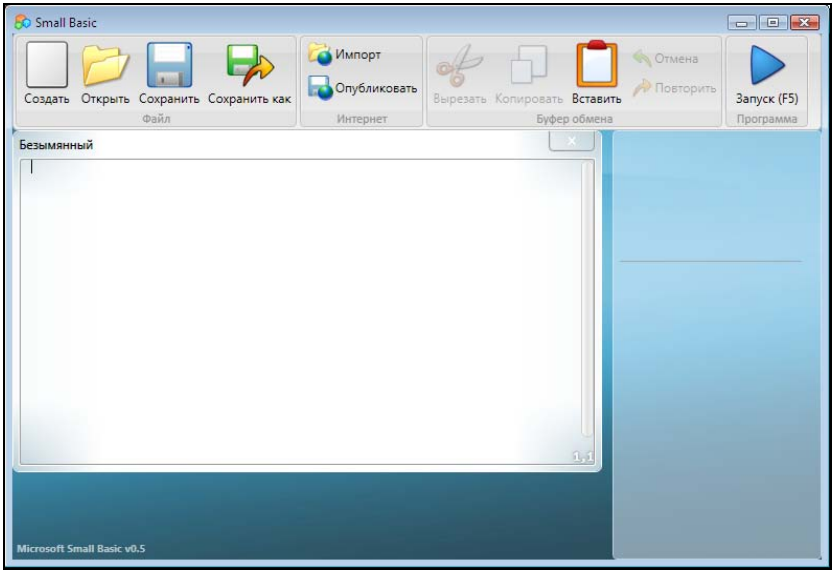


Рис. 1.2. Окно Small Basic в начале работы над новой программой

В верхней части окна находится панель инструментов (рис. 1.3).

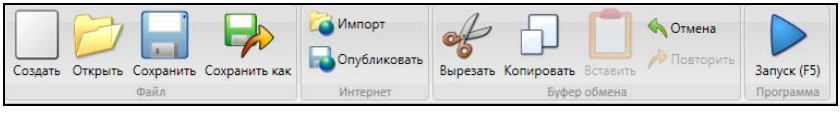


Рис. 1.3. Панель инструментов

Кнопки панели инструментов (табл. 1.1) используются для активизации действий, необходимых в процессе разработки программы.

Таблица 1.1. *Кнопки панели инструментов*

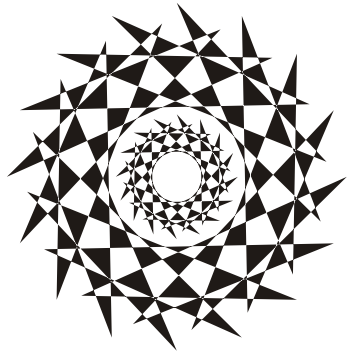
Кнопка	Название	Действие
	**Создать**	Создает и открывает в новом окне файл исходного текста программы
	**Открыть**	Отображает диалоговое окно **Открыть**, в котором можно выбрать файл программы (или текстовый файл) для загрузки в редактор кода
	**Сохранить**	Если файл ни разу не был сохранен на диске, то открывает диалоговое окно **Сохранить**, в котором можно задать имя файла программы. Если файл был хотя бы раз сохранен или файл загружен с дика, сохраняет программу в файле (диалоговое окно **Сохранить** на экране не появляется)
	**Сохранить как**	Отображает диалоговое окно **Сохранить как**, в котором программист может задать имя файла, отличное от того, из которого была загружена программа
	**Опубликовать**	Активизирует процесс публикации (передачи) текста программы на странице **SmallBasic.com/programs**. Опубликованная программа получает уникальный идентификатор
	**Вырезать**	Вырезает из текста программы выделенный фрагмент и помещает его в буфер обмена. Кнопка доступна только в том случае, если фрагмент действительно выделен
	**Копировать**	Копирует выделенный фрагмент и помещает его в буфер обмена. Кнопка доступна только в том случае, если фрагмент действительно выделен
	**Вставить**	Вставляют в текст программы, в то место, где в данный момент находится курсор, копию содержимого буфера обмена. Кнопка доступна только в том случае, если в буфере обмена находится текст, помещенный в него в результате выполнения команды **Вырезать** или **Копировать**

Таблица 1.1 *(окончание)*

Кнопка	Название	Действие
	**Отменить**	Отменяет действия по изменению текста программы (действием считается, например, ввод строки текста, удаление фрагмента)
	**Повторить**	Отменяет изменения в тексте программы, сделанные в результате щелчка на кнопке **Отменить**
	**Запустить**	Активизирует процесс компиляции и в случае отсутствия в программе синтаксических ошибок запускает программу, находящуюся в активном окне редактора кода

В центре окна Small Basic находится окно редактора кода (рис. 1.4), в котором можно набирать текст программы. Слово **Безымянный** в заголовке окна показывает, что текст программы еще ни разу не был сохранен на диске.

Рис. 1.4. Окно редактора кода (текста программы)



Глава 2

Первая программа

Процесс создания программы в Microsoft Small Basic рассмотрим на примере. Создадим программу, при помощи которой можно пересчитать вес из фунтов в килограммы (1 фунт = = 453.59237 грамма).

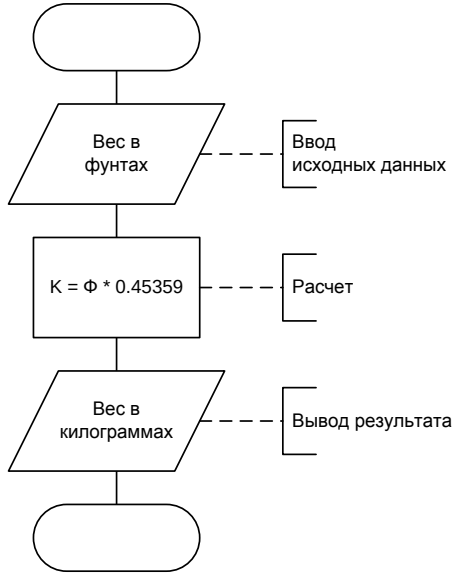


Рис. 2.1. Алгоритм программы пересчета веса из фунтов в килограммы

Пересчет выполняется по формуле:

$$К = Ф \times 0{,}45359,$$

где Ф — вес в фунтах; К — вес в килограммах.

Перед тем как приступить к непосредственной работе в Small Basic на компьютере, рекомендуется разработать (составить) алгоритм решения задачи и написать программу на бумаге. Алгоритм программы пересчета веса из фунтов в килограммы приведен на рис. 2.1, программа — в листинге 2.1. (Текст программы принято называть листингом.)

Листинг 2.1. Пересчет веса из фунтов в килограммы

```
' Пересчет веса из фунтов в килограммы

' Ввод исходных данных
TextWindow.Write("Введите вес в фунтах -> ")
funt = TextWindow.ReadNumber()

' Расчет
kg = funt * 0.45359

' Вывод результата
TextWindow.Write(funt)
TextWindow.Write(" ф. = ")
TextWindow.Write(kg)
TextWindow.WriteLine(" кг")
```

Первая строка программы, текст

```
' Пересчет веса из фунтов в килограммы
```

это комментарий (обратите внимание на апостроф в начале строки, именно он отмечает начало комментария). Комментарии включают в программу, чтобы пояснить назначение программы,

переменных, ключевые действия. Комментарии облегчают восприятие программы, позволяют понять, как она работает. Строка `' Ввод исходных данных` — тоже комментарий. Она выделяет инструкции программы, обеспечивающие ввод исходных данных. Строка `TextWindow.Write("Введите вес в фунтах -> ")` — это команда, которая выводит на экран текст `Введите вес в фунтах ->`. Следующая строка — инструкция ввода данных с клавиатуры. Она записывает в *переменную* `funt` число, введенное пользователем с клавиатуры. Строка `kg = funt * 0.45359` — это инструкция присваивания. Она умножает значение переменной `funt` (число, которое ввел пользователь) на 0.45359 и записывает полученное значение в переменную `kg`. Далее следуют инструкции, обеспечивающие вывод результата расчета на экран. Инструкция `TextWindow.Write(funt)` выводит на экран значение переменной `funt`. Следующие инструкции выводят на экран, соответственно, текст `ф. =`, значение переменной `kg` и текст `кг`. То есть результат расчета выводится в виде

вес в фунтах = вес в килограммах.

Например, если пользователь введет число 2, то программа выведет: `2 ф. = 0.90718 кг`.

Набор текста программы

При запуске Small Basic автоматически открывается окно редактора кода, в котором можно набирать текст программы.

Текст программы в окне редактора кода набирают обычным образом.

Редактор кода автоматически выделяет цветом константы (числовые и символьные) и ключевые слова языка Small Basic (`if`, `then`, `else`, `do` и др.), а также курсивом — комментарии, что облегчает восприятие структуры программы.

В процессе набора текста необходимо соблюдать правила хорошего стиля программирования — использовать несущие смысловую нагрузку имена переменных, добавлять в текст комментарии, записывать инструкции ветвления и циклов с отступами.

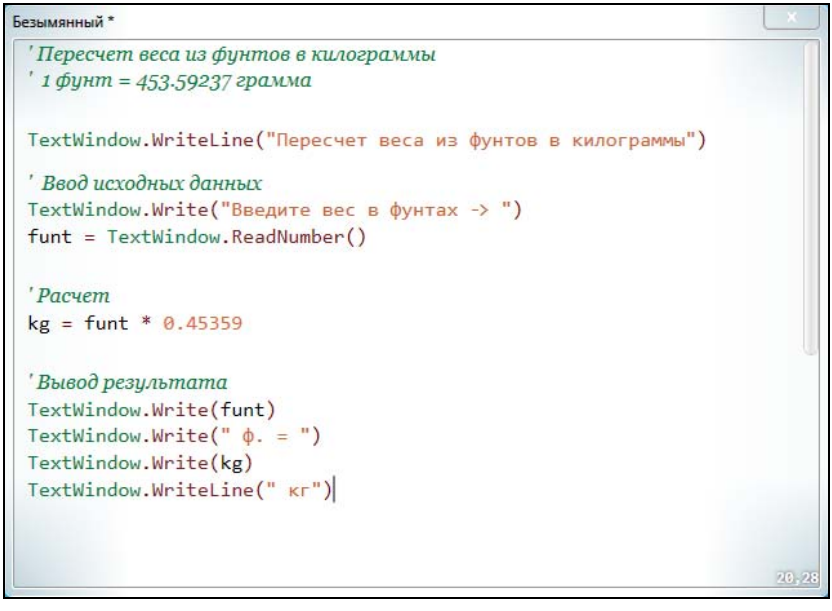


Рис. 2.2. Окно редактора кода

В качестве примера на рис. 2.2 приведено окно редактора кода, в котором находится программа пересчета веса из фунтов в килограммы. В заголовке окна отражается имя файла программы. Слово **Безымянный** показывает что программа еще ни разу не была сохранена или пользователь, хотя и сохранил программу, но не задал имя файла, а оставил имя "по умолчанию", а звездочка — что с момента последнего сохранения в программу были внесены какие-то изменения.

Сохранение программы

После того как тест программы будет набран, его надо сохранить на диске. Для этого нужно сделать щелчок на находящейся в панели инструментов кнопке **Сохранить** (рис. 2.3), в поле **Имя файла** появившегося окна **Сохранить как** (рис. 2.4) ввести имя файла и сделать щелчок на кнопке **OK**.

Рис. 2.3. Чтобы сохранить текст программы, надо сделать щелчок на кнопке **Сохранить**

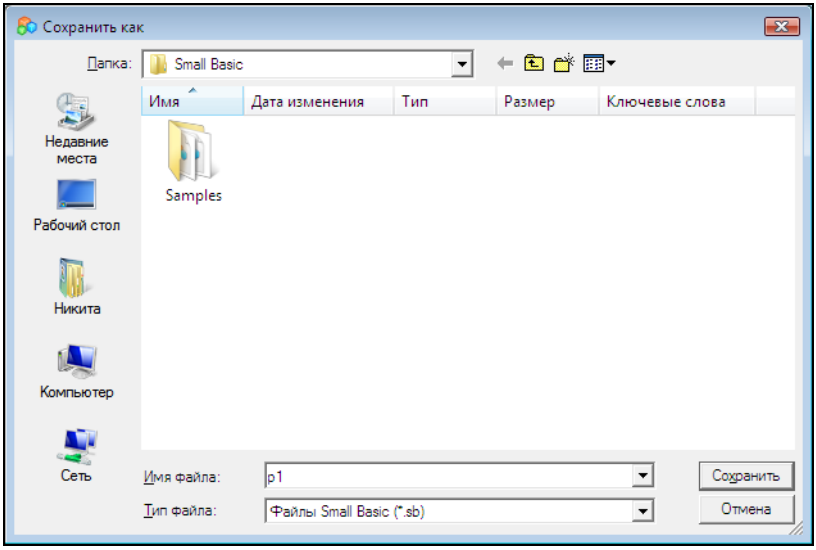


Рис. 2.4. Сохранение программы

Расширение имени файла (sb, сокращение от Small Basic) можно не указывать, оно будет добавлено к имени автоматически. Следует обратить внимание, после того как программа будет сохранена на диске, имя файла появится в заголовке окна редактора кода.

Следует обратить внимание на то, что по умолчанию среда разработки предложит сохранить файл программы в папке Документы\Small Basic. Папка Small Basic создается автоматически в папке Документы в процессе установки Microsoft Small Basic на компьютер.

Запуск программы

Чтобы увидеть, как работает программа, текст которой находится в окне редактора кода, надо сделать щелчок на находящейся в панели инструментов кнопке **Запуск** (рис. 2.5) или нажать клавишу <F5>. Если в программе нет ошибок, то на экране появится окно программы (рис. 2.6).

Рис. 2.5. Чтобы запустить программу, надо нажать кнопку **Запуск**

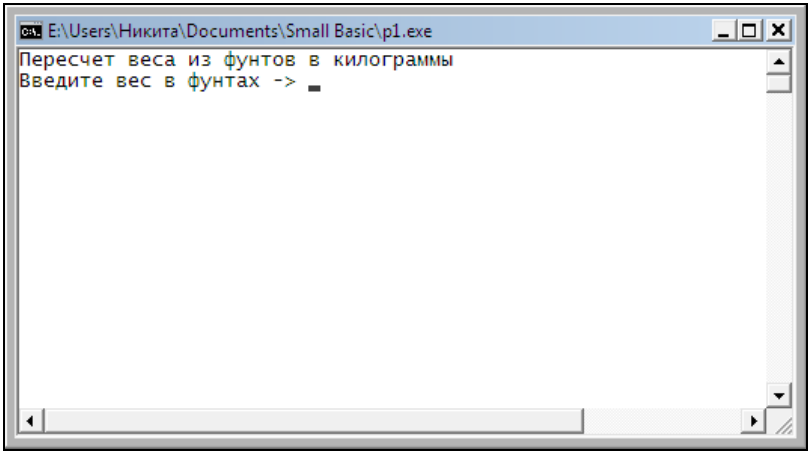


Рис. 2.6. Окно программы пересчета веса из фунтов в килограммы сразу после ее запуска

Ошибки в программе

Как было сказано ранее, если в программе нет ошибок, то в результате нажатия кнопки **Запуск** программа запускается. Однако если в программе есть ошибки, то она не запускается, а в нижнюю часть окна редактора кода выводится сообщение об ошибке. В сообщении указывается номер строки программы, в которой обнаружена ошибка, и номер ошибочного символа. В качестве примера на рис. 2.7 приведено окно редактора с сообщением об ошибке. Ошибка заключается в использовании в качестве десятичного разделителя при записи дробного числа неверного символа — запятой.

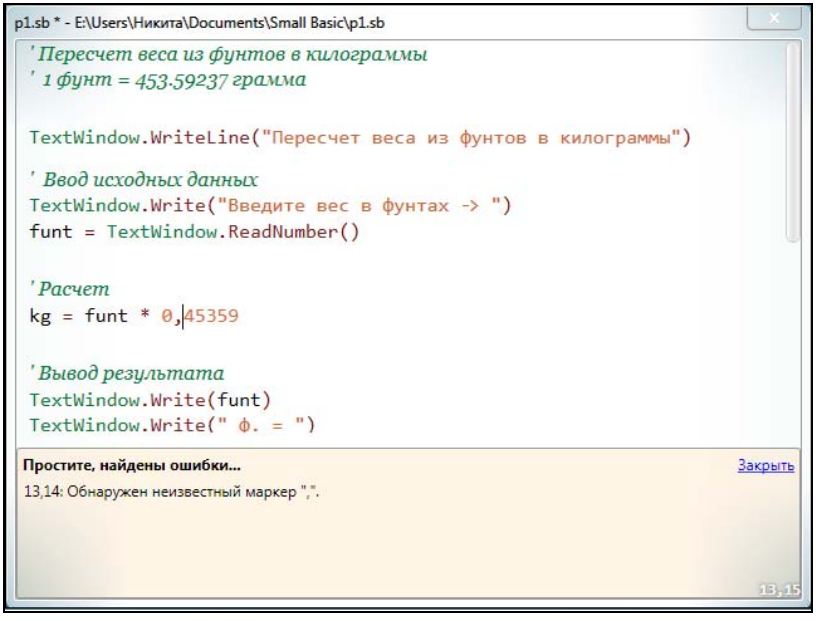


Рис. 2.7. Пример сообщения об ошибке в программе

После анализа причины ошибки и ее устранения (в приведенном примере надо заменить запятую точкой) нужно снова попытаться запустить ее. Таким образом, последовательно исправляя обнаруживаемые средой разработки ошибки, можно добиться того, что в программе их не будет.

В табл. 2.1 приведены сообщения о наиболее типичных ошибках.

Таблица 2.1. *Сообщения о типичных ошибках*

Сообщение об ошибке	**Вероятная причина**
Обнаружен неизвестный маркер	В программе обнаружен непредусмотренный синтаксисом языка программирования символ. Например, при записи дробного числа в качестве десятичного разделителя вместо точки указана запятая
Переменная ... используется, но ей не было присвоено значение	В программе нет инструкции, которая задает значение переменной — инструкции присваивания или инструкции ввода значения с клавиатуры. Неправильно записано имя переменной. Например: исходные данные вводятся в переменную `funt`, а формула пересчета веса из фунтов в килограммы записана так: `kg = f * 0.45359` Результат расчета находится в переменной `kg`, а в инструкции вывода значения переменной на экран (`TextWindow.Write` или `TextWindow.WriteLine`) указана переменная `k`
Здесь ожидается RightParens	В выражении (формуле) нарушен баланс скобок, нет необходимой правой (закрывающей) скобки

Различают синтаксические и алгоритмические ошибки. *Синтаксические ошибки* — это ошибки записи инструкций программы. *Алгоритмические ошибки* — это ошибки логики работы программы, например ошибки записи формул. Синтаксические ошибки выявляет среда разработки. С алгоритмическими ошибками дело обстоит иначе. Обнаружить их можно только в процессе *тестирования* программы, путем сравнения результата "выданного" программой с данными "ручного" расчета. Отсутствие в программе синтаксических ошибок не гарантирует правильную работу программы.

Завершение работы

Чтобы завершить работу со Small Basic, надо просто закрыть его главное окно.

Если находящаяся в окне редактора кода программа еще ни разу не была сохранена на диске или с момента последнего сохранения в нее были внесены какие-либо изменения, то на экране появится вопрос о необходимости сохранения программы (рис. 2.8). Чтобы сохранить изменения, надо сделать щелчок на кнопке **Да**. Щелчок на кнопке **Отменить** отменяет команду завершения работы со Small Basic (в результате вновь становится доступным окно редактора кода).

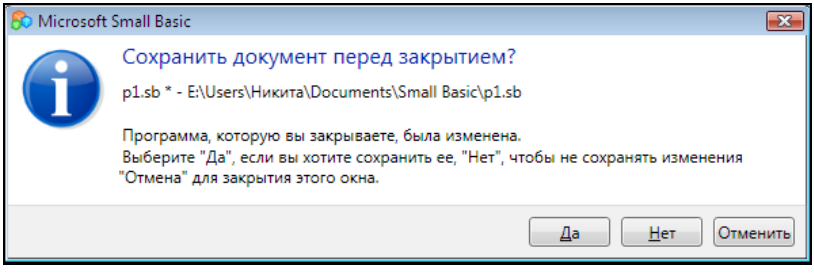


Рис. 2.8. Чтобы сохранить изменения, надо сделать щелчок на кнопке **Да**

Внесение изменений в программу

Открыть файл программы, загрузить его в редактор кода для внесения изменений можно несколькими способами.

Если Small Basic уже запущен, то надо сделать щелчок на кнопке **Открыть** (рис. 2.9), в появившемся окне открыть папку, в которой находится нужная программа, указать файл программы (рис. 2.10) и нажать кнопку **Открыть**. В результате описанных действий, программа будет загружена в редактор кода.

Рис. 2.9. Кнопка **Открыть**

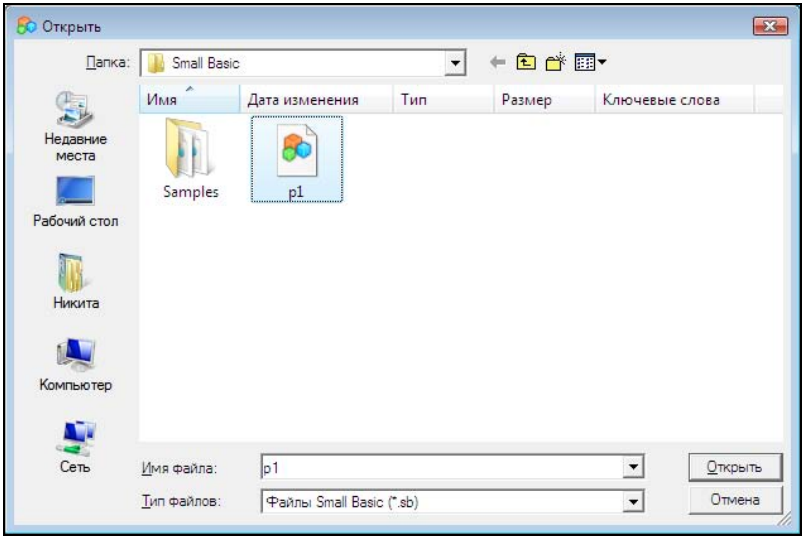


Рис. 2.10. Выбор файла программы для загрузки в редактор кода

Если среда разработки не запущена, то для того чтобы внести изменения в программу, можно открыть папку, в которой находится файл программы (sb-файл), и сделать двойной щелчок мышью на его значке (рис. 2.11).

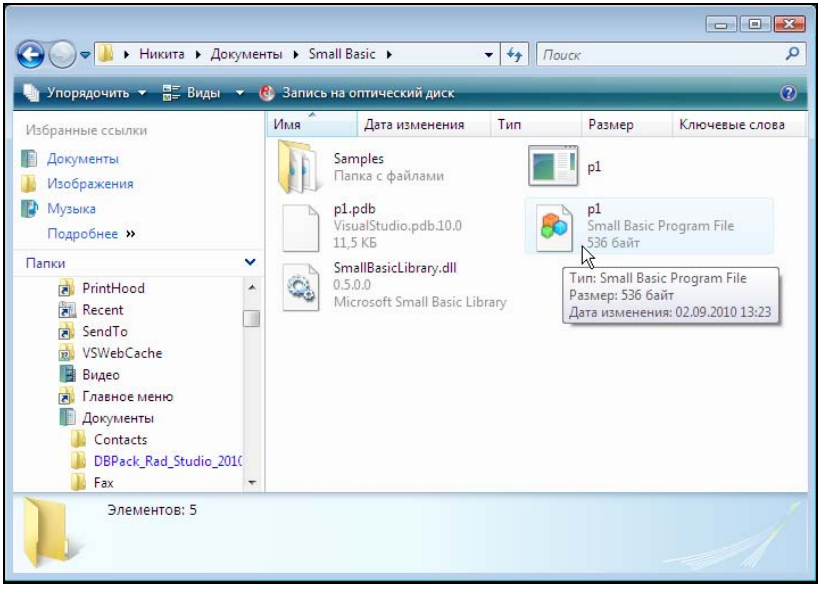


Рис. 2.11. Чтобы загрузить программу в редактор кода, сделайте щелчок на значке файла программы

Запуск программы из Windows

По умолчанию компилятор создает exe-файл программы в той папке, в которой находится файл текста программы. Поэтому, чтобы запустить созданную в Small Basic программу из Windows, надо раскрыть папку, в которой находится текст программы (sb-файл), и сделать двойной щелчок на значке выполняемого (exe) файла программы (рис. 2.12).

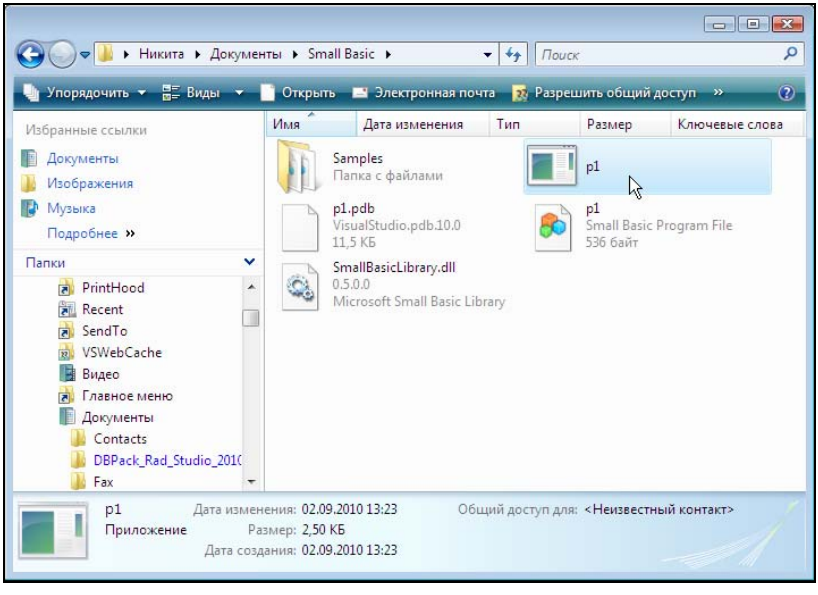
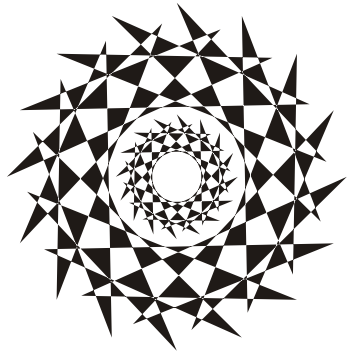


Рис. 2.12. Запуск программы из Windows: сделайте двойной щелчок на значке выполняемого файла программы

Установка программы на другой компьютер

В результате щелчка на кнопке **Запуск** активизируется процесс компиляции программы и, в случае отсутствия в программе синтаксических ошибок, в каталоге, где находится файл исходной программы (sb-файл), создается выполняемый (exe) файл программы. Именно его, а также файл динамической библиотеки SmallBasicLibrary.dll, и надо передать пользователю, чтобы он смог запустить программу на своем компьютере. При этом следует понимать, что программа или *приложение*, созданное в Small Basic, является .NET-приложением, и необходимым условием его работы является наличие на компьютере среды Microsoft .NET Framework. Эта среда является неотъемлемой частью операционных систем Microsoft Windows последних версий, и, скорее всего, на компьютере пользователя она есть. Так как размер выполняемого файла не большой, то перенести программу, созданную в Small Basic, на компьютер пользователя можно при помощи флэшки.



Глава 3

Введение в программирование

Язык программирования Small Basic является диалектом языка Basic, имеет схожий синтаксис — правила записи инструкций. Различие состоит, прежде всего, в организации ввода и вывода данных.

Алгоритм и программа

Программа, работающая на компьютере, нередко отождествляется с самим компьютером, т. к. пользователь (человек, работающий с программой) "вводит в компьютер" исходные данные с клавиатуры, а "компьютер выдает результат". На самом деле преобразование исходных данных, введенных с клавиатуры, в результат, отображаемый на экране монитора, выполняет процессор, и делает он это в соответствии с программой — последовательностью команд (инструкций), составленной программистом. Таким образом, чтобы компьютер выполнил некоторую работу, необходимо составить или, как часто говорят программисты, "написать программу". Строго говоря, сначала надо разработать *алгоритм* программы и затем представить его в виде команд на языке программирования. Следует обратить внимание, выражение "написать программу" отражает только один из этапов создания компьютерной программы, когда программист действительно пишет команды на бумаге или в редакторе текста среды разработки.

Этапы разработки программы

Программирование — это процесс создания программы, который может быть представлен как последовательность следующих шагов (этапов):

- постановка задачи (определение требований к программе);
- разработка (составление) алгоритма решения поставленной задачи;
- написание команд (кодирование);
- отладка;
- тестирование.

Постановка задачи — один из важнейших этапов. На этом этапе подробно описывается исходная информация и формулируются требования к результату. Например, требование к программе вычисления сопротивления электрической цепи, состоящей из двух сопротивлений, которые могут быть соединены последовательно или параллельно, может быть сформулировано так:

- исходной информацией для программы являются величины сопротивлений, выраженные в омах, и способ соединения сопротивлений;
- если сопротивления соединены последовательно, то величина сопротивления цепи вычисляется по формуле: $R = R_1 + R_2$;
- если сопротивления соединены параллельно, то величина сопротивления цепи вычисляется по формуле:

$$R = \frac{R_1 \times R_2}{R_1 + R_2};$$

- результат расчета (сопротивление цепи) должен быть выведен в омах, если величина сопротивления цепи меньше

1000 Ом, или в килоомах, если величина сопротивления цепи больше 1000 Ом.

На этапе *разработки алгоритма* необходимо определить последовательность элементарных действий (шагов), которые надо выполнить для достижения поставленной цели (получения результата).

После того как будут определены требования к программе и составлен алгоритм решения поставленной задачи, алгоритм *записывается* на языке программирования.

Под *отладкой* понимается процесс устранения ошибок в программе. Различают синтаксические и алгоритмические ошибки. Синтаксические ошибки — это ошибки записи инструкций. Они наиболее легко устранимы, т. к. об их наличии программиста информирует компилятор — специальная программа, которая переводит *исходную* программу в *выполняемую*. Алгоритмические ошибки обнаружить труднее. Действительно, если формула, по которой выполняется расчет, записана неверно (например, вместо знака + поставлен –), то как компилятор может это определить? Этап отладки можно считать законченным, если программа правильно работает на одном-двух тестовых наборах исходных данных.

Этап *тестирования* особенно важен, если вы предполагаете, что вашей программой будут пользоваться другие люди. На этом этапе следует проверить, как ведет себя программа при обработке различных данных, причем, возможно, и неверных. Например, в программе вычисления тока в электрической цепи следует проверить, как будет вести себя программа, если пользователь введет ноль в качестве значения ее сопротивления. Вспомните, величина тока вычисляется по формуле $I = \frac{U}{R}$, а делить на ноль, как известно, нельзя.

Алгоритм

Алгоритм — точное предписание, определяющее процесс перехода от исходных данных к результату.

Предписание считается алгоритмом, если оно обладает следующими тремя свойствами:

- *определенностью*, т. е. точностью, не оставляющей место произволу при выполнении предписания;
- *универсальностью*, т. е. возможностью обработки различных, находящихся в заранее оговоренных пределах, исходных данных;
- *результативностью*, т. е. направленностью на получение результата.

Далее приведен алгоритм вычисления сопротивления электрической цепи, состоящей из двух сопротивлений (резисторов), которые могут быть соединены последовательно или параллельно. Алгоритм состоит из 3-х шагов. Шаги алгоритма выполняются, начиная с первого.

1. Получить исходные данные: R_1 — величина сопротивления первого резистора; R_2 — величина сопротивления второго резистора; S — способ соединения резисторов (будем считать, что $S = 1$, если резисторы соединены последовательно, и $S = 2$, если резисторы соединены параллельно).
2. Если резисторы соединены последовательно ($S = 1$), то по формуле $R = R_1 + R_2$ вычислить сопротивление цепи, иначе (если $S \neq 1$) вычислить сопротивление цепи по формуле:

$$R = \frac{R_1 \times R_2}{R_1 + R_2}.$$

3. Вывести на экран значение сопротивления цепи (R).

Приведенное предписание обладает всеми тремя свойствами алгоритма:

- определенностью: в предписании указаны исходные данные, условия и формулы, по которым надо выполнять расчет; формулы указаны для каждого из двух возможных способов соединения сопротивлений;
- универсальностью: в предписании указаны не конкретные значения сопротивлений, а формулы;
- результативностью: при выполнении предписания получается конкретный результат — значение сопротивления цепи.

Следует обратить внимание, при описании алгоритма используются обобщенные понятия, например "величина сопротивления первого резистора", "способ соединения резисторов". При решении задачи "по алгоритму" эти понятия конкретизируются, получают конкретные значения.

Алгоритм решения задачи может быть представлен в виде словесного описания или графически — в виде блок-схемы.

В блок-схемах для обозначения логически различающихся фрагментов программы (действий) используются определенные стандартные символы, основные из которых показаны на рис. 3.1.

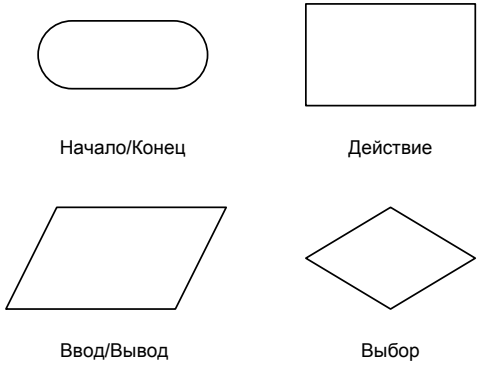


Рис. 3.1. Символы, используемые для графического представления алгоритмов

В качестве примера на рис. 3.2 представлена блок-схема алгоритма вычисления сопротивления электрической цепи, состоящей из двух резисторов, которые могут быть соединены последовательно или параллельно.

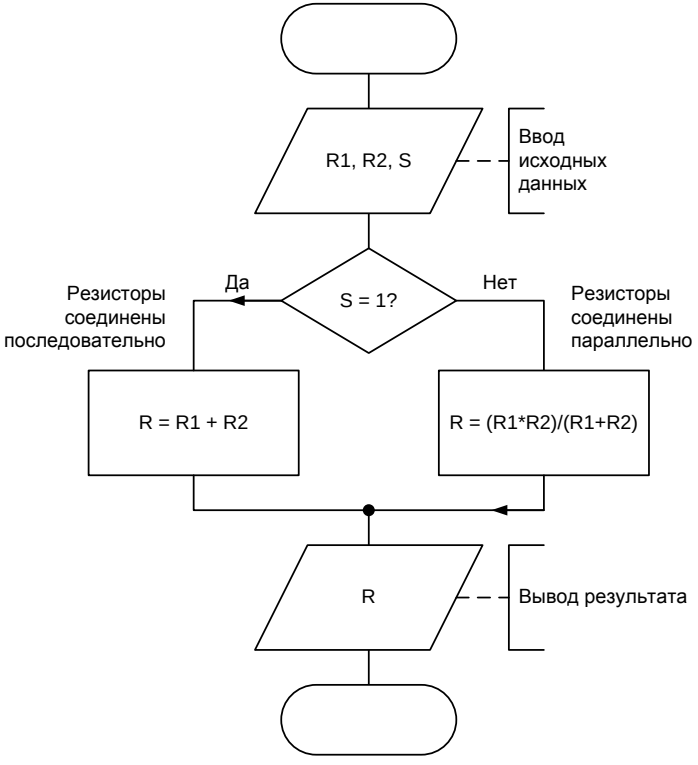


Рис. 3.2. Блок-схема алгоритма вычисления сопротивления электрической цепи

Изображение алгоритма в виде блок-схемы позволяет наглядно представить последовательность действий, которые необходимо выполнить для решения поставленной задачи, убедиться, в том числе и самому программисту, в правильности понимания поставленной задачи, процесса ее решения.

