So You Want To Learn to Programm?

James M. Reneau, M.S.
Assistant Professor
Shawnee State University
Portsmouth Ohaio USA

http://www.basicbook.org

James M. Reneau
P.O. Box 278
Rassell, Kentucky 41169-2078 USA

Book Version: 20101113a
For BASIC-256 Version 0.9.6.48 or later

Хотите научиться программировать?

Джеймс М. Рено, М.С.
Ассистент-профессор
государственного университета Шауни
Портсмут, Огайо, США

http://www.basicbook.org

James M. Reneau
P.O. Box 278
Rassell, Kentucky 41169-2078 USA

Версия книги: 20101113a
Для BASIC-256 версии 0.9.6.48 или старше

Хотите научиться программировать?
James M. Reneau, M.S. - jim@renejm.com

Createspace Print ISBN: 978-1456329044
Перевод на русский С. Ирюпин, В. Чёрный

ISBN 978-5-905167-06-5

Москва 2011 г.

Оглавление

Предисловие

На BASIC-256 я натолкнулся совсем не случайно. Когда мы (команда Ростовского LUG[1]), начали помогать школам внедрять свободное ПО, а точнее, школьную сборку от ALT Linux, я целенаправленно искал, какую свободную программу предложить школьникам для освоения основ алгоритмизации и программирования, и обнаружил BASIC-256, тогда ещё версии 0.9.2. Понравилось, что в программе, кроме редактора текста, есть ещё и дополнительные окна для ввода-вывода текста и вывода графики. Порадовало и наличие сборок как для Linux, так и для Windows. Забегая вперёд, скажу, что нынешняя версия работает со спрайтами, звуком и даже способна проговаривать текст (русский и английский в Linux, и только английский в Windows).

Посетив http://sourceforge.net/projects/kidbasic/, я увидел, что уже существует сборка 0.9.5, перекинулся парой писем с автором, Джимом Рено, потом пообщался с мейнтейнером[2] этого пакета в ALT Linux, Дамиром Шайхутдиновым. В итоге, сам стал мейнтейнером BASIC-256 и даже, в дальнейшем, внёс небольшой вклад в его дизайн и код. Перевёл встроенную справку, начал переводить сайт http://basic256.org/. Тем временем Джим взялся активно развивать программу, добавляя новые функции и начал писать книгу по BASIC-256. Ту самую, которую вы сейчас держите в руках. Переводил я её вместе с Владимиром Чёрным, руководителем отдела образовательных проектов ООО «ALT Linux». По-моему, книга получилась неплохая — материал изложен ясно и просто, без лишнего теоретизирования, много примеров программ.

Хочу сказать спасибо Джиму Рено, — за классную программу, открытость и простоту в общении. Дамиру Шайхутдинову, — за внимательность и терпение, с которым он выслушивал дурацкие вопросы начинающего мейнтейнера, за постоянную готовность помочь. Владимиру Чёрному — за общение, сотрудничество и терпение, когда я затягивал отправку очередной порции переведённого материала.

Всем нашим читателям желаю успехов в освоении BASIC. Что бы там не говорили скрептики и нытики, — это простой и удобный язык для всех, кто желает познакомиться с программированием.

Сергей Ирюпин
Ростов-на-Дону

[1] LUG — аббревиатура от Linux User Group, — группа пользователей Линукс *(Прим. редактора)*.

[2] В сообществе **С**вободного **П**рограммного **О**беспечения мейнтейнером (от анг. термина mainteiner — поддерживать, сопровождать) называют человека, который собирает бинарный пакет из исходников для конкретного репозитория (хранилища пакетов), следит за обновлениями и изменениями, своевременно внося изменения в бинарный пакет. Кроме того, мейнтейнер подписывает пакет цифровой подписью чтобы можно было убедиться в подлинности сборки *(Прим. редактора)*.

Благодарности

Приношу огромную благодарность всем, кто работал над проектом BASIC-256 на Sourceforge[3], особенно Яну Ларсену (Ian Larsen aka[4]: DrBlast) за создание языка BASIC-256 и самобытные взгляды.

Я также считаю необходимым поблагодарить ребят, участников весенней школы программистов 2010 года из средней школы Рассела (Russel Midnight School) и Джулию Мур (Julia Moore). Приношу также благодарность моим последователям Сергею Ирюпину и Джоелю Кану (Joel Khan).

Посвящения

Моей жене Ненси и дочери Анне.

Предисловие к русскому изданию

Русское издание этой книги было заботливо и усердно подготовлено Сергеем Ирюпиным и Владимиром Чёрным. Без их преданности сообществу Свободного Программного Обеспечения и особенно этому проекту это было бы невозможно.

Самое большое моё желание, чтобы ты, мой читатель, получил удовольствие от изучения программирования. Не будь просто пользователем технологии — стань её властелином!

James Reneau

[3] http://sourceforge.net/projects/kidbasic/

[4] aka — сокращение от **a**lso **k**nown **a**s, означающее «также известен как» (*прим. переводчика*).

Глава 1

Знакомство с BASIC-256 — скажи «Привет»[1]

В этой главе вы познакомитесь со средой BASIC-256, на примере операторов print и say. Вы увидите разницу между командами, которые вы отдаёте компьютеру, а также разницу между текстовыми строками и числами, которые будут использованы программой. Мы также исследуем простую математику для того, чтобы показать, насколько умен ваш компьютер. Наконец, вы узнаете, что такое синтаксическая ошибка и как её можно исправить.

1.1 Окно BASIC-256

Окно BASIC-256 разделено на 5 секций: строка меню, панель инструментов, область текста программы, окно ввода-вывода текста, окно вывода графики (см. рис. 1.1).

1.1.1 Верхнее меню

Верхнее меню содержит несколько различных раскрывающихся меню. Она включает в себя: «Файл», «Правка», «Просмотр», «Старт», «Справка». Меню «Файл» позволит вам сохранять и загружать сохранённые ранее программы, печатать и выходить из BASIC-256. Меню «Правка» позволяет вырезать, копировать, вставлять текст и изображения из программы, текстового и графического окна. Меню «Просмотр» позволит просмотреть или скрыть различные окна BASIC-256. Меню «Старт» позволит выполнять и отлаживать вашу программу. Меню «Справка» покажет окно с информацией о BASIC-256, также какую версию вы сейчас используете.

1.1.2 Панель инструментов

Большинство пунктов меню, которые вы будете использовать, доступны на панели инструментов.

 Новый — начать новую программу.

 Открыть — загрузить сохранённую программу.

 Сохранить — сохраняет программу на диск или USB устройство.

[1] В оригинале «say Hello». Во всех учебниках программирования принято начинать изучение языка с вывода на экран фразы «Hello World», автор обыгрывает этот момент в заголовке. (прим. переводчика)

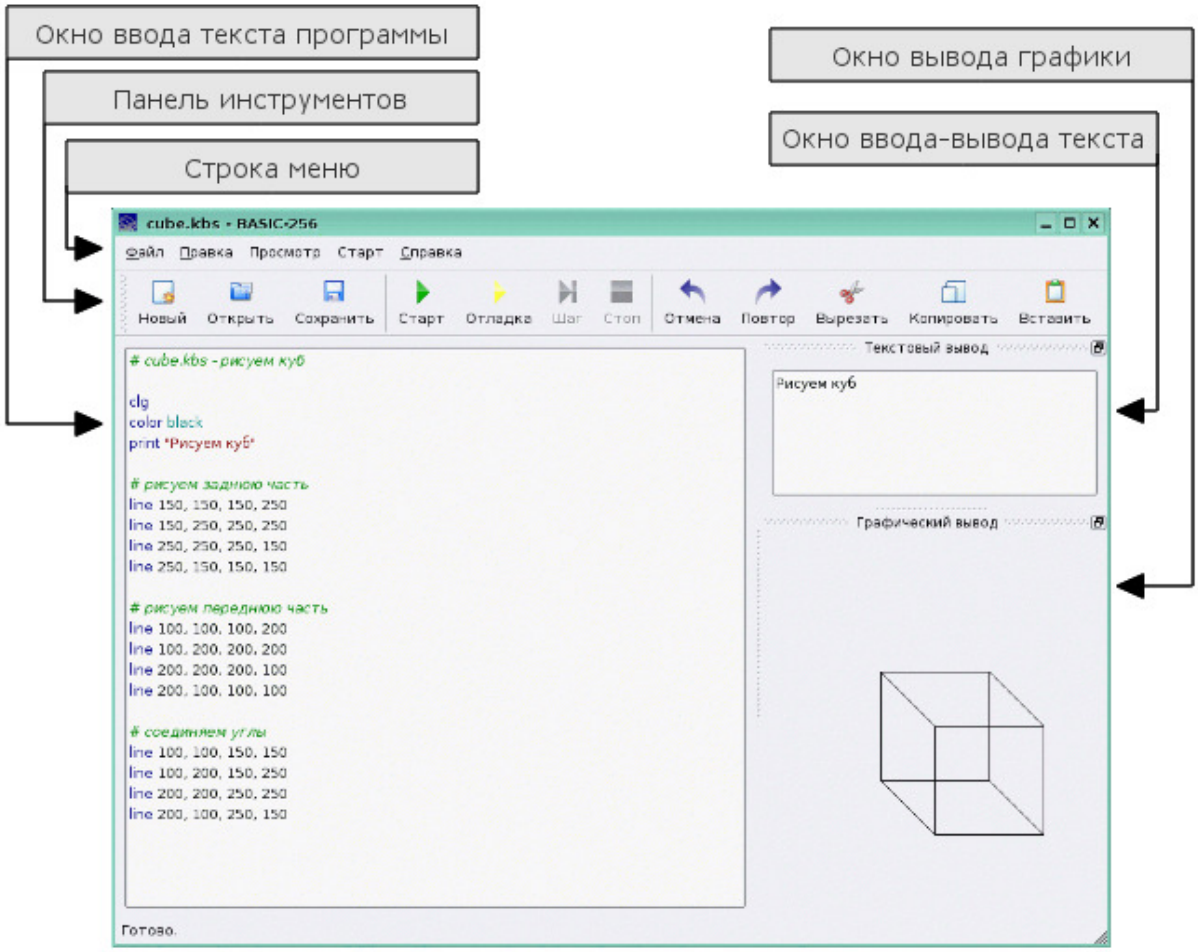


Рис. 1.1. Экран BASIC-256


Старт — выполняет текущую программу.

Отладка — начинает построчное выполнение программы.


Шаг — при отладке — перейти на новую строку.


Стоп — прекращает выполнение текущую программу.


Отмена — отменяет последнее изменение в программе.


Повтор — возвращает последнее отменённое изменение.


Вырезать — переносит выделенный текст в буфер обмена.


Копировать — помещает копию выделенного текста в буфер обмена.


Вставить — вставляет текст из буфера обмена в необходимое место.

1.1.3 Окно текста программы

Текст программы состоит из инструкций, которые указывают компьютеру, что и как нужно делать. Вы будете набирать текст программ, изменять и исправлять их код именно в этом окне, а также загружать сюда сохранённые ранее программы.

1.1.4 Окно ввода-вывода текст

Эта окно будет отображать вывод текста из ваших программ. Это могут быть и слова и числа. Если программа захочет задать вам вопрос, то вопрос (а также и то, что вы напечатаете в ответ) тоже появится здесь.

1.1.5 Окно вывода графики

BASIC-256 — это язык, умеющий управлять графикой (в дальнейшем вы это увидите). Картинки, формы и образы, созданные вами, будут отображаться в этом окне.

1.2 Ваша первая программа — оператор say

Давайте создадим компьютерную программу и посмотрим, поприветствует ли нас BASIC-256. В окне текста программы напечатайте следующую команду в одну строку:

```
say "Hello! Привет!"
```

Программа 1. Скажи привет.

После того, как вы наберёте эту команду, щёлкните мышью по кнопке  «Старт» на панели инструментов. BASIC-256 поздоровался с вами через динамики компьютера?[2]

say *выражение*
Оператор **say** используется для того, чтобы BASIC-256 прочитал выражение вслух, в компьютерные динамики.

"Hello! Привет!"
BASIC-256 рассматривает буквы, цифры и знаки препинания, которые находятся внутри двойных кавычек, как единый блок. Этот блок называется строкой.

 «Старт» на панели инструментов или «Старт» в меню Вы должны сказать BASIC-256, когда вы хотите приступить к выполнению программы. Автоматически он не узнает, что вы закончили вводить код программы. Запустить программу на выполнение можно либо нажав на кнопку «Старт» на панели инструментов, либо выбрав пункт «Старт» в выпадающем меню.

[2]Оператор **say** появился в версии **0.9.4**. Если вы используете BASIC-256 в ALT Linux, для работы **say** необходимо установить пакет **espeak** (консольной командой от root: **apt-get install** espeak или с помощью Synaptic). В противном случае вы ничего не услышите. При работе в Windows™английские слова будут проговариваться без необходимости установки дополнительных программ, русские — нет. (*прим. разработчика*)

Для того, чтобы полностью удалить программу, в которой вы работаете и начать новую, мы используем кнопку  «Новый» на панели инструментов. Нажатие этой кнопки вызовет появление следующего диалогового окна (см. рис. 1.2):

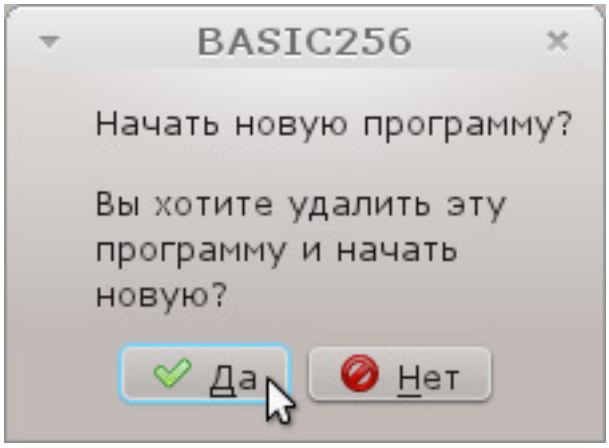


Рис. 1.2. BASIC-256 — окно начала новой программы

Если вы действительно хотите удалить программу, нажмите кнопку «Да»(«Yes»). Если вы случайно нажали «Новый» и не хотите начинать другую программу, нажмите кнопку «Нет»(«No»).

Кнопка  «Новый» на панели инструментов или «Файл» ⟶ «Новый» в меню
Команда «Новый» сообщает BASIC-256 о том, что вы хотите удалить текущую программу и начать новую. Если вы не сохранили программу (Глава 2), то все изменения, сделанные в программе, не будут сохранены.

Попробуйте несколько разных программ, используя оператор say. Поприветствуйте своего лучшего друга, попросите компьютер назвать ваш любимый цвет, в общем — развлекитесь.

Оператор **say** также может называть числа. Попробуйте следующую программу:

```
say 123456789
```

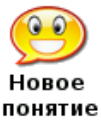
Программа 2. Назови число.