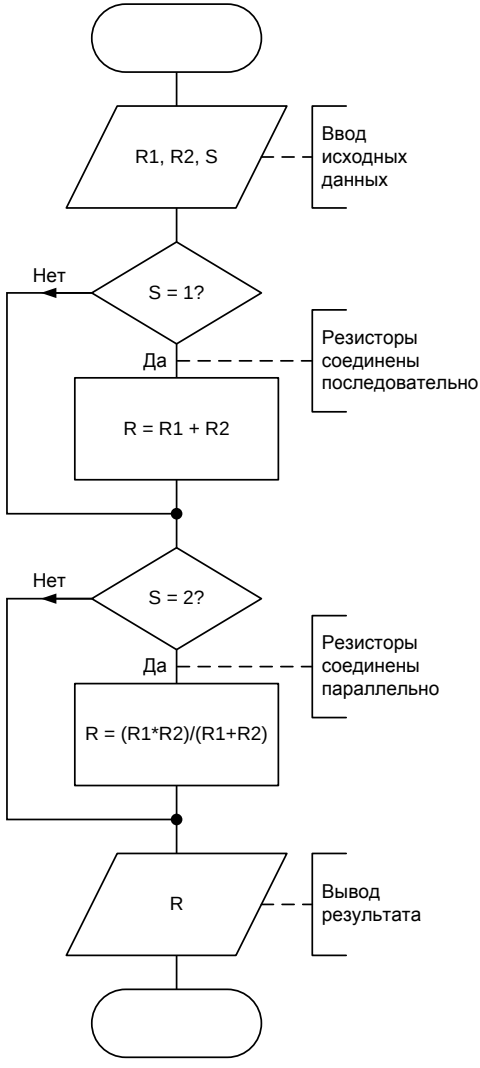


Рис. 3.3. Алгоритм вычисления сопротивления электрической цепи (вариант 2)

Следует обратить внимание, что одну и ту же задачу можно решить по-разному, следуя разным алгоритмам. Так, например, алгоритм рассмотренной задачи вычисления сопротивления электрической цепи можно представить и так, как показано на рис. 3.3.

После разработки алгоритма решения задачи и представления его в виде блок-схемы можно перейти к составлению программы — записи алгоритма на выбранном языке программирования.

Алгоритмические структуры

Алгоритм решения любой задачи можно представить как совокупность следующих алгоритмических структур:

- следование;
- выбор;
- цикл.

Следование

Действия, которые надо выполнить, чтобы пересчитать расстояние из верст в километры (расчет по формуле), можно представить так: получить исходные данные (ввод), посчитать (расчет), отобразить результат (вывод). Это пример последовательного алгоритма.

В алгоритмической структуре "следование" все действия (шаги алгоритма) выполняются последовательно, одно за другим и всегда в одном и том же порядке (рис. 3.4).

Рис. 3.4. Алгоритмическая структура "следование"

Выбор

На практике редко встречаются задачи, алгоритмы решения которых линейны. Действия, которые необходимо выполнить для достижения результата, как правило, зависят от исходных условий и промежуточных (полученных во время работы программы) данных. Например, в программе расчета сопротивления электрической цепи, состоящей из двух резисторов, формула, которую следует использовать для расчета, выбирается в зависимости от способа соединения сопротивлений.

Алгоритмическая структура, соответствующая ситуации, в которой надо выбрать действие (путь дальнейшего развития алгоритма) в зависимости от выполнения (или невыполнения) некоторого *условия*, называется "выбором". Наиболее часто встречаются ситуации, когда надо выбрать один из двух возможных вариантов действия (рис. 3.5), выполнить некоторое действие только при условии выполнения определенного условия (рис. 3.6) или выбрать одно действие из нескольких возможных вариантов (рис. 3.7). Эти ситуации действия моделируются, соответственно, алгоритмическими структурами "выбор" и "множественный выбор".

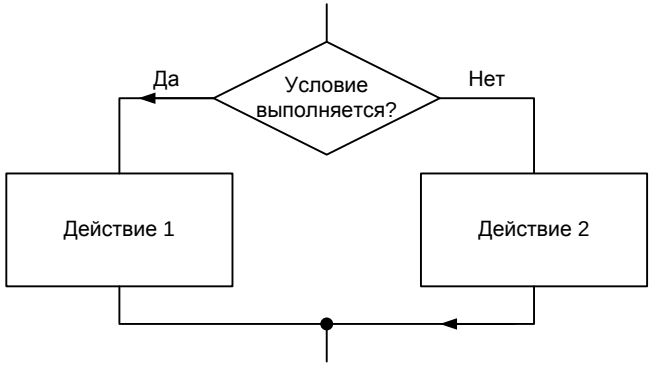


Рис. 3.5. Алгоритмическая структура "выбор"

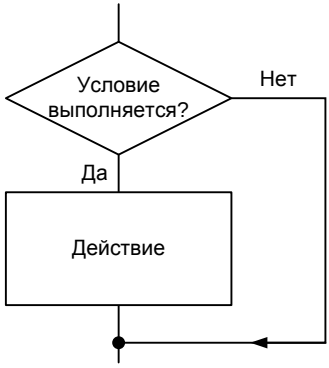


Рис. 3.6. Вариант алгоритмической структуры "выбор"

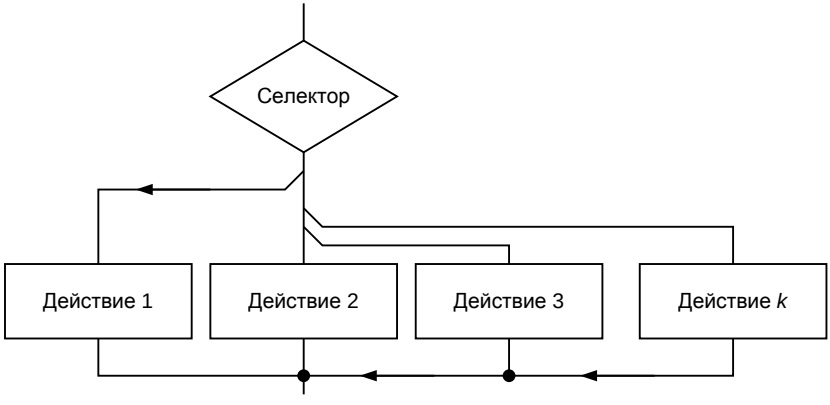


Рис. 3.7. Алгоритмическая структура "множественный выбор"

В Small Basic алгоритмическая структура "выбор" реализуется инструкцией `If`. Инструкции, соответствующей алгоритмической структуре "множественный выбор", в Small Basic нет (в Visual Basic алгоритмической структуре "множественный выбор" соответствует инструкция `Select`).

Цикл

Часто для достижения результата одну и ту же последовательность действий необходимо выполнить несколько раз. Например, чтобы построить таблицу значений функции, надо вычислить значение функции, вывести на экран значение аргумента и функции, изменить значение аргумента, затем повторить описанные выше действия еще раз. Такой алгоритм решения задачи называется *циклическим* или *циклом*.

Различают циклы с предусловием (рис. 3.8), постусловием (рис. 3.9) и циклы с фиксированным количеством повторений (рис. 3.10).

В Small Basic циклу с предусловием соответствует инструкция `While`, циклу с фиксированным числом повторений — инструкция `For`. Инструкции, реализующей цикл с постусловием, в Small Basic нет (в Visual Basic циклу с постусловием соответствует инструкция `Do Loop Until`).

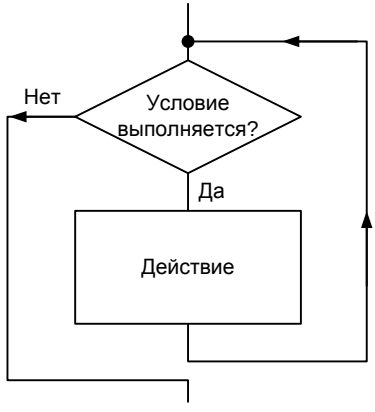


Рис. 3.8. Цикл с предусловием

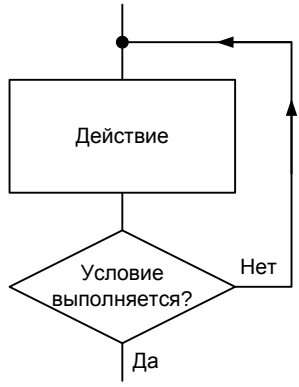


Рис. 3.9. Цикл с постусловием

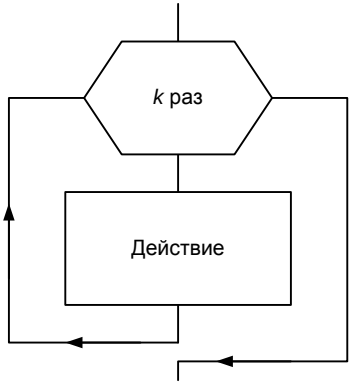


Рис. 3.10. Цикл с фиксированным количеством повторений

Стиль программирования

Работая над программой, программист, особенно начинающий, должен хорошо представлять, что программа, которую он разрабатывает, предназначена, с одной стороны, для пользователя, с другой — для самого программиста. Текст программы нужен прежде всего самому программисту, а также другим людям, с которыми он совместно работает над проектом. Поэтому, для того чтобы работа была эффективной, программа должна быть легко читаемой, ее структура должна соответствовать структуре и алгоритму решаемой задачи. Как этого добиться? Надо следовать правилам хорошего стиля программирования. Стиль программирования — это набор правил, которым следует программист (осознано или потому, что "так делают другие") в процессе своей работы. Очевидно, что хороший программист должен следовать правилам хорошего стиля.

Хороший стиль программирования предполагает:

- использование имеющих смысловую нагрузку имен переменных и подпрограмм;
- использование отступов и пустых строк при записи текста программы;
- комментирование назначения переменных и ключевых точек программы.

Использование имеющих смысловую нагрузку имен переменных делает программу более понятной. В принципе, переменной можно присвоить любое имя. Однако лучше, чтобы имя переменной было связано с ее назначением. Например, в программе вычисления дохода по вкладу, переменные лучше назвать `Sum`, `Percent` и `Period`, чем `s`, `p` , `n` и, тем более, `a`, `b`, `c`.

Отступы следует использовать при записи инструкций выбора, варианта и циклов. Они позволяют выделить логическую струк-

туру инструкции. Пустые строки между последовательностями инструкций программы, реализующими некоторую часть задачи подпрограммы, выделяют логическую структуру программы.

Комментарии существенно облегчают понимание текста программы. Программист пишет комментарии, прежде всего, для себя и если предполагается, что с программой будут работать другие люди, то и для них. Когда комментарии пишутся для других, то понятно, для чего они нужны, — чтобы читающий программу мог в ней разобраться. Комментарий "для себя" программист пишет для того, чтобы облегчить процесс отладки, а также для того, чтобы, если через некоторое время потребуется внести изменения в программу, не тратить много времени на вспоминание, что делает программа и как она работает. Обычно комментируют назначение переменных в инструкциях их объявления и основные точки программы, в том числе назначение подпрограмм. Текст комментариев должен быть кратким, но содержательным.

Следование правилам хорошего стиля программирования значительно уменьшает вероятность появления ошибок на этапе набора текста, делает программу легко читаемой, что в свою очередь облегчает процесс отладки и внесения изменений.

Четкого критерия оценки степени соответствия программы хорошему стилю программирования не существует. Вместе с тем, достаточно одного взгляда, чтобы понять, соответствует ли программа хорошему стилю или нет.

Сводить понятие стиля программирования только к правилам записи текста программы было бы неверно. Стиль, которого придерживается программист, проявляется и во время работы программы. Хорошая программа должна быть прежде всего надежной и дружественной к пользователю.

Надежность подразумевает, например, что программа, не полагаясь на "разумное" поведение пользователя, контролирует исходные данные.

Дружественность предполагает разумное и предсказуемое, с точки зрения пользователя, поведение программы.

Контрольные вопросы

1. Перечислите алгоритмические структуры.
2. Перечислите алгоритмические структуры типа "выбор".
3. Перечислите циклические алгоритмические структуры.
4. Что такое условие?
5. В цикле с предусловием задается условие выполнения инструкций тела цикла?
6. В цикле с постусловием задается условие повторного выполнения инструкций тела цикла или условие завершения цикла?

Программа

Программа Small Basic представляет собой последовательность инструкций. Каждую инструкцию записывают в отдельной строке. Некоторые инструкции, например `If`, принято записывать в несколько строк.

С точки зрения синтаксиса, программа представляет собой последовательность идентификаторов (слов). Идентификаторы могут представлять собой ключевые слова языка программирования (`If`, `Then`, `Else`, `For`, `While` и др.), переменные, числовые константы (дробные или целые), символьные константы (отдельные символы или строки символов).

Комментарии

Для пояснения логики работы программы в ее текст рекомендуется включать комментарии. Комментарии обычно размещают в отдельной строке текста программы или после инструкции, действие которой поясняется. Перед поясняющим текстом ставят апостроф (одинарную кавычку) — `'`.

Типы данных и переменные

Программа Small Basic позволяет обрабатывать числовые и символьные данные, т. е. в Small Basic есть только два типа данных: числовой и символьный. К числовому типу относятся целые и дробные числа, к символьному — символы и строки символов.

Для хранения в программе данных (исходных, промежуточных и результата) необходимы переменные. *Переменная* — это область памяти компьютера, в которой находятся данные. Обращение к переменной, например, для получения или сохранения данных, осуществляется по имени.

В отличие от языков Паскаль (Pascal) и Си (C), в которых все переменные должны быть объявлены явно (перечисление переменных, используемых в программе, с указанием их типа называется *объявлением переменных*), в программе Small Basic (впрочем как и в Visual Basic) заранее перечислять все переменные не надо. Память для данных выделяется в момент обращения к переменной, например, при выполнении инструкции присваивания значения переменной или инструкции ввода значения с клавиатуры.

В качестве имени переменной можно использовать любую последовательность из букв латинского или русского алфавитов и цифр, однако первым символом должна быть буква. Следует обратить внимание на то, что в качестве имен переменных нельзя использовать ключевые слова языка программирования Small Basic, имена встроенных функций Small Basic и некоторые другие идентификаторы.

Важно отметить, что Small Basic различает прописные и строчные буквы в именах переменных. То есть идентификаторы (имена) `Sum` и `sum` обозначают разные переменные.

Контрольные вопросы

1. Перечислите типы данных языка Small Basic.

2. Что такое переменная?

3. Для чего используются переменные?

4. Какие символы можно использовать при записи имен переменных?

5. Составьте список переменных, необходимых в программе вычисления:

   - площади прямоугольника;
   - величины силы тока в электрической цепи;
   - стоимости печати фотографий;
   - дохода по вкладу в банке.

Константы

Константы используются в программе для представления конкретных значений. Различают числовые (целые и дробные) и строковые константы.

Числовые константы

В тексте программы числовые константы записываются обычным образом. Следует обратить внимание, в *тексте программы* при записи дробных констант в качестве десятичного разделителя используется *точка*.

Примеры числовых констант:

```
0
123
0.05
-1.0
```

Строковые константы

Строковые константы, для того чтобы их можно было отличить от имен переменных, в тексте программы заключаются в кавычки. Вот примеры строковых констант:

```
"Microsoft Small Basic"
"руб."
"$"
""
```

Здесь следует обратить внимание на последнюю из приведенных констант (между двумя кавычками нет ни одного символа). Это так называемая *пустая строка*, т. е. строка, не содержащая ни одного символа. Она часто используется для инициализации строковых переменных.

Инструкция присваивания

Инструкция присваивания является основной вычислительной инструкцией. В результате выполнения инструкции присваивания значение переменной меняется, ей *присваивается* новое значение.

Примеры:

```
Sum = Cena * Kol
Kg = 0.495 * Funt
Total = 0
```

В общем виде инструкция присваивания записывается так:

Переменная = *Выражение*

Здесь:

- *Переменная* — переменная, значение которой надо изменить (присвоить значение);
- = — оператор "присвоить";
- *Выражение* — выражение, значение которого надо записать в переменную.

Выполняется инструкция присваивания следующим образом: сначала вычисляется значение выражения, указанного справа от символа присваивания (=), затем полученное значение записывается в переменную, имя которой указано слева от символа присваивания.

Например, в результате выполнения инструкций:

`i = 0` значение переменной `i` становится равным нулю;

`a = b + c` значение переменной `a` становится равным сумме значений переменных `b` и `c`;

`n = n + 1` значение переменной `n` увеличивается на единицу.

Выражение

Выражение состоит из *операндов* и *операторов*. Операнды, в качестве которых могут выступать константы, переменные, а также функции — это объекты, над которыми выполняются действия. Операторы (табл. 3.1) обозначают действия, выполняемые над операндами.

Таблица 3.1. *Операторы*

Оператор	Действие
+	Сложение
–	Вычитание
*	Умножение
/	Деление

Простое выражение состоит из двух операндов и находящегося между ними оператора.

Например:

```
i + 1
Cena * Kol
sum – discount
cena * 0.05
U / R
```

Если выражение состоит из одного-единственного операнда, представляющего собой константу, переменную или функцию, то оно называется *простейшим*.

Например:

```
0.05
K
0
```

При вычислении значения выражения следует учитывать порядок выполнения действий. В общем случае действует известное правило вычислений значений выражений: сначала выполняется умножение и деление, затем — сложение и вычитание. Другими словами, сначала выполняются действия, соответствующие операторам, имеющим более высокий приоритет (`*`, `/`), затем более низкий (`+`, `-`). Если приоритет операторов одинаков, то сначала выполняется действие, соответствующее оператору, находящемуся левее.

Для задания требуемого порядка выполнения операторов в выражении можно использовать скобки. Например:

```
(a + b) / 2
```

Выражение, заключенное в скобки, трактуется как один операнд. Это значит, что операторы выражения, стоящего в скобках, будут выполняться в обычном порядке, но раньше, чем операторы, находящиеся за скобками.

При записи выражений, содержащих скобки, должна соблюдаться парность скобок, т. е. число открывающих скобок должно быть равно числу закрывающих скобок. Нарушение парности скобок — наиболее распространенная ошибка при записи выражений.

Контрольные вопросы

Запишите инструкцию присваивания, обеспечивающую вычисление:

- площади прямоугольника;
- величины силы тока в электрической цепи;
- стоимости печати фотографий;
- дохода по вкладу в банке;
- увеличения на единицу значения переменной `N`.

Понятие функции

Функция — это подпрограмма, обеспечивающая решение некоторой задачи и возвращающая в вызвавшую ее программу результат, который называется *значением функции*. Например, значением стандартной функции `Math.Sin` является синус угла, указанного в качестве параметра функции, а значением функции `Math.SquareRoot` — квадратный корень числа.

Чтобы получить значение функции, ее имя надо вызвать — указать функцию в качестве операнда выражения инструкции присваивания.

Например:

```
r = Math.SquareRoot(x)
y = L * Math.Sin(a)
```

В табл. 3.2 приведены некоторые математические функции Small Basic (префикс `Math` для облегчения восприятия не указан).

Таблица 3.2. *Математические функции*

Функция	Значение
`Abs(x)`	Абсолютное значение `x`
`Power(a,b)`	Число `a` в степени `b`
`SquareRoot(x)`	Квадратный корень из `x`
`Log(x)`	Десятичный (по основанию 10) логарифм `x`
`NaturalLog(x)`	Натуральный (по основанию *e*) логарифм `x`
`Remainder(a,b)`	Остаток от деления `a` на `b` как целое значение
`Sin(a)`	Синус угла `a`, заданного в радианах
`Cos(a)`	Косинус угла `a`, заданного в радианах
`Tan(a)`	Тангенс угла `a`, заданного в радианах
`Round(x)`	Целое — `x`, округленное в соответствии с известными правилами
`Ceiling(x)`	Целое — `x`, округленное "с избытком"
`Floor(x)`	Целое, x округленное "с недостатком"

Ввод и вывод

Обычно данные, необходимые программе, вводятся с клавиатуры, а результат отображается на экране монитора. Ввод данных с клавиатуры обеспечивают инструкции `TextWindow.ReadNumber` и `TextWindow.ReadLine`, вывод результата — `TextWindow.Write` и `TextWindow.WriteLine`.

Вывод информации на экран

Операция вывода заключается в отображении результата работы программы, например значений переменных, в окне программы.

Инструкции `TextWindow.Write` и `TextWindow.WriteLine` предназначены для вывода на экран монитора сообщений и значений переменных.

Вывод сообщений

В общем виде инструкция вывода сообщения записывается так:

```
TextWindow.Write(Сообщение)
```

где *Сообщение* — текст (строковая константа), который надо вывести на экран.

Пример:

```
TextWindow.Write("Microsoft Small Basic");
```

Положение текста на экране определяется текущим положением курсора в окне программы. В начале работы программы курсор находится в начале первой строки окна программы, а после вывода сообщения — за последним символом сообщения. Таким образом, следующая инструкция `TextWindow.Write` выведет сообщение сразу за текстом, выведенным предыдущей инструкцией

`TextWindow.Write`. Например, в результате выполнения инструкций:

```
TextWindow.Write("Microsoft ")
TextWindow.Write("Small ")
TextWindow.Write("Basic")
```

на экране появится строка `Microsoft Small Basic`.

Часто на экран надо вывести несколько строк текста. Очевидно, чтобы вывести текст с новой строки, надо после вывода сообщения переместить курсор в начало следующей строки экрана. Чтобы после вывода сообщения курсор переместился в начало следующей строки, вместо инструкции `TextWindow.Write` следует использовать инструкцию `TextWindow.WriteLine`. Например, в результате выполнения инструкций

```
TextWindow.WriteLine("Microsoft")
TextWindow.WriteLine("Small")
TextWindow.WriteLine("Basic")
```

на экране появятся три строки.

Иногда возникает необходимость разделить строки сообщения. Сделать это можно, вставив инструкцию вывода пустой строки между инструкциями вывода строк сообщения. Например:

```
TextWindow.WriteLine("Microsoft")
TextWindow.WriteLine("")
TextWindow.WriteLine("Small Basic")
```

Вывод значений переменных

Инструкция вывода значения переменной в общем виде выглядит так:

```
TextWindow.Write(Переменная)
```

где *Переменная* — имя переменной, значение которой надо вывести на экран монитора.

Пример:

```
TextWindow.Write(profit)
```

Несколько следующих одна за другой инструкций вывода сообщений и значений переменных можно объединить в одну, разделив сообщения и имена переменных знаком "плюс". Например, следующие три инструкции

```
TextWindow.Write("Доход: ")
TextWindow.Write(profit)
TextWindow.Write(" руб.")
```

можно заменить одной

```
TextWindow.Write("Доход: " + profit + " руб.")
```

Программист может изменить цвет символов, выводимых инструкциями `TextWindow.Write` и `TextWindow.WriteLine`. Цвет символов определяет глобальная переменная `TextWindow.ForegroundColor`. Чтобы задать требуемый цвет, надо изменить ее значение. В качестве значения `TextWindow.ForegroundColor` следует указать одну из приведенных в табл. 3.3 констант.

Таблица 3.3. *Кодировка цвета*

Константа	**Цвет**
`"Black"`	Черный
`"Blue"`	Синий
`"Cyan"`	Бирюзовый
`"Gray"`	Серый
`"Green"`	Зеленый
`"Magenta"`	Сиреневый
`"Red"`	Красный
`"White"`	Белый
`"Yellow"`	Желтый

Таблица 3.3 *(окончание)*

Константа	Цвет
"DarkBlue"	Темно-синий
"DarkCyan"	Бирюзовый темный
"DarkGray"	Серый темный
"DarkGreen"	Темно-зеленый
"DarkMagenta"	Сиреневый темный
"DarkRed"	Темно-красный
"DarkYellow"	Желтый темный

Пример:

```
TextWindow.ForegroundColor = "Blue"
TextWindow.WriteLine("Microsoft Small Basic")
```

Аналогично, путем присвоения значения переменной `TextWindow.BackgroundColor` устанавливается цвет фона.

Пример:

```
TextWindow.ForegroundColor = "Blue" ' цвет символов
TextWindow.BackgroundColor = "Gray" ' цвет фона
TextWindow.WriteLine("Microsoft Small Basic")
```

Следует обратить внимание на то, что константа `TextWindow.BackgroundColor` задает цвет фона только для области вывода текста. Если необходимо изменить цвет фона всего окна, то надо закрасить экран цветом фона. Сделать это можно, вызвав функцию `TextWindow.Clear()`, предварительно задав значение `TextWindow.BackgroundColor`.

Пример:

```
TextWindow.ForegroundColor = "Blue" ' цвет символов
TextWindow.BackgroundColor = "Gray" ' цвет фона
TextWindow.Clear() ' очистить экран (закрасить цветом фона)
TextWindow.WriteLine("Microsoft Small Basic")
```

Контрольные вопросы

1. Переменная `Sum` содержит значение суммы покупки. Запишите инструкцию, обеспечивающую вывод значения переменной. После числового значения должен отображаться текст `"руб."`.

2. Переменные `Sum` и `Profit` содержат значения величины дохода и суммы в конце срока вклада. Запишите инструкцию, которая обеспечивает отображение значения этих переменных. Значения должны быть выведены каждое в отдельной строке. Перед значением переменной `Profit` должен отображаться текст `"Доход:"`, перед значением переменной `Sum` — `"Сумма в конце срока вклада:"`.

Задание

Напишите программу, которая выводит на экран четверостишие. Стихотворение должно быть выведено серыми буквами на синем фоне, а имя автора написано белым.

У лукоморья дуб зеленый;
Златая цепь на дубе том:
И днем и ночью кот ученый
Все ходит по цепи кругом...
А. С. Пушкин

Ввод данных

Операция ввода данных заключается в получении от пользователя исходных данных, например, значений переменных, используемых в расчетных формулах.

Инструкции `TextWindow.ReadNumber()` и `TextWindow.Read()` обеспечивают ввод данных с клавиатуры.

Ввод чисел

Функция `TextWindow.ReadNumber()` используется для ввода с клавиатуры чисел.

В общем виде инструкция ввода числа с клавиатуры выглядит так:

```
переменная = TextWindow.ReadNumber()
```

где *переменная* — имя переменной, значение которой надо ввести с клавиатуры во время работы программы.

Пример:

```
sum = TextWindow.ReadNumber()
```

Функция `TextWindow.ReadNumber()` работает так. Она ждет, пока на клавиатуре не будут набраны данные и нажата клавиша <Enter>. Набираемые *правильные* символы (цифры, точка и знак "минус", если он введен первым) появляются на экране, неправильные — не отображаются. После нажатия клавиши <Enter> введенные данные записываются в указанную переменную. Затем выполняется следующая инструкция программы.

Чтобы пользователь знал, какие данные ждет от него программа, перед каждой инструкцией ввода данных рекомендуется поместить инструкцию отображения подсказки.

Пример:

```
TextWindow.Write("Ширина (см) ->")
width = TextWindow.ReadNumber()
TextWindow.Write("Высота (см) ->")
height = TextWindow.ReadNumber()
```

Ввод строк

Функция `TextWindow.Read()` используется для ввода с клавиатуры строк символов.

В общем виде инструкция ввода строки с клавиатуры выглядит так:

переменная = `TextWindow.Read()`

где *переменная* — имя переменной, значение которой надо ввести с клавиатуры во время работы программы.

Пример:

```
name = TextWindow.Read()
```

Функция `TextWindow.Read()` работает так. Она ждет, пока на клавиатуре не будут набраны данные и нажата клавиша <Enter>. Набираемые символы появляются на экране. После нажатия клавиши <Enter> введенная строка записывается в указанную переменную. Затем выполняется следующая инструкция программы.

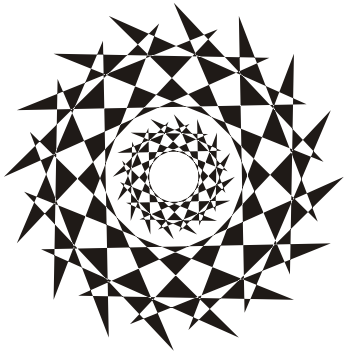
Чтобы пользователь знал, какие данные ждет от него программа, перед инструкцией ввода данных рекомендуется поместить инструкцию отображения подсказки.

Пример:

```
TextWindow.Write("Ваше имя ->")
name = TextWindow.Read()
```

Контрольные вопросы

1. В программе для расчета стоимости покупки используется переменная `Cena`. Запишите инструкцию, обеспечивающую ввод значения этой переменной.
2. Какую функцию следует использовать для ввода с клавиатуры значения числовой переменной?
3. Какую функцию следует использовать для ввода с клавиатуры значения символьной переменной?
4. Какой символ следует использовать при вводе с клавиатуры дробных чисел?
5. Программа не реагирует на нажатие запятой при вводе числа с клавиатуры. Почему?



Глава 4

Инструкции управления

Как было сказано ранее, алгоритмы решения большинства задач не являются линейными. Действия, которые необходимо выполнить для решения поставленной задачи, как правило, зависят от выполнения определенных условий.

Условие

Условие — это выражение логического типа, которое может принимать одно из двух значений: "истина" или "ложь".

На естественном языке условие формулируется в виде вопроса, на который можно ответить "да" или "нет". Например: "A равно B?", "Сумма больше 1000?". То есть "истина" эквивалентна "да", а "ложь" — нет.

Операторы сравнения

Простое условие состоит из двух операндов и оператора сравнения, находящегося между ними. Операндами выражения-условия могут быть переменные, константы (числовые, строковые) и функции.

Вот примеры условий:

```
Sum > 1000
A = B
```

Операторы сравнения приведены в табл. 4.1.

Таблица 4.1. *Операторы сравнения*

Оператор	Условие	Значение
> (больше)	`a > b`	`Истина` (Да), если значение операнда `a` больше значения операнда `b`, иначе `Ложь` (Нет)
< (меньше)	`a < b`	`Истина` (Да), если значение операнда `a` меньше значения операнда `b`, иначе `Ложь` (Нет)
= (равно)	`a = b`	`Истина` (Да), если значение операнда `a` равно значению операнда `b`, иначе `Ложь` (Нет)
<> (не равно)	`a <> b`	`Истина` (Да), если значение операнда `a` равно значению операнда `b`, иначе `Ложь` (Нет)
>= (больше или равно)	`a >= b`	`Истина` (Да), если значение операнда `a` больше или равно значению операнда `b`, иначе `Ложь` (Нет)
<= (меньше или равно)	`a <= b`	`Истина` (Да), если значение операнда `a` меньше или равно значению операнда `b`, иначе `Ложь` (Нет)

Применение операторов сравнения к операндам числового типа не вызывает затруднений. Например, если значение переменной `n` равно `10`, то значение выражения `n < 10` равно `Ложь`, а выражения `n = 10` — `Истина`.

Переменную символьного типа можно сравнить с другой переменной символьного типа или с символьной константой. Строки сравниваются посимвольно, начиная с первого символа. Если количество символов в сравниваемых строках одинаково и все символы совпадают, то такие строки считаются равными. В противном случае строки считаются разными. В табл. 4.2 приведены примеры сравнения строк.

Таблица 4.2. *Примеры сравнения строк*

Строка 1	**Строка 2**	**Результат сравнения**
`Small Basic`	`Small Basic`	Строки равны
`Small Basic`	`small Basic`	Строки не равны

Логические операторы

Операторы сравнения позволяют записывать простые условия, из которых путем применения логических операторов `And` (логическое И), `Or` (логическое ИЛИ) и `Not` (отрицание) можно строить сложные условия.

В табл. 4.3 приведены результаты применения логических операторов к операндам-условиям `У1` и `У2`.

Таблица 4.3. *Действие логических операторов*

У1	**У2**	**У1 And У2**	**У1 Or У2**
Ложь (Нет)	Ложь (Нет)	Ложь (Нет)	Ложь (Нет)
Ложь (Нет)	Истина (Да)	Ложь (Нет)	Истина (Да)
Истина (Да)	Ложь (Нет)	Ложь (Нет)	Истина (Да)
Истина (Да)	Истина (Да)	Истина (Да)	Истина (Да)

Из таблицы видно, что выражение `У1 And У2` истинно только в том случае, если значение обоих операндов истинно, т. е. оба условия выполняются. Выражение `У1 Or У2` истинно, если значение хотя бы одного операнда истинно, т. е. в том случае, когда хотя бы одно из условий (выполняется) истинно.

Оператор `Not` изменяет значение логического выражения на противоположное.

Примеры сложных условий:

```
(x >= x1) And (x <= x2)
(x < x1) Or (x > x2)
(Sum >= 1000) And (sum < 10000)
```

Контрольные вопросы

1. Что такое условие?

2. Из чего состоит простое условие?

3. Перечислите операторы сравнения.

4. Скидка предоставляется, если сумма покупки больше 1000 руб. Запишите условие предоставления скидки.

5. При печати фотографий скидка предоставляется, если количество заказанных экземпляров не мене 15. Запишите условие предоставления скидки.

6. Запишите условие, позволяющее определить способ соединения (последовательно или параллельно) сопротивлений электрической цепи.

7. Перечислите логические операторы.

8. Запишите условие принадлежности значения переменной `x` интервалу `x1 .. x2`.

9. Скидка предоставляется при печати не менее 5 фотографий формата 18×24 или не менее 10 фотографий формата 10×15. Запишите условие предоставления скидки.

Инструкция *If*

Выбор

Инструкция `If` используется в том случае, если нужно выбрать одно из двух возможных действий. В общем виде она записывается так:

```
If Условие Then
    Действие 1
Else
    Действие 2
EndIf
```

Выполняется инструкция `If` следующим образом. Сначала вычисляется значение выражения *`Условие`*. Затем, если условие истинно (выполняется), выполняются инструкции, находящиеся между `Then` и `Else`. Если условие не выполняется (значение выражения *`Условие`* равно `Ложь`), то выполняются инструкции, находящиеся между `Else` и `EndIf`. Алгоритм, реализуемый инструкцией `If`, приведен на рис. 4.1.

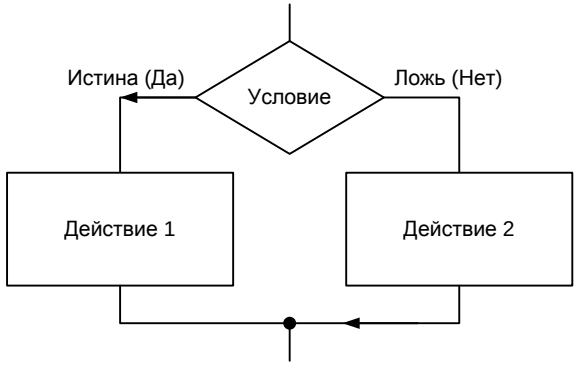


Рис. 4.1. Блок-схема инструкции `If`

Примеры:

```
If t  = 1 Then
    r = r1+r2
Else
    r = r1*r2/(r1+r2)
EndIf
```

В качестве примера использования инструкции `If` в листинге 4.1 приведена программа вычисления дохода по вкладу в банке. Предполагается, что процентная ставка зависит от суммы вклада: если сумма меньше 10 тыс. руб., то ставка 9.5%, если больше, то 11.5%. Пример работы программы "Доход" приведен на рис. 4.2.

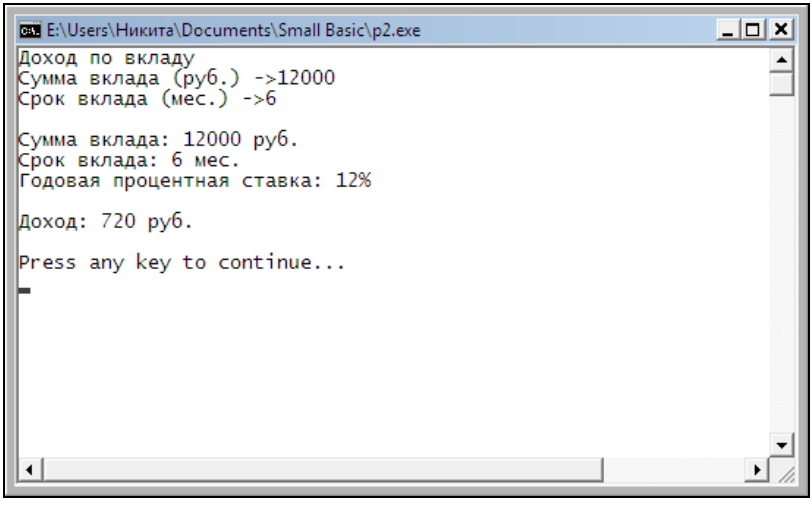


Рис. 4.2. Пример работы программы "Доход"

Листинг 4.1. Доход

```
' Доход по вкладу

TextWindow.BackgroundColor = "DarkGreen" ' цвет фона
TextWindow.Clear() '  очистить экран

TextWindow.WriteLine("Доход по вкладу")

' Ввод исходных данных
TextWindow.Write("Сумма вклада (руб.) ->")
sum = TextWindow.ReadNumber()

TextWindow.Write("Срок вклада (мес.) ->")
period = TextWindow.ReadNumber()

' Определим величину процентной ставки
If sum < 10000 Then
  percent = 9.5
Else
  percent = 11.5
EndIf

' Расчет
profit = sum * (percent / 100 /12) * period

' Вывод результата
TextWindow.WriteLine("")
TextWindow.WriteLine("Сумма вклада: " + sum + " руб.")
TextWindow.WriteLine("Срок вклада: " + period + " мес.")
TextWindow.WriteLine("Годовая процентная ставка: " + percent + "%")

TextWindow.WriteLine("")
TextWindow.ForegroundColor = "White"
TextWindow.WriteLine("Доход: " + Math.Round(profit) + " руб.")

TextWindow.WriteLine("")
TextWindow.ForegroundColor = "Gray"
```

Если какое-либо действие требуется выполнить только в случае выполнения какого-либо условия, а в случае невыполнения этого условия действие надо пропустить (ничего делать не надо), то инструкция `If` записывается так:

```
If Условие Then
    Действие
EndIf
```

При помощи нескольких инструкций `If` можно осуществить множественный выбор. Например, если необходимо выбрать один из трех вариантов, то реализовать это можно так:

```
If Условие 1 Then
    Действие 1
Else
    If Условие 2 Then
        Действие 2
    Else
        Действие 3
    EndIf
EndIf
```

В качестве примера рассмотрим программу "Контроль веса". Вес человека может быть недостаточным, избыточным или оптимальным. В простейшем случае оптимальным принято считать вес, вычисляемый по формуле `Рост – 100`. Программа "Контроль веса" запрашивает у пользователя его рост и вес, вычисляет оптимальный вес, сравнивает его с реальным и выводит соответствующее сообщение. Алгоритм программы представлен на рис. 4.3, текст — в листинге 4.2. Пример работы программы приведен на рис. 4.4.

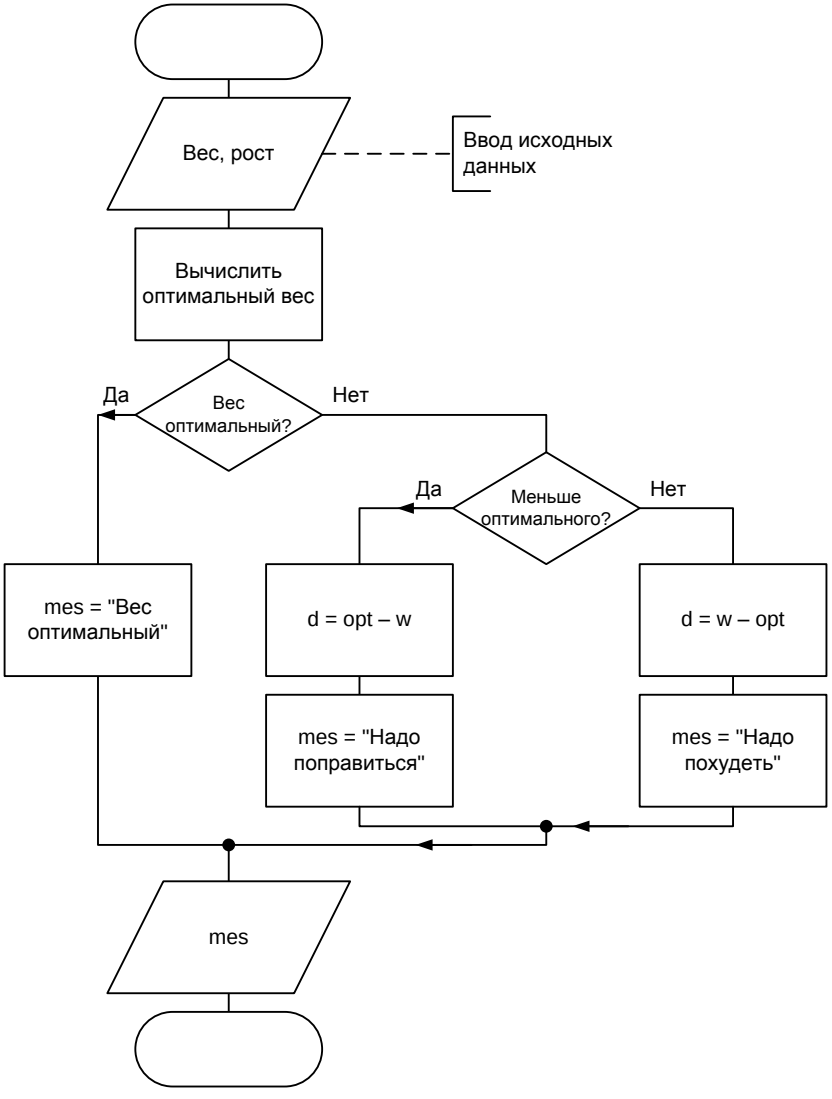


Рис. 4.3. Алгоритм программы "Контроль веса"

Листинг 4.2. Контроль веса (множественный выбор)

```
' Контроль веса

TextWindow.BackgroundColor = "DarkBlue" ' цвет фона
TextWindow.Clear() ' очистить экран

TextWindow.WriteLine("Контроль веса")
TextWindow.WriteLine("Введите ваш рост и вес")

' Ввод исходных данных
TextWindow.Write("Рост (см.) ->")
height = TextWindow.ReadNumber()

TextWindow.Write("Вес (кг.) ->")
weight = TextWindow.ReadNumber()

' Вычислим оптимальный вес
optimal = height - 100

' Сравним реальный вес с оптимальным
If weight = optimal Then
  msg = "Ваш вес оптимален!"
Else
  If (weight < optimal) Then
    dw = optimal - weight
    msg = "Вам надо поправиться на " + dw + " кг."
  Else
    dw = weight - optimal
    msg = "Вам надо похудеть на " + dw + " кг."
  EndIf
EndIf
```

```
' Вывод результата
TextWindow.WriteLine("")
TextWindow.ForegroundColor = "White"
TextWindow.WriteLine(msg)

TextWindow.WriteLine("")
TextWindow.ForegroundColor = "Gray"
```

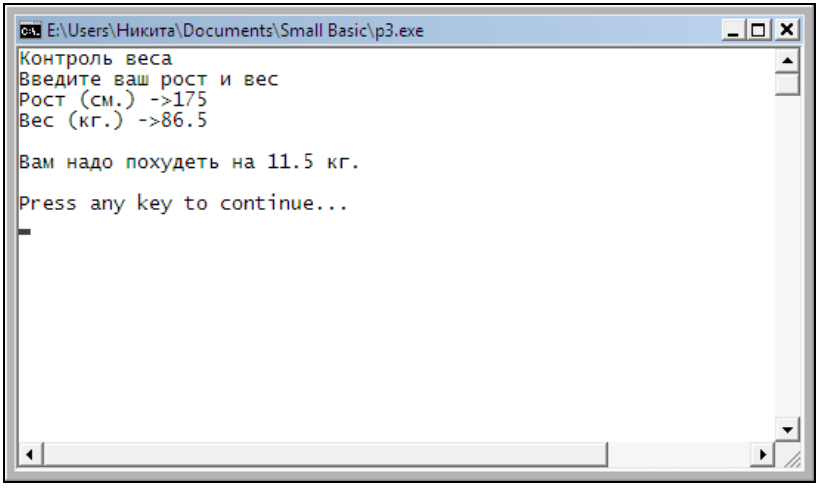


Рис. 4.4. Пример работы программы "Контроль веса"

В качестве еще одного примера использования инструкции `If` для реализации множественного выбора в листинге 4.3 приведена программа, которая позволяет посчитать стоимость заказа печати фотографий в зависимости от их формата. Программа (рис. 4.5) считает стоимость заказа с учетом скидки, которая предоставляется в случае, если количество заказанных фотографий больше 10.

Листинг 4.3. Фото (множественный выбор)

```
' Фото (множественный выбор)

TextWindow.BackgroundColor = "DarkGreen" ' цвет фона
TextWindow.Clear() ' очистить экран

TextWindow.WriteLine("Фото")

' Ввод исходных данных
TextWindow.WriteLine("Формат:")
TextWindow.WriteLine(" 1 - 9x12")
TextWindow.WriteLine(" 2 - 10x15")
TextWindow.WriteLine(" 3 - 18x24")
format = TextWindow.ReadNumber()

TextWindow.Write("Количество (шт.) -> ")
k = TextWindow.ReadNumber()

' Определить цену одной фотографии
If format = 1 Then
  cena = 2.5
  fname = "9x12"
Else
  If format = 2 Then
    cena = 4
    fname = "10x15"
  Else
    cena = 8
    fname = "18x24"
  EndIf
EndIf

' Расчет
sum = cena * k

' Скидка 5%, если количество фотографий больше 10
```

```
If k > 10 Then
  discount = sum * 0.05
Else
  discount = 0
EndIf

total = sum - discount

' Вывод результата
TextWindow.WriteLine("")
TextWindow.WriteLine("Формат: " + fname)
TextWindow.WriteLine("Количество: " + k  + " шт.")
TextWindow.WriteLine("Цена: " + cena  + " руб./шт.")
TextWindow.WriteLine("Сумма: " + sum  + " руб.")
TextWindow.WriteLine("Скидка (5%): " + discount + " руб.")
TextWindow.WriteLine("---------------------------------------")
TextWindow.WriteLine("К оплате: " + total + " руб.")

TextWindow.WriteLine("")
TextWindow.ForegroundColor = "Gray"
```

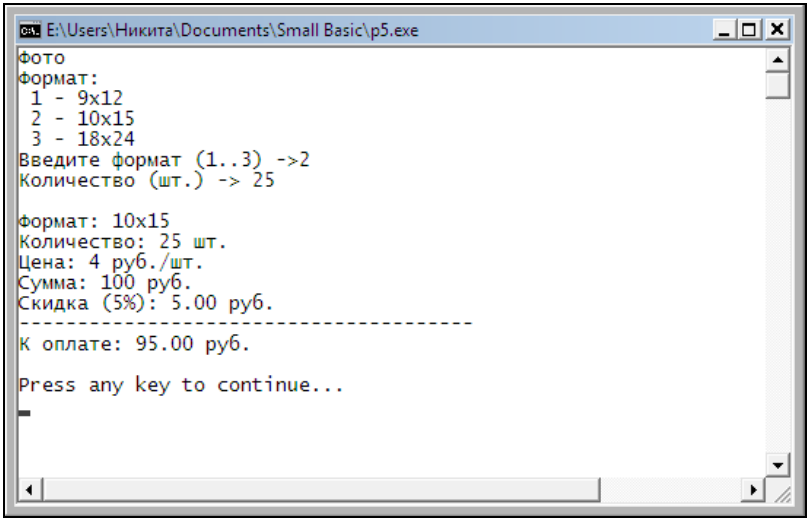


Рис. 4.5. Пример работы программы "Фото"

Контрольные вопросы

1. Какой алгоритмической структуре соответствует инструкция `If`?

2. Запишите инструкцию `If`, позволяющую вычислить сопротивление электрической цепи, состоящей из двух резисторов, которые могут быть соединены последовательно или параллельно.

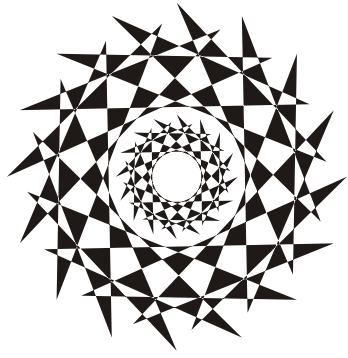
3. Запишите инструкцию `If`, позволяющую вычислить величину скидки, предоставляемой покупателю, если сумма покупки больше 1000 руб.

4. Цена минуты разговора по телефону 1 руб. при условии, что длительность разговора не менее 5 минут, и 1,5 руб. в противном случае. Запишите инструкцию `If`, позволяющую вычислить стоимость разговора.

5. В субботу и воскресенье всем покупателям предоставляется скидка в 10%. Запишите инструкцию `If`, позволяющую вычислить величину скидки.

6. Процент дохода зависит от суммы вклада: 10%, если сумма меньше 5 тыс. руб., 11%, если сумма не превышает 20 тыс. руб., и 12%, если сумма вклада больше 20 тыс. руб. Запишите инструкцию `If`, позволяющую определить величину процентной ставки.

7. Нарисуйте блок-схему алгоритма приведенной в этом разделе программы "Фото".



Глава 5

Циклы

При решении многих задач одну и ту же последовательность действий приходится выполнять несколько раз. Например, чтобы построить таблицу значений функции в диапазоне `[x1, x2]` с шагом изменения аргумента `dx`, надо вычислить значение функции в точке `x`, вывести результат (значения аргумента и функции), увеличить значение `x` на `dx` и затем повторить описанные выше действия еще раз (столько раз, в скольких точках надо вычислить значение функции). Такие повторяющиеся действия называются *циклами* и реализуются в программе с использованием *инструкций циклов*. В Small Basic циклические вычисления могут быть реализованы при помощи инструкций `For` (цикл с фиксированным количеством повторений) и `While` (цикл с предусловием).

Инструкция *For*

Инструкция `For` используется, если некоторую последовательность действий надо выполнить несколько раз, причем число повторений можно задать (вычислить) заранее.

В общем виде инструкция `For` записывается так:

```
For Счетчик = Нач_знач To Кон_знач Step Приращение
    Инструкции
EndFor
```

Здесь:

- *Счетчик* — переменная — счетчик циклов;
- *Нач_знач* — выражение, определяющее начальное значение счетчика циклов;
- *Кон_знач* — выражение, определяющее конечное значение счетчика циклов;
- *Приращение* — выражение, определяющее величину изменения счетчика циклов после каждого выполнения инструкций тела цикла.

> ***Замечание***
>
> Если значение приращения равно единице, то слово `Step` и величину приращения можно не указывать.

Выполняется инструкция `For` следующим образом. Сначала переменной *Счетчик* присваивается значение выражения *Нач_знач*. Затем, если значение счетчика меньше значения выражения *Кон_знач* или равно ему, выполняются инструкции цикла. После этого значение счетчика увеличивается на величину *Приращение*. Если значение счетчика меньше или равно *Кон_знач*, то снова выполняются инструкции цикла. И так до тех пор, пока значение счетчика не превысит значение *Кон_знач* (рис. 5.1).

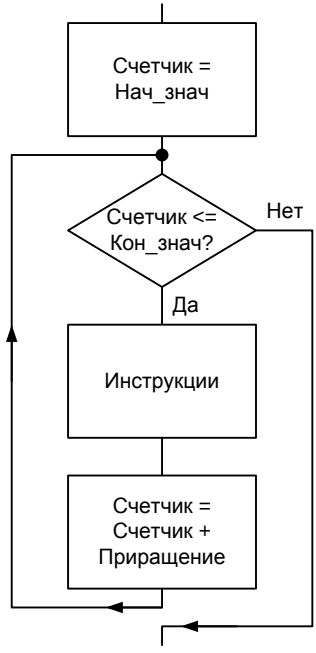


Рис. 5.1. Блок-схема инструкции `For`

Инструкции цикла `For` выполняются (*Кон_знач* − *Нач_знач* + 1) раз. Обратите внимание: если начальное значение счетчика больше указанного конечного значения счетчика, инструкции цикла ни разу не будут выполнены.

В качестве выражений, определяющих начальное и конечное значения счетчика циклов, обычно используют переменные или константы.

В качестве примера использования инструкции `For` в листинге 5.1 приведена программа "Таблица значений функции". Она выводит на экран (рис. 5.2) таблицу значений функции $0.5x^2 + x - 5$.

Листинг 5.1. Таблица значений функции (цикл `For`)

```
' Таблица функции (цикл For)
x1 = -2
x2 = 2
dx = 0.25

n = (x2 - x1)/dx +1

' "Шапка" таблицы
TextWindow.WriteLine("----------------------")
TextWindow.WriteLine("  X          Y")
TextWindow.WriteLine("----------------------  ")

' Таблица
x = x1
For i=1 To n
  ' вычислить значение функции
  y = 0.5 * x* x + x - 5

  ' вывести строку таблицы
  TextWindow.Write(x + "      ")
  TextWindow.WriteLine(y)

  x = x + dx
EndFor
```

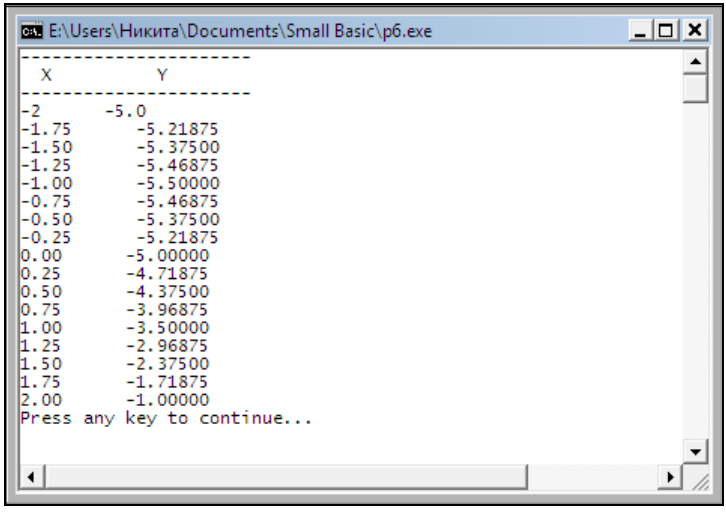


Рис. 5.2. Таблица значений функции (цикл `For`)

Контрольные вопросы

1. Какой алгоритмической структуре соответствует инструкция `For`? Изобразите эту структуру.

2. В каком случае величину приращения переменной счетчика циклов можно не указывать?

3. Значение переменной `n` равно 5. Чему будет равно значение переменной `s` после выполнения приведенного далее кода?

```
s = 0
For i=1 To n
    s = s + i
EndFor
```

4. Что делает следующий фрагмент кода? Чему будет равно значение переменной `m`, если значение `n` равно нулю?

```
m=1
For i=1 To n
   m = m * 2
EndFor
```

Инструкция *While*

Инструкция `While` используется для реализации циклов с предусловием. В общем виде она записывается так:

```
While Условие
    Инструкции
EndWhile
```

Здесь:

- *Инструкции* — инструкции, которые надо выполнить несколько раз (тело цикла);
- *Условие* — условие выполнения инструкций тела цикла.

Выполняется инструкция `While` следующим образом. Сначала вычисляется значение выражения *Условие*. Если условие истинно, то выполняются инструкции цикла. Затем снова вычисляется значение выражения *Условие*. Если условие истинно, то инструкции цикла выполняются еще раз, а если ложно, то инструкции цикла не выполняются, и на этом выполнение инструкции `While` завершается. Таким образом, инструкции цикла выполняются до тех пор, пока значение выражение *Условие* истинно (рис. 5.3).

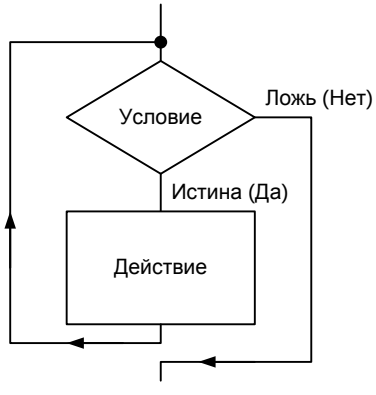


Рис. 5.3. Блок-схема инструкции `While`

Следует обратить особое внимание на то, что каждый раз после выполнения инструкций цикла проверяется значение выражения *Условие*. Поэтому, для того чтобы цикл `While` завершился, инструкции цикла обязательно должны влиять на значения переменных, входящих в выражение *Условие*. Кроме того, возможна ситуация, когда инструкции тела цикла не будут выполнены ни разу.

В качестве примера использования инструкции `While` в листинге 5.2 приведен вариант программы "Таблица функции".

Листинг 5.2. Таблица значений функции (цикл `While`)

```
' Таблица функции (цикл While)
x1 = -2
x2 = 2
dx = 0.25

' "Шапка" таблицы
TextWindow.WriteLine("----------------------")
TextWindow.WriteLine("  X          Y")
TextWindow.WriteLine("----------------------  ")

' Таблица
x = x1
While (x <= x2)
  ' вычислить значение функции
  y = 0.5 * x* x + x - 5

  ' вывести строку таблицы
  TextWindow.Write(x + "      ")
  TextWindow.WriteLine(y)

  x = x + dx
EndWhile
```

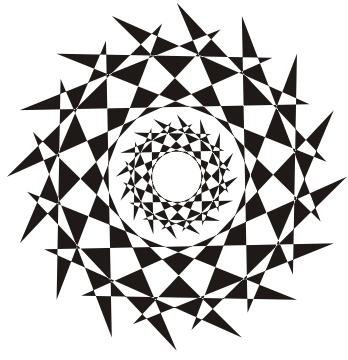
Контрольные вопросы

1. Какой алгоритмической структуре соответствует инструкция `While`? Изобразите эту структуру.

2. Возможна ли ситуация, при которой инструкции тела цикла `While` не будут выполнены ни одного раза?

3. Замените приведенный далее цикл `For` на цикл `While`. Что делает этот цикл?

```
For i=1 To k
     n = TextWindow.ReadNumber()
     s = s + n
EndFor
```



Глава 6

Массивы

Массив — это структура данных, которая представляет собой совокупность переменных одного типа.

Переменные, образующие массив, принято называть элементами массива.

Различают одномерные и двумерные массивы. Графически одномерный массив можно изобразить так, как показано на рис. 6.1.

Массивы используют для хранения однородной по своей структуре информации: одномерные — списков, двумерные — таблиц.

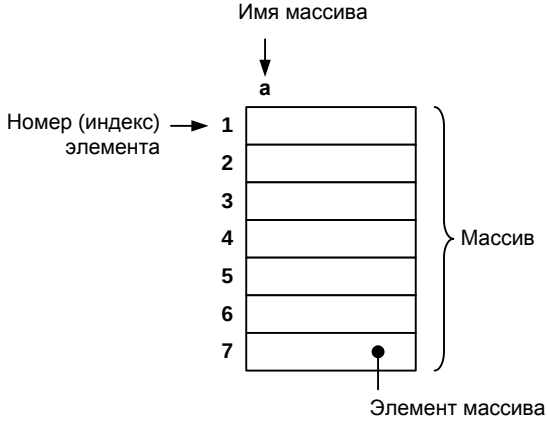


Рис. 6.1. Графическое представление массива

Доступ к элементу массива

Чтобы получить доступ к элементу одномерного массива, надо указать имя массива и номер (индекс) элемента, заключив его в квадратные скобки. В качестве индекса следует использовать выражение целого типа (в простейшем случае — константу или переменную).

Пример:

```
a[1]
a[i]
a[i+1]
```

Элементы массива могут быть пронумерованы с нуля, с единицы или с другого целого числа. Какой способ нумерации выбрать, зависит от решаемой задачи. Например, в программе вычисления значения многочлена, использующей массив для хранения коэффициентов, элементы массива удобно нумеровать с нуля. Тогда нулевой элемент будет содержать коэффициент при нулевой степени, первый — при первой степени, второй — при второй степени и т. д. В задаче обработки метеорологической информации элементы массива логично нумеровать с единицы. При этом первый элемент будет содержать температуру первого дня месяца, второй — второго и т. д.

Все операции, которые можно выполнять над переменными, можно выполнять и над элементами массива. Например, элементы массива можно использовать в качестве операндов выражений и условий.

Например:

```
a[3] = 0
sum = sum + a[i]
a[i] > a[max]
```

Контрольные вопросы

1. Что такое массив? Дайте определение.
2. Объявите массив, состоящий из 10 элементов целого типа.
3. Как получить доступ к нужному элементу массива?

Ввод массива

Под операцией ввода массива понимается процесс ввода значений всех его элементов. Ввести значения элементов массива можно с клавиатуры. Например, следующий фрагмент программы вводит с клавиатуры массив `a`, состоящий из четырех элементов.

```
TextWindow.Write("a[1]->")
a[1] = TextWindow.ReadNumber()
TextWindow.Write("a[2]->")
a[2] = TextWindow.ReadNumber()
TextWindow.Write("a[3]->")
a[3] = TextWindow.ReadNumber()
TextWindow.Write("a[4]->")
a[4] = TextWindow.ReadNumber()
```

Приведенные выше инструкции можно заменить циклом `For`:

```
For i = 1 To 4
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor
```

В качестве примера в листинге 6.1 приведена программа "Среднее арифметическое", которая вводит с клавиатуры массив и вычисляет среднее арифметическое его элементов.

Листинг 6.1. Ввод и обработка массива

```
' Ввод и обработка массива
TextWindow.Write("Среднее арифметическое элементов массива")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 5
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

' Вычислить сумму элементов
s = 0 ' сумма элементов массива
For i=1 To 5
  s = s + a[i]
EndFor

' Вычислить среднее арифметическое
m = s / 5

TextWindow.WriteLine("----") ' пустая строка
TextWindow.WriteLine("Сумма элементов массива:" + s)
TextWindow.WriteLine("Среднее арифметическое:" + m)
```

Вывод массива

Под выводом массива понимается вывод на экран значений всех элементов массива. Наиболее просто вывести массив можно при помощи инструкции `For`, используя счетчик циклов для доступа к элементам массива.

Примеры:

```
For i=1 To 10
  TextWindow.WriteLine(a[i])
EndFor


' Перед каждым значением выводится имя элемента
For i=1 To 10
  TextWindow.WriteLine("a[" + i + "]=" + a[i])
EndFor
```

В качестве примера в листинге 6.2 приведена программа "Отклонение от среднего", которая путем обработки массива `A`, введенного пользователем с клавиатуры, формирует массив `B`, элементы которого содержат отклонения от среднего арифметического элементов массива `A`. Значение `i`-го элемента массива `B` вычисляется по формуле: `B[i]=A[i]-M`, где `M` — среднее арифметическое элементов массива `A`.

Листинг 6.2. Ввод и обработка массива

```
' Ввод и обработка массива
TextWindow.Write("Отклонение от среднего")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 5
```

```
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

' Вычисление суммы элементов
s = 0 ' сумма элементов массива
For i=1 To 5
  s = s + a[i]
EndFor

' Вычисление среднего арифметического
m = s / 5

' Формирование массива b
For i=1 To 5
  b[i] = a[i] - m
EndFor

TextWindow.WriteLine("----------")
TextWindow.WriteLine("Сумма элементов массива: " + s)
TextWindow.WriteLine("Среднее арифметическое: " + m)

TextWindow.WriteLine("Отклонение от среднего:")
' Вывод массива b
For i = 1 To 5
  TextWindow.Write(b[i] + "     ")
EndFor

TextWindow.WriteLine("")
```

Поиск минимального элемента

Задача поиска минимального элемента массива заключается в определении номера элемента, значение которого не превышает значения остальных элементов массива. Решается задача путем перебора (просмотра) всех элементов массива, т. к. минимальным может быть любой элемент массива.

Алгоритм поиска минимального элемента можно сформулировать так:

- предположим, что первый элемент массива является минимальным;
- будем последовательно сравнивать элементы массива (начиная со второго) с минимальным. Если текущий элемент меньше минимального, то будем считать минимальным этот элемент (запомним его номер).

Программа, реализующая описанный алгоритм, приведена в листинге 6.3, пример ее работы — на рис. 6.2.

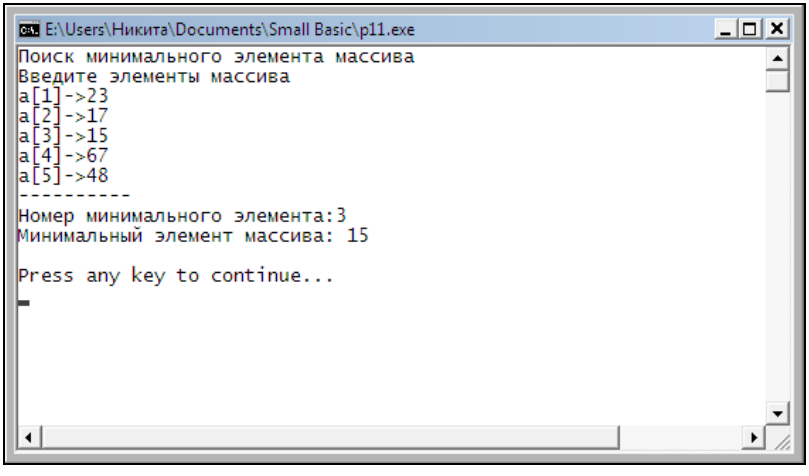


Рис. 6.2. Поиск минимального элемента массива

Листинг 6.3. Поиск минимального элемента массива

```
' Поиск минимального элемента массива
TextWindow.WriteLine("Поиск минимального элемента массива")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 5
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

' Предположим, что первый элемент массива минимальный
m = 1    ' номер минимального элемента

' Сравним i-й элемент с минимальным
For i=2 To 5
  If a[i] < a[m] Then  ' i-й элемент меньше минимального?
    ' да, меньше
    m = i
  EndIf
EndFor

' Вывод результата
TextWindow.WriteLine("----------")
TextWindow.WriteLine("Номер минимального элемента:" + m)
TextWindow.WriteLine("Минимальный элемент массива: " + a[m])

TextWindow.WriteLine("")
```

Сортировка массива

Под сортировкой массива понимается процесс перестановки элементов с целью упорядочивания их в соответствии с каким-либо критерием.

Различают сортировку по возрастанию и убыванию.

Массив, для элементов которого выполняется условие:

$a(1) \le a(2) \le \ldots a(i-1) \le a(i) \le a(i+1) \ldots \le a(k)$

называется упорядоченным по возрастанию.

Массив, для элементов которого выполняется условие:

$a(1) \ge a(2) \ge \ldots a(i-1) \ge a(i) \ge a(i+1) \ldots \ge a(k)$

называется упорядоченным по убыванию.

Задача сортировки распространена в информационных системах и часто используется как предварительный этап задачи поиска, т. к. поиск в упорядоченном (отсортированном) массиве можно выполнить значительно быстрее, чем в неупорядоченном.

Существует много методов сортировки массивов. Здесь мы рассмотрим два:

- ❒ метод прямого выбора;
- ❒ метод обмена ("пузырька").

Сортировка методом прямого выбора

Алгоритм сортировки массива по возрастанию методом прямого выбора может быть представлен так:

1. Просматривая массив от первого элемента, найти минимальный элемент и поместить его на место первого элемента, а первый — на место минимального (поменять элементы местами).
2. Просматривая массив от второго элемента, найти минимальный элемент и поместить его на место второго элемента, а второй — на место минимального.
3. Повторить описанные действия для всех (кроме последнего) оставшихся элементов массива.

Программа сортировки массива по возрастанию представлена в листинге 6.4. Для демонстрации процесса сортировки состояние массива отображается после каждого цикла сортировки (рис. 6.3).

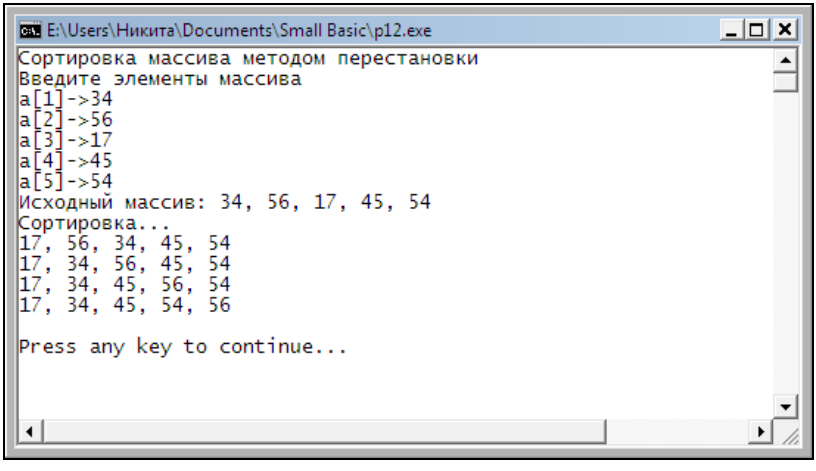


Рис. 6.3. Сортировка массива методом прямого выбора

Листинг 6.4. Сортировка массива по возрастанию методом прямого выбора

```
' Сортировка массива методом перестановки элементов
TextWindow.WriteLine("Сортировка массива методом
перестановки")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 5
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

' Исходный массив
TextWindow.Write("Исходный массив: ")
For i=1 To 4
  TextWindow.Write(a[i] +", ")
EndFor
TextWindow.WriteLine(a[5])

TextWindow.WriteLine("Сортировка...")
For i = 1 To 4
  ' Поиск минимального элемента в части массива
  ' от i-го элемента массива

  m = i  ' предположим, что i-й элемент массива минимальный
  ' Сравним остальные элементы с минимальным
  For j = i + 1 To 5
    If a[j] < a[m] Then
      m = j
    EndIf
```

```
  EndFor

  ' Здесь найден минимальный элемент в области от a[i].
  ' Обменяем местами i-й и минимальный элементы
  buf = a[i]
  a[i] = a[m]
  a[m] = buf

  ' Здесь цикл сортировки завершен

  ' Отладочная "печать" — массив после цикла сортировки
  For k=1 To 4
    TextWindow.Write(a[k] +", ")
  EndFor
  TextWindow.WriteLine(a[5])

EndFor
```

Сортировка методом "пузырька"

В основе алгоритма лежит идея обмена соседних элементов массива, если следующий элемент меньше предыдущего (при сортировке по возрастанию).

Каждый элемент массива, начиная с первого, сравнивается со следующим, и если он больше следующего, то элементы меняются местами. Таким образом, элементы с меньшим значением продвигаются к началу массива ("всплывают"), а элементы с большим значением — "тонут" (поэтому данный метод сортировки часто называют методом "пузырька"). Чтобы отсортировать массив, описанный выше процесс обменов надо повторить $N-1$ раз, где N — количество элементов массива.