После того, как вы наберёте эту команду, щёлкните мышью по кнопке  «Старт» на панели инструментов. Сказал ли BASIC-256 то, что вы хотели?[3]

[3]Если вы работаете в ALT Linux и пакет **espeak** установлен, вы услышите: «сто двадцать три миллиона четыреста пятьдесят шесть тысяч семьсот восемьдесят девять», — на русском языке. При работе с Windows™(без установки дополнительных программ) цифры будут называться только на английском языке. (*прим. разработчика*)

Таблица 1.1. Основные математические операции

Оператор	Операция
+	Сложение *выражение1* + *выражение2*
−	Вычитание *выражение1* − *выражение2*
*	Умножение *выражение1* * *выражение2*
/	Деление *выражение1* / *выражение2*



числа BASIC-256 позволяет вводить числа в десятичной форме. Не используйте запятые при вводе больших чисел. Если вам нужно число меньше нуля, поставьте перед ним знак минус. Например: 1.56, 23456, -6.45 и 5.

1.3 BASIC-256 действительно хорошо работает с числами — простая арифметика

Мозг компьютера (который называется Центральным Процессором или кратко — ЦП) работает только с числами. Всё, что он делает, начиная с графики, звука и заканчивая всем остальным, делается при помощи умелого обращения с числами.

Четыре основные действия: сложение, вычитание, умножение и деление приводятся в исполнение, используя операторы, показанные в Таблице 1.1

Попробуйте эту программу и послушайте разговаривающий супер-калькулятор.

```
say 12 * (2 + 10)
```

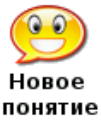
Программа 3. Скажи ответ.

Компьютер должен сказать вам: «144»

```
say 5 / 2
```

Программа 4. Скажи другой ответ.

Сказал ли компьютер «2.5»?



+ − */()
Четыре основных математических оператора: сложение (+), вычитание (−), деление (/) и умножение (*) работают с числами для выполнения вычислений. Числа должны быть по обе стороны этих операторов. Вы также можете использовать скобки () для группировки операций.

Попробуйте написать несколько коротких программ, используя оператор say, а также четыре основные математические операции. Обязательно используйте все четыре операции.

1.4 Другое использование + (конкатенация)

Оператор + также соединяет строки. Эта операция называется конкатенация. Конкатенация добавляет строку к строке, как вагоны в составе поезда, чтобы сделать её длиннее.

Давайте попробуем:

```
say "Привет " + "Сергей."
```

Программа 5. Скажи «Привет, Сергей».

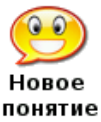
Компьютер должен поприветствовать Сергея.

Попробуем другую программу.

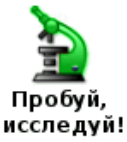
```
say 2 + " жды два - четыре"
```

Программа 6. Сказать «Дважды два — четыре».

Оператор + в последнем примере был использован для объединения, потому что второй операнд является строкой и компьютер не знает как выполнить математическое действие со строкой (поэтому — «конкатенация»).



+ (конкатенация)
Другое применение знака плюс (+), — сказать компьютеру выполнить конкатенацию (объединение) строк. Если одно или оба операнда — строки, будет выполнена конкатенация; если оба операнда — числа, произойдёт их сложение.



Попробуйте несколько разных программ, используя команду say и оператор + (конкатенации). Соединяйте строки и числа вместе с другими строками и числами.

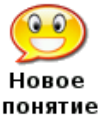
1.5 Окно ввода-вывода текста — оператор print

Программы, использующие **say**, могут быть очень полезными и развлекающими, но часто бывает необходимо написать информацию (слова и числа) на экране так, чтобы их можно было прочесть. Эту задачу выполняет оператор **print**. В окне для ввода текста программы наберите программу из двух строк:

```
print "Привет"
print "всем"
```

Программа 7. Напечатать «Привет всем».

После того, как вы наберёте текст этой программы, щёлкните мышкой по кнопке  «Старт» на панели инструментов. В окне для ввода-вывода текста появятся слова: "Привет"на первой строке и "всем— на второй.

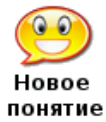


print *выражение* **print** *выражение;* Оператор **print** используется, чтобы отображать текст и числа в окне ввода-вывода текста BASIC-256. Напечатав что-либо, print переходит на новую строку, но можно напечатать несколько знаков в одной и той же строке, используя ; (точку с запятой) в конце выражения.

Оператор **print** по умолчанию действует так, что последующий текст оказывается на новой строке. Если вы используете ; (точку с запятой) в конце выводимого выражения, то последующие выводимые знаки останутся на этой же строке.


```
cls
print "Привет ";
print "вам, ";
print "мои друзья."
```

Программа 8. Несколько print выводят в одну строку.



cls
Оператор cls стирает всю информацию в окне ввода-вывода текста[4].

Попробуйте разные программы, используя оператор **print**. Используйте слова, числа, математику и конкатенацию.

1.6 Что такое «Ошибка синтаксиса»?

Программисты — обычные люди и иногда совершают ошибки. «Ошибка синтаксиса» — один из видов ошибок, с которыми мы можем столкнуться. Такая ошибка возникает, когда BASIC-256 не понимает программу, которую вы набрали. Обычно синтаксические ошибки вызваны неправильным написанием операторов (команд), пропущенными запятыми, ненужными пробелами, незакрытыми кавычками, непарными скобками. BASIC-256 сообщит вам, в какой строке находится ошибка и даже попытается сказать, в какой позиции строки её можно найти.

[4]Каждый раз, при запуске программы для исполнения, окно ввода-вывода текста очищается автоматически.(*прим. разработчика*)

Глава 2

Рисуем основные фигуры

В этой главе мы начнём работать с графикой. Вы научитесь рисовать прямоугольники, окружности, линии и точки разных цветов. Программы будут становиться всё более и более сложными, поэтому вы также узнаете как сохранять программы и как загружать их, чтобы снова запустить на выполнение и отредактировать.

2.1 Рисуем прямоугольники и окружности

Давайте начнём с написания графической программы для нашей любимой спортивной команды «Серые пятнышки». Её цвета — голубой и серый.

```
1 # c2_greyspots.kbs
2 # программа для нашей команды - Серые пятнышки
3 clg
4 color blue
5 rect 0,0,300,300
6 color grey
7 circle 149,149,100
8 say "Серые пятнышки, вперёд!"
```

Программа 9. Серые пятнышки.

Обрати внимание!

Предупреждение: с этого момента листинги программ будут идти с пронумерованными строками. Не печатайте эти номера строк, когда будете вводить программу.

Давайте изучим каждую строку приведённой выше программы. Первая строка называется ремарка или оператор комментария. Комментарий — это место, где программист оставляет свои пометки в компьютером коде, игнорируемые системой. Такие пометки хороши для того, чтобы описать что делает тот или иной фрагмент кода, название программы, почему мы написали программу, или кто программист.

Новое понятие

\#
rem

Оператор # или **rem** называется комментарием. Он позволяет программисту оставлять свои заметки (зачем, как и что работает) в тексте программы. Когда компьютер видит # или **rem**, он игнорирует весь оставшийся текст в строке.

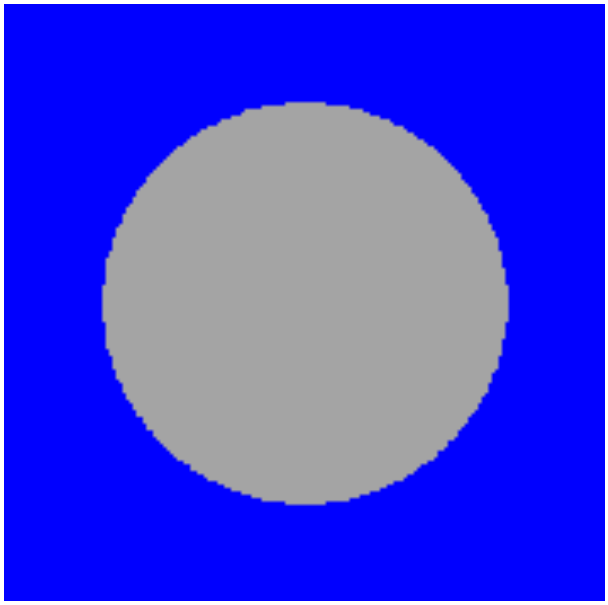


Рис. 2.1. Вывод программы 9. Серые пятнышки.

В второй строке вы видите оператор **clg**. Он очень похож на оператор **cls** из первой главы, за исключением того, что **clg** очищает окно, куда выводится графика.

clg

Оператор **clg** очищает окно вывода графики для того, чтобы у нас было чистое место для рисования.

В третьей строке мы видим оператор **color**. Он сообщает BASIC-256 какой цвет нужно использовать для следующего действия рисования. Вы можете устанавливать цвет, указав одно из 18 стандартных названий (см. Таблицу 2.1), или определив один из 16 миллионов вариантов, смешивая основные цвета (красный (**R**ed), зелёный (**G**reen) и синий (**B**lue)) разной интенсивности.

Если вы используете цифровой способ определения цвета, учтите, что числа должны быть в диапазоне от 0 до 255. Нуль (0) говорит об отсутствии яркости у выбранного цвета, а 255 означает максимальную яркость. Ярко-белый цвет представлен числами 255,255,255 (все цвета максимальной яркости), чёрный — как 0,0,0 (нулевая яркость всех цветов). Такое числовое представление известно как «**RGB**-триплет». Таблица 2.1 показывает имена некоторых цветов и их числовые значения.

color *имя_цвета*
color *красный, зелёный, голубой*
color *RGB_число*

вместо **color** можно также использовать **colour**[1].

Оператор **color** позволяет установить цвет, которым вы будете рисовать дальше. Вы можете использовать **color** с именем цвета (**black**, **white**, **red**, **darkred**, **green**, **darkgreen**, **blue**, **darkblue**, **cyan**, **darkcyan**, **purple**, **darkpurple**, **yellow**, **darkyellow**, **orange**, **darkorange**, **grey**/**gray**, **darkgrey**/**darkgray**), с тремя цифрами (0-255), описывающими интенсивность красного, зелёного и голубого цветов (R,G,B) или одним значением, полученным в результате вычисления выражения: **красный**×256^2+**зелёный**×256+**голубой**.

Таблица 2.1: Числовые значения цветов

Образец цвета	код	название цвета
	black (0,0,0)	чёрный
	white (248,248,248)	белый
	red (255,0,0)	красный
	darkred (128,0,0)	тёмно-красный
	green (0,255,0)	зелёный
	darkgreen (0,128,0)	тёмно-зелёный
	blue (0,0,255)	синий
	darkblue (0,0,128)	темносиний
	cyan (0,255,255)	голубой
	darkcyan (0,128,128)	тёмно-голубой
	purple (255,0,255)	пурпурный
	darkpurple (128,0,128)	тёмно-пурпурный
	yellow (255,255,0)	жёлтый
	darkyellow (128,128,0)	тёмно-жёлтый
	orange (255,102,0)	оранжевый

[1]Это сделано из-за того, что в разных англоязычных странах разное написание слова «цвет» (*прим. переводчика*)

Таблица 2.1 — продолжение

Образец цвета	код	название цвета
	darkorange (170,51,0)	тёмно-оранжевый
	gray или grey (164,164,164)	серый
	darkgray или darkgrey (128,128,128)	тёмно-серый
	clear (-1)	прозрачный

По умолчанию окно для вывода графики имеет размер 300 пикселей в ширину (x) и 300 пикселей в высоту (y). Пиксель — это самая маленькая точка, которая может быть изображена на мониторе вашего компьютера. Координаты верхнего левого угла — (0,0), а правого нижнего — (299,299). Каждый пиксель может быть представлен двумя числами, первое (x) показывает смещение вправо, второе (y) — смещение вниз. Такой способ маркировки точек известен в математике как Декартова прямоугольная система координат.

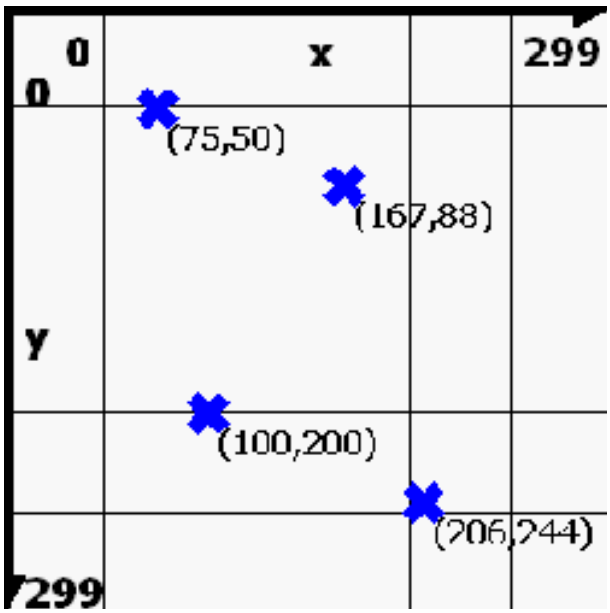


Рис. 2.2. Декартова система координат окна вывода графики

Следующий оператор (строка 5) — **rect**. Он позволяет рисовать прямоугольники. **Rect** использует четыре цифры, разделённые запятыми: (1) координата верхнего левого угла прямоугольника по оси x, (2) координата этого угла по оси y, (3) ширина, (4) высота. Все четыре цифры задаются в пикселях (размер самой маленькой точки, которая может быть изображена на экране).

Вы можете видеть, что прямоугольник в программе начинается в верхнем левом углу и далее заполняется в окне вывода графики.

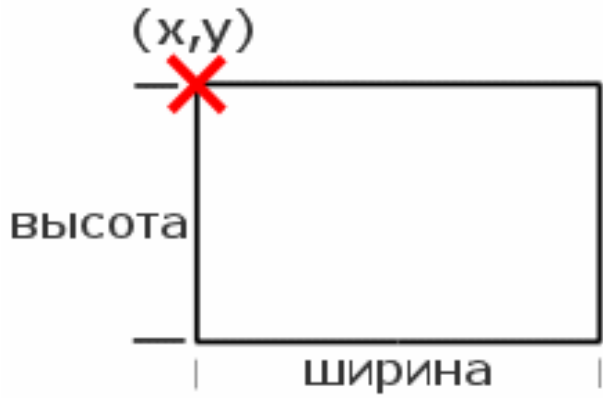
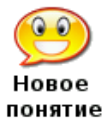


Рис. 2.3. Прямоугольник



rect *x, y, ширина, высота*
Оператор **rect** использует текущий цвет и рисует прямоугольник в окне вывода графики. Верхний левый угол прямоугольника задан двумя первыми числами, а ширина и высота — двумя другими.

Строка 7 содержит оператор **circle**, который рисует окружность. Он использует три числовых аргумента, первые два — это декартовы координаты центра окружности, а третий — её радиус (в пикселях).