Рисунок 6.4 показывает процесс сортировки массива методом "пузырька". Дуги отмечают элементы, которые следует обменять местами на очередном шаге цикла обменов.

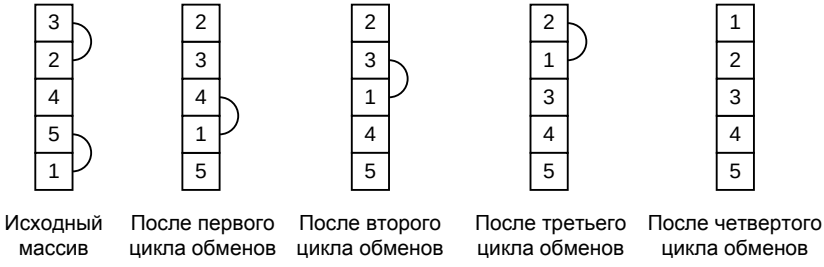


Рис. 6.4. Процесс сортировки массива методом "пузырька"

Программа, реализующая алгоритм сортировки массива по возрастанию методом "пузырька", приведена в листинге 6.5. Для демонстрации процесса сортировки состояние массива отображается после выполнения каждого цикла обменов (рис. 6.5).

Листинг 6.5. Сортировка массива методом "пузырька"

```
' Сортировка массива методом "пузырька"
TextWindow.WriteLine("Сортировка массива методом пузырька")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 5
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

' Исходный массив
TextWindow.Write("Исходный массив: ")
For i=1 To 4
  TextWindow.Write(a[i] +", ")
EndFor
TextWindow.WriteLine(a[5])

TextWindow.WriteLine("Сортировка...")
' Количество циклов обмена на 1 меньше размера массива
For i = 1 To 4
  For j = 1 To 4
    If a[j] > a[j + 1] Then
      ' обменять местами a[j] и a[j+1]
      buf = a[j]
      a[j] = a[j + 1]
      a[j + 1] = buf
    EndIf
  EndFor
```

```
  ' Здесь цикл обменов завершен


  ' Отладочная "печать" - массив после цикла сортировки
  For k=1 To 4
    TextWindow.Write(a[k] +", ")
  EndFor
  TextWindow.WriteLine(a[5])


EndFor
```

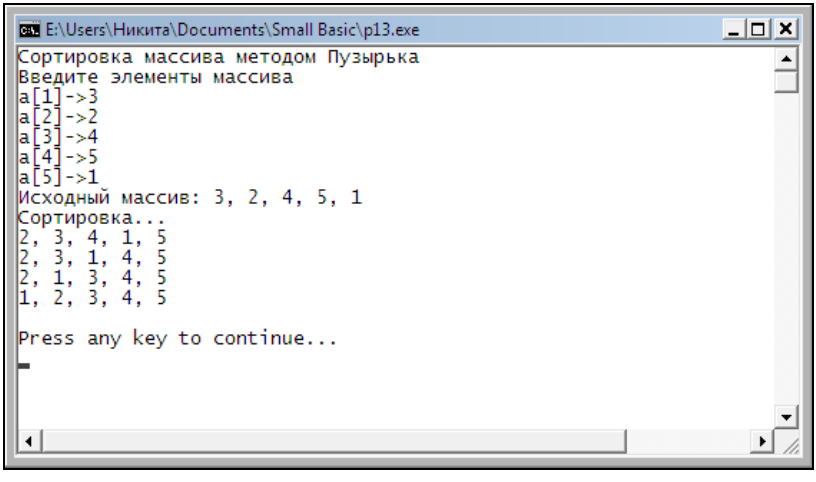


Рис. 6.5. Сортировка массива методом "пузырька"

Поиск в массиве

При решении многих задач часто возникает необходимость установить, содержит ли массив определенную информацию или нет. Например, проверить, есть ли в массиве фамилий фамилия Петров. Задачи такого типа называются поиском в массиве.

Метод перебора

Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой — это алгоритм перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

В листинге 6.6 приведен текст программы поиска в массиве целых чисел методом перебора элементов. Перебор элементов массива осуществляется в цикле `While`, в теле которого инструкция `If` сравнивает текущий (`i`-й) элемент массива с образцом. Если текущий элемент равен образцу, то переменной `found` присваивается значение 1. Цикл выполняется до тех пор, пока не будет найден искомый элемент (в этом случае значение переменной `found` равно 1) и номер текущего (проверяемого) элемента меньше или равен количеству элементов массива. По завершении цикла, проверив значение переменной `found`, можно определить, успешен поиск или нет. Пример работы программы показан на рис. 6.6.

Листинг 6.6. Поиск в массиве (метод перебора)

```
' Поиск в массиве методом перебора

TextWindow.WriteLine("Поиск в массиве методом перебора")

TextWindow.WriteLine("Введите элементы массива")
For i = 1 To 5
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

TextWindow.Write("Образец для поиска -> ")
obr = TextWindow.ReadNumber()

TextWindow.Write("Массив: ")
For i=1 To 4
  TextWindow.Write(a[i] +", ")
EndFor
TextWindow.WriteLine(a[5])

TextWindow.WriteLine("Поиск...")
found = 0 ' пусть нужного элемента в массиве нет
i = 1     ' номер проверяемого элемента массива

While (found = 0) And (i <= 5)

  If a[i] = obr Then
    ' нужный элемент найден
    found = 1
  Else
    i = i + 1
```

```
  EndIf

EndWhile

If found = 1 Then
  TextWindow.WriteLine("Совпадение с элементом номер " + i)
Else
   TextWindow.WriteLine("Искомого элемента в массиве нет")
EndIf
```

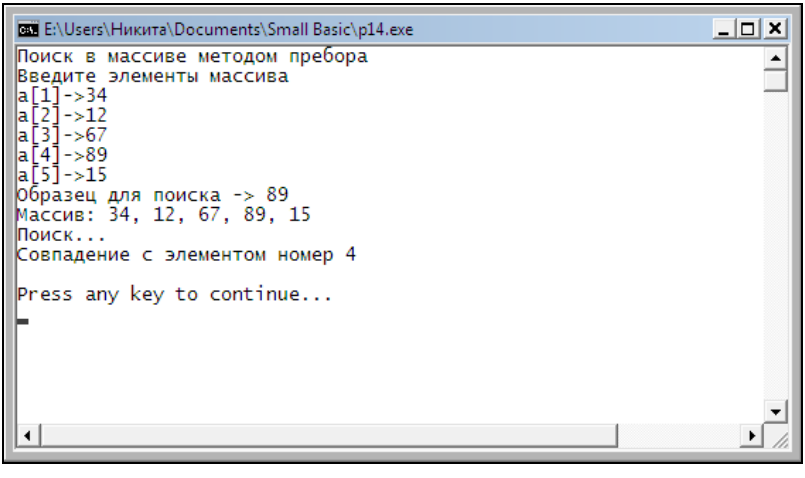


Рис. 6.6. Поиск в массиве методом перебора

Алгоритм перебора на практике применяется редко, т. к. его эффективность невысока: если массив неупорядочен, то нужный элемент может быть как в начале, так и в конце массива. Очевидно, чем больше массив, тем в среднем дольше программа будет искать нужный элемент.

Бинарный поиск

На практике часто приходится иметь дело с информацией, которая упорядочена по некоторому критерию. Например, список фамилий, как правило, упорядочен по алфавиту, массив метеорологических данных — по датам наблюдений. Если информация упорядочена, то можно сделать предположение, в какой части списка (ближе к началу или ближе к концу) она находится, и искать ее именно там.

Для поиска в упорядоченных массивах применяют другие, более эффективные по сравнению с методом простого перебора, алгоритмы, один из которых — метод бинарного поиска.

Суть метода бинарного поиска заключается в следующем. Выбирается средний (по номеру) элемент упорядоченного массива (элемент с номером `m`), и образец сравнивается с этим элементом (рис. 6.7).

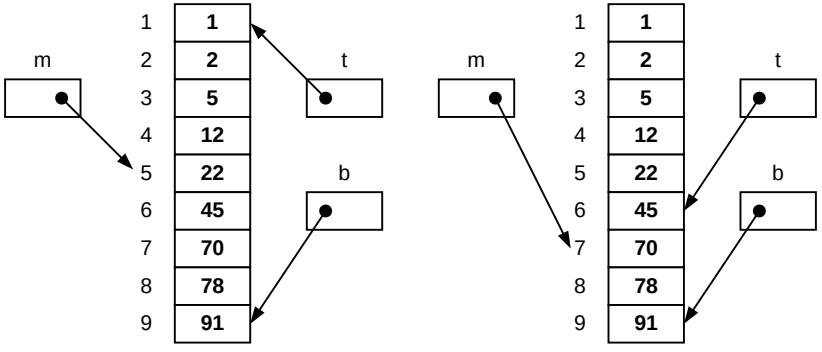


Рис. 6.7. Выбор среднего элемента массива при бинарном поиске

Если средний элемент равен образцу, то задача решена. Если образец меньше среднего элемента (предполагается, что массив упорядочен по возрастанию), то искомый элемент расположен выше (до) среднего элемента (между элементами с номерами `t` и `m-1`).

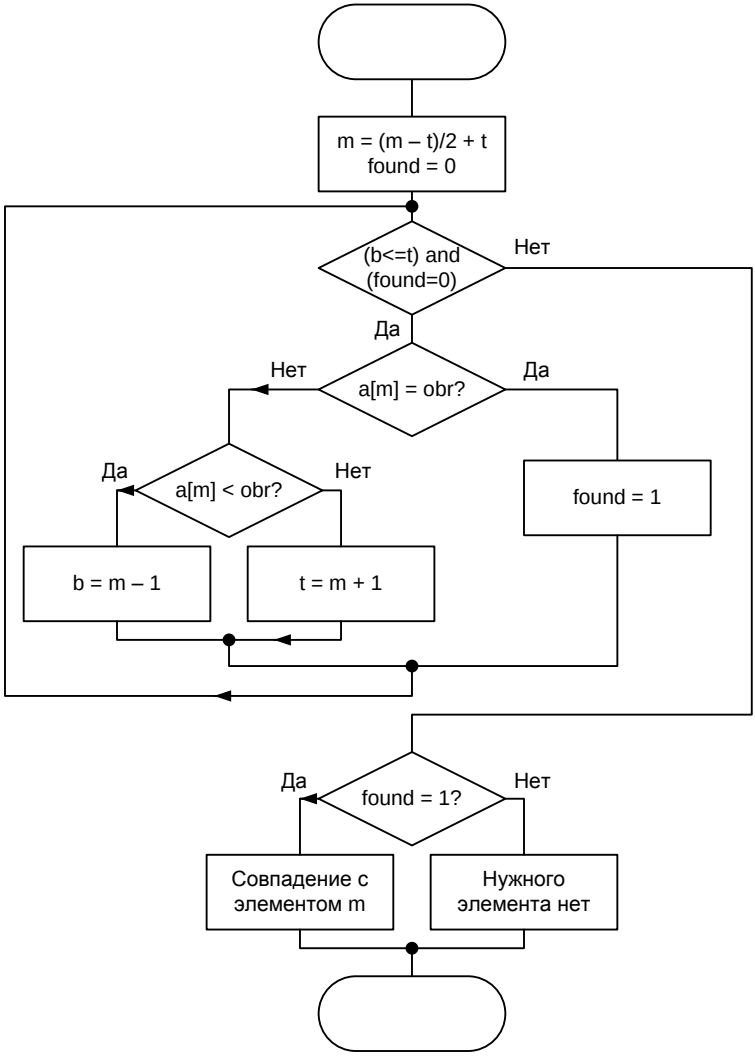


Рис. 6.8. Алгоритм бинарного поиска в упорядоченном по возрастанию массиве

Если образец больше среднего элемента, то искомый элемент расположен ниже (после) среднего (между элементами с номерами `m+1` и `b`). После того как будет определена часть массива, в которой может находиться искомый элемент, поиск проводят в этой части. Номер среднего элемента вычисляется по формуле `(b – t) / 2 + t`. Следует обратить внимание на то, что в программе для вычисления номера среднего элемента используется функция `Math.Round`, которая округляет результат деления.

Алгоритм бинарного поиска в упорядоченном массиве представлен на рис. 6.8, программа — в листинге 6.7.

В программу добавлены инструкции вывода значений переменных `t`, `b` и `m`. Эта информация полезна для понимания сути алгоритма. Пример работы программы приведен на рис. 6.9.

Листинг 6.7. Бинарный поиск в упорядоченном массиве

```
' Бинарный поиск в упорядоченном по возрастанию массиве
TextWindow.WriteLine("Бинарный поиск в упорядоченном
массиве")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 9
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

TextWindow.Write("Образец для поиска -> ")
obr = TextWindow.ReadNumber()

TextWindow.Write("Массив: ")
```

```
For i=1 To 8
  TextWindow.Write(a[i] +", ")
EndFor
TextWindow.WriteLine(a[9])

TextWindow.WriteLine("Поиск...")

t = 1  '
b = 9
m = Math.Round ((b - t) / 2) + t

found = 0
k = 0   ' количество сравнений с образцом

TextWindow.WriteLine("t  b  m")

While (found = 0) And (t <= b)
  m = Math.Round ((b - t) / 2) + t
  TextWindow.Write(t + "  " )
  TextWindow.Write(b + "  " )
  TextWindow.WriteLine( m)

  k = k + 1
  If a[m] = obr Then
    found = 1
  Else
    If obr < a[m] Then
      ' искомый элемент в "верхней" части массива
      b = m - 1
    Else
```

```
      ' искомый элемент в "нижней" части массива
      t = m + 1
    EndIf
  EndIf
EndWhile

If found = 1 Then
  TextWindow.WriteLine("Совпадение с элементом номер " + m)
  TextWindow.WriteLine("Количество сравнений с образцом: " +
k)
Else
  TextWindow.WriteLine("Искомого элемента в массиве нет")
EndIf
```

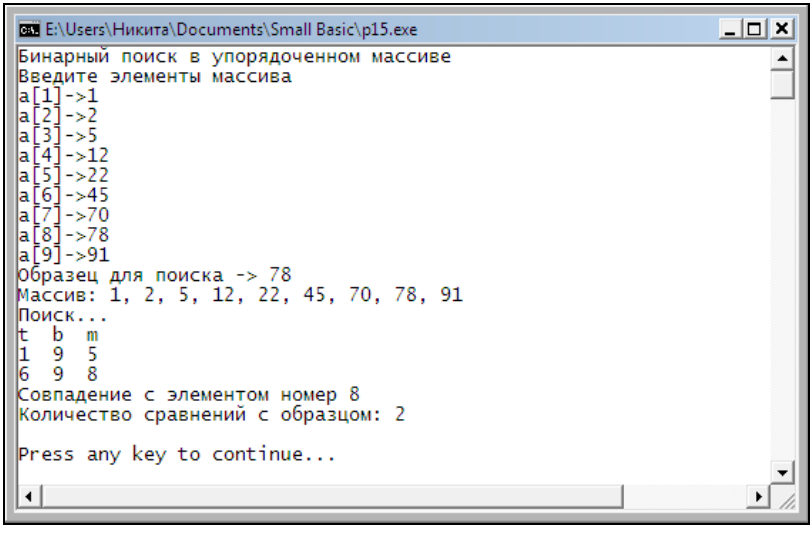


Рис. 6.9. Бинарный поиск в упорядоченном массиве

Контрольные вопросы

1. Что делает следующий фрагмент кода?

```
For i = 1 To 10
    a[i] = 0
EndFor
```

2. Что делает следующий фрагмент кода?

```
For i  = 1 To 10
    s = s + a[i]
EndFor
m = s / 10
```

3. Какие методы сортировки массивов вы знаете?
4. Какие методы поиска в массиве вы знаете? В какой ситуации следует применять каждый из них?
5. Напишите программу, которая проверяет, является ли массив, введенный пользователем с клавиатуры, упорядоченным по возрастанию.

Двумерные массивы

Исходные данные для решения многих задач часто представляют в табличной форме. Например, ниже приведена таблица результата продаж автомобилей за год.

	I кв.	II кв.	III кв.	IV кв.
Модель_1				
Модель_2				
Модель_3				
Модель_4				
Модель_5				

В программе приведенную таблицу можно представить как совокупность массивов `m1`, `m2`, `m3`, `m4` и `m5`. При этом предполагается, что каждый из массивов состоит из четырех элементов и хранит информацию о количестве проданных автомобилей одной марки. Эту таблицу можно представить по-другому, как совокупность массивов `q1`, `q2`, `q3` и `q4`. В этом случае каждый массив, состоящий из пяти элементов, будет хранить информацию о количестве проданных автомобилей всех моделей, соответственно, в первом, втором, третьем и четвертом кварталах.

Для представления таблиц вместо совокупности одномерных массивов можно использовать двумерные массивы. Графически двумерный массив можно изобразить так, как показано на рис. 6.10.

Чтобы получить доступ к элементу двумерного массива, нужно указать имя массива и индексы строки и столбца, на пересече-

нии которых находится нужный элемент. Индексы заключаются в квадратные скобки. В качестве индекса следует использовать выражение целого типа — константу или переменную.

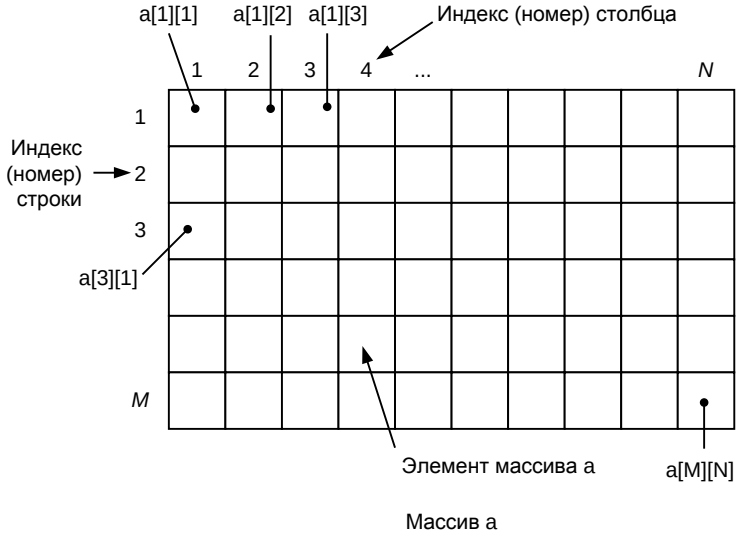


Рис. 6.10. Графическое представление двумерного массива

Примеры:

```
a[5][1]
a[i][j]
```

Двумерные массивы обычно вводят с клавиатуры и выводят на экран по строкам, т. е. сначала все элементы первой строки, затем второй и т. д. Это удобно делать при помощи "вложенных" инструкций `For`. Приведенная в листинге 6.8 программа демонстрирует процессы ввода и вывода двумерного массива.

Листинг 6.8. Ввод и вывод двумерного массива

```
' Ввод и вывод двумерного (3x4) массива
TextWindow.WriteLine("Ввод и вывод двумерного (3x4) массива")

'  Ввод двумерного массива
TextWindow.WriteLine("Введите элементы массива")
For i=1 To 3 ' 3 строки
  ' ввод элементов i-ой строки
  For j=1 To 4    ' 4 элемента в строке
    ' ввод j-го элемента i-ой строки
    TextWindow.Write("a["+i+"]["+j+"] ->")
    a[i][j] = TextWindow.ReadNumber()
  EndFor
EndFor

TextWindow.WriteLine("")

' Вывод двумерного массива
TextWindow.WriteLine("Введенный массив")
For i=1 To 3    ' 3 строки
  ' вывод i-ой строки
  For j=1 To 4  ' 4 элемента в строке
    ' вывод j-го элемента i-ой строки
    TextWindow.Write(a[i][j] + "  ")
  EndFor
  ' После вывода последнего элемента строки
  ' курсор надо переместить в следующую строку
  TextWindow.WriteLine("")
EndFor
```

В приведенной программе каждый раз, когда внутренний цикл (цикл "по `j`") завершается, внешний цикл (цикл "по `i`") увеличивает `i` на единицу, и внутренний цикл выполняется снова. Таким образом, компоненты массива `a` вводятся и выводятся в следующем порядке: `a[1][1]`, `a[1][2]`, `a[1][3]`, `a[1][4]`, `a[2][1]`, `a[2][2]`, `a[2][3]`, `a[2][4]`, `a[3][1]` и т. д. (рис. 6.11).

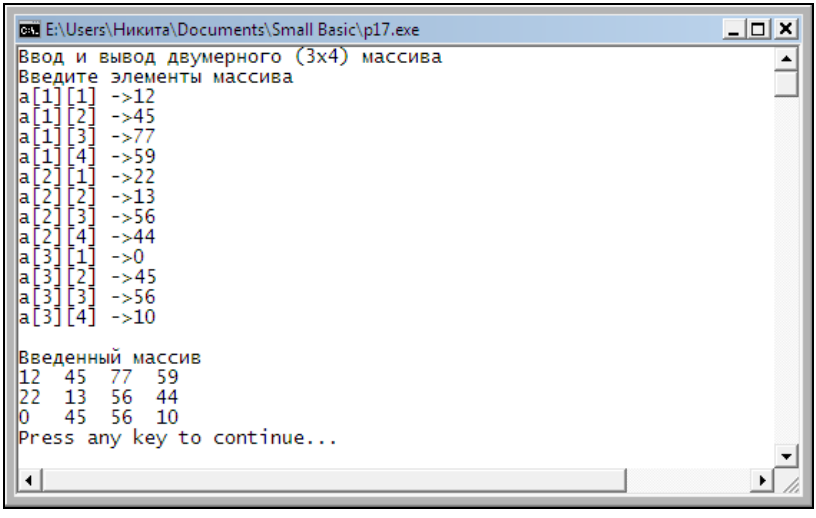


Рис. 6.11. Ввод и вывод двумерного массива

В качестве примера рассмотрим следующую задачу. Пусть есть данные о продажах автомобилей, и надо посчитать: общее количество проданных автомобилей, количество автомобилей, проданных в каждом квартале, и количество автомобилей каждой марки, проданных за год. Другими словами, надо найти сумму элементов массива, сумму элементов по столбцам и по строкам.

Результат обработки таблицы и исходные данные удобно объединить в одну таблицу:

	I кв.	II кв.	III кв.	IV кв.	Всего
Модель_1					
Модель_2					
Модель_3					
Всего					

В программе приведенную таблицу можно представить как двумерный массив 4×5 (4 строки, 5 столбцов).

Текст программы, которая решает поставленную задачу, приведен в листинге 6.9, ее окно — на рис. 6.12.

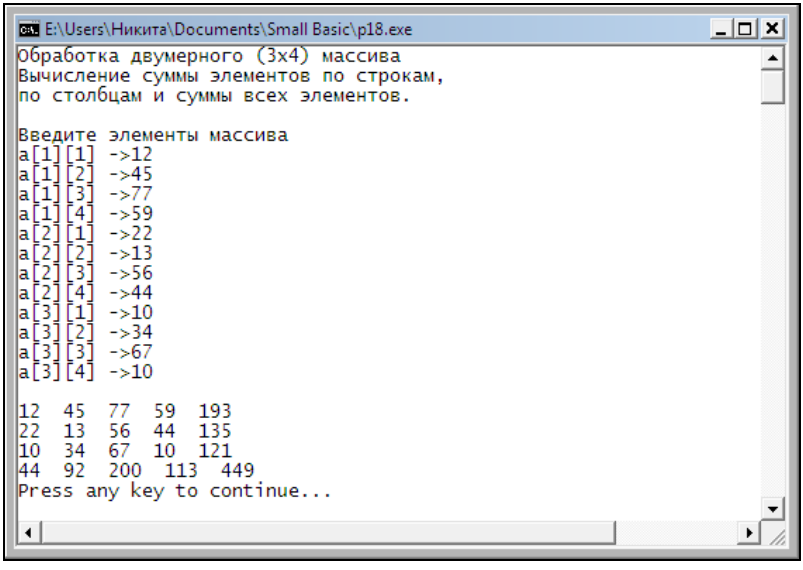


Рис. 6.12. Обработка двумерного массива

Листинг 6.9. Ввод и обработка двумерного массива

```
' Обработка двумерного массива
TextWindow.WriteLine("Обработка двумерного (3x4) массива")
TextWindow.WriteLine("Вычисление суммы элементов по
строкам,")
TextWindow.WriteLine("по столбцам и суммы всех элементов")

TextWindow.WriteLine("")

' Ввод  массива
TextWindow.WriteLine("Введите элементы массива")
For i=1 To 3 ' 3 строки
  ' Ввод элементов i-ой строки
  For j=1 To 4 ' 4 элемента в строке
    ' ввод j-го элемента i-ой строки
    TextWindow.Write("a["+i+"]["+j+"] ->")
    a[i][j] = TextWindow.ReadNumber()
  EndFor
EndFor

' Сумма элементов по строкам.
' Результат запишем в 5-й столбец массива
For i = 1 To 3 ' 3 строки
  ' Сумма 1-4 элементов i-ой строки
  sum = 0
  For j = 1 To 4
    sum = sum + a[i][j]
  EndFor
  a[i][5] = sum
EndFor

' Сумма элементов по столбцам.
' Результат запишем в 4-ю строку
```

```
For j = 1 To 4 ' 4 столбца
  ' Сумма 1-3 элементов столбца
  sum = 0
  For i = 1 To 3
    sum = sum + a[i][j]
  EndFor
  a[4][j] = sum
EndFor

' Сумма элементов массива.
' Можно сложить значения элементов 5-го столбца или
' 4-й строки или 1..4 элементы 1..3 строк.
' Реализуем последний способ
sum = 0
For i = 1 To 3
  For j = 1 To 4
    sum = sum + a[i][j]
  EndFor
EndFor

a[4][5] = sum

' Вывод результата обработки - массива 4x6
For i=1 To 4 ' 4 строки
  ' Вывод i-ой строки
  For j=1 To 5 ' 5 элементов в строке
    ' Вывод j-го элемента i-ой строки
    TextWindow.Write(a[i][j] + "  ")
  EndFor
  ' После вывода последнего элемента строки
  ' курсор надо переместить в следующую строку
  TextWindow.WriteLine("")
EndFor
```

Следующая программа, ее текст приведен в листинге 6.10, после вычисления суммы элементов строк, сортирует массив (строки таблицы) по содержимому 5-го столбца, хранящего суммы элементов строк. Сортировка выполняется методом "пузырька". Для обмена `i` и (`i`+1)-й строк массива в качестве буфера используется последняя (дополнительная) строка массива (количество строк и столбцов в массиве на единицу больше, чем количество строк и столбцов исходной таблицы). Пример работы программы приведен на рис. 6.13.

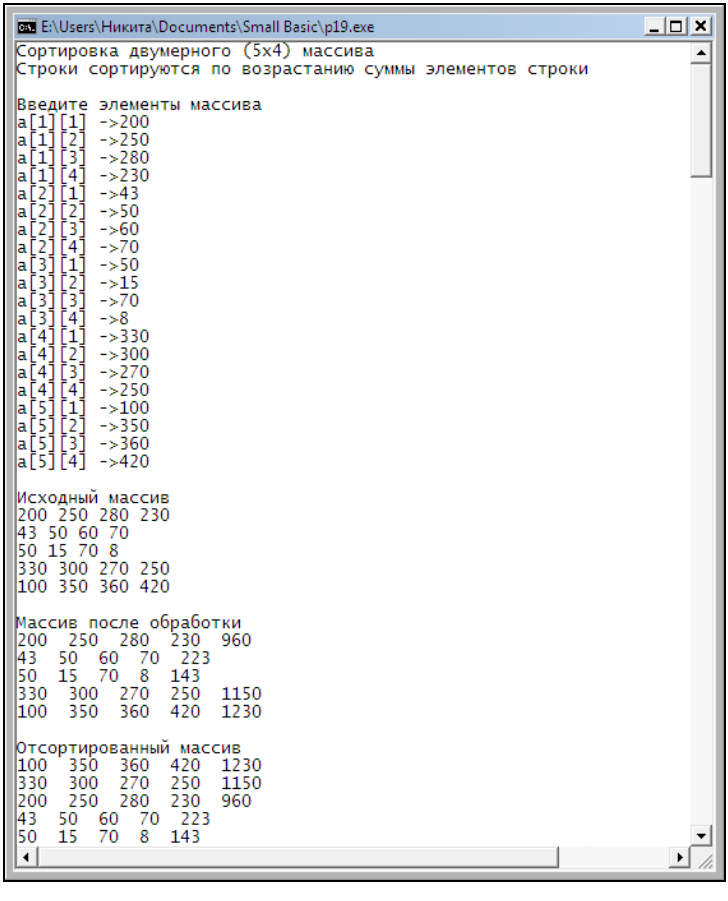


Рис. 6.13. Сортировка двумерного массива

Листинг 6.10. Сортировка двумерного массива

```
' Сортировка двумерного массива
TextWindow.WriteLine("Сортировка двумерного (5x4) массива")
TextWindow.WriteLine("Строки сортируются по возрастанию суммы
элементов строки")
TextWindow.WriteLine("")

' Ввод  массива
TextWindow.WriteLine("Введите элементы массива")
For i=1 To 5 ' 5 строк
  ' Ввод элементов i-ой строки
  For j=1 To 4 ' 4 элемента в строке
    ' Ввод j-го элемента i-ой строки
    TextWindow.Write("a["+i+"]["+j+"] ->")
    a[i][j] = TextWindow.ReadNumber()
  EndFor
EndFor

' Вывод  исходного массива
TextWindow.WriteLine("")
TextWindow.WriteLine("Исходный массив")
For i=1 To 5
  For j=1 To 4
    TextWindow.Write(a[i][j]+ " ")
  EndFor
  TextWindow.WriteLine("")
EndFor

' Вычислить сумму элементов по строкам.
' Результат запишем в 5-й столбец массива
For i = 1 To 5 ' 5 строк
```

```
  ' Сумма 1-4 элементов i-ой строки
  sum = 0
  For j = 1 To 4
    sum = sum + a[i][j]
  EndFor
  a[i][5] = sum
EndFor

' Вывод  массива после обработки
TextWindow.WriteLine("")
TextWindow.WriteLine("Массив после обработки")
For i=1 To 5
  For j=1 To 5
    TextWindow.Write(a[i][j] + "  ")
  EndFor
  TextWindow.WriteLine("")
EndFor

' Сортировка массива по содержимому 5-го столбца
For i = 1 To 4 ' кол-во циклов обменов на 1 меньше кол-ва
строк
  For j = 1 To 4 ' кол-во обменов на 1 меньше кол-ва строк
    ' Сортировка по содержимому 5-го столбца
    If a[j][5] < a[j+ 1][5] Then
      ' обменять j-ю и j+1 строки
      ' 6-я строка - буфер
      For k = 1 To 5 ' в строке 5 ячеек
        a[6][k] = a[j][k]
        a[j][k] = a[j + 1][k]
        a[j + 1][k] = a[6][k]
      EndFor
    EndIf
```

```
  EndFor
EndFor

' Вывод отсортированного массива (3x5)
TextWindow.WriteLine("")
TextWindow.WriteLine("Отсортированный массив")
For i=1 To 5     '  5 строк
  ' Вывод i-ой строки
  For j=1 To 5  ' 5 элементов в строке
    ' Вывод j-го элемента i-ой строки
    TextWindow.Write(a[i][j] + "  ")
  EndFor
  TextWindow.WriteLine("")
EndFor
```

Ошибки при работе с массивами

При работе с массивами наиболее распространенной ошибкой является обращение к несуществующему элементу. Причиной этой ошибки является выход значения индексного выражения за допустимые границы.

В качестве примера рассмотрим программу, приведенную в листинге 6.11. Синтаксических ошибок в ней нет. Однако программа работает неправильно (рис. 6.14). Причина этого в том, что в качестве границы цикла `For` по `j` указана константа 5. Поэтому, когда значение переменной `j` становится равным 5, в условии инструкции `If` сравнивается пятый и несуществующий шестой элементы массива. Чтобы устранить эту ошибку, в качестве верхней границы изменения индекса надо указать 4.

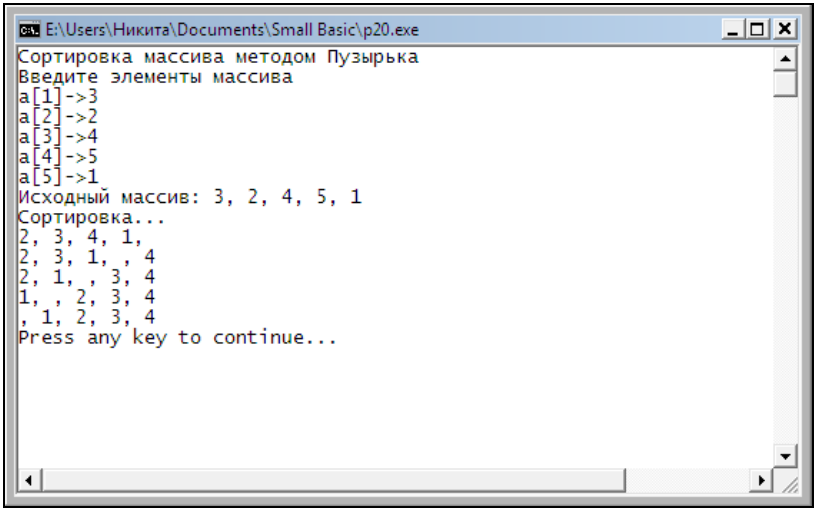
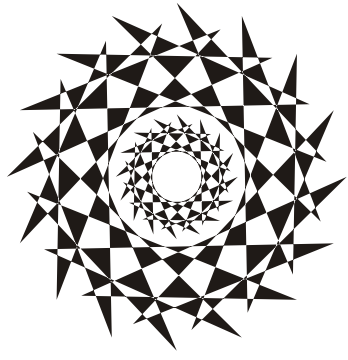


Рис. 6.14. Сортировка массива (программа работает неправильно)

Листинг 6.11. Сортировка массива методом "пузырька" (программа с ошибкой)

```
' Сортировка массива методом "пузырька".
' В программе есть ошибка!!!
' Программа работает неправильно!!!
TextWindow.WriteLine("Сортировка массива методом Пузырька")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 5
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

' Исходный массив
TextWindow.Write("Исходный массив: ")
For i=1 To 4
  TextWindow.Write(a[i] +", ")
EndFor
TextWindow.WriteLine(a[5])

TextWindow.WriteLine("Сортировка...")
' Количество циклов обмена на 1 меньше размера массива
For i = 1 To 5
  For j = 1 To 5 ' ЗДЕСЬ ОШИБКА! Должно быть 4
    If a[j] > a[j + 1] Then
      ' Обменять местами a[j] и a[j+1]
      buf = a[j]
      a[j] = a[j + 1]
      a[j + 1] = buf
```

```
    EndIf
  EndFor

  ' Здесь цикл обменов завершен

  ' Отладочная "печать" - массив после цикла сортировки
  For k=1 To 4
    TextWindow.Write(a[k] +", ")
  EndFor
  TextWindow.WriteLine(a[5])

EndFor
```



Глава 7

Подпрограмма

Довольно часто в программе есть фрагменты, состоящие из одинаковых инструкций. Например, в рассмотренной в *главе 6* (см. листинг 6.4) программе сортировки двумерного массива массив выводится на экран два раза: исходный и отсортированный. Чтобы не набирать несколько раз одну и ту же последовательность инструкций, их можно оформить как *подпрограмму* (вместо термина "подпрограмма" часто используют термин "процедура") и затем в то место программы, где должны быть выполнены эти инструкции, поместить команду вызова подпрограммы.