Рис. 2.4. Окружность

circle *x, y, радиус*
Оператор **circle** использует текущий цвет и рисует заполненную этим цветом окружность с центром в точке (x,y) и заданным радиусом.

Можете ли вы, используя **color**, **rect** и **circle**, создать эмблему для вашей школы или любимой спортивной команды?

Вот несколько примеров простых программ, которые используют новые операторы (**clg**, **color**, **rect** и **circle**). Наберите эти программы и попробуйте их изменить. Сделайте «хмурое лицо», «лицо пришельца» или лицо кого-нибудь, кого вы знаете.

```
 1 # c2_rectanglesmile.kbs
 2
 3 # очищаем экран
 4 clg
 5
 6 # рисуем лицо
 7 color yellow
 8 rect 0,0,299,299
 9
10 # рисуем рот
11 color black
12 rect 100,200,100,25
13
14 # рисуем глаза
15 color black
16 rect 75,75,50,50
17 rect 175,75,50,50
18
19 say "Привет!"
```

Программа 10. Лицо, составленное из прямоугольников.

```
 1 # c2_circlesmile.kbs
 2
 3 # очищаем экран
 4 clg
 5 color white
 6 rect 0,0,300,300
 7
 8 # рисуем лицо
 9 color yellow
10 circle 150,150,150
11
12 # рисуем рот
13 color black
14 circle 150,200,70
15 color yellow
16 circle 150,150,70
17
18 # рисуем глаза
19 color black
20 circle 100,100,30
```

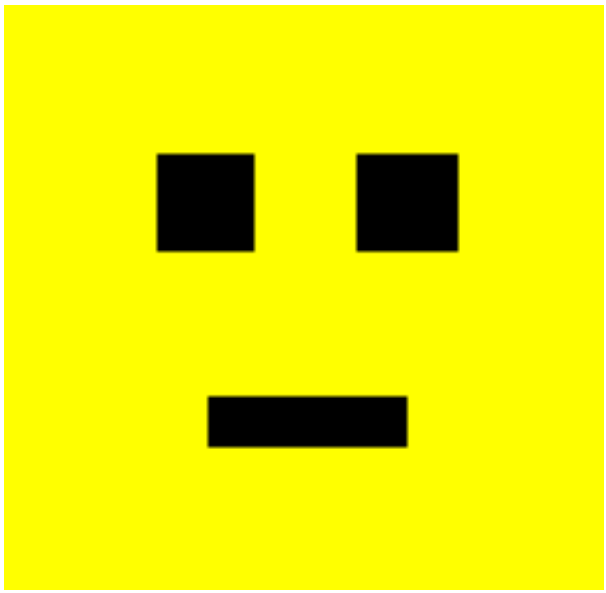
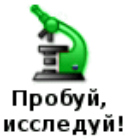


Рис. 2.5. Вывод программы 10. Лицо, составленное из прямоугольников.

```
21 circle 200,100,30
```

Программа 11. Улыбающееся лицо, составленное из окружностей.

Рис. 2.6. Вывод программы 11. Улыбающееся лицо, составленное из окружностей.



Попробуйте скомбинировать прямоугольники и окружности, чтобы создать графическое изображение своего собственного лица.

2.2 Сохранение вашей программы и её повторная загрузка

Теперь, когда программы становятся всё более сложными, вы, видимо, захотите сохранить их для того, чтобы в будущем загрузить снова.

Вы можете сохранить программу, нажав кнопку  «Сохранить» на панели инструментов или выбрав пункт «Сохранить» в выпадающем меню «Файл». Появится диалоговое окно с запросом имени файла, если речь идёт о новой программе, или же окно, в котором вас попросят подтвердить запись сделанных изменений (перезаписать старый файл).

Если вы не хотите стирать старую версию программы, используйте пункт «Сохранить как» в меню «Файл», чтобы записать копию под другим именем.

Чтобы открыть ранее сохранённую программу, используйте кнопку  «Открыть» на панели инструментов, либо пункт «Открыть» в выпадающем меню «Файл».

2.3 Рисуем линии

Следующий оператор рисования — это **line**. Он рисует линию шириной один пиксель от одной точки к другой, используя текущий цвет. Программа 12 показывает пример использования оператора **line**.

```
1 #c2_triangle.kbs - рисуем треугольник
2
3 clg
4 color black
5
6 line 150, 100, 100, 200
7 line 100, 200, 200, 200
8 line 200, 200, 150, 100
```

Программа 12. Рисуем треугольник.

line *старт_x, старт_y, финиш_x, финиш_y*
Рисует линию шириной один пиксель от стартовой точки до конечной, используя текущий цвет.

Используйте лист из альбома для черчения, чтобы изобразить другие фигуры, а затем напишите программу для их рисования. Попробуйте нарисовать прямоугольный треугольник, пятиугольник, звезду и другие фигуры.

Следующая программа — пример, что вы можете сделать, используя только одни линии. Она рисует куб.

```
1 # c2_cube.kbs - рисуем куб
2
3 clg
4 color black
5
6 # рисуем заднюю часть
7 line 150, 150, 150, 250
8 line 150, 250, 250, 250
9 line 250, 250, 250, 150
```

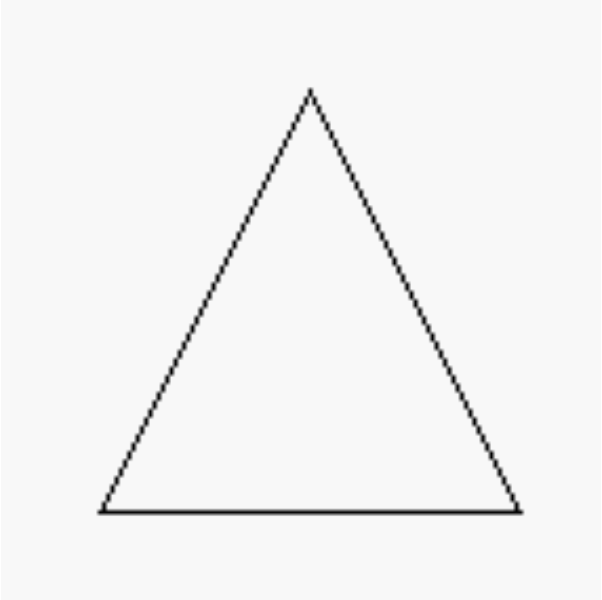


Рис. 2.7. Вывод программы 12. Рисуем треугольник.

```
10 line 250, 150, 150, 150
11
12 # рисуем переднюю часть
13 line 100, 100, 100, 200
14 line 100, 200, 200, 200
15 line 200, 200, 200, 100
16 line 200, 100, 100, 100
17
18 # соединяем углы
19 line 100, 100, 150, 150
20 line 100, 200, 150, 250
21 line 200, 200, 250, 250
22 line 200, 100, 250, 150
```

Программа 13. Рисуем куб.

2.4 Рисуем отдельные точки на экране

Последний графический оператор, рассматриваемый в этой главе — **plot**. Оператор **plot** закрашивает текущим цветом одну точку (пиксель) на экране. Для большинства из нас эти точки настолько малы, что их трудно увидеть. Позднее мы напишем программы, которые будут рисовать группы точек для того, чтобы создать очень детальное изображение.

```
1 # c2_plot.kbs - используем plot для рисования точек
2
3 clg
4
5 color red
6 plot 99,100
7 plot 100,99
```

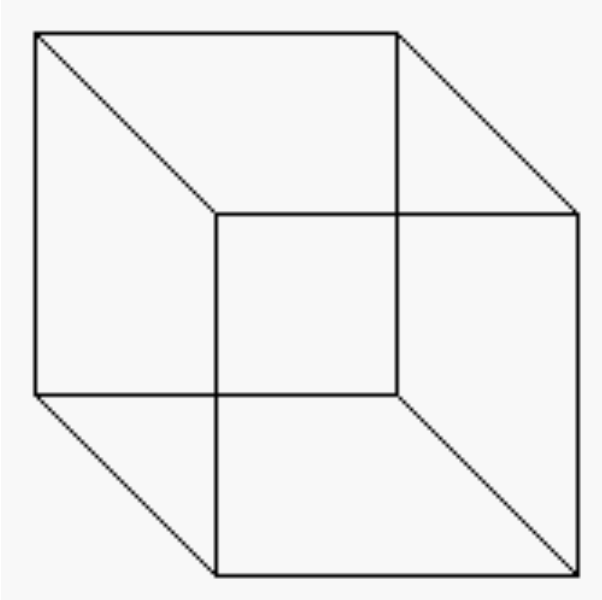



Рис. 2.8. Вывод программы 13. Рисуем куб.

```
 8 plot 100,100
 9 plot 100,101
10 plot 101,100
11
12 color darkgreen
13 plot 200,200
```

Программа 14. Используем **plot** для рисования точек.

Новое понятие

plot *x, y*
Закрашивает один пиксель (точку на экране) текущим цветом.

Большая программа

В конце каждой главы вы найдёте одну или несколько «Больших программ», для того, чтобы вы могли взглянуть на них, набрать текст и поэкспериментировать с ними. Эти программы будут содержать только то, что вы изучили до сих пор в этой книге.

Приведённая ниже программа изображает лицо и делает его «говорящим». Перед тем, как программа скажет очередное слово, нижняя часть лица будет менять форму рта, перерисовывая его. Это создаёт эффект примитивной анимации и делает лицо более забавным.

```
1 # c2_talkingface.kbs
2 # рисует лицо с глазами
3 color yellow
4 rect 0,0,300,300
5 color black
6 rect 75,75,50,50
7 rect 175,75,50,50
8
```

Рис. 2.9. Вывод программы 14. Используем **plot** для рисования точек (обведено для привлечения внимания).

```
 9 # стирает старый рот
10 color yellow
11 rect 0,150,300,150
12 # рисует новый рот
13 color black
14 rect 125,175,50,100
15 # говорит слово
16 say "я"
17
18 color yellow
19 rect 0,150,300,150
20 color black
21 rect 100,200,100,50
22 say "очень"
23
24 color yellow
25 rect 0,150,300,150
26 color black
27 rect 125,175,50,100
28 say "рад"
29
30 color yellow
31 rect 0,150,300,150
32 color black
33 rect 125,200,50,50
34 say "что"
35
36 color yellow
37 rect 0,150,300,150
```



```
38 color black
39 rect 100,200,100,50
40 say "ты"
41
42 color yellow
43 rect 0,150,300,150
44 color black
45 rect 125,200,50,50
46 say "мой"
47
48 # рисует новое лицо с круглой улыбкой
49 color yellow
50 rect 0,0,300,300
51 color black
52 circle 150,175,100
53 color yellow
54 circle 150,150,100
55 color black
56 rect 75,75,50,50
57 rect 175,75,50,50
58 say "друг"
```

Программа 15. Большая программа — разговаривающее лицо.

Рис. 2.10. Вывод программы 15. Большая программа — разговаривающее лицо.

Глава 3

Звуки и музыка

Теперь, когда у нас есть цвет и графика, давайте добавим звук и немного музыки. В этой главе вы узнаете основные понятия акустики и музыкальной нотации, познакомитесь с понятием числовой переменной, научитесь переводить мелодию в частоты и длительности для того, чтобы компьютер синтезировал музыку.

3.1 Основы звука — всё, что вам нужно знать о звуках

Звук возникает благодаря колебаниям воздуха, воздействующим на барабанную перепонку. Эти колебания называются звуковыми волнами. Когда воздух колеблется быстро, вы слышите высокую ноту, а когда воздух колеблется медленно — низкую ноту. Уровень колебаний называют частотой.

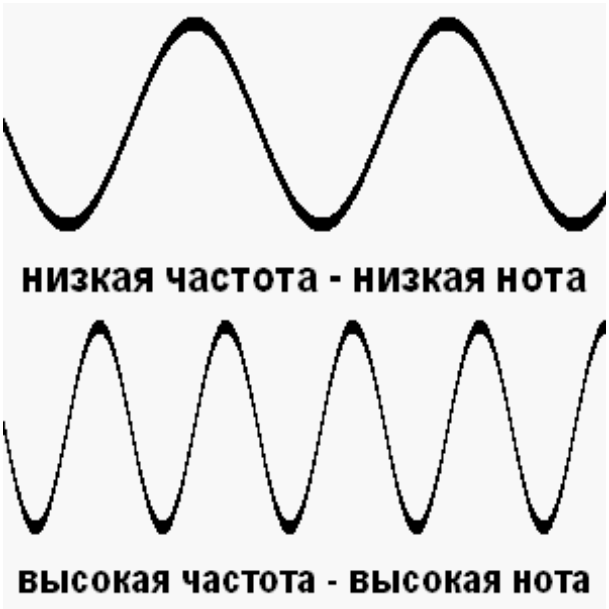


Рис. 3.1. Звуковые волны

Частота измеряется в единицах, которые называются герц (Гц). Она показывает, как много циклов колебаний (вверх и вниз) в секунду совершает звуковая волна. Обычный

человек может слышать очень низкие звуки частотой 20 Гц и очень высокие звуки частотой 20000 Гц. BASIC-256 может воспроизводить звуки в диапазоне от 50 до 7000 Гц.

Другое свойство звука — его длительность. Компьютеры работают очень быстро и позволяют измерять время с точностью до миллисекунд (мс). Миллисекунда составляет 1/1000 (одну тысячную) долю секунды.

Давайте создадим некоторые звуки.

```
1 # c3_sounds.kbs
2 sound 233,1000
3 sound 466,500
4 sound 233,1000
```

Программа 16. Сыграем три отдельные ноты

Возможно, вы услышали щёлкающий звук в колонках между звуками, сыгранными в этом примере. Это вызвано тем, что компьютер создаёт звук и ему необходимо остановиться и «подумать» примерно миллисекунду. Оператор **sound** может быть также записан с указанием списка частот и длительностей, чтобы сгладить переход от одной ноты к другой.

```
1 # c3_soundslist.kbs
2 sound {233,1000,466,500,233,1000}
```

Программа 17. Список звуков

Эта вторая звуковая программа воспроизводит те же самые три тона той же длительности, но компьютер создаёт и проигрывает все звуки сразу, делая их более гладкими.

sound *частота, длительность*
sound {*частота1, длительность1, частота2, длительность2, ...*}
sound *массив_чисел*

Основная форма оператора **sound** использует два аргумента: (1) частоту звука в Гц (колебаний в секунду) и (2) длительность звука в миллисекундах (мс). Вторая форма **sound** позволяет указывать несколько пар значений (частота, длительность) в списке, заключённом в фигурные скобки {}. Третья форма **sound** использует массив, который содержит частоты и длительности. Речь о массивах пойдёт в Главе 13.

Как же BASIC-256 воспроизводит мелодию? Первое, что мы должны сделать — преобразовать ноты на нотном стане в частоты. На рисунке 3.2 показаны две октавы нот, их названия и приблизительная частота, которая их создаёт. В музыке есть ещё особое понятие — пауза. Пауза означает — «не воспроизводить музыку в определённый интервал времени». Если вы используете список звуков, то можете вставить паузу, указав частоту нуль (0) и необходимое время паузы.