Объявление подпрограммы в общем виде выглядит так:

```
Sub Имя
    ' здесь инструкции подпрограммы
EndSub
```

Слово `Sub` (от англ. *subprogram* — подпрограмма) показывает, что далее следуют инструкции подпрограммы, слово `EndSub` отмечает конец подпрограммы. Идентификатор `Имя` определяет имя подпрограммы.

В качестве примера в листинге 7.1 приведена подпрограмма WriteArray, которая выводит на экран массив `a`.

Листинг 7.1. Пример подпрограммы (процедуры)

```
' Процедура WriteArray выводит на экран массив
Sub WriteArray
  For k=1 To 4
    TextWindow.Write(a[k] +",")
  EndFor
  TextWindow.WriteLine(a[5])
EndSub
```

Чтобы процедура была выполнена, ее надо *вызвать* — указать имя процедуры в тексте программы. В качестве примера в листинге 7.2 приведена программа сортировки массива, в которой для вывода массива на экран используется процедура. Следует обратить внимание на то, что процедуре доступны все переменные программы. Поэтому, если процедура вызывается в цикле (а в рассматриваемом примере это так), она не должна менять значения переменной счетчика циклов, т. е. в качестве переменной счетчика циклов внутри процедуры следует использовать другую переменную, отличную от переменной счетчика циклов в главной программе.

Листинг 7.2. Сортировка массива (пример объявления и использования процедуры)

```
' Сортировка массива методом перестановки элементов.
' Для вывода массива на экран используется процедура.

' Процедура WriteArray выводит на экран массив
Sub WriteArray
  For k=1 To 4
    TextWindow.Write(a[k] +",")
  EndFor
  TextWindow.WriteLine(a[5])
EndSub

' Основная программа
TextWindow.WriteLine("Сортировка массива методом
перестановки")

TextWindow.WriteLine("Введите элементы массива")

' Ввод массива
For i = 1 To 5
  TextWindow.Write("a[" + i +"]->")
  a[i] = TextWindow.ReadNumber()
EndFor

' Вывести исходный массив
TextWindow.WriteLine("Исходный массив")
WriteArray() ' вызов процедуры

TextWindow.WriteLine("Сортировка...")
For i = 1 To 4
  ' Поиск минимального элемента в части массива
```

```
  ' от i-го элемента массива

  m = i ' предположим, что i-й элемент массива минимальный
  ' Сравним остальные элементы с минимальным
  For j = i + 1 To 5
    If a[j] < a[m] Then
      m = j
    EndIf
  EndFor

  ' Здесь найден минимальный элемент в области от a[i].
  ' Обменяем местами i-й и минимальный элементы.
  buf = a[i]
  a[i] = a[m]
  a[m] = buf

  ' Здесь цикл сортировки завершен

  ' Отладочная "печать" - вывод массива после цикла сортировки
  WriteArray()

EndFor

' Здесь массив отсортирован
TextWindow.WriteLine("Сортировка выполнена")
WriteArray() ' Вызов процедуры
```

Следует обратить внимание на то, что подпрограмма Small Basic, в отличие от подпрограмм в других языках программирования (Visual Basic, Pascal, C++ и др.), для передачи в подпрограмму информации не используют механизм параметров. Информация в подпрограмму передается через внешние (глобальные) переменные. Поясним это на примере. Предположим, необходимо напечатать (вывести на экран) таблицу значений функции. Для рисования линий "шапки" и линии после последней строки будем использовать процедуру `Line`. Чтобы нарисовать три линии, процедуру `Line` надо вызвать три раза. Первый и третий раз линия должна рисоваться символом "равно", второй — символом "минус". Так как указать символ, которым надо рисовать линию в инструкции вызова процедуры `Line`, нельзя, то объявим в программе переменную `ch` и перед вызовом процедуры `Line` будем записывать символ, которым надо рисовать линию. Текст программы приведен в листинге 7.3, вид экрана — на рис. 7.1.

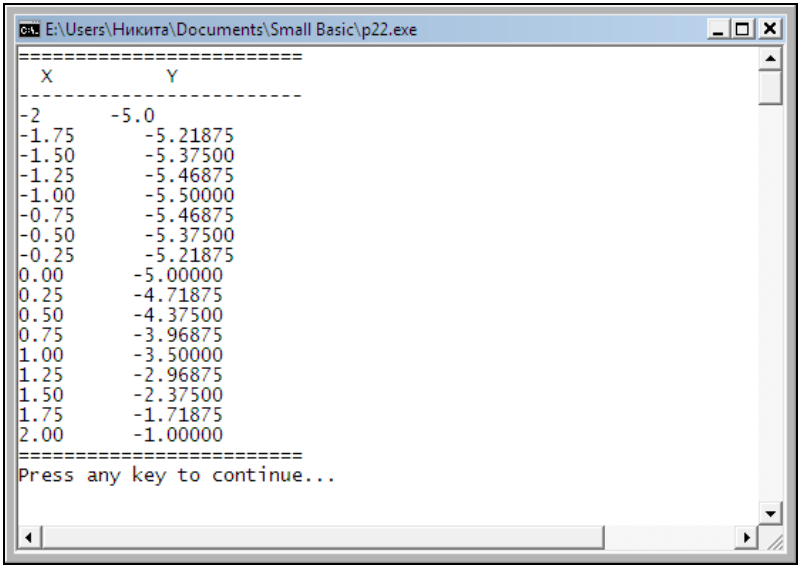


Рис. 7.1. Таблица значений функции (линии рисует процедура `Line`)

Листинг 7.3. Таблица функции (процедура с параметром)

```
' Использование подпрограммы с параметром

' Параметр процедуры Line
ch = ""    ' символ, которыми рисуется линия

' Процедура рисует линию из ch-символов
Sub Line
  For i=1 to 25
    TextWindow.Write(ch)
  EndFor
  TextWindow.WriteLine("")
EndSub

x1 = -2
x2 = 2
dx = 0.25

ch = "="    ' символ, которыми рисуется линия
Line()      ' рисовать линию

TextWindow.WriteLine("  X          Y")

ch = "-"    ' символ, которыми рисуется линия
Line()      ' рисовать линию

' Таблица
For x=x1 To x2 Step dx
  y = 0.5 * x* x + x - 5
  TextWindow.Write(x + "       ")
  TextWindow.WriteLine(y)
EndFor

ch = "="    ' символ, которыми рисуется линия
Line()      ' рисовать линию
```

Подпрограммы используют не только для того, чтобы избежать дублирования кода, но и для того, чтобы облегчить процесс разработки за счет разбиения большой (сложной) задачи на несколько небольших подзадач и реализации каждой подзадачи как подпрограммы. В качестве примера рассмотрим программу "Конвертер", которая позволяет пересчитать вес из фунтов в килограммы или из килограммов в фунты. Алгоритм программы "Конвертер" приведен на рис. 7.2, текст — в листинге 7.4, пример работы — на рис. 7.3.

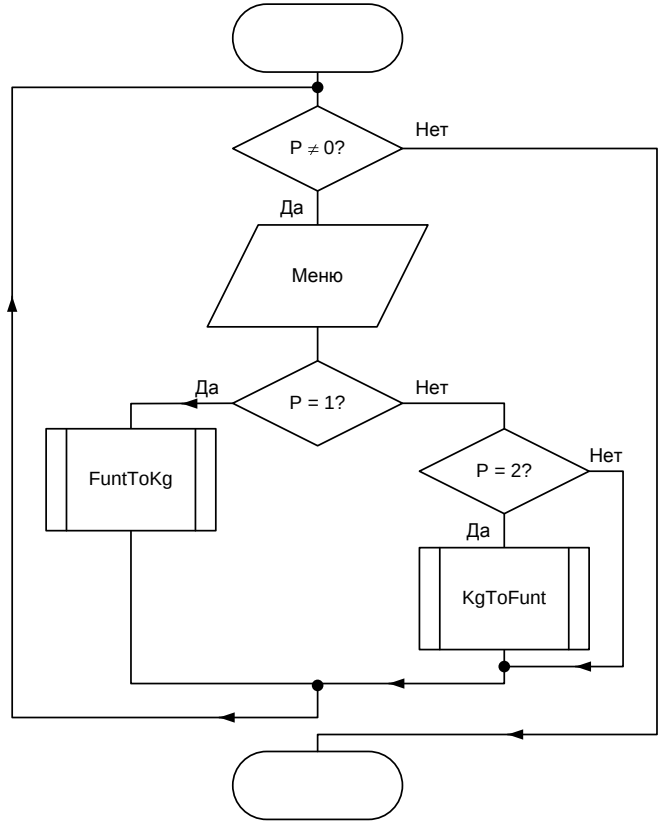


Рис. 7.2. Алгоритм программы "Конвертер"

Листинг 7.4. Конвертер

```
' Конвертер. Пересчет веса из фунтов в килограммы
' или из килограммов в фунты
' 1 фунт = 453,9 гр.

'  ПОДПРОГРАММЫ

' Пересчет веса из фунтов в килограммы
Sub FuntToKg
  TextWindow.Clear()
  TextWindow.WriteLine("Пересчет веса из фунтов в килограммы")
  TextWindow.Write("Вес в фунтах-> ")
  fnt = TextWindow.ReadNumber()
  kg = fnt * 0.4539

  TextWindow.Write(fnt + "ф. = ")
  TextWindow.WriteLine(kg + "кг.")

  TextWindow.WriteLine("")
  TextWindow.Write("Для возврата в меню нажмите <Enter>")
  TextWindow.Read()
EndSub

' Пересчет веса из килограммов в фунты
Sub KgToFunt
  TextWindow.Clear()
  TextWindow.WriteLine("Пересчет веса из килограммов в фунты")
  TextWindow.Write("Вес в килограммах-> ")
  kg = TextWindow.ReadNumber()
  fnt = kg / 0.4539
  TextWindow.Write(kg + "кг. = ")
  TextWindow.WriteLine(fnt + "ф.")
```

```
  TextWindow.WriteLine("")
  TextWindow.Write("Для возврата в меню нажмите <Enter>")
  TextWindow.Read()
EndSub


' ОСНОВНАЯ ПРОГРАММА
p = 1     ' чтобы войти в цикл While

While (p <> 0)
  ' Меню
  TextWindow.Clear()
  TextWindow.WriteLine("== КОНВЕРТЕР ==")
  TextWindow.WriteLine("1 - фунты -> килограммы")
  TextWindow.WriteLine("2 - килограммы -> фунты")
  TextWindow.WriteLine("0 - выход")
  TextWindow.WriteLine("")
  TextWindow.Write("Ваш выбор-> ")
  p = TextWindow.ReadNumber()

  ' Вызвать подпрограмму в соответствии
  ' с выбором пользователя
  If (p = 1) Then
    ' пересчет веса из фунтов в килограммы
    FuntToKg()
  else
    If (p = 2) then
      ' пересчет веса из килограммов в фунты
      KgToFunt()
    EndIf
  endif
EndWhile
```

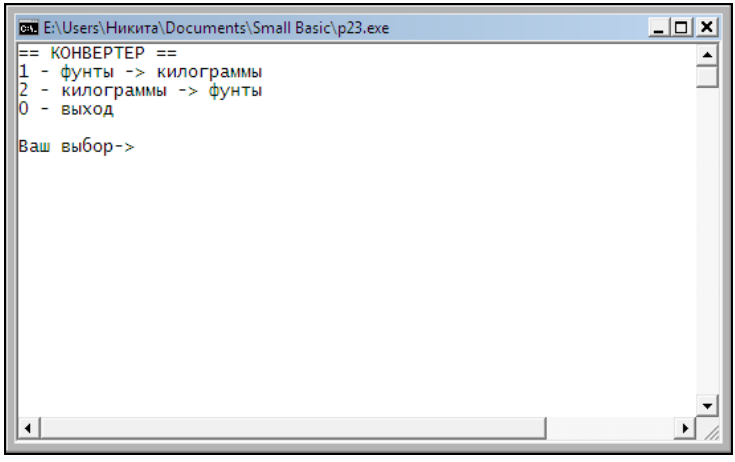


Рис. 7.3. Программа "Конвертер" (меню)

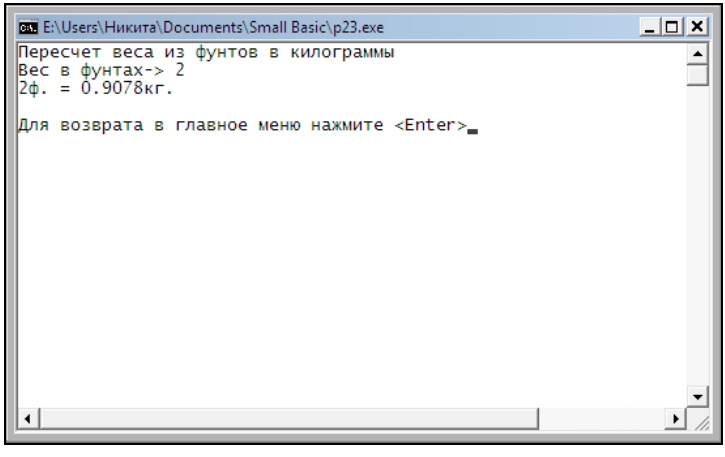
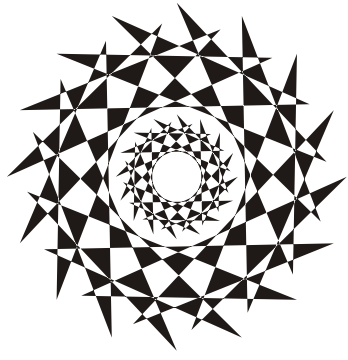


Рис. 7.3. Программа "Конвертер"

Контрольные вопросы

1. Для чего используются подпрограммы?
2. Как "вызвать" подпрограмму?
3. Как в подпрограмму Small Basic передать информацию?



Глава 8

Графика

В этой главе рассказывается, что надо сделать, чтобы в окне программы появилась фотография, картинка, сформированная из линий, прямоугольников, окружностей, точек, график или диаграмма. Также вы познакомитесь с принципами создания анимации, узнаете, как "оживить" картинку.

Программа Small Basic формирует графику на поверхности графического окна (Graphics Window). Для того чтобы графическое окно появилось на экране и внутри него был нарисован, например, прямоугольник, в программу надо поместить инструкцию вызова *метода*, рисующего прямоугольник. Так в результате выполнения инструкции

```
GraphicsWindow.DrawRectangle(10,10,50,30)
```

будет нарисован прямоугольник шириной в 50 и высотой в 30 пикселов, левый верхний угол которого будет находиться в точке с координатами (10, 10).

Графическая поверхность

Методы, рисующие графические примитивы внутри графического окна, рассматривают его поверхность как *холст*, состоящий из отдельных точек — пикселов. Положение пиксела характеризуется его горизонтальной (`x`) и вертикальной (`y`) координатами. Координаты отсчитываются от левого верхнего угла и возрастают слева направо (`x`) и сверху вниз (`y`). Точка, находящаяся в левом верхнем углу поверхности, имеет координаты (0, 0) (рис. 8.1).

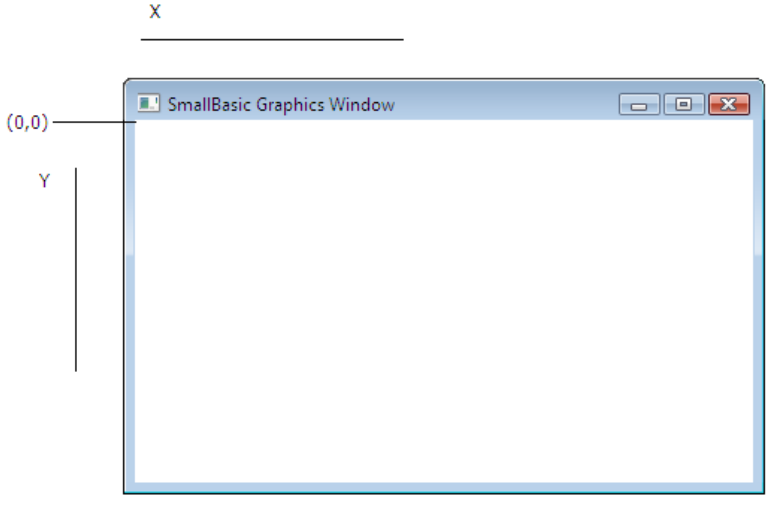


Рис. 8.1. Координаты точек графической поверхности отсчитываются от левого верхнего угла

Свойства графического окна `GraphicsWindow` (табл. 8.1) определяют вид графических примитивов, рисуемых на графической поверхности соответствующими методами. Так, например, свойство `PenColor` (Цвет пера) задает цвет линий и границ фигур, а свойство `BrushColor` (Цвет кисти) — цвет закраски областей.

Таблица 8.1. *Свойства* `GraphicsWindow`

Свойство	Описание
`PenColor`	Задает цвет линий и границ геометрических фигур
`PenWidth`	Задает толщину линий и границ геометрических фигур, рисуемых пером
`BrushColor`	Задает цвет закраски внутренних областей геометрических фигур
`FontName`	Задает шрифт, используемый для вывода текста на графическую поверхность
`FontSize`	Задает размер шрифта, используемого для вывода текста на графическую поверхность
`FontItalic`	Устанавливает, что шрифт, используемый для вывода текста на графическую поверхность, должен быть курсивный (italic)
`FontBold`	Устанавливает, что шрифт, используемый для вывода текста на графическую поверхность, должен быть полужирным (bold)
`Width`	Ширина окна
`Height`	Высота окна
`CanResize`	Устанавливает вид границы окна. Если свойству присвоить значение `False`, то граница станет тонкой, поэтому у пользователя не будет возможности изменить размер окна

Цвет линий и областей задается путем изменения соответственно значений свойств `PenWidth` и `PenColor`. В качестве значения свойства, определяющего цвет, следует использовать название цвета, например, `"Red"`, `"DarkRed"`, `"Green"`, `"YellowGreen"`, `"Gold"`, `"SkyBlue"`. Всего в Small Basic определены 142 цвета (9 оттенков красного цвета, 6 — розового (Pink), 6 — оранжевого, 11 — желтого, 18 пурпурного (Purple), 23 — зеленого, 25 — синего, 17 — коричневого, 17 — белого и 10 оттенков серого цвета). Некоторые из них приведены в табл. 8.2.

Таблица 8.2. *Некоторые цвета графического режима*

Константа	Цвет
"Red"	Красный
"DarkRed"	Темно-красный
"Pink"	Розовый
"LightPink"	Светло-розовый
"DeepPink"	Темно-розовый
"Orange"	Оранжевый
"DarkOrange"	Темно-оранжевый
"OrangeRed"	Оранжево-красный
"Yellow"	Желтый
"LightYellow"	Светло-желтый
"Gold"	Золотой
"Fuchsia"	Фуксия
"Magenta"	Маджента
"DarkViolet"	Темно-фиолетовый
"Purple"	Фиолетовый
"GreenYellow"	Зелено-желтый
"Lime"	Лайм
"SpringGreen"	Весенняя зелень
"SeaGreen"	Морская зелень
"ForestGreen"	Зеленый лес
"DarkGreen"	Темно-зеленый
"YellowGreen"	Желто-зеленый
"Olive"	Оливковый
"Aqua"	Морская волна
"SteelBlue"	Синяя сталь
"SkyBlue"	Небесно-синий

***Таблица 8.2** (окончание)*

Константа	Цвет
"Blue"	Синий
"DarkBlue"	Темно-синий
"Chocolate"	Шоколадный
"Brown"	Коричневый
"White"	Белый
"Snow"	Снег
"LightGray"	Светло-серый
"Silver"	Серебро
"Gray"	Серый
"Black"	Черный

Программист также может при помощи функции `GetColorFromRGB` определить свой собственный цвет. Метод-функция `GetColorFromRGB` возвращает код цвета, полученного путем смешивания красной, зеленой и синей красок в указанных пропорциях. У функции `GetColorFromRGB` три параметра: первый задает долю красной (`Red`), второй — зеленой (`Green`), третий — синей (`Blue`) составляющей. Значение каждого из параметров должно находиться в диапазоне от 0 до 255. Например, значением `GetColorFromRGB(205,127,50)` является код "золотого" цвета.

Далее приведен фрагмент кода, который показывает, как программист может определить и использовать в дальнейшем этот цвет.

```
' "золотой" цвет
color = GraphicsWindow.GetColorFromRGB(205,127,50)
GraphicsWindow.PenColor = color        ' теперь перо "золотое"
GraphicsWindow.DrawLine(10,10,100,10)  ' "золотая" линия
```

Графические примитивы

Картинку, чертеж или схему можно представить как совокупность графических *примитивов*: точек, линий, окружностей, дуг, текста и др.

Рисование графических примитивов обеспечивают соответствующие методы (табл. 8.3).

Таблица 8.3. *Методы рисования графических примитивов*

Метод	Действие
`DrawLine(x1,y1,x2,y2)`	Рисует линию из точки (`x1`, `y1`) в точку (`x2`, `y2`)
`DrawRectangle(x,y,w,h)`	Рисует прямоугольник шириной в `w` и высотой в `h` пикселов, левый верхний угол которого находится в точке с координатами (`x`, `y`)
`DrawEllipse(x,y,w,h)`	Рисует эллипс шириной в `w` и высотой в `h` пикселов. Параметры `x` и `y` задают координаты левого верхнего угла прямоугольника, в который вписывается эллипс
`SetPixel(x,y,color)`	Окрашивает точку с координатами (`x`, `y`) в цвет, заданный параметром `color`
`FillRectangle(x,y,w,h)`	Рисует закрашенный прямоугольник шириной в `w` и высотой в `h` пикселов, левый верхний угол которого находится в точке с координатами (`x`, `y`)
`FillEllipse(x,y,w,h)`	Рисует закрашенный эллипс шириной в `w` и высотой в `h` пикселов. Параметры `x` и `y` задают координаты левого верхнего угла прямоугольника, в который вписывается эллипс
`DrawText(x,y,st)`	Выводит от точки (`x`, `y`) текст `st`

Точка

Точку на графической поверхности рисует метод `SetPixel`.

Инструкция вызова метода `SetPixel` в общем виде выглядит так:

```
GraphicsWindow.SetPixel(x, y, Color)
```

Параметры *x* и *y* задают координаты точки графической поверхности, цвет которой надо изменить. Параметр *Color* задает цвет точки.

Пример:

```
GraphicsWindow.SetPixel(10,20,"Red")
```

Линия

Рисование прямой линии выполняет метод `DrawLine`.

Инструкция вызова метода `Line` в общем виде выглядит так:

```
GraphicsWindow.DrawLine(x1, y1, x2, y2)
```

Параметры *x1* и *y1* задают координаты точки начала линии, а параметры *x2*, *y2* — координаты точки конца.

Пример:

```
GraphicsWindow.DrawLine(10,20,100,20)
```

Цвет линии, рисуемой методом `DrawLine`, определяет значение свойства `PenColor`. Поэтому, чтобы нарисовать линию нужного цвета, надо, перед тем как вызвать метод `DrawLine`, присвоить соответствующее значение свойству `PenColor`.

Пример:

```
GraphicsWindow.PenColor = "Tomato"
GraphicsWindow.DrawLine(10,20,100,20)
GraphicsWindow.PenColor = "ForestGreen"
```

```
GraphicsWindow.DrawLine(10,30,100,30)
GraphicsWindow.PenColor = "SkyBlue"
GraphicsWindow.DrawLine(10,40,100,40)
```

Толщину линии, рисуемой методом `DrawLine`, определяет свойство `PenWidth` (по умолчанию линия рисуется толщиной в два пиксела). Поэтому, чтобы изменить толщину линии, надо присвоить значение свойству `PenWidth`.

Использование метода `DrawLine` демонстрирует программа "Сетка" (ее окно приведено на рис. 8.2, а текст — в листинге 8.1).

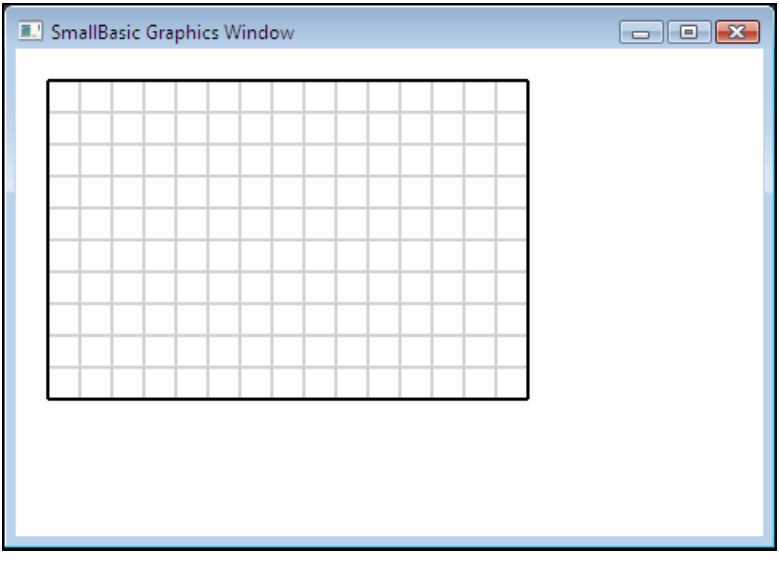


Рис. 8.2. Окно программы "Сетка"

Листинг 8.1. Сетка

```
' Координатная сетка

' левый верхний угол области отображения сетки
x0 = 20
y0 = 20

' размер области отображения сетки
w = 300
h = 200

' шаг сетки
dx = 20
dy = 20

' координаты точки начала линии
x = x0
y = y0

' цвет линий сетки
GraphicsWindow.PenColor = "LightGray"

' горизонтальные  линии
While (x <= x0+w)
  GraphicsWindow.DrawLine(x,y,x,y+h)
  x = x + dx
EndWhile

' вертикальные линии
x = x0
While (y <= y0+h)
```

```
  GraphicsWindow.DrawLine(x,y,x+w,y)
  y = y + dy
EndWhile

' граница
GraphicsWindow.PenColor = "Black"
GraphicsWindow.DrawLine(x0,y0,x0+w,y0)      ' верхняя граница
GraphicsWindow.DrawLine(x0,y0+h,x0+w,y0+h)  ' нижняя граница
GraphicsWindow.DrawLine(x0,y0,x0,y0+h)      ' левая гор. линия
GraphicsWindow.DrawLine(x0+w,y0,x0+w,y0+h)  ' правая гор. линия
```

Контрольные вопросы

1. Как задать цвет линии, рисуемой методом `DrawLine`?
2. Как задать толщину линии, рисуемой методом `DrawLine`?

Прямоугольник

Метод `DrawRectangle` рисует прямоугольник.

Инструкция вызова метода `DrawRectangle` в общем виде выглядит так:

```
GraphicsWindow.DrawRctangle(x, y, w, h)
```

Параметры `x` и `y` задают координаты левого верхнего угла прямоугольника, `w` и `h` — его ширину и высоту (рис. 8.3).

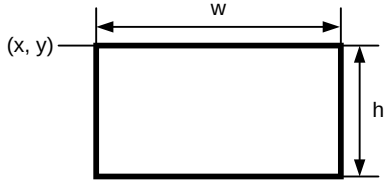


Рис. 8.3. Метод `DrawLine` рисует прямоугольник

Пример;

```
GraphicsWindow.DrawRectangle(10,20,100,20)
```

Цвет и толщину границы прямоугольника, рисуемого методом `DrawRectangle`, определяют, соответственно, значения свойств `PenColor` и `PenWidth`.

Метод `FillRectangle` рисует закрашенный прямоугольник.

Инструкция вызова метода `FillRectangle` в общем виде выглядит так:

```
GraphicsWindow.FillRctangle(x, y, w, h)
```

Параметры `x` и `y` задают координаты левого верхнего угла прямоугольника, `w` и `h` — его ширину и высоту.

Пример:

```
GraphicsWindow.FillRectangle(10,20,100,20)
```

Цвет закраски прямоугольника, рисуемого методом `FillRectangle`, определяет значение свойств `BrushenColor`.

Использование методов `DrawRectangle` и `FillRectangle` демонстрирует программа "Прямоугольники" (листинг 8.2). Она рисует на поверхности формы итальянский флаг (рис. 8.4).

Листинг 8.2. Прямоугольники

```
' Итальянский флаг

' Размер полосы флага
w = 30
h = 50

' Положение левого верхнего угла
x0 = 20
y0 = 20

x = x0
y = y0

' Зеленая полоса
GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillRectangle(x,y,w,h)

' Белая полоса
GraphicsWindow.BrushColor = "White"
x = x + w
GraphicsWindow.FillRectangle(x,y,w,h)

' Красная полоса
GraphicsWindow.BrushColor = "Red"
```

```
x = x + w
GraphicsWindow.FillRectangle(x,y,w,h)

' Контур
GraphicsWindow.PenColor = "Gray"
GraphicsWindow.DrawRectangle(x0,y0,3*w,h)

GraphicsWindow.BrushColor = "Gray"
GraphicsWindow.DrawText(x0, y0+h+10,"Италия")
```

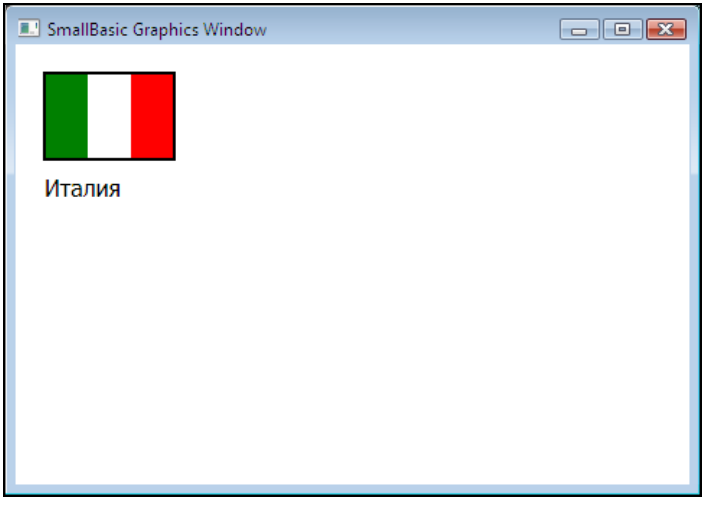


Рис. 8.4. Контур прямоугольника рисует метод `DrawRectangle`, прямоугольник — метод `FillRectangle`

Контрольные вопросы

1. Что рисует метод `DrawRectangle`?
2. Что рисует метод `FillRectangle`?
3. Как задать цвет границы прямоугольника?
4. Как задать цвет закраски прямоугольника?

Эллипс, окружность и круг

Метод `DrawEllipse`, в зависимости от значения параметров, рисует эллипс или окружность.

Инструкция вызова метода `DrawEllipse` в общем виде выглядит так:

```
GraphicsWindow.DrawEllipse(x, y, w, h)
```

Параметры *x* и *y* задают координаты левого верхнего угла, а *w* и *h* — соответственно, ширину и высоту прямоугольника, в которую "вписывается" эллипс (рис. 8.5). Очевидно, что если значения *w* и *h* будут равны, метод `DrawEllipse` нарисует окружность.

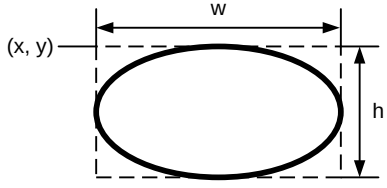


Рис. 8.5. Метод `DrawEllipse` рисует эллипс

Толщину и цвет линии эллипса определяют, соответственно, значения свойств `PenWidth` и `PenColor`.

Закрашенный эллипс (или круг, если значения параметров `w` и `h` равны) рисует метод `FillEllipse`. Цвет закраски внутренней области эллипса определяет свойство `BrushColor`.

Использование метода `DrawEllipse` демонстрирует программа "Олимпиада" (ее текст приведен в листинге 8.3), она рисует на поверхности формы символ Олимпийских игр (рис. 8.6). Следует обратить внимание на то, что в программе явно указаны координаты только первой (левой верхней) окружности, координаты каждой следующей отсчитываются от центра предыдущей.

Листинг 8.3. Олимпиада

```
' Олимпийская эмблема
r = 30 ' радиус кольца
x = 50
y = 40

GraphicsWindow.PenWidth = 3

' Рисуем от левого кольца первого ряда
GraphicsWindow.PenColor = "MediumBlue"
GraphicsWindow.DrawEllipse(x,y,2*r, 2*R)

' Второе кольцо сдвинуто относительно первого вправо на 2.25r
x = x+2.25*r
GraphicsWindow.PenColor = "Black"
GraphicsWindow.DrawEllipse(x,y,2*r, 2*R)

' Третье кольцо сдвинуто относительно второго вправо на 2.25r
x = x+2.25*r
GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(x, y, 2*r, 2*R)

' Второй ряд колец рисуем справа налево.
' Правое кольцо второго ряда сдвинуто вниз и влево
' относительно правого кольца первого ряда.
y = y+1.25*r

x = x-1.25*r
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawEllipse(x,y,2*r, 2*R)

x = x-2.25*r
```

```
GraphicsWindow.PenColor = "ForestGreen"
GraphicsWindow.DrawEllipse(x, y, 2*r, 2*R)

GraphicsWindow.FontSize = 14
GraphicsWindow.FontBold = "False"
GraphicsWindow.DrawText(50,160,"Быстрее, выше, сильнее!")
```

Рис. 8.6. Символ и девиз Олимпийских игр

Контрольные вопросы

1. Как задать цвет эллипса?
2. Как задать толщину границы эллипса?
3. Как при помощи метода `DrawEllipse` нарисовать окружность?

Текст

Вывести текст в графическое окно можно при помощи метода `DrawText`. Инструкция вызова метода `DrawText` в общем виде выглядит так:

```
GraphicsWindow.DrawText(x, y, Текст)
```

Параметры *x* и *y* задают координаты левого верхнего угла области вывода текста, параметр *Текст* — текст, который надо вывести.

Пример:

```
GraphicsWindow.DrawText(50,20,"Microsoft Small Basic")
```

Шрифт, который метод `DrawText` использует для отображения текста, определяет свойство `FontName`, размер символов — свойство `FontSize`, цвет символов — свойство `BrushColor`. По умолчанию текст выводится полужирным шрифтом. Если надо, чтобы использовался обычный шрифт, свойству `FontBold` надо присвоить значение `False`. Чтобы текст был выведен курсивом, свойству `FontItalic` следует присвоить значение `True`.

Приведенная в листинге 8.4 программа демонстрирует вывод текста в графическое окно (рис. 8.7).

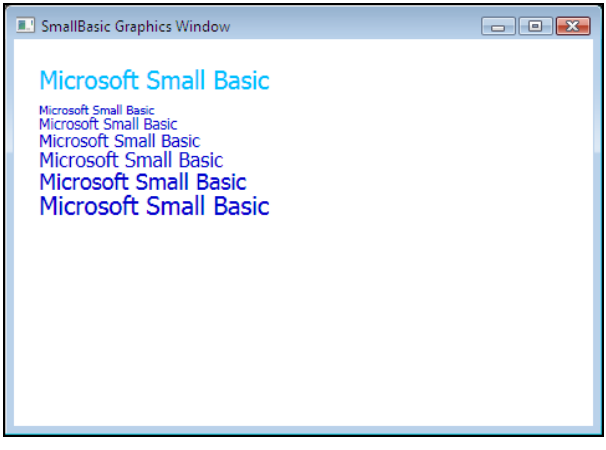


Рис. 8.7. Вывод текста в графическое окно выполняет метод `DrawText`

Листинг 8.4. Вывод текста

```
GraphicsWindow.FontName = "Tahoma"
GraphicsWindow.FontSize = 20    ' размер символов в пикселах
GraphicsWindow.FontBold = "False"
GraphicsWindow.BrushColor = "DeepSkyBlue"
GraphicsWindow.DrawText(20,20,"Microsoft Small Basic")

x = 20
y = 50
GraphicsWindow.BrushColor = "MediumBlue"
For s =10 To 20 Step 2
  GraphicsWindow.FontSize = s
  GraphicsWindow.DrawText(x,y,"Microsoft Small Basic")
  y=y+s
EndFor
```

Контрольные вопросы

1. Перечислите свойства, которые задают характеристики шрифта, используемого методом `DrawText` для вывода текста в графическое окно.
2. В каких единицах задается размер шрифта?
3. Как вывести текст курсивом?
4. Как задать цвет символов текста, выводимого методом `DrawText`?