Возьмите небольшой музыкальный фрагмент, а затем посмотрите значения частоты для каждой ноты. Почему бы нам не попросить компьютер сыграть «Атака!» (смотрите рисунок 3.3). Вы наверное заметили, что нота «соль» (G) второй октавы находится выше нотного стана. Если нота располагается не на нотном стане, её частоту можно удвоить, чтобы сделать выше, или уменьшить наполовину, чтобы сделать ниже. Получается та же самая нота, только на октаву выше или ниже[2].

[1] Латинские названия нот можно посмотреть в википедии (http://ru.wikipedia.org/wiki/Нотные_знаки). За эталон частоты ноты берётся нота ля (A) первой октавы, частота которой должна быть равной 440 Гц, что и видно на рисунке (*прим. редактора*).

[2] Отношение частот одинаковых нот из соседних октав равно двум или 1/2. (*прим. редактора*).

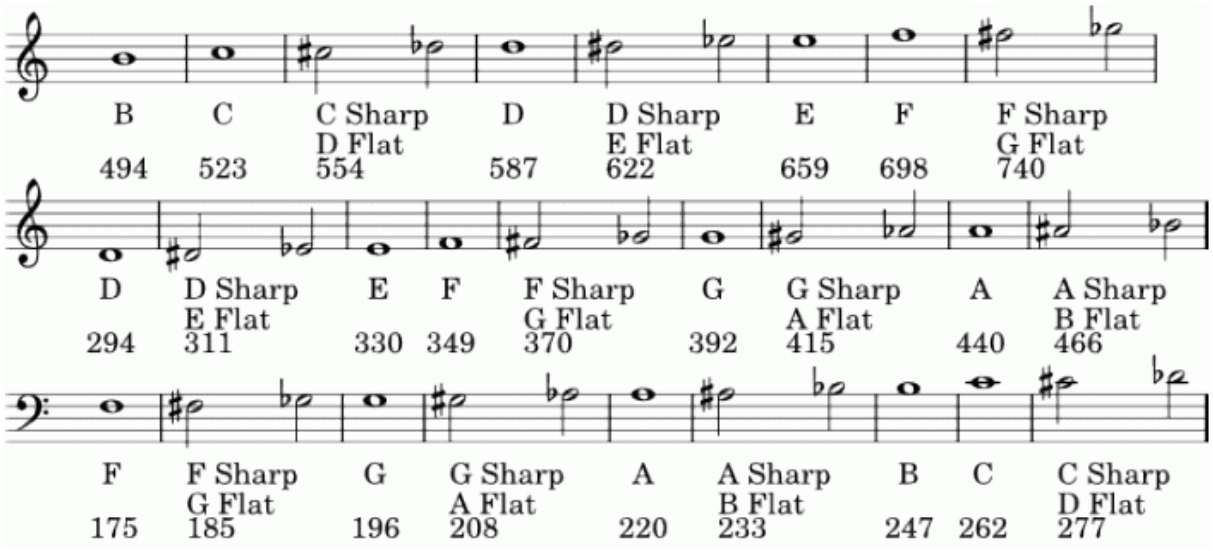


Рис. 3.2. Ноты[1]

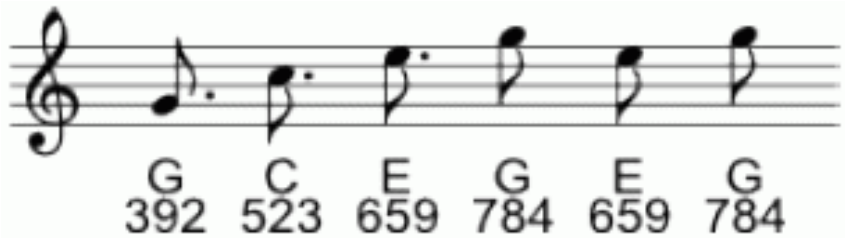


Рис. 3.3. Атака!

Теперь, когда у нас есть частоты, нам нужны ещё длительности звучания для каждой из нот. Таблица 3.1 показывает наиболее распространённые длительности нот и пауз, насколько они продолжительны в сравнении друг с другом, и несколько типовых длительностей.

Продолжительность в миллисекундах (мс) можно вычислить, если вы знаете скорость музыки в ударах в минуту (BPM — beats per minute), используя формулу 1.

$$\textit{Длительность ноты} = 1000 \times 60 / BPM \times \textit{Относительная длина}$$

Формула 1. Вычисление длительности ноты

Таблица 3.1: Ноты и обычная длительность

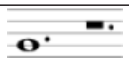
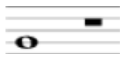
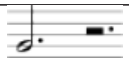
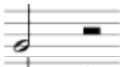
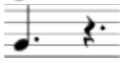
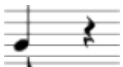
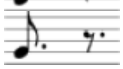
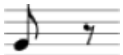
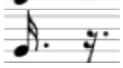
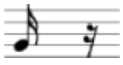
Название ноты	**Символ ноты и паузы**	**Относительная длина**	**100 BPM длительность мс**	**120 BPM длительность мс**	**140 BPM длительность мс**
Целая с точкой		6.000	3600	3000	2571
Целая		4.000	2400	2000	1714

Таблица 3.1 — продолжение

Название ноты	**Символ ноты и паузы**	**Относительная длина**	**100 BPM длительность мс**	**120 BPM длительность мс**	**140 BPM длительность мс**
Половина с точкой		3.000	1800	1500	1285
Половина		2.000	1200	1000	857
Четверть с точкой		1.500	900	750	642
Четверть		1.000	600	500	428
Восьмая с точкой		0.750	450	375	321
Восьмая		0.500	300	250	214
Шестнадцатая с точкой		0.375	225	187	160
Шестнадцатая		0.250	150	125	107

Теперь с формулой и таблицей для расчёта длительности, мы можем написать программу, чтобы сыграть сигнал «Атака!».

```
1 # c3_charge.kbs --- играем сигнал «Атака!»
2 sound {392,375,523,375,659,375,784,250,659,250,784,250}
3 say "Атака!"
```

Программа 18. Атака!

Зайдите в интернет и найдите музыку для «Row-row-row Your Boat» или другую мелодию, напишите программу для её воспроизведения.

3.2 Числовые переменные

Компьютеры действительно хорошо запоминают разные вещи, в то время как у нас, людей, с этим бывают проблемы. Язык BASIC-256 позволяет нам давать названия областям компьютерной памяти, а затем хранить в них информацию. Эти именованные области называются переменными.

Есть четыре типа переменных: числовые переменные, строковые, числовые массивы и массивы строк. В этой главе вы узнаете, как использовать числовые переменные, а другие — в остальных главах.

Числовая переменная

Числовая переменная позволяет присвоить имя блоку ячеек в оперативной памяти компьютера. Вы можете хранить и извлекать цифровые (целые или десятичные) значения из числовой переменной в вашей программе. Имя числовой переменной должно начинаться с буквы, оно может содержать латинские буквы[3]и числа и чувствительно к регистру. Запрещается использовать слова, принадлежащие языку BASIC-256 при наименовании переменных (см. Приложение I).

Примеры правильных имён переменных: **a, b6, reader, x** и **zoo**.

Имена переменных чувствительны к регистру. Это означает что имя переменной, записанное буквами в верхнем регистре и такое же имя, но записанное в нижнем регистре, представляют разные области памяти компьютера.