Иллюстрации

Отображение иллюстрации в графическом окне обеспечивают методы `DrawImage` и `DrawResizedImage`. Они позволяют вывести в графическое окно иллюстрацию формата JPG, GIF, PNG, BMP, ICO. Различие методов состоит в том, что `DrawImage` выводит иллюстрацию "как есть", а `DrawResizedImage` позволяет выполнить ее масштабирование.

Инструкция вызова метода `DrawImage` в общем случае выглядит так:

```
GraphicsWindow.DrawImage(img, x, y)
```

Параметр *img*, в качестве которого обычно указывается полное имя файла, задает иллюстрацию, параметры *x* и *y* — координаты правого верхнего угла области отображения.

Пример:

```
GraphicsWindow.DrawImage("C:\users\Никита\Pictures\
p1.jpg", 10, 10)
```

Если файл иллюстрации находится в той же папке, что и exe-файл программы, то получить имя папки, в которой находится программа и, следовательно, файл иллюстрации, можно, обратившись к свойству `Program.Directory`.

Пример:

```
f = Program.Directory + "\p1.jpg"
GraphicsWindow.DrawImage(f, 10, 10)
```

Если размер иллюстрации больше размера окна программы (или области окна, которая предназначена для отображения иллюстрации), то для того чтобы иллюстрация была видна полностью, следует использовать метод `DrawResizedImage`, инструкция вызова которого в общем случае выглядит так:

```
GraphicsWindow.DrawResizedImage(img, x, y, w, h)
```

Параметр `img` задает иллюстрацию, параметры `x` и `y` — координаты правого верхнего угла области ее отображения, а параметры `w` и `h` — размер области отображения. Очевидно, чтобы иллюстрация отображалась без искажения, размеры области отображения должны быть пропорциональны размеру иллюстрации. Например, чтобы фотография размером 1980×1080 была выведена в окно программы без искажения, размер области должен быть, например, 990×540 (коэффициент масштабирования 0.5) или 495×270 (коэффициент масштабирования 0.25).

В листинге 8.5 приведена программа (рис. 8.8), которая демонстрирует использование метода `DrawResizedImage`.

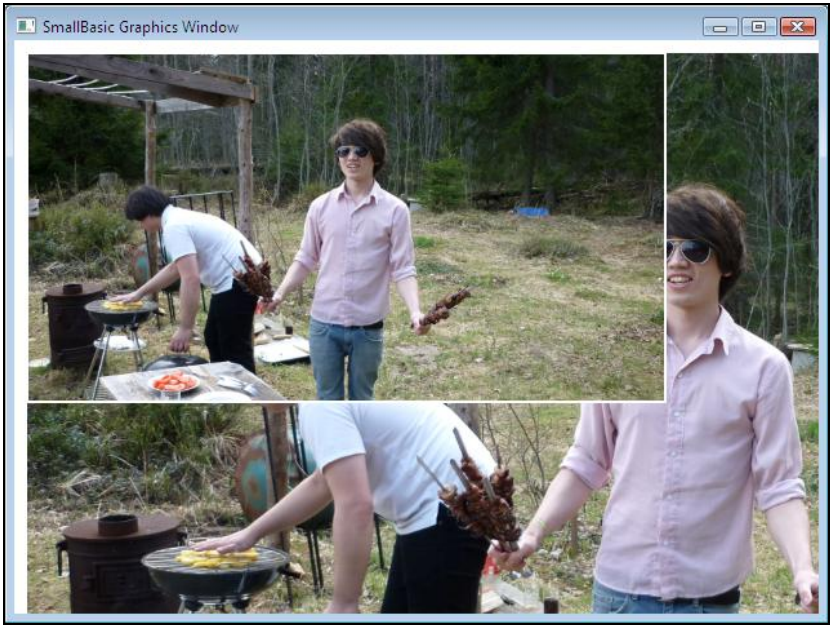


Рис. 8.8. Масштабирование иллюстрации

Листинг 8.5. Вывод иллюстрации методом `DrawResizedImage`

```
' Масштабирование иллюстрации.
' Предполагается, что файл иллюстрации находится
' в той же папке, что и exe-файл программы

img = Program.Directory + "\p1.jpg"
' Размер иллюстрации 1980 x 1080

' Масштаб 0.5
GraphicsWindow.DrawResizedImage(img,10,10,990,540)

' Масштаб 0.25
GraphicsWindow.DrawResizedImage(img,10,10,495,270)

' Рамка для наглядности
GraphicsWindow.PenColor = "White"
GraphicsWindow.DrawRectangle(10,10,495,270)
```

Задачу масштабирования иллюстрации можно возложить на программу (листинг 8.6).

Листинг 8.6. Масштабирование иллюстрации

```
' Отображение иллюстрации без искажения

' Предполагается, что файл p1.jpg находится
' в каталоге программы
f = Program.Directory + "\p1.jpg"

' Загрузить картинку
img = ImageList.LoadImage(f)

' Определим размер загруженной картинки
```

```
w = ImageList.GetWidthOfImage(img)
h = ImageList.GetHeightOfImage(img)

' Определим размер области графического окна,
' предназначенной для отображение картинки.
wgw = GraphicsWindow.Width - 20
hgw = GraphicsWindow.Height -20

If (w <= wgw) And (h <= hgw) Then
  ' Масштабировать не надо
  GraphicsWindow.DrawImage(f,10,10)
Else
  ' Необходимо масштабировать
  kx = wgw / w    ' коэф. масштабирования по X
  ky = hgw / h    ' коэф. масштабирования по Y

  ' Чтобы не было искажения, коэффициенты масштабирования
  ' по X и Y должны быть равны
  k = Math.Min(kx,ky)

  ' Вычислим размер области, обеспечивающий отображение
  ' картинки без искажения
  iw = w * k
  ih = h * k

  ' Вывести картинку
  GraphicsWindow.DrawResizedImage(img,10,10,iw,ih)
EndIf

' Показать информацию об иллюстрации
GraphicsWindow.BrushColor = "White"

GraphicsWindow.DrawText(15,15,f)
st =  "Image size:" + w +"x" + h
GraphicsWindow.DrawText(15,27,st)
```

В начале работы программа загружает картинку в память. Делает это метод `LoadImage`, которому в качестве параметра передается имя файла иллюстрации. Затем программа определяет размер загруженной иллюстрации (функции `GetWidthOfImage` и `GetHeightOfImage` возвращают, соответственно, ширину и размер картинки, находящейся в переменной `img`). Далее вычисляется размер области, предназначенной для отображения иллюстрации. Ее ширина и высота на 20 пикселов меньше, соответственно, высоты и ширины графического окна (10 пикселов сверху, снизу и справа это — поля). Затем размер картинки сравнивается с размером области отображения. Если картинка целиком помещается в область отображения, то она выводится методом `DrawImage`. Если это не так, то вычисляются коэффициенты масштабирования по ширине и высоте. Чтобы картинка отображалась без искажения, коэффициенты масштабирования по x и y должны быть одинаковыми. Поэтому в качестве коэффициента масштабирования принимается минимальный из коэффициентов. Для выявления минимального значения в программе использована функция `Min`. Далее вычисляется размер области отображения иллюстрации и метод `DrawResizedImage` выводит иллюстрацию. Окно программы приведено на рис. 8.9.

Метод `DrawImage` можно использовать для формирования сложных изображений путем наложения картинок. В качестве примера в листинге 8.7 приведена программа, которая в графическом окне формирует изображение летящего в облаках самолета (рис. 8.10). Сначала в окно выводится изображение неба, затем самолета (рис. 8.11). Следует обратить внимание, картинка самолета должна быть с "прозрачным" фоном.

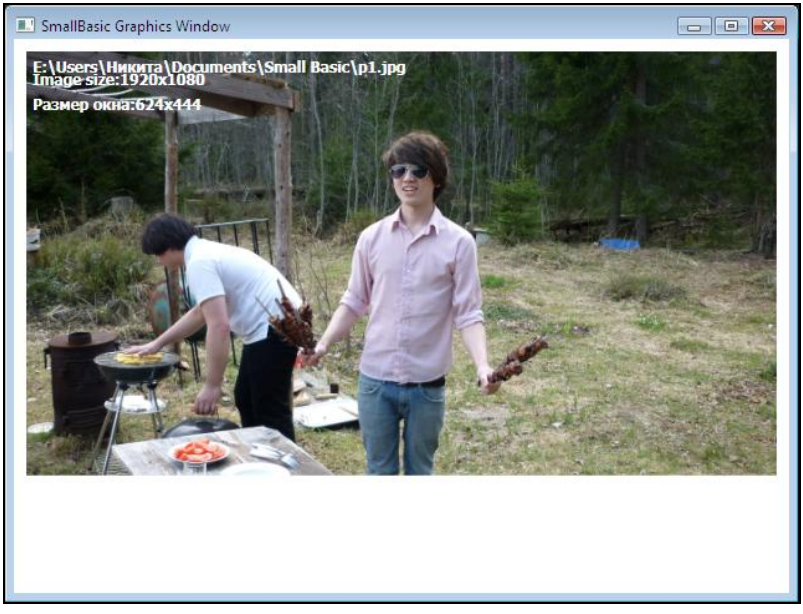


Рис. 8.9. Результат масштабирования картинки

Рис. 8.10. Изображение, сформированное путем наложения одной картинки на другую

Рис. 8.11. Элементы, из которых сформирована картинка

Листинг 8.7. Формирование изображения путем наложения картинок

```
' Установить размер окна в соответствии
' с размером фоновой картинки
GraphicsWindow.Width = 445
GraphicsWindow.Height = 200

' Сделать границу окна тонкой
GraphicsWindow.CanResize = "False"

' Загрузить картинки
sky = Program.Directory + "\sky.png"
plane = Program.Directory + "\plane.png"

' Вывести фон
GraphicsWindow.DrawImage(sky,0,0)
' Вывести самолет
GraphicsWindow.DrawImage(plane,300,100)
```

Контрольные вопросы

1. Перечислите методы отображения иллюстраций.
2. Какой метод следует использовать, если размер иллюстрации больше размера графического окна или области, предназначенной для отображения иллюстрации?
3. Как узнать истинный размер иллюстрации?

Анимация

Анимация (дословно оживление) — это технология создания "живых" изображений. Анимированное изображение или анимация, в отличие от обычной, статичной, картинки, представляет собой изображение, элементы которого ведут себя подобно объектам реального мира.

Существуют два подхода к реализации анимации. Первый предполагает наличие заранее подготовленной серии картинок (кадров), на которых изображены фазы движения объекта. Последовательное отображение кадров создает эффект анимации. Этот подход используют создатели мультфильмов. Второй подход, который используют разработчики компьютерных игр, предполагает создание кадров анимации "на лету", во время работы программы. При реализации этого подхода изображение фаз движения объектов формируется из графических примитивов непосредственно на "экране", в качестве которого обычно используется фоновый рисунок.

Движение — один из основных анимационных эффектов. Реализовать его достаточно просто: сначала нужно нарисовать объект (вывести изображение объекта), затем через некоторое время стереть и снова вывести, но уже на некотором расстоянии от его первоначального положения. Подбором времени между выводом и удалением изображения, а также расстояния между новым и предыдущим положением объекта, можно добиться того, что у наблюдателя (зрителя) будет складываться впечатление, что объект равномерно движется.

Следующая программа, ее текст приведен в листинге 8.8, показывает, как можно реализовать движение объекта в графическом окне. Объект (окружность) движется от левой границы окна к правой. Окружность стирает и рисует процедура обработки сигнала от *таймера*. Таймер — это устройство, которое с задан-

ным периодом генерирует сигнал. В программе таймер — это объект, который запускает процедуру, связанную с таймером. Настройку таймера выполняют инструкции

```
Timer.Tick = onTimer
Timer.Interval = 50
```

Первая инструкция задает процедуру, которую следует запускать при появлении сигнала от таймера, вторая — период сигналов от таймера. Период задается в миллисекундах. Процедура `onTimer` стирает объект, вычисляет его новое положение и рисует его на новом месте.

Листинг 8.8. Движение простого объекта

```
' Движение окружности от левой границы окна к правой

d =10 ' размер (диаметр) объекта

' Начальное положение объекта
x = -d
y = 50

' Приращения координаты
dx = 2

GraphicsWindow.BrushColor = "White"
GraphicsWindow.FillRectangle(0,0,GraphicsWindow.Width,
GraphicsWindow.Height)

GraphicsWindow.PenWidth = 1

' Настройка таймера
Timer.Interval = 50  ' период сигнала от таймера
Timer.Tick = onTimer ' при появлении сигнала от таймера
```

```
' Запускать процедуру onTimer

' Процедура обрабатывает сигнал от таймера
Sub onTimer
  ' Стереть объект.
  ' Стираем объект путем его закраски цветом фона
  GraphicsWindow.FillRectangle(x-1,y-1,d+2,d+2)

  ' Вычислить новое положение объекта
  x = x + dx
  If (x > GraphicsWindow.Width) Then
    x= -d
  EndIf

  ' Нарисовать объект на новом месте
  GraphicsWindow.PenColor = "ForestGreen"
  GraphicsWindow.DrawEllipse(x,y,d,d)
EndSub
```

При программировании движения объектов, состоящих из множества элементов, инструкции, рисующие объект, рекомендуется оформить как процедуру. В качестве примера в листинге 8.9 приведена программа, в окне которой летит "тарелка" — неопознанный летающий объект", UFO. Тарелку рисует процедура UFO, которую вызывает процедура обработки сигнала от таймера. При рисовании объекта используется метод "базовой точки". Суть его заключается в следующем. Выбирается некоторая точка объекта, которая принимается за базовую. Координаты остальных точек отсчитываются от базовой точки. На рис. 8.12 приведено изображение UFO. Базовой точкой является точка (x_0, y_0). Координаты остальных точек отсчитываются именно от нее. Окно программы UFO приведено на рис. 8.13.

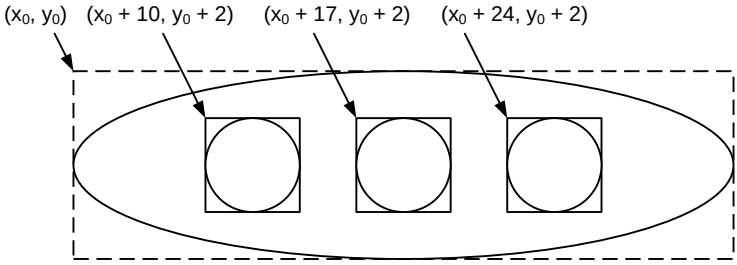


Рис. 8.12. Пример определения координат изображения относительно базовой точки

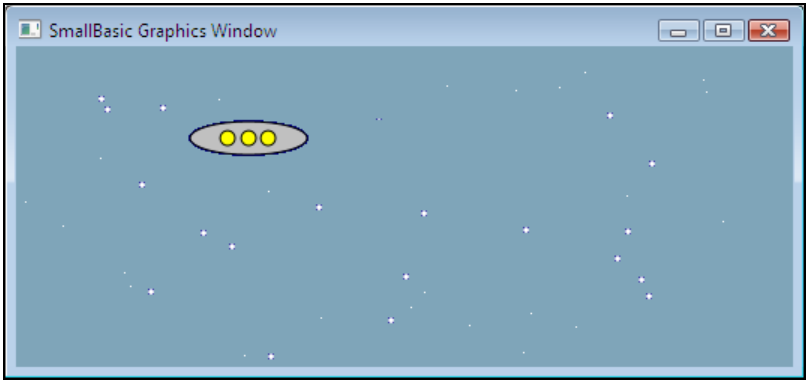


Рис. 8.13. Окно программы UFO

Листинг 8.9. Движение объекта (полет UFO)

```
' Движение UFO - "неопознанного летающего объекта"

' Размер объекта
w = 39
h = 11

' Начальное положение объекта
x = -w
y = 50

' Приращения координаты X
dx = 2

bColor = "DarkBlue" ' цвет фона
GraphicsWindow.BrushColor = bColor
GraphicsWindow.FillRectangle(0,0,GraphicsWindow.Width,
GraphicsWindow.Height)

' Разбросать звезды
GraphicsWindow.PenColor = "White"
For i=1 To 100
  sx = Math.GetRandomNumber(GraphicsWindow.Width)
  sy = Math.GetRandomNumber(GraphicsWindow.Height)

  ' Диаметр звезды: 1 или 2
  d = Math.GetRandomNumber(2)
  If ( d = 2) Then
    GraphicsWindow.DrawEllipse(sx,sy,2,2)
  Else
    GraphicsWindow.SetPixel(sx,sy,"White")
  EndIf
```

```
EndFor

GraphicsWindow.PenWidth = 1

' Настройка таймера
Timer.Interval = 50  ' период сигнала от таймера
Timer.Tick = onTimer ' при появлении сигнала от таймера

' Запускать процедуру onTimer

' Процедура обрабатывает сигнал от таймера
Sub onTimer
  ' Стираем объект путем его закраски цветом фона
  GraphicsWindow.BrushColor = bColor
  GraphicsWindow.FillRectangle(x-1,y-1,w+2,h+2)

  ' Вычислить новое положение объекта
  x = x + dx
  If (x > GraphicsWindow.Width) Then
    x= -w
  EndIf

  ' Нарисовать объект на новом месте
  UFO() ' вызов процедуры UFO, которая рисует объект
EndSub

' Рисует UFO
Sub UFO
  ' Корпус
  GraphicsWindow.BrushColor = "Silver"
  GraphicsWindow.PenColor = "Black"
```

```
  GraphicsWindow.FillEllipse(x,y,w,h)
  GraphicsWindow.DrawEllipse(x,y,w,h)
  ' Иллюминаторы
  GraphicsWindow.BrushColor = "Yellow"
  GraphicsWindow.FillEllipse(x+10,y+3,5,5)
  GraphicsWindow.DrawEllipse(x+10,y+3,5,5)
  GraphicsWindow.FillEllipse(x+17,y+3,5,5)
  GraphicsWindow.DrawEllipse(x+17,y+3,5,5)
  GraphicsWindow.FillEllipse(x+24,y+3,5,5)
  GraphicsWindow.DrawEllipse(x+24,y+3,5,5)
EndSub
```

"Черепашья" графика

Идея использования "черепашки", способной двигаться по экрану в соответствии с реализованным в виде последовательности команд алгоритмом, для обучения программированию была сформулирована и реализована в языке Лого (Logo) профессором математики и педагогики Массачусетского технологического университета Сеймуром Пейпертом. Наличие черепашки и возможность управлять ею позволили сделать процесс обучения основам программирования занимательным, а сам язык доступным даже детям дошкольного возраста: составляя простые "программы", они могли заставить черепашку двигаться по экрану, рисовать геометрические фигуры и простые картинки.

В Small Basic тоже есть черепашка (*turtle* — черепаха). Она может двигаться по экрану, не оставляя следа или рисуя линию. Команды управления черепашкой приведены в табл. 8.4. Цвет и толщину линии, рисуемой черепашкой, определяют, соответственно, свойства `GraphicsWindow.PenColor` и `GraphicsWindow.PenWidth`.

В качестве примера в листинге 8.10 приведена программа, демонстрирующая черепашью графику — черепашка рисует домик, забор и елочку (рис. 8.14).

Таблица 8.4. *Команды управления черепашкой*

Команда	Действие
`Turtle.Show()`	Показать, сделать видимой, черепашку
`Turtle.Move(d)`	Перемещение черепашки вперед на заданное параметром `d` расстояние
`Turtle.MoveTo(x,y)`	Перемещение черепашки в точку с указанными координатами
`Turtle.TurnLeft()`	Поворот налево на 90°
`Turtle.TurnRight()`	Поворот направо на 90°
`Turtle.Turn(α)`	Поворот налево (против часовой стрелки) на заданный параметром `α` угол
`Turtle.PenDown()`	Опустить карандаш. После выполнения команды при движении черепашка оставляет след. Цвет следа определяет значение свойства `PenColor`
`Turtle.PenUp()`	Поднять карандаш. После выполнения команды при движении черепашка след не оставляет
`Turtle.Hide()`	Скрыть, сделать невидимой, черепашку
`Turtle.Angle = α`	Задает направление движения — угол "оси" черепашки относительно оси *Oy*. Например: 0 — вверх по вертикали; 90 — вправо по горизонтали; –90 — влево по горизонтали; 180 — вниз
`Turtle.Speed = s`	Задает скорость движения черепашки. Значение `s` должно быть в диапазоне от 1 до 10

Листинг 8.10. "Черепашья" графика

```
Turtle.Speed = 8

' Забор
Turtle.PenUp()
Turtle.MoveTo(30,200)
Turtle.Angle = 0 ' голову вверх
GraphicsWindow.PenColor = "DarkGreen"
Turtle.PenDown()
For i=1 To 5      ' 5 "дощечек"
  Turtle.Move(40)
  Turtle.TurnRight()
  Turtle.Move(10)
  Turtle.TurnRight()
  Turtle.Move(40)
  Turtle.TurnLeft()
  Turtle.Move(10)
  Turtle.TurnLeft()
EndFor

' Домик
Turtle.PenUp()
' стену рисуем от левого верхнего угла
Turtle.MoveTo(130,120)
Turtle.Angle = 0     ' голову вверх
GraphicsWindow.PenColor = "DarkRed"
Turtle.PenDown()

For i=1 To 4
  Turtle.TurnRight()
```

```
  Turtle.Move(80)
EndFor

' Крыша — равносторонний треугольник
Turtle.Angle = 30
For i=1 To 2
  Turtle.Move(80)
  Turtle.Turn(120)
EndFor

' Окно
Turtle.PenUp()
Turtle.MoveTo(150,140)

Turtle.Angle = 90   ' голову вправо
GraphicsWindow.PenColor = "DarkRed"
Turtle.PenDown()
For i=1 To 4
  Turtle.Move(40)
  Turtle.TurnRight()
EndFor

' Елка
x=250
y=110
Turtle.PenUp()
For j=1 To 3
  Turtle.PenUp()
  Turtle.MoveTo(x,y)
  Turtle.Angle = 30
```

```
  GraphicsWindow.PenColor = "ForestGreen"
  Turtle.PenDown()
  For i=1 To 3
    Turtle.Move(50)
    Turtle.Turn(120)
  EndFor
  y=y+40
EndFor

Turtle.Hide()
```

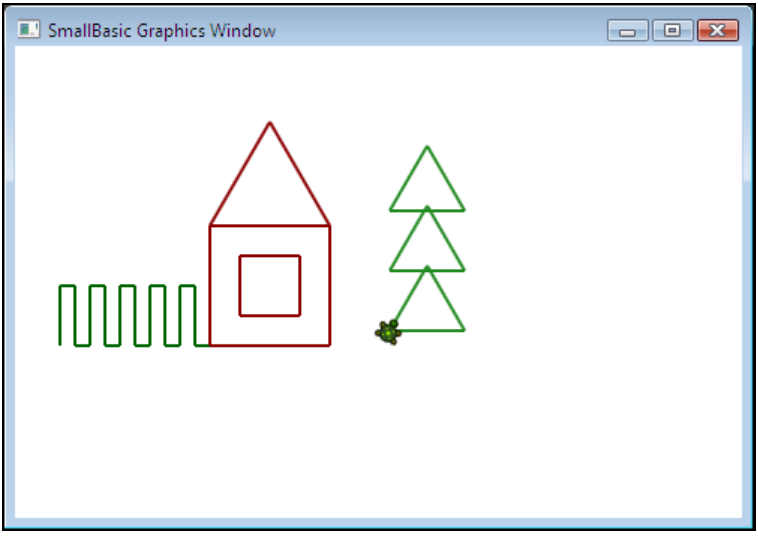
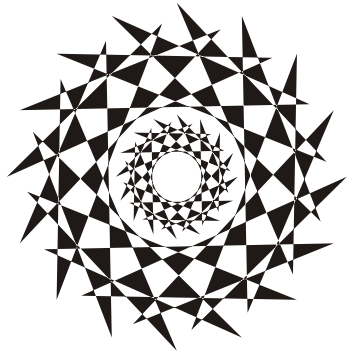


Рис. 8.14. Черепашья графика



Глава 9

Файлы

До этого момента наши программы вводили исходные данные с клавиатуры и выводили результат своей работы на экран, что не всегда удобно. В этой главе рассказывается, как вывести данные в файл, а также о том, как прочитать данные из файла.

Файл — это структура данных, представляющая собой последовательность элементов одного типа. Количество элементов файла практически не ограничено.

Различают текстовые и двоичные файлы. Текстовый файл — это последовательность символов, его можно просмотреть, загрузив в редактор текста. Например, текст, набранный в редакторе Блокнот и сохраненный на диске, представляет собой текстовый файл. Файл, содержащий программу Small Basic, тоже текстовый. Увидеть, что находится в двоичном файле, можно только при помощи программы, предназначенной для просмотра файлов соответствующего типа. Например, чтобы увидеть, что находится в png-файле, его надо загрузить в графический редактор, например Paint.

Замечание

Расширение текстового файла не обязательно должно быть txt. Так, например, в момент сохранения текстового файла в Блокноте, автоматически добавляемое к имени файла расширение txt

можно заменить, скажем, на dat. Расширение требуется операционной системе для сопоставления содержимого файла с программой, которую надо использовать "по умолчанию" для работы с файлами этого типа, запуска соответствующей программы.

Создать файл данных можно программно (записать данные в файл) или "вручную", в Блокноте или в Small Basic (рис. 9.1). В последнем случае в момент сохранения файла расширение sb следует заменить на txt.

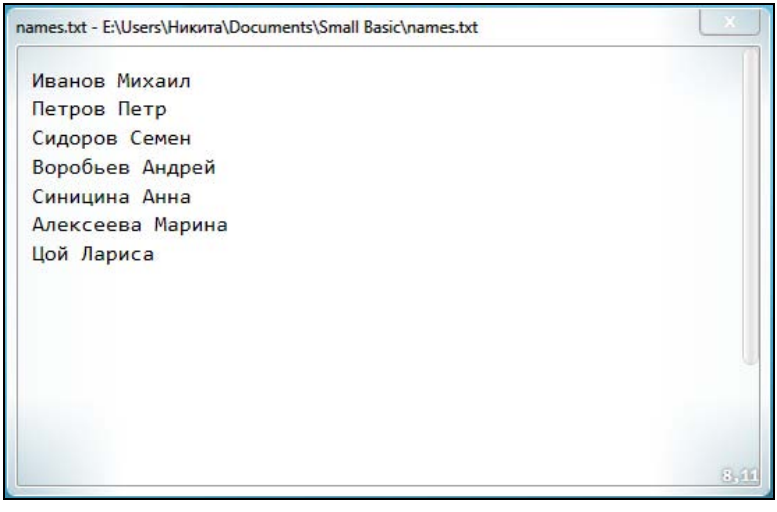


Рис. 9.1. Создание файла данных в редакторе Small Basic

Чтение данных из файла

Операция получения данных из файла называется чтением. Данные из файла обычно читают по строкам.

Инструкция чтения строки из файла в общем виде выглядит так:

```
a = File.ReadLile(f, n)
```

где *f* — имя файла, из которого читаются данные; *n* — номер строки, которую надо прочитать; a — переменная, в которую считываются данные.

Пример:

```
st = File.ReadLine("C:\Users\Никита\Documents\
Small Basic\data.txt",5)
```

Если файл данных находится в том же каталоге, что и файл программы, то получить имя каталога программы можно, обратившись к методу `Program.Directory`.

Пример:

```
f =  Program.Directory. + "\data.txt"
st = File.ReadLine(f,5)
```

Следует обратить внимание, если строки с указанным номером в файле нет, то инструкция `File.ReadLine` возвращает пустую строку. Пустая строка будет возвращена также и в том случае, если в инструкции чтения из файла указан несуществующий файл.

Приведенная в листинге 9.1 программа демонстрирует процесс чтения данных из файла. Она считывает из файла строки и выводит их на экран. Программа считывает все строки. Факт достижения конца файла определяется путем проверки значения переменной, в которую считываются данные. Если значением переменной является пустая строка, то это значит, что достигнут конец файла. Переменной `eof` присваивается значение `True`, и цикл `While` завершается. Окно программы приведено на рис. 9.2. Предполагается, что файл names.txt существует и находится в том же каталоге, что и программа.

Листинг 9.1. Чтение строк из файла

```
' Чтение данных из файла
f = Program.Directory + "\names.txt"

i=1
eof = "False"   ' чтобы войти в цикл
While (eof <> "True")
  name = File.ReadLine(f,i)
  If (name <> "") Then
    ' Строка прочитана
    TextWindow.Write(i + ". ")
    TextWindow.WriteLine(name)
    i = i + 1
  Else
    ' Строк больше нет
    eof = "True"
  EndIf
EndWhile
```

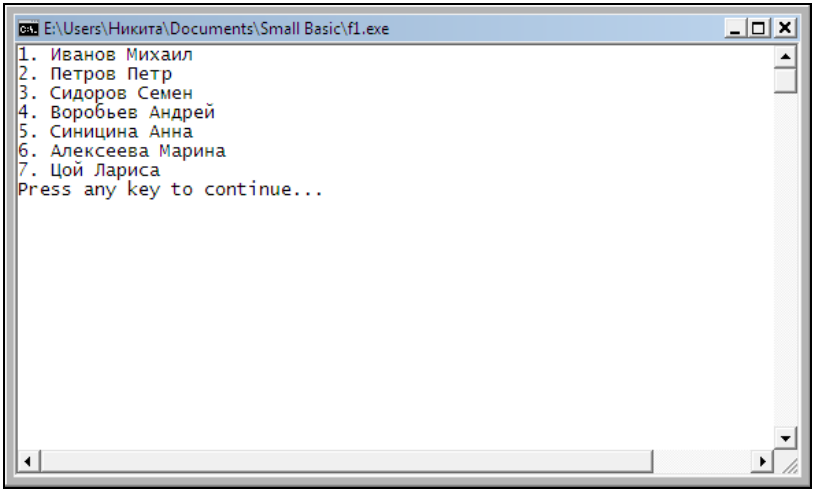


Рис. 9.2. Чтение данных из файла

Файл может содержать не только символьную, но и числовую информацию (рис. 9.3). Здесь следует обратить внимание на то, что если в строке графического редактора отображается число (последовательность цифр), то формально это не число, а строка символов. В момент чтения строки из файла, если строка является изображением числа, она преобразуется в соответствующее число. При записи "вручную" в файл дробных чисел в качестве десятичного разделителя следует использовать *точку*.

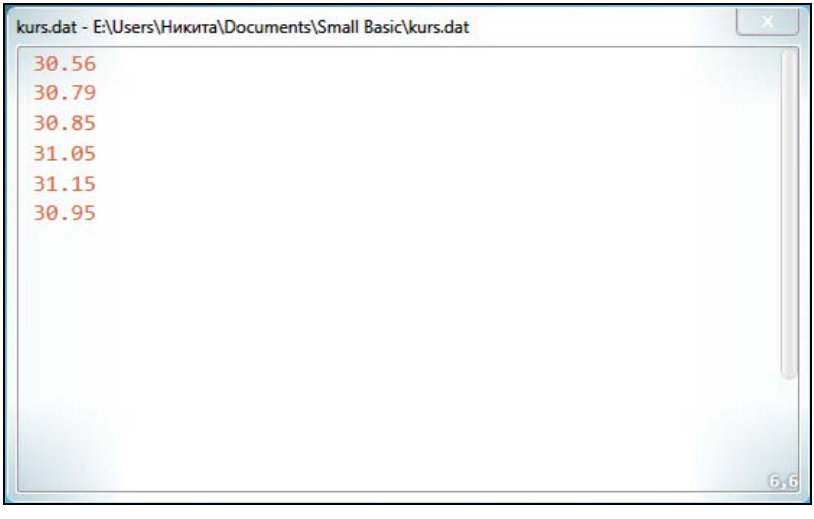


Рис. 9.3. Текстовый файл, содержащий числовую информацию

Приведенная в листинге 9.2 программа демонстрирует чтение числовой информации. Она читает числа из файла в массив и затем вычисляет среднее арифметическое элементов массива (рис. 9.4).

Листинг 9.2. Чтение числовых данных из файла

```
' Чтение числовых данных из файла
f = Program.Directory + "\kurs.dat"
```

```
i=1
eof = "False"    ' чтобы войти в цикл
While (eof <> "True")
  st = File.ReadLine(f,i)

  If (st <> "") Then
    ' Строка прочитана

    a[i] = st
    TextWindow.WriteLine(a[i])
    i=i+1
  Else
    ' Строк больше нет
    eof = "True"
  EndIf
EndWhile

' Здесь i на единицу больше, чем кол-во прочитанных чисел
n = i - 1    ' кол-во прочитанных строк

' Вычислим сумму элементов массива
s = 0
For i = 1 To n
  s = s + a[i]
EndFor

m = s / n
TextWindow.WriteLine("Среднее арифметическое:" +  m)
```

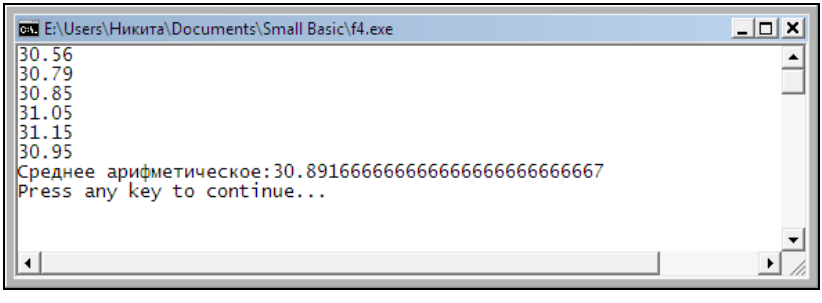


Рис. 9.4. Чтение чисел из файла

Запись данных в файл

Запись данных в файл возможна в режиме добавления или перезаписи. В режиме добавления, данные добавляются в конец файла. В режиме перезаписи новые данные записываются поверх существующих (если файла нет, то он создается).

Приведенная в листинге 9.3 программа (рис. 9.5) демонстрирует добавление информации в файл. Непосредственно запись информации в файл выполняет инструкция `File.AppendContents`.

Листинг 9.3. Добавление данных в файл

```
' Запись данных в файл в режиме добавления

TextWindow.WriteLine("Добавление данных в файл")
TextWindow.WriteLine("Для завершения ввода нажмите <Enter")

f = Program.Directory + "\names.txt"

i = 0        ' кол-во строк, записанных в файл
name = "*"   ' чтобы войти в цикл

While (name <> "")
  TextWindow.Write("->")
  name = TextWindow.Read()
  If (name <> "") Then
    ' Пользователь ввел строку.
    ' Запишем ее в файл.
    i = i+1
    File.AppendContents(f,name)
  EndIf
EndWhile
```

```
TextWindow.WriteLine("Добавлено строк:" + i)

' Показать содержимое файла
i=1
eof = "False"    ' чтобы войти в цикл
While (eof <> "True")
  name = File.ReadLine(f,i)
  If (name <> "") Then
    TextWindow.Write(i + ". ")
    TextWindow.WriteLine(name)
    i = i + 1
  Else
    eof = "True"
  EndIf
EndWhile
```

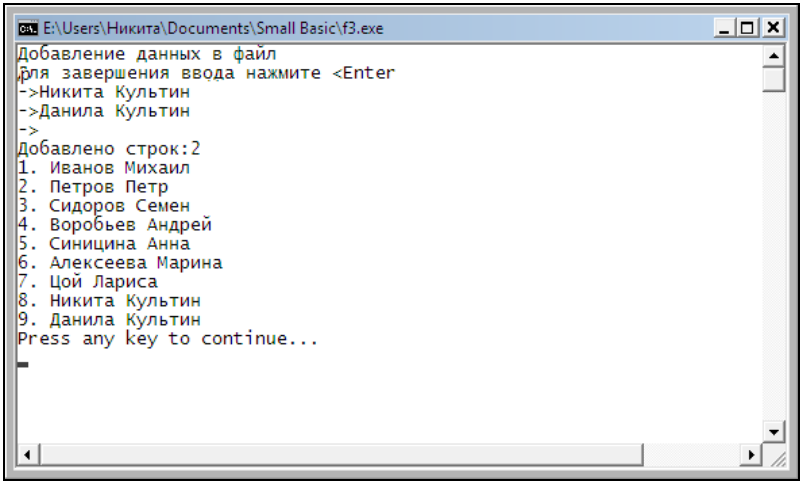


Рис. 9.5. Добавление строк в файл

Следующая программа, ее текст приведен в листинге 9.4, демонстрирует процесс замены информации в файле. Программа позволяет заменить (перезаписать) строку с указанным номером. Непосредственно перезапись строки выполняет инструкция `File.WriteLine(f, n, st)`, параметр `f` которой задает файл, `n` — номер заменяемой строки, а параметр `st` — новое содержимое строки. Следует обратить внимание, если указанный пользователем во время работы программы номер заменяемой строки будет больше количества строк, находящихся в файле, то строка будет добавлена в конец файла. Пример работы программы приведен на рис. 9.6.

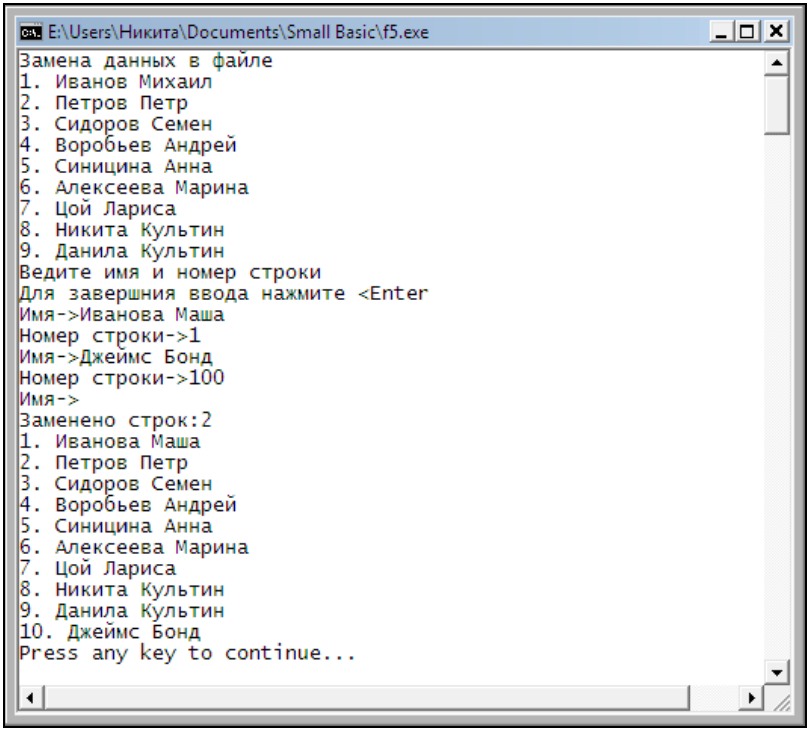


Рис. 9.6. Замена информации в файле

Листинг 9.4. Замена информации в файле

```
' Запись данных в файл в режиме замены

TextWindow.WriteLine("Замена данных в файле")

f = Program.Directory + "\names.txt"

' Показать содержимое файла
i=1
eof = "False"    ' чтобы войти в цикл
While (eof <> "True")
  name = File.ReadLine(f,i)
  If (name <> "") Then
    TextWindow.Write(i + ". ")
    TextWindow.WriteLine(name)
    i = i + 1
  Else
    eof = "True"
  EndIf
EndWhile

TextWindow.WriteLine("Ведите имя и номер строки")
TextWindow.WriteLine("Для завершения ввода нажмите <Enter")

k = 0        ' кол-во замененных строк

name = "*"   ' чтобы войти в цикл
While (name <> "")
  TextWindow.Write("Имя->")
  name = TextWindow.Read()
  If (name <> "") Then
    ' Пользователь ввел имя
```

```
    TextWindow.Write("Номер строки->")
    n = TextWindow.Read()

    ' Заменить строку в файле
    File.WriteLine(f,n,name)
    k = k +1
  EndIf
EndWhile

TextWindow.WriteLine("Заменено строк:" + k)

' Показать содержимое файла
i=1
eof = "False"     ' чтобы войти в цикл
While (eof <> "True")
  name = File.ReadLine(f,i)
  If (name <> "") Then
    TextWindow.Write(i + ". ")
    TextWindow.WriteLine(name)
    i = i + 1
  Else
    eof = "True"
  EndIf
EndWhile
```

Ошибки при работе с файлами

Типичной ошибкой при работе с файлами является обращение к несуществующему файлу. В Small Basic обращение к несуществующему файлу не является критической ошибкой, т. е. не приводит к аварийному завершению программы. При обращении к находящемуся в рабочей папке (здесь под рабочей понимается папка, в которой находится программа) несуществующему файлу для чтения данных инструкция `ReadLine` "ничего не возвращает" (возвращает пустую строку). Инструкция `WriteLine` при попытке записи в несуществующий файл автоматически создает файл с указанным именем. Вместе с тем возможна ситуация, когда обращение к файлу может привести к возникновению *исключения* (критической ошибки). Например, если программа должна читать данные с "флэшки", а флэш-накопителя USB в момент запуска программы в USB-разъеме нет, то при попытке чтения данных возникает исключение и на экране появляется окно с сообщением о прекращении работы программы (рис. 9.7). Для завершения работы программы, в которой возникла ошибка, в окне сообщения следует сделать щелчок на кнопке **Закрыть программу**.

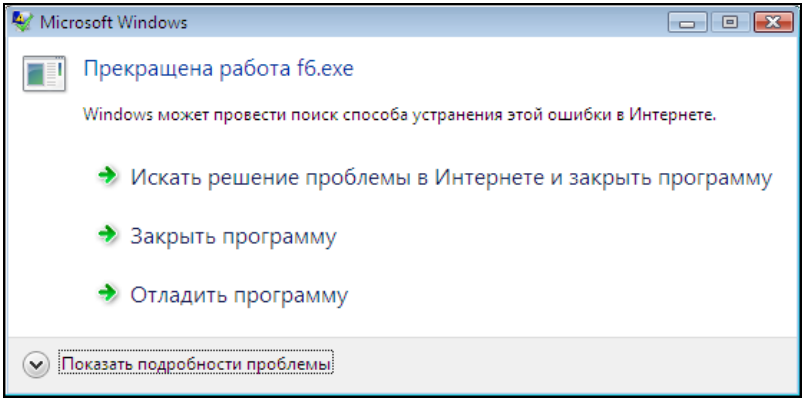
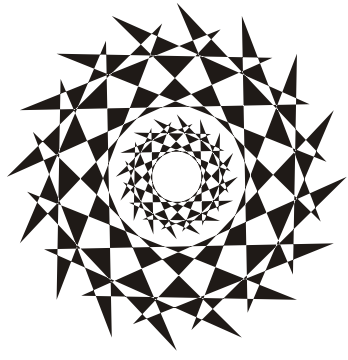


Рис. 9.7. Сообщение об ошибке при обращении к файлу, находящемуся на недоступном диске



Глава 10

Примеры программ

Экзаменатор

Тестирование широко применяется для оценки уровня знаний в учебных заведениях, при приеме на работу, для оценки квалификации персонала. Испытуемому предлагается *тест* — последовательность вопросов, на которые он должен ответить. Обычно к каждому вопросу дается несколько вариантов ответа, из которых надо выбрать правильный. Каждому варианту ответа соответствует некоторая оценка. Суммированием оценок за ответы получается общий балл, на основе которого делается вывод об уровне подготовленности испытуемого (выставляется оценка).

Рассмотрим программу "Экзаменатор" (рис. 10.1), которая позволяет автоматизировать процесс тестирования.

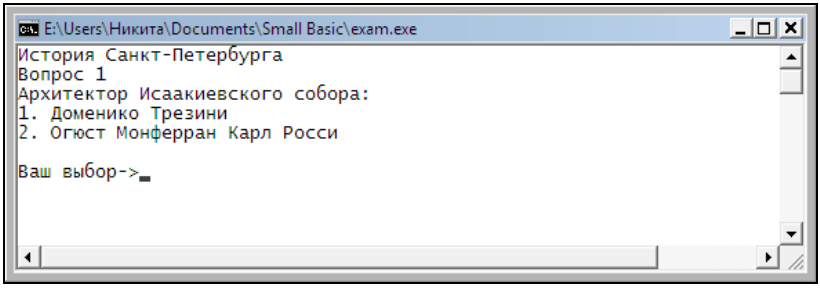


Рис. 10.1. Программа "Экзаменатор"

Требования к программе

В результате анализа различных тестов можно сформулировать следующие требования к программе тестирования:

- программа должна обеспечить работу с тестом произвольной длины (не должно быть ограничений на количество вопросов в тесте);
- каждому вопросу может соответствовать до четырех возможных вариантов ответа, только один из которых является правильным;
- результат тестирования должен быть отнесен к одному из четырех уровней. Например, "отлично", "хорошо", "удовлетворительно" или "плохо";
- тест должен представлять собой текстовый файл;
- программа не должна обеспечивать возврат к предыдущему вопросу. Если вопрос предложен, то на него должен быть получен ответ.

Алгоритм программы

Укрупненный алгоритм работы программы "Экзаменатор" приведен на рис. 10.2.

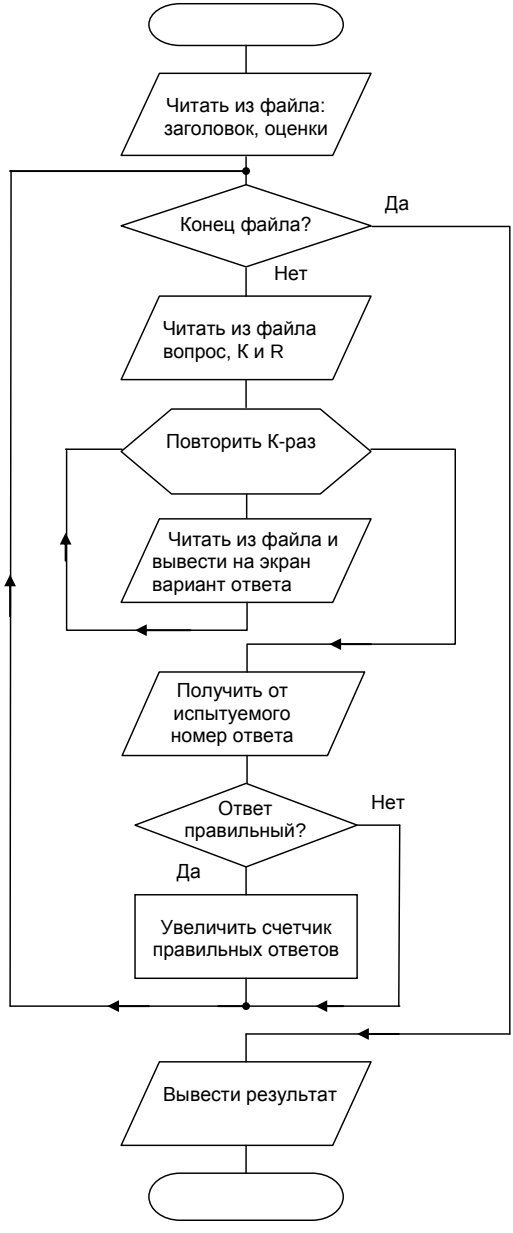


Рис. 10.2. Алгоритм программы "Экзаменатор"

Структура файла теста

Реализация алгоритма программы в значительной мере зависит от представления обрабатываемых программой данных. Для рассматриваемой программы файл теста имеет следующую структуру:

Заголовок (название) теста

Сообщение для уровня 1

Сумма баллов для достижения уровня 1

Сообщение для уровня 2

Сумма баллов для достижения уровня 2

Сообщение для уровня 3

Сумма баллов для достижения уровня 3

Сообщение для уровня 4

Сумма баллов для достижения уровня 4

Вопрос 1

Количество вариантов ответа к вопросу 1

Номер правильного ответа на вопрос 1

Вариант ответа

Вариант ответа

Вариант ответа

Вопрос 2

Количество вариантов ответа к вопросу 2

Номер правильного ответа на вопрос 2

Вариант ответа

Вариант ответа

Вариант ответа

...

В листинге 10.1 в качестве примера приведен файл теста, целью которого является проверка знания истории Санкт-Петербурга.

Листинг 10.1. Пример файла теста (spb.tst)

```
История Санкт-Петербурга
Отлично
7
Хорошо
6
Удовлетворительно
5
Плохо
4
Архитектор Исаакиевского собора:
3
2
Доменико Трезини
Огюст Монферран
Карл Росси
Александровская колона воздвигнута в 1836 г. как памятник,
посвященный:
2
1
деяниям императора Александра I
подвигу народа в Отечественной войне 1812 года
Архитектор Зимнего дворца:
```

3

1

Бартоломео Растрелли

Карл Росси

Огюст Монферран

Михайловский замок построен по проекту:

3

3

Воронихина Андрея Никифоровича

Старова Ивана Егоровича

Баженова Василия Ивановича

Остров, на котором находится Ботанический сад, называется:

3

3

Заячий

Медицинский

Аптекарский

Невский проспект получил свое название

3

2

по имени реки, на которой стоит Санкт-Петербург

по имени близко расположенного монастыря, Александро-Невской лавры

в память о знаменитом полководце - Александре Невском

Скульптура памятника Петру I "Медный всадник" выполнена

2

1

Фальконе

Клодтом

Подготовить файл теста можно при помощи редактора Small Basic (рис. 10.3).

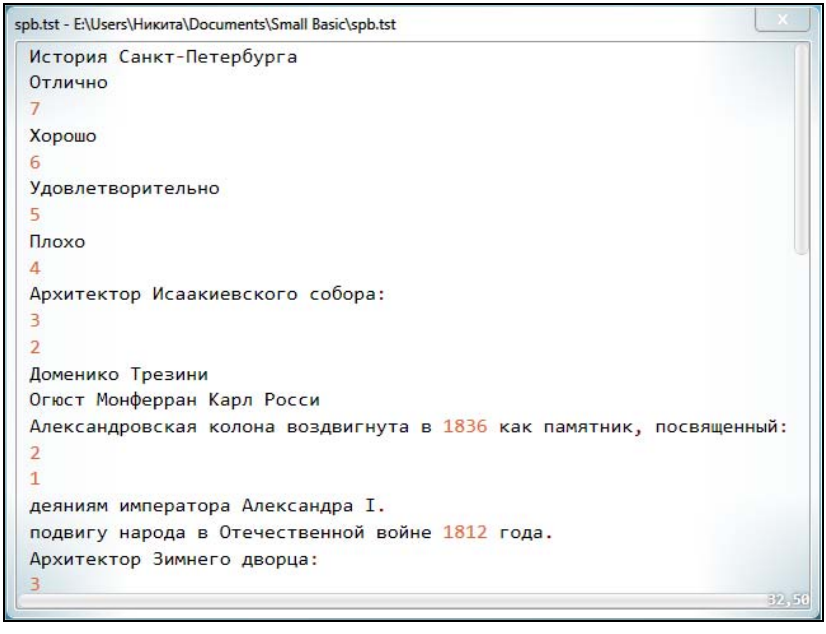


Рис. 10.3. Файл теста

Доступ к файлу теста

Чтобы программа могла работать с разными файлами тестов, надо каким-либо образом передать ей имя файла теста. Наиболее просто это можно сделать, указав имя файла теста в качестве параметра команды запуска программы (рис. 10.4).

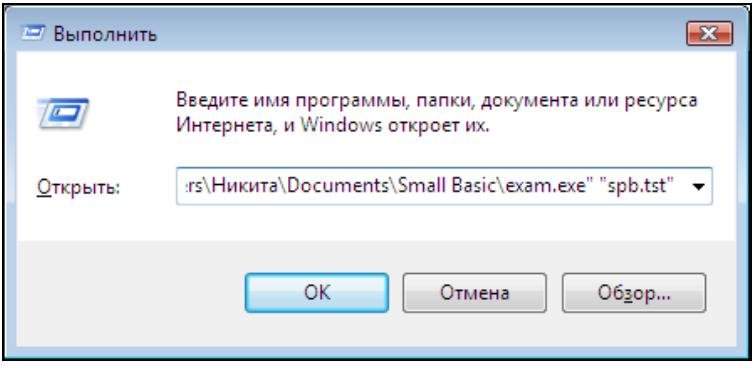


Рис. 10.4. Имя файла теста можно указать как параметр команды запуска программы тестирования

В приведенном примере `exam` — имя exe-файла программы тестирования, `spb.tst` — файл теста "История Санкт-Петербурга" (предполагается, что файл теста находится в том же каталоге, что и программа).

Программа, во время своей работы, может получить параметр (аргумент), указанный в команде ее запуска, обратившись к функции `GetArguments` (в качестве параметра функции необходимо указать номер нужного аргумента команды запуска).

Приведенный далее фрагмент кода демонстрирует использование функции `GetArguments` для достуа к аргументу команды запуска.

```
If (Arguments.Count = 0) Then
```

```
  TextWindow.WriteLine("В команде запуска программы аргумент
не указан.")
Else
  arg = Arguments.GetArgument(1)
  TextWindow.Write("Аргумент команды запуска программы: ")
  TextWindow.WriteLine(arg)
EndIf
```

Следует обратить внимание на то, что в Small Basic нет возможности передать параметр запускаемой программе. Поэтому одно из предполагаемых значений параметра следует указать в тексте программы, а затем, когда программа будет отлажена, добавить инструкции получения параметра из командной строки.

Программа

Текст программы "Экзаменатор" приведен в листинге 10.2.

Листинг 10.2. Программа "Экзаменатор"

```
'  Экзаменатор
' (c) Никита Культин, 2010

' Внимание!
' После того как программа будет отлажена,
' следующую инструкцию надо "раскомментировать"
'if ( Arguments.Count = 0 ) Then
' TextWindow.WriteLine("В команде запуска программы надо ука-
зать файл теста.")
'  TextWindow.Write("Для завершения нажмите <Enter>")
'  TextWindow.Read()
'  Program.End()
'Else
'  f = Arguments.GetArgument(1)
'EndIf

' Внимание!
' После того как программа будет отлажена, следующую
' инструкцию надо "закомментировать"
f = "spb.tst"  ' файл теста

nq = 0     ' количество вопросов теста
nRight = 0 ' количество правильных ответов

' для текущего вопроса
na = 0    ' количество вариантов ответа
right = 0 ' номер правильного ответа
```

```
answ = 0  ' номер выбранного ответа

' mes - массив  сообщений
' level - массив, кол-во правильных ответов,
'         необходимое для достижения уровня

n = 1 ' номер строки, читаемой из файла

' название теста
title = File.ReadLine(f,n)

n=n+1

TextWindow.BackgroundColor = "DarkBlue"

' Читаем критерии оценки - сообщение и количество
' правильных ответов, необходимое для достижения уровня
For i=1 To 4
  mes[i] = File.ReadLine(f,n)
  n=n+1
  level[i] = File.ReadLine(f,n)
  n= n + 1
EndFor

eof = "False"   ' для входа в цикл While
While (eof <> "True")
  ' Прочитать вопрос
  q = File.ReadLine(f,n)

  If (q <> "") Then
    ' Прочитан очередной вопрос
    nq = nq + 1 ' счетчик вопросов
```

```
TextWindow.Clear()
TextWindow.WriteLine(title)
TextWindow.WriteLine("Вопрос " + nq)

' Вывести вопрос
TextWindow.ForegroundColor = "White"
TextWindow.WriteLine(q)
TextWindow.ForegroundColor = "Gray"
n = n + 1

' Прочитать количество вариантов ответа
na = File.ReadLine(f,n)
n = n + 1

' Прочитать номер правильного ответа
right = File.ReadLine(f,n)
n = n + 1

' Прочитать и вывести варианты ответа
For i =1 To na
  s = File.ReadLine(f,n)
  n = n+1
  ' Вывод варианта ответа
  TextWindow.Write(i+". ")
  TextWindow.WriteLine(s)
endfor

' Здесь выведен вопрос и варианты ответа

' Получить от испытуемого номер ответа
TextWindow.WriteLine("")
```

```
    TextWindow.Write("Ваш выбор->")
    answ = TextWindow.ReadNumber()

    If (answ = right) Then
      ' Испытуемый выбрал правильный ответ
      nRight = nRight +1
    EndIf

  else
    ' Не удалось прочитать очередной вопрос
    eof = "True" ' достигнут конец файла
  endif
EndWhile

' Процесс тестирования завершен
TextWindow.Clear()

TextWindow.WriteLine(title)
TextWindow.WriteLine("Всего вопросов: " + nq)
TextWindow.WriteLine("Правильных ответов: " + nRight)

' Обработка результата тестирования
i = 1 '  Пусть правильных ответов достаточно для
'  достижения наивысшего уровня (лучшая оценка).
'  Если это не так, входим в цикл
while (nRight  < level[i]) and (i < 4)
  i = i + 1 ' понизим уровень оценки
EndWhile

TextWindow.WriteLine("Оценка: " + mes[i] )
TextWindow.WriteLine("")
```

Оценка результата тестирования носит интервальный характер. Например, если тест состоит из 10 вопросов, то чтобы получить "отлично", надо правильно ответить на 10 вопросов, "хорошо" — от 8 до 9, "удовлетворительно" — от 6 до 7. Оценка за тест получается сравнением количества правильных ответов и количества ответов, характеризующих уровень. В тесте "История Санкт-Петербурга" 7 вопросов. Уровень "отлично" характеризуется числом 7, уровень "хорошо" — числом 6, уровень "удовлетворительно" — числом 5.

В программе при определении оценки количество правильных ответов (`nRight`) сравнивается со значениями, характеризующими уровни оценок (массив `level`). Первоначально предполагается, что набранное количество баллов позволяет получить оценку первого уровня — "отлично". Если предположение неверно, то уровень оценки понижается, и набранное количество баллов сравнивается с оценкой этого уровня, и так до тех пор, пока не будет найден уровень, соответствующий набранному количеству баллов. После того как будет определен уровень, выводится соответствующее сообщение — оценка.

Запуск "теста"

Программа "Экзаменатор" должна получить имя файла теста из командной строки. Поэтому чтобы "запустить тест", пользователь должен в окне **Выполнить** (см. рис. 10.4) набрать имя программы тестирования и файла теста. Это не совсем удобно. Чтобы облегчить жизнь пользователю, можно настроить операционную систему так, что программа "Экзаменатор" будет запускаться в результате щелчка на значке файла теста. Настройка выполняется следующим образом. Сначала надо раскрыть папку, в которой находится какой-либо файл теста (tst-файл), и сделать двойной щелчок на значке файла. Так как расширение tst не является стандартизированным, то операционная система "не знает", какую программу надо запустить, чтобы открыть tst-файл. Поэтому она предложит указать программу, с помощью которой надо открыть файл — на экране появится окно **Выбор программы**. В этом окне нужно сделать щелчок на кнопке **Обзор**, раскрыть папку, в которой находится программа "Экзаменатор", и указать файл exam.exe. После этого надо установить флажок **Использовать выбранную программу для всех файлов такого типа**.

Вид окна **Выбор программы** после выполнения всех перечисленных действий приведен на рис. 10.5.

В результате описанных действий в реестр операционной системы будет внесена информация о том, что файлы с расширением tst надо открывать с помощью программы "Экзаменатор" (имя файла, на котором сделан щелчок, передается программе как параметр).

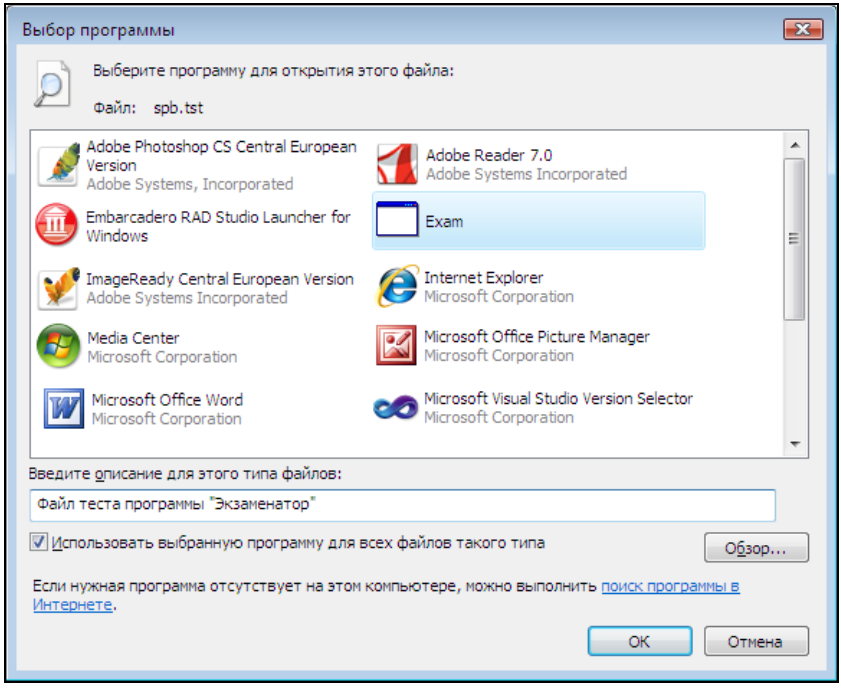


Рис. 10.5. Теперь файлы с расширением tst будет открывать "Экзаменатор"

Диаграмма

Программа "Диаграмма" демонстрирует работу с деловой графикой. Данные, используемые для построения диаграммы, загружаются из текстового файла. Характерная особенность программы состоит в том, что для построения диаграммы она использует "по максимуму" всю область окна, выделенную для диаграммы (ширина столбиков зависит от количества элементов данных, высота столбика, соответствующего максимальному значению ряда, равна высоте области построения диаграммы). Диаграмма строится в отклонениях от минимального значения. Делается это для того, чтобы сделать более наглядным разброс высот столбиков в случае незначительного разброса значений ряда данных.

Пример диаграммы и соответствующий ей файл данных приведены соответственно на рис. 10.6 и 10.7.

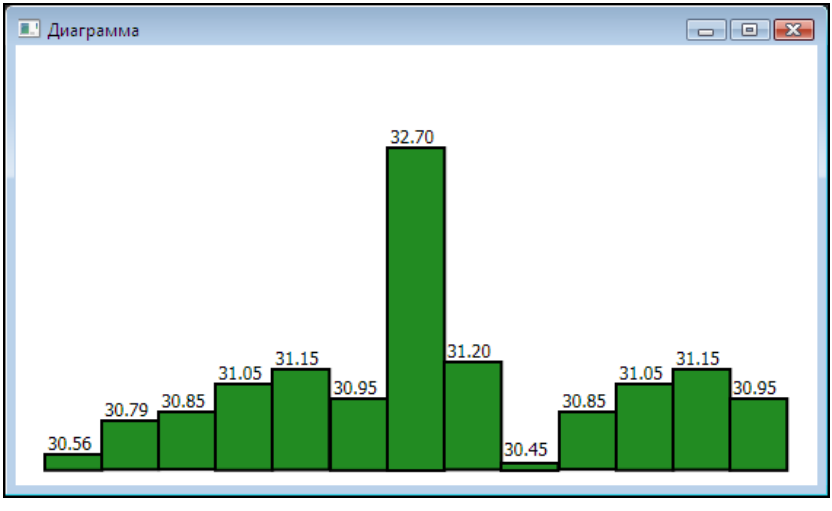


Рис. 10.6. Пример диаграммы

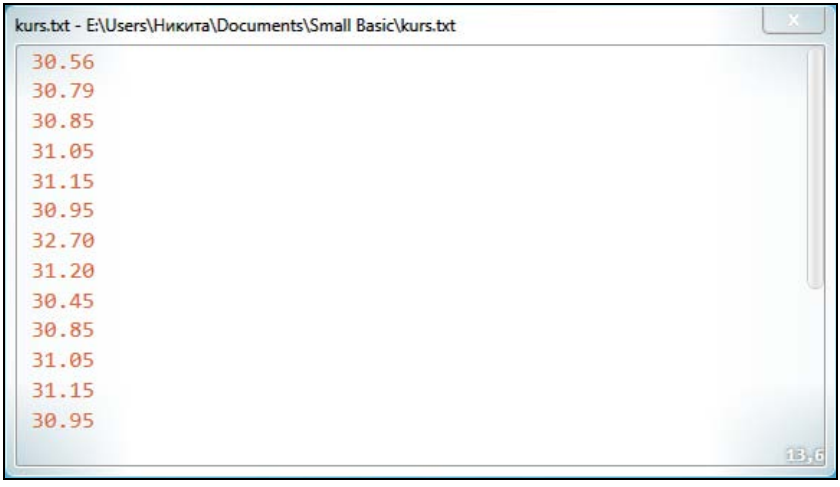


Рис. 10.7. Файл данных

Работает программа "Диаграмма" (листинг 10.3) следующим образом.

Сначала данные загружаются в массив `a`. Данные находятся в массиве `a`. Затем они копируются в массив `h` (предполагается, что массив `h` будет содержать значения высот столбиков в пикселах) и одновременно умножаются на 100. Умножение выполняется, чтобы преобразовать дробное число (исходные данные) в целое (высота столбика). Значения ряда данных (элементы массива `h`) могут отличаться друг от друга незначительно, то имеет смысл строить диаграмму в отклонениях от минимального значения. Поэтому выполняется поиск минимального элемента массива и минимальное значение вычитается из всех элементов массива `h`. Так как возможна ситуация, когда значение максимального элемента массива `h` (высота столбика, соответствующая максимальному элементу массива `a`) больше высоты области выделенной для построения диаграммы, то, в общем случае, необходимо выполнить масштабирование диаграммы (значений элементов массива `h`). Для вычисления коэффициента масштабирования

ищется максимальный элемент массива h. Затем элементы массива h умножаются на коэффициент масштабирования. Следует обратить внимание, если максимальное значение элементов массива h меньше высоты области построения диаграммы, то значение коэффициента масштабирования больше единицы. То есть диаграмма "растягивается" по вертикали. Последняя подготовительная операция — вычисление ширины столбика. Ширина вычисляется путем деления ширины области построения диаграммы на количество элементов данных. Далее в цикле For строится диаграмма.

Листинг 10.3. Диаграмма

```
' Столбчатая диаграмма
GraphicsWindow.Title = "Диаграмма"
GraphicsWindow.Width = 550
GraphicsWindow.Height = 300

' Читать данные из файла
f = Program.Directory + "\kurs.txt

i=1
eof = "False" ' чтобы войти в цикл
While (eof <> "True")
  st = File.ReadLine(f,i)

  If (st <> "") Then
    ' Строка прочитана
    a[i] = st
    'TextWindow.WriteLine(a[i])
    i=i+1
  Else
    ' Строк больше нет
```

```
    eof = "True"
  EndIf
EndWhile

' Данные находятся в массиве a.
' Так как значения отличаются незначительно,
' то будем строить диаграмму не значений,
' а отклонений от минимального значения.
' Кроме того, чтобы не потерять сотые,
' домножим значения на 100.

n = Array.GetItemCount(a) ' количество элементов массива
' Домножим каждый элемент массива на 100
For i=1 To n
  h[i] = a[i]*100
EndFor

' Найдем минимальный элемент массива
m = 1
For i=2 To n
  If h[i] < h[m] Then
    m = i
  endif
EndFor
min = h[m]

' От каждого элемента массива отнимем минимальный,
' в результате получим отклонения от минимального
For i=1 To n
  h[i] = h[i] - min
EndFor

' Чтобы диаграмма поместилась в области окна,
```

```
' выделенной для ее построения, или если максимальное
' значение ряда меньше высоты области построения,
' надо выполнить масштабирование.
' Для этого сначала найдем максимальный элемент массива.

' Поиск максимального элемента массива
m = 1 ' пусть первый элемент максимальный
For i=2 To n
  If h[i] > h[m] Then
    m=i
  endif
EndFor
max = h[m]

' Высота области построения диаграммы
hw = GraphicsWindow.Height - 80

' Вычислим коэффициент масштабирования
k = hw / max

' Промасштабируем данные
For i=1 To n
  h[i] = h[i]*k
EndFor

' Построение диаграммы
x0 =20

' Вертикальная координата оси
y0= GraphicsWindow.Height - 10

' Ширина столбика
w = (GraphicsWindow.Width - 40)/n
```

```
' Шрифт подписи данных
fs = 12
GraphicsWindow.FontSize = fs
GraphicsWindow.FontBold = "False"

' Строим
x=x0
For i=1 To n

  ' Рисуем столбик
  If (h[i])> 0  then

    GraphicsWindow.BrushColor = "ForestGreen"
    GraphicsWindow.FillRectangle(x,y0-h[i],w,h[i])
    GraphicsWindow.DrawRectangle(x,y0-h[i],w,h[i])

    ' Подпись
    GraphicsWindow.BrushColor = "Black"
    GraphicsWindow.DrawText(x+2,y0-h[i]-fs-3,a[i])

  Else
    GraphicsWindow.BrushColor = "ForestGreen"
    GraphicsWindow.FillRectangle(x,y0-5,w,5)
    GraphicsWindow.DrawRectangle(x,y0-5,w,5)

    ' Подпись
    GraphicsWindow.BrushColor = "Black"
    GraphicsWindow.DrawText(x+2,y0-5-fs-5,a[i])
  Endif

  x=x+w
EndFor
```

Игра "15"

Перед тем как приступить к рассмотрению игры, обсудим, как пользователь может взаимодействовать с программой, используя мышь.

Программа может реагировать на различные *события*, например, на щелчок кнопкой мыши или нажатие клавиши. Щелчок кнопкой мыши и нажатие клавиши — это пример того, что называют *событием*.

Программа Small Basic может воспринимать следующие события: нажатие кнопки мыши, отпускание кнопки мыши, перемещение указателя мыши, нажатие клавиши клавиатуры.

У каждого события есть имя (табл. 10.1).

***Таблица 10.1.** События, которые может воспринимать программа*

Событие	Описание
`MouseDown`	Нажатие кнопки мыши
`MouseUp`	Отпускание кнопки мыши
`MouseMove`	Перемещение указателя мыши
`KyeDown`	Нажатие клавиши клавиатуры
`KeyUp`	Отпускание нажатой клавиши клавиатуры

Для того чтобы программа реагировала на событие, необходимо написать процедуру обработки этого события и указать имя этой процедуры в качестве обработчика события. Процедура — обработчик задается путем присвоения значения соответствующему свойству (табл. 10.2).

Таблица 10.2. *Свойства, определяющие процедуры обработки событий*

Свойство	Задает процедуру обработки события
`GraphicsWindow.MouseDown`	`MouseDown`
`GraphicsWindow.MouseUp`	`MouseUp`
`GraphicsWindow.MouseMove`	`KyeDown`
`GraphicsWindow.KyeDown`	`KyeDown`
`GraphicsWindow.KeyUp`	`KeyUp`

Часто программе необходима информация, в какой точке окна находился указатель мыши в момент нажатия кнопки мыши. Получить информацию о положении указателя мыши можно, обратившись к свойствам `GraphicsWindow.MouseX` и `GraphicsWindow.MouseY`.

В качестве примера в листинге 10.4 приведена программа, которая демонстрирует обработку нажатия кнопки мыши. В ее окне (рис. 10.8), в точке, в которой пользователь нажал кнопку мыши, рисуется прямоугольник и отображаются координаты точки.

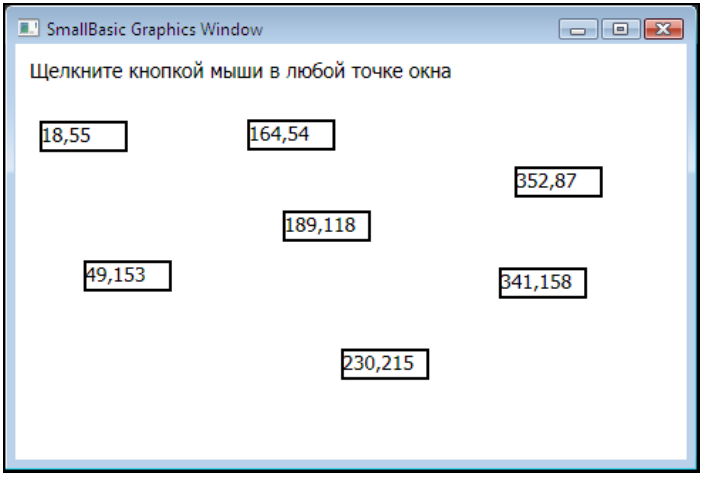


Рис. 10.8. Обработка нажатия кнопки мыши в окне программы

Листинг 10.4. Обработка нажатия кнопки мыши

```
' Обработка нажатия (щелчка) кнопки мыши

' Зададим процедуру обработки нажатия кнопки мыши
GraphicsWindow.MouseDown = md

GraphicsWindow.DrawText(10,10,"Щелкните кнопкой мыши в любой
точке окна")

' Это процедура обработки нажатия кнопки мыши
Sub md
  ' Получить координаты точки, в которой находился
  ' указатель мыши в момент нажатия кнопки
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY

  ' Рисуем прямоугольник
  GraphicsWindow.DrawRectangle(x,y,60,20)

  ' Показать координаты нажатия кнопки
  s = x + "," + y
  GraphicsWindow.DrawText(x,y,s)
EndSub
```

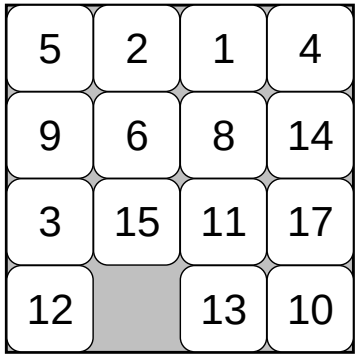


Рис. 10.9. В начале игры фишки перемешаны

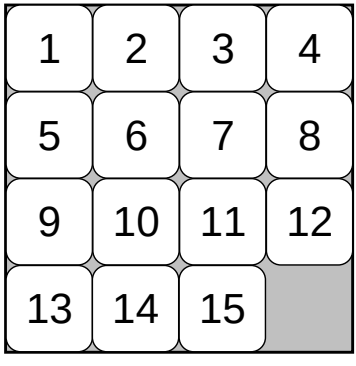


Рис. 10.10. Правильный порядок фишек

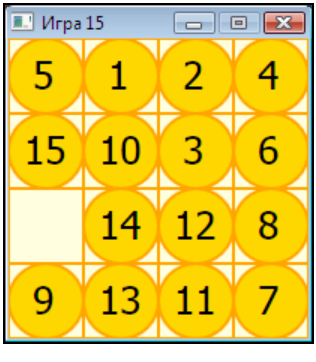


Рис. 10.11. Игра "15"

Теперь можно перейти к рассмотрению компьютерной реализации логической игры "15". Вот ее правила. В прямоугольной коробочке находятся 15 фишек, на которых написаны числа от 1 до 15. Размер коробочки — 4×4, таким образом, в ней есть одна пустая ячейка. В начале игры фишки перемешаны (рис. 10.9). Задача игрока состоит в том, чтобы, не вынимая фишки из коробочки, выстроить их в правильном порядке (рис. 10.10). Программа "Игра 15", ее текст приведен в листинге 10.5, реализует описанную игру. Окно программы приведено на рис. 10.11. Фишка, на изображении которой игрок делает щелчок левой кнопкой мыши, перемещается в пустую клетку.

Листинг 10.5. Игра "15"

```
' Игра "15"
' (c) Никита Культин, 2010

' Возможны 5 вариантов оформления
p = Math.GetRandomNumber(5)

If p = 1 then
  ' Barby
  fColor = "White"              ' цвет цифры на фишке
  bColor = "HotPink"            ' цвет фишки
  lColor = "MediumVioletRed"    ' цвет линий
  eColor = "Pink"               ' цвет пустой клетки
else
  If p = 2 Then
    ' Military
    fColor = "Black"            ' цвет цифры на фишке
    bColor = "ForestGreen"      ' цвет фишки
    lColor = "DarkGreen"        ' цвет линий
    eColor = "Green"            ' цвет пустой клетки
```

```
  Else
    If p =3 Then
      ' Classic
      fColor = "White"          ' цвет цифры на фишке
      bColor = "Black"          ' цвет фишки
      lColor = "White"          ' цвет линий
      eColor = "DimGray"        ' цвет пустой клетки
    Else
      If p = 4 Then
        ' Sky
        fColor = "White"        ' цвет цифры на фишке
        bColor = "SkyBlue"      ' цвет фишки
        lColor = "SteelBlue"    ' цвет линий
        eColor = "LightBlue"    ' цвет пустой клетки
      Else
        ' Gold
        fColor = "Black"        ' цвет цифры на фишке
        bColor = "Gold"         ' цвет фишки
        lColor = "Orange"       ' цвет линий
        eColor = "LightYellow"  ' цвет пустой клетки
      EndIf
    EndIf
  EndIf
endif

W = 4 ' кол-во клеток по горизонтали
H = 4 ' кол-во клеток по вертикали

' Размер клетки 50x50
ch = 50
cw = 50
```

```
' Положение (координаты) пустой клетки
ex = 4
ey = 4

' Поместить фишки на поле в правильном порядке
For i = 1 To W
  For j = 1 To H
    pole[i][j] = (i-1)*W + j
  EndFor
EndFor

pole[ex][ey] = 0 ' пустая клетка

' Настройка окна программы
GraphicsWindow.Hide()
GraphicsWindow.CanResize = "False"
GraphicsWindow.Width = W * cw
GraphicsWindow.Height = H * ch
GraphicsWindow.Title = "Игра 15"
GraphicsWindow.Top = 200
GraphicsWindow.Left = 300
GraphicsWindow.Show()

' Шрифт текста на фишке
GraphicsWindow.FontSize = 26
GraphicsWindow.FontBold = "False"

' Перемешать фишки
' (x1,y1) - координаты пустой клетки
' (x2,y2) - координаты перемещаемой фишки
x1 = W
y1 = H
```

```
d = 0 ' направление перемещения относительно пустой клетки

For i = 1 To 100
  ' Выбрать фишку, которую переместим в пустую клетку

  found = "False"
  While found = "False"

    ' Определим фишку, которую двигаем в пустую клетку.
    ' В общем случае есть 4 фишки. Однако если пустая клетка
    ' на границе, то количество фишек меньше.
    d = Math.GetRandomNumber(4)

    If d = 1 Then
      ' Берем фишку справа от пустой
      x2=x1+1
      y2 = y1
    else
      If d = 2 Then
        ' Берем фишку слева от пустой
        x2 = x1-1
        y2 = y1
      else
        If d = 3 Then
          ' Берем фишку сверху от пустой
          y2 = y1 - 1
          x2 = x1
        Else
          ' Берем фишку снизу от пустой
          y2 = y1 +1
          x2 = x1
```

```
        EndIf
      EndIf
    EndIf

    ' Убедимся, что координаты перемещаемой фишки
    ' определены правильно
    If (x2 >=1) And (x2 <=W) And (y2>=1) And (y2 <=H) then
      found = "True"
    endif
  EndWhile

  ' Здесь определили фишку, которую надо переместить
  ' в пустую клетку

  '  Переместим
  pole[y1][x1] = pole[y2][x2]
  pole[y2][x2] = 0
  x1 = x2
  y1 = y2
EndFor

' Здесь фишки перемешаны
ex = x1
ey = y1

' Показать поле
y = 0
GraphicsWindow.PenColor = lColor
For i=1 To W
  x = 0
  For j = 1 To H
    ' Рисуем клетку
```

```
    GraphicsWindow.BrushColor = eColor
    GraphicsWindow.FillRectangle(x,y,cw,ch)
    GraphicsWindow.DrawRectangle(x,y,cw,ch)

    If pole[i][j] > 0 Then
      GraphicsWindow.BrushColor = bColor
      GraphicsWindow.FillEllipse(x,y,cw,ch)
      GraphicsWindow.DrawEllipse(x,y,cw,ch)
      GraphicsWindow.BrushColor = fColor

      If (pole[i][j] <= 9) then
        GraphicsWindow.DrawText(x+16,y+8,pole[i][j])
      else
        GraphicsWindow.DrawText(x+10,y+8,pole[i][j])
      EndIf
    else
      ' Пустая клетка
      '  GraphicsWindow.BrushColor = eColor
      '  GraphicsWindow.FillRectangle(x,y,cw,ch)
      '  GraphicsWindow.DrawRectangle(x,y,cw,ch)
      'GraphicsWindow.BrushColor = "White"
    endif
    x = x + cw
  EndFor
  y = y+ch
EndFor

' Задать процедуру, обрабатывающую щелчок мыши
' на игровом поле
GraphicsWindow.MouseDown = md
```

```
' Процедура md обрабатывает нажатие кнопки мыши
Sub md
  ' Определим координаты клетки, в которой сделан щелчок

  ' Сначала определим координаты щелчка в окне программы:
  ' 3  - ширина левой границы окна;
  ' 23 - высота заголовка окна
  mx = Mouse.MouseX - GraphicsWindow.Left - 2
  my = Mouse.MouseY - GraphicsWindow.Top - 22

  cx = Math.Floor(mx / cw) + 1
  cy = Math.Floor(my / ch) + 1

  ' Переместить фишку, на которой сделан щелчок,
  ' в соседнюю клетку.
  ' (ex, ey) - координаты пустой клетки

  If ((Math.Abs(cx-ex)=1) And (cy = ey)) Or ((cx = ex) And
(Math.Abs(cy-ey) = 1)) Then
    ' Переместить фишку, на которой сделан щелчок,
    ' в пустую клетку
    pole[ey][ex] = pole[cy][cx]
    pole[cy][cx] = 0

    b = ex
    ex = cx
    cx = b

    b = ey
    ey = cy
    cy = b

    ' Перерисовать измененные клетки
```

```
x = (cx-1) * cw
y = (cy-1) * ch

GraphicsWindow.BrushColor = bColor
GraphicsWindow.PenColor = lColor
GraphicsWindow.FillEllipse(x,y,cw,ch)
GraphicsWindow.DrawEllipse(x,y,cw,ch)
GraphicsWindow.BrushColor = fColor

If (pole[cy][cx] <= 9) then
  GraphicsWindow.DrawText(x+16,y+8,pole[cy][cx])
else
  GraphicsWindow.DrawText(x+10,y+8,pole[cy][cx])
EndIf

' Пустая клетка
x = (ex-1) * cw
y = (ey-1) * ch

GraphicsWindow.BrushColor = eColor
GraphicsWindow.FillRectangle(x,y,cw,ch)
GraphicsWindow.DrawRectangle(x,y,cw,ch)

' Проверим, правильно ли выстроены клетки.
' Если клетки выстроены правильно, то элементы массива
' pole по строкам содержат числа от 1 до 15
ok = "True"
i = 1
r = 1
c = 1
While (ok = "True") And (i < W*H)
  If pole[r][c] <> i Then
```

```
        ok = "False"
      else
        i=i+1
        ' Переход к следующей клетке
        If c < W Then
          c = c + 1
        Else
          c = 1
          r = r + 1
        EndIf
      EndIf
    EndWhile

    If ok = "True" Then
      ' Фишки выстроены в правильном порядке
      GraphicsWindow.ShowMessage("Поздравляю! Вы справились
      с поставленной задачей!", "Игра 15")
      Program.End()
    EndIf
  EndIf
EndSub
```

Игра "Парные картинки"

Игра "Парные картинки" развивает внимание. Вот ее правила. Игровое поле разделено на клетки, за каждой из которых скрыта картинка. Картинки — парные, т. е. на игровом поле есть две клетки, в которых находятся одинаковые картинки. В начале игры все клетки "закрыты". Щелчок левой кнопкой мыши "открывает" клетку, в клетке появляется картинка. Теперь надо найти клетку, в которой находится такая же картинка. Щелчок в другой клетке открывает вторую картинку. Если картинки в открытых клетках одинаковые, то эти клетки "исчезают". Если разные — клетки остаются открытыми. Следующий щелчок закрывает две открытые клетки и открывает ту, в которой сделан щелчок. При этом две открытые клетки закрываются даже в том случае, если в открываемой в данный момент клетке находится такая же картинка, как и в одной из двух открытых. Игра заканчивается, когда игрок откроет (найдет) все пары картинок.

В приведенной реализации игры 8 пар картинок, размер поля — 4×4. Картинки загружаются из файлов.

Рис. 10.12. Пример окна программы "Парные картинки"

На рис. 10.12 приведен пример окна программы, после того как игрок "нашел" одну пару картинок. Текст программы "Парные картинки" приведен в листинге 10.6.

Листинг 10.6. Игра "Парные картинки"

```
' Парные картинки.
' (c) Никита Культин, 2010

' P[][] - игровое поле
' P[i][j] - номер (1..8) картинки
' P[i][j] = 0 - в клетке картинка, для которой найдена пара

' Размер клетки (картинки)
WP = 96  ' ширина клетки
HP = 96  ' высота клетки

' Настройка окна программы
GraphicsWindow.Hide()
GraphicsWindow.Width = WP * 4
GraphicsWindow.Height = HP * 4
GraphicsWindow.CanResize = "False"
GraphicsWindow.Title = "Парные картинки"
GraphicsWindow.Left = 400
GraphicsWindow.Top = 300
GraphicsWindow.Show()

' **** Разбросать картинки
' В массив P запишем номера картинок (числа от 1 до 8).
' Каждое число должно быть записано в массив P два раза.
' Количество записанных чисел будем фиксировать в массиве B.
' Элемент B[i] хранит информацию о том, сколько раз
```

```
' число i было записано в массив P.

' Инициализация массива B
For i=1 To 8
  B[i] = 0
EndFor

For i = 1 To 4
  For j = 1 To 4
    ' Будем генерировать число, до тех пор,
    ' пока не получим число,
    ' которое в массив P записано менее 2-х раз
    r = Math.GetRandomNumber(8)
    While (B[r] = 2)
      r = Math.GetRandomNumber(8)
    EndWhile

    P[i][j] = r
    B[r] = B[r] + 1
  endfor
endfor

'  Загрузить картинки
For i = 1 To 8
  img[i] = ImageList.LoadImage(Program.Directory + "\" +  i
+".bmp")
EndFor

' Рисуем поле (клетки)
For i=1 to 4
  x=0
  For j=1 To 4
    GraphicsWindow.DrawRectangle(x,y,WP,HP)
```

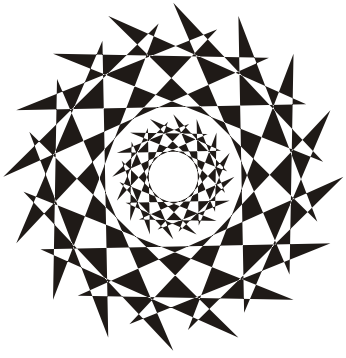
```
    x=x+WP
  EndFor
  y=y+HP
endfor

no = 0  ' количество открытых пар картинок
np = 0  ' количество картинок, открытых в данный момент

' Зададим процедуру обработки щелчка кнопкой мыши
GraphicsWindow.MouseDown = click

' Номера строк (r) и столбцов (c) открытых клеток
r1=0
c1=0
r2=0
c2=0

' Щелчок кнопкой мыши на игровом поле
Sub click
  ' Определим координаты ячейки, в которой сделан щелчок.
  ' Сначала определим координаты щелчка в окне программы
  ' (3  - ширина левой границы окна;
  '  23 - высота заголовка окна)
  mx =   Mouse.MouseX - GraphicsWindow.Left - 2
  my = Mouse.MouseY - GraphicsWindow.Top - 22

  c = Math.Floor(mx / WP) + 1
  r = Math.Floor(my / HP) + 1

   If (P[r][c] = 0) Then
    ' Щелчок сделан в ячейке, в которой находилась
    '  картинка, для которой найдена пара
```

```
    Goto e1
  EndIf

  If (np = 0) Then
    ' Нет открытых клеток.
    ' Откроем клетку (покажем картинку),
    ' в которой сделан щелчок.
    ' (x, y) - координаты левого верхнего угла клетки
    x = (c - 1) * WP
    y = (r - 1) * HP

    k = P[r][c]
    GraphicsWindow.DrawImage(img[k],x,y)

    ' Запомнить положение открытой клетки
    r1=r
    c1=c
    np = 1    ' количество открытых клеток
  else
    If (np = 1) Then
      ' Открыта одна клетка.
      ' Откроем клетку, в которой сделан щелчок
      x = (c - 1) * WP
      y = (r - 1) * HP

      k = P[r][c]
      GraphicsWindow.DrawImage(img[k],x,y)

      ' Запомнить положение открытой клетки
      r2=r
      c2=c
```

```
    np = 2

    ' Проверим, возможно, открыты одинаковые картинки
    If (P[r1][c1] = P[r2][c2] ) Then
      ' Открыты клетки с одинаковыми картинками
      no = no + 1
      ' В клетках находятся одинаковые картинки,
      ' и они найдены
      P[r1][c1] = 0
      P[r2][c2] = 0

      ' Запускаем таймер: tuk — процедура обработки
      '                          сигнала от таймера
      Timer.Tick = tuk
      Timer.Interval = 100

      np = 0
    endif
  else
    ' Открыты две клетки.

    ' Закрыть открытые и открыть ту,
    ' в которой сделан щелчок
    GraphicsWindow.BrushColor = "WhiteSmoke"

    x = (c1 - 1) * WP
    y = (r1 - 1) * HP
    GraphicsWindow.FillRectangle(x,y,WP,HP)
    GraphicsWindow.DrawRectangle(x,y,WP,HP)

    x = (c2 - 1) * WP
    y = (r2 - 1) * HP
```

```
      GraphicsWindow.FillRectangle(x,y,WP,HP)
      GraphicsWindow.DrawRectangle(x,y,WP,HP)

      ' Откроем клетку, в которой сделан щелчок
      ' (x, y) — координаты левого верхнего угла клетки
      x = (c - 1) * WP
      y = (r - 1) * HP

      k = P[r][c]
      GraphicsWindow.DrawImage(img[k],x,y)

      ' Запомнить положение открытой клетки
      r1=r
      c1=c
      np = 1
    endif
  endif

  e1:
EndSub

' Сигнал от таймера
Sub tuk
  Timer.Pause() ' остановить таймер

  ' Убрать клетки, в которых находится
  ' найденная пара картинок
  GraphicsWindow.BrushColor = "Silver"
  x = (c1 - 1) * WP
  y = (r1 - 1) * HP
  GraphicsWindow.FillRectangle(x,y,WP,HP)
  GraphicsWindow.DrawRectangle(x,y,WP,HP)
```

```
  x = (c2 - 1) * WP
  y = (r2 - 1) * HP
  GraphicsWindow.FillRectangle(x,y,WP,HP)
  GraphicsWindow.DrawRectangle(x,y,WP,HP)

  If (no = 8) Then
    ' Найдены все 8 пар картинок
    GraphicsWindow.ShowMessage("Game over!", "Парные
картинки")
    Program.End()
  endif
EndSub
```



Приложение

Описание DVD

Прилагаемый к книге DVD содержит дистрибутивы Microsoft Small Basic и Visual Studio Express 2010, справочную информацию, а также программы (исходные тексты), приведенные в книге в качестве примеров (табл. П1).

Таблица П1. *Содержимое DVD*

Папки	**Файл**	**Описание**
Документация	Vvedenie_v_Small_Basic.pdf	Документация по программированию на Small Basic на русском языке ("Знакомство с программированием") [1]
Документация\SmallBasicDocumentation	index.htm	Список функций Small Basic и компонентов FC[2] на английском языке. (Запускать из папки файл index.htm)[3]
Документация	Регистрация MS Visual Studio 2010 Express.pdf	Инструкция по регистрации MS Visual Studio 2010 Express
Дистрибутивы	SmallBasic.msi	Дистрибутив Small Basic
Дистрибутивы\Microsoft .NET Framework 3.5	dotNetFx35setup.exe	.NET Framework 3.5
Дистрибутивы\Пакет обновления 1 (SP1) Microsoft .NET Framework 3.5	dotnetfx35setup.exe	Пакет обновления 1 (SP1) для Microsoft .NET Framework 3.5

[1] Размещено с разрешения автора Михаила Пчельникова.
[2] FC — FremyCompany.
[3] Размещено с разрешения автора Франсуа Реми.

Таблица П1 *(окончание)*

Папки	Файл	Описание
Дистрибутивы\ Microsoft .NET Framework 3.5 Service pack 1 (Full Package) — Русский	dotnetfx35.exe	Microsoft .NET Framework 3.5 Service Pack 1 (Full Package)
Дистрибутивы\ Visual Studio 2010		Дистрибутив Visual Studio Express 2010 — Русский
Дистрибутивы\ Библиотека расширения от FremyCompany		Библиотека расширения от компании FremyCompany, которая *значительно расширяет* возможности языка (появляются дополнительные объекты, такие как `Clipboard`, `Controls`, `DataFile`, `Dialogs`, `Drawings`, `Keyboard`, `Xml` и прочие)
Примеры		Примеры из книги

Примечание

Дополнительную документацию и примеры по Small Basic можно получить на сайте русскоязычного сообщества для начинающих программистов Small Basic: **http://www.smallbasic.ru.**

Дистрибутив Microsoft Small Basic находится в папке Дистрибутивы. Чтобы установить Microsoft Small Basic на компьютер, откройте папку Дистрибутивы, сделайте двойной щелчок на значке файла SmallBasic.msi и следуйте инструкциям мастера установки.

Примечание

Для установки Small Basic требуется операционная система: Windows XP/Vista/7 и обязательно .NET Framework 3.5. .NET Framework 3.5 по умолчанию установлен в ОС Windows 7.

Microsoft .NET Framework 3.5 располагается в папке Дистрибутивы\Microsoft .NET Framework 3.5. Для его установки сделайте двойной щелчок мыши на файле dotNetFx35setup.exe и следуйте инструкциям мастера установки. Затем установите пакеты обновления, размещенные в папках Дистрибутивы\ Пакет обновления 1 (SP1) Microsoft .NET Framework 3.5 и Дистрибутивы\Microsoft .NET Framework 3.5 Service Pack 1 (Full Package) — Русский.

Примечание

Более подробную информацию о загрузке и установке Microsoft .NET Framework можно найти на сайте **http://msdn.microsoft.com/ru-ru/netframework** или **http://www.smallbasic.ru/download**.

Исходные тексты программ находятся в папке Примеры. Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте папку Примеры в папку Документы.

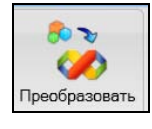
Тот, кто захочет пойти дальше в изучении программирования, может установить Visual Basic 2010 Express. Дистрибутив Visual Studio 2010 Express находится в папке Дистрибутивы\ Visual Studio 2010. Чтобы установить Visual Studio 2010 на компьютер, сделайте двойной щелчок на значке файла Setup.hta и следуйте инструкциям мастера установки.

Внимание!

Microsoft® Visual Studio® 2010 Express необходимо зарегистрировать в течение 30 дней после установки. Регистрация осуществляется вводом бесплатного регистрационного ключа продукта. Инструкция по регистрации находится в файле Документация\Регистрация MS Visual Studio 2010 Express.pdf.

Примечание

Вы также сможете экспортировать свою программу на Small Basic в Visual Basic 2010 Express, нажав кнопку **Преобразовать** в главном меню программы.



Предметный указатель

www.bhv.ru

Дукин А. Н., Пожидаев А. А.
Самоучитель Visual Basic 2010

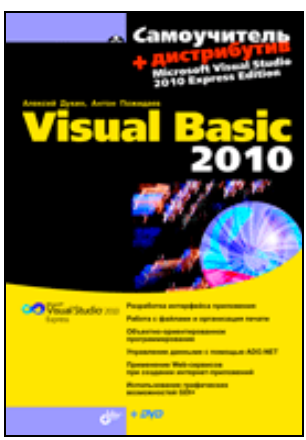
Магазин "Новая техническая книга"

СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10

Отдел оптовых поставок

E-mail: opt@bhv.spb.su

Ваш учитель по быстрому созданию приложений



- Разработка интерфейса приложения
- Работа с файлами и организация печати
- Объектно-ориентированное программирование
- Управление данными с помощью ADO.NET
- Применение Web-сервисов при создании интернет-приложений
- Использование графических возможностей GDI+

В книге рассмотрен инструмент для разработки приложений Visual Basic 2010, который позволяет в кратчайшие сроки создавать профессиональные проекты для работы в Windows, интеренет-приложения, а также программы для карманных компьютеров и сотовых телефонов. Шаг за шагом на многочисленных примерах обсуждаются интегрированная среда разработки, вопросы построения проекта и создания законченного приложения. Большое внимание уделяется созданию информационных систем с использованием СУБД. Приводится множество готовых решений, которые читатель сможет использовать при разработке собственных приложений. Рассматриваются вопросы, связанные с созданием собственного установочного диска и справочной системы. Книга легко читается, доступна даже для тех, кто имеет минимальный опыт программирования в Visual Basic или не имеет его вовсе.

На компакт-диске размещен дистрибутив пакета Microsoft Visual Studio 2010 Express Edition, содержащий Visual Basic 2010 Express Edition и другие компоненты пакета.

www.bhv.ru

Зиборов В. В.
Visual Basic 2010 на примерах

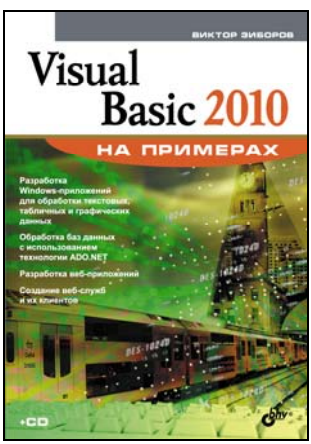
Магазин "Новая техническая книга"

СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10

Отдел оптовых поставок

E-mail: opt@bhv.spb.su

В книге подробно рассмотрено более сотни типичных примеров, встречающихся в практике реального программирования для платформы .NET Framework в среде Microsoft Visual Basic 2010.



Примеры расположены в порядке от простого к сложному:

- простейшие программы с экранной формой и элементами управления,
- инициирование и обработка событий мыши и клавиатуры,
- чтение/запись текстовых и бинарных файлов,
- редактирование графических данных,
- управление буфером обмена,
- ввод и вывод табличных данных,
- использование функций MS Word, MS Excel и AutoCAD,
- обработка баз данных с использованием технологии ADO.NET,
- создание веб-служб и их клиентов,
- программирование веб-приложений и др.

Последовательное изучение примеров позволит начинающим быстро освоить новую среду разработки, а опытные пользователи найдут готовые решения для использования в своих приложениях.

ЗИБОРОВ Виктор Владимирович, кандидат технических наук, доцент кафедры геоинформатики и фотограмметрии Киевского национального университета строительства и архитектуры. Автор более 50 научных и учебно-методических работ.

www.bhv.ru

Дейтел П., Дейтел Х., Эйр Г.
Просто о Visual Basic 2008.
Обучение на практических примерах.
3 изд.

Магазин "Новая техническая книга"

СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10

www.techkniga.com

Отдел оптовых поставок

E-mail: opt@bhv.spb.su

Книга посвящена разработке приложений в среде Visual Basic 2008. Материал излагается в виде 32 уроков, подготовленных в соответствии с методикой обучения программированию компании Deitel & Associates. На примере разработки более 100 готовых приложений рассматриваются: среда разработки, визуальное программирование, библиотека классов .NET Framework Class Library, элементы управления WinForm, обработка событий, отладчик, управляющие операторы, массивы, классы, объекты, базы данных, Web-приложения и др. Показано применение технологий LINQ, ASP.NET 3.5, ASP.NET AJAX, WPF, XAML и Silverlight. На DVD размещен дистрибутив Microsoft Visual Studio 2008 Express Edition, содержащий Visual Basic 2008 Express Edition и другие компоненты пакета.

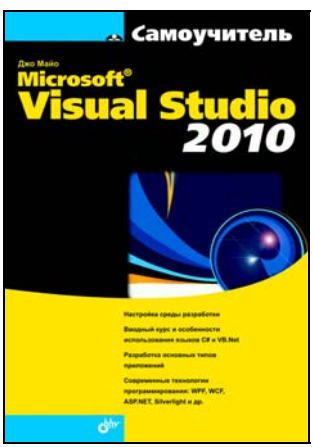
На DVD размещен дистрибутив Microsoft Visual Studio 2008 Express Edition, содержащий Visual Basic 2008 Express Edition и другие компоненты пакета.

Информационная поддержка книги на сайте **www.deitel.com**

www.bhv.ru

Джозеф Майо
Самоучитель Microsoft Visual Studio 2010

Магазин	СПб., Измайловский пр., д. 29,
Новая техническая книга	тел.: (812) 251-41-10, www.techkniga.com
Отдел оптовых поставок	E-mail: opt@bhv.spb.su



Джозеф Майо (*Joseph Mayo*) — профессиональный программист, занимается разработкой программного обеспечения с 1986 г., является сертифицированым специалистом Microsoft MVP, специализируется на использовании технологии Microsoft .NET. Автор многочисленных книг по языкам программирования и использованию технологий Microsoft для разработчиков.

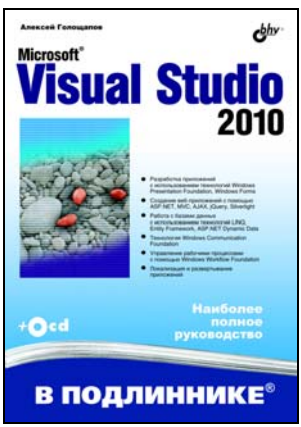
- Показано *создание различных типов приложений* в интегрированной среде разработки Microsoft Visual Studio 2010.
- Рассмотрены *основы программирования на языках C# и VB*, работа с решениями, проектами, сборками и библиотеками классов.
- Описаны *инструменты* предназначенные для анализа и отладки кода, поиска и исправления ошибок.
- Рассмотрена *работа с базами данных* с помощью языка интегрированных запросов LINQ. Приведена информация о языках XML и XAML.
- Описаны основные концепции работы с системой *Windows Presentation Foundation*, технология *Silverlight*, построение Web-приложений с помощью технологии *ASP.NET MVC*, создание Web-сервисов с помощью *Windows Communications Foundation*.
- Рассмотрено создание собственной программы-мастера для работы над проектами, шаблонов для автоматизации генерируемых фрагментов кода и рутинных задач, добавочных модулей и др.

Основные навыки работы в Visual Studio 2010 — легко и доступно!

www.bhv.ru

Голощапов А. Л.
Microsoft Visual Studio 2010

Магазин **Новая техническая книга**:	СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10, www.techkniga.com
Отдел оптовых поставок:	E-mail: opt@bhv.spb.su



Голощапов Алексей Леонидович, ведущий программист и архитектор программного обеспечения и баз данных. Сертифицированный специалист Microsoft, разработчик с многолетним опытом работы, специализирующийся на разработке приложений для настольных и мобильных систем и баз данных. Автор книги «Google Android: программирование для мобильных устройств»

Прочитав книгу, вы узнаете об инструментах, которые входят в состав Visual Studio 2010, познакомитесь с многочисленными технологиями, предоставляемыми средой для разработки практических приложений. Рассмотрены:

- Разработка приложений с использованием технологий Windows Presentation Foundation, Windows Forms
- Создание Web-приложений с помощью ASP.NET, MVC, AJAX, jQuery, Silverlight
- Работа с базами данных с использованием технологий LINQ, Entity Framework, ASP.NET Dynamic Data
- Технология Windows Communication Foundation
- Управление рабочими процессами с помощью Windows Workflow Foundation
- Локализация и развертывание приложений

На компакт-диске размещены примеры программ из книги.

Быстрое погружение в среду разработки!

Об авторах:

«Идеальный как для детей, так и для взрослых, Small Basic поможет начинающим сделать первые шаги в удивительный мир программирования»
с сайта www.smallbasic.com

Культин Никита Борисович, кандидат технических наук, доцент Санкт-Петербургского государственного политехнического университета, где читает курс «Теория и технология программирования». Автор серии книг по программированию в Turbo Pascal, Delphi, Microsoft Visual C++, Microsoft Visual Basic и др., которые вышли общим тиражом более 350 тыс. экземпляров.

Цой Лариса Борисовна, специалист в области информационно-коммуникационных технологий, обладает большим опытом преподавательской деятельности, соавтор серии книг по офисным приложениям и программированию для начинающих, в том числе популярной книги «Visual Basic для студентов и школьников».

Small Basic

для начинающих

Книга подготовлена при поддержке корпорации Microsoft

Ресурсы Microsoft для начинающих разработчиков:
msdn.microsoft.com/beginner
microsoft.ru/express

+DVD

Диск содержит дистрибутив программы, примеры, рассмотренные в книге, и справочную информацию.

Если вы хотите научиться программировать и не знаете с чего начать, то эта книга именно то, что вам нужно. В ней в доступной форме изложены основы теории программирования и приведено описание доступного для начинающих современного языка программирования Small Basic. Показан процесс создания программы: от составления алгоритма до записи инструкций на языке программирования и отладки. Вы узнаете:

- Как представить алгоритм программы в виде блок-схемы?
- Как записать инструкции программы?
- Что такое переменная?
- Как ввести исходные данные с клавиатуры?
- Как вывести результат работы программы на экран?
- Как использовать инструкции выбора и циклов?
- Как работать с массивами?
- Как вывести на экран графику?

Ответы на эти и многие другие вопросы начинающего программиста вы найдете в этой книге.

bhv®

БХВ-Петербург

190005, Санкт-Петербург, Измайловский пр., д 29

E-mail: mail@bhv.ru
Internet: www.bhv.ru

Тел.: (812) 251-42-44
Тел./факс: (812) 320-01-79

ISBN 978-5-9775-0664-9