Программа 19 — это пример программы, использующей числовые переменные.

```
1 # c3_numericvariables.kbs
2 numerator = 30
3 denominator = 5
4 result = numerator / denominator
5 print result
```

Программа 19. Простые числовые переменные

Программа использует три переменные. В строке 2 значение 30 сохраняется в переменной под названием «numerator». В строке 3 значение 5 сохраняется в переменную «denominator». В строке 4 значение из «numerator» делится на значение из переменной «denominator» и сохраняет результат в переменной с именем «result»[4].

Теперь, когда мы увидели переменные в действии, мы можем переписать программу «Атака!», используя переменные и формулу для расчёта длительности (Формула 1).

```
1 # c3_charge2.kbs
2 # играем сигнал атаки, используя переменные
3 beats = 120
4 dottedeighth = 1000 * 60 / beats * .75
5 eighth = 1000 * 60 / beats * .5
6 sound {392, dottedeighth, 523, dottedeighth, 659, dottedeighth, 784,
   eighth, 659, eighth, 784, eighth}
7 say "Атака!"
```

Программа 20. Атака! с переменными

Изменяйте скорость воспроизведения музыки, регулируя значение, хранящееся в переменной «beats».

[3]В именах переменных можно использовать только латинские буквы (A..Z,a..z), использование русских букв недопустимо (*прим. переводчика*)

[4]Имена переменныем выбирают в соответствии со смыслом хранимых данных, поэтому автор использует: numerator (англ) - числитель, denominator (англ) - знаменатель, result (англ)- результат. В итоге программа вычисляет: результат = числитель / знаменатель (*прим. редактора*)

В этой главе для «Большой программы» мы выберем фрагмент музыки И.С. Баха и напишем программу для её воспроизведения.
Этот фрагмент — часть Маленькой Фуги И.С. Баха в соль-мажор.

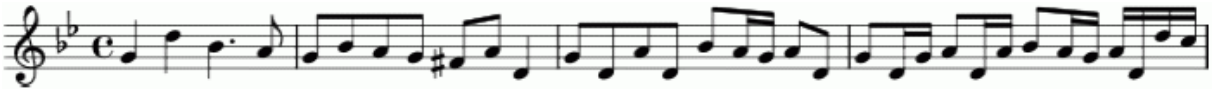


Рис. 3.4. Первая строка Маленькой Фуги И.С. Баха в соль-мажор

```
 1 # c3_littlefuge.kbs
 2 # Музыка И.С. Баха - XVIII Фуга в соль-мажор.
 3 tempo = 100 # ритм - удары в минуту
 4 milimin = 1000 * 60 # количество миллисекунд в минуте
 5 q = milimin / tempo # ритм задаётся четвертями (целая = 4 четверти) - это
    четверть
 6 h = q * 2 # Это половинка - 2 четверти
 7 e = q / 2 # это восьмая - половинка от четверти
 8 s = q / 4 # шестнадцатая = 1/4 от четверти
 9 de = e + s # восьмая с точкой = восьмая + шестнадцатая
10 dq = q + e # четверть с точкой = четверть + восьмая
11
12 sound {392, q, 587, q, 466, dq, 440, e, 392, e, 466, e, 440, e, 392,
    e, 370, e, 440, e, 294, q, 392, e, 294, e, 440, e, 294, e, 466, e,
    440, s, 392, s, 440, e, 294, e, 392, e, 294, s, 392, s, 440, e, 294,
    s, 440, s, 466, e, 440, s, 392, s, 440, s, 294, s}
```

Программа 21. Маленькая Фуга в соль-мажор

Глава 4

Мыслить как программист

Одна из самых трудных вещей — научиться думать как программист. Программистом не станешь, просто читая книги или посещая занятия, нужно приложить собственные усилия.

Чтобы быть хорошим программистом, необходимо испытывать страсть к технологиям, самообучению, логическому мышлению, а также стремление творить и исследовать.

Вы подобны великим исследователям: Христофору Колумбу (открывшему Америку), Нейлу Армстронгу (сделавшему первый шаг на Луне) и Юрию Гагарину (первому космонавту). Перед вами — заключённая в компьютере бесконечная вселенная для исследования и творчества, где ограничить вас могут лишь ваши собственные творческие способности и готовность познавать новое.

Программа для разработки игры или какое-нибудь интересное приложение часто содержит более тысячи строк программного кода. Это может быстро вымотать даже самого опытного программиста. Обычно программисты, разбираясь со сложной задачей, используют систему «трёх шагов», что-то вроде:

1. Поразмышлять над задачей.
2. Разбить задачу на части и формально описать каждую из них.
3. Преобразовать эти части в код на языке программирования, который вы используете.

4.1 Псевдокод

Псевдокод — причудливое слово, которое используется для описания, шаг за шагом, того, что должна делать ваша программа. Слово псевдокод происходит от греческой приставки «*псевдо*» (*pseudo*), что в переводе означает «подделка» и слова «*код*» (*code*), обозначающего фактические операторы программы. Псевдокод создаётся не для компьютера, а для того, чтобы помочь вам осознать сложность задачи и разбить её на смысловые части.

Нет «самого лучшего» способа написания псевдокода. Существуют десятки стандартов, и каждый из них подходит для определённого типа задач. В этом введении мы используем простые команды на русском языке для описания решения нашей задачи.

Давайте напишем простую программу, рисующую школьный автобус (как на рисунке 4.1).

Разобьём эту задачу на две части:

- нарисовать колёса
- нарисовать корпус

Теперь разобьём эти части на более мелкие и напишем наш псевдокод:

Чтобы наша программа заработала, всё, что осталось сделать — это записать её:

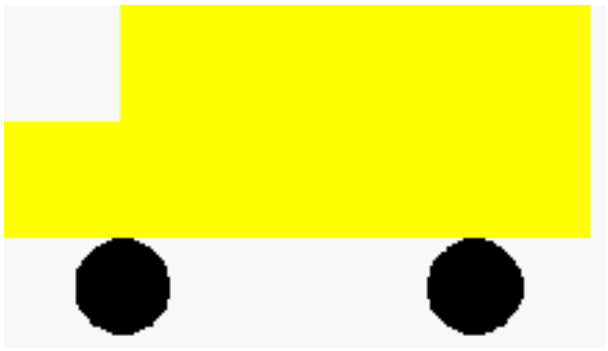


Рис. 4.1. Школьный автобус

Таблица 4.1. Школьный автобус — псевдокод

Выбрать чёрный цвет. Нарисовать колёса. Выбрать жёлтый цвет. Нарисовать кузов автобуса. Нарисовать переднюю часть автобуса.

Таблица 4.2. Школьный автобус — псевдокод и команды BASIC-256

Выбрать чёрный цвет. Нарисовать колёса.	color black circle 50,120,20 circle 200,120,20
Выбрать жёлтый цвет.	color yellow
Нарисовать кузов автобуса.	rect 50,0,200,100
Нарисовать переднюю часть автобуса.	rect 0,50,50,50

Полная программа рисования школьного автобуса (программа 22) приведена ниже. Взгляните на окончательный вариант программы, и вы увидите комментарии, предназначенные для того, чтобы помочь программисту вспомнить части, на которые была разделена задача в момент первоначального осмысления.

```
 1 # schoolbus.kbs - школьный автобус
 2 clg
 3 # рисуем колёса
 4 color black
 5 circle 50,120,20
 6 circle 200,120,20
 7 # рисуем корпус автобуса
 8 color yellow
 9 rect 50,0,200,100
10 rect 0,50,50,50
```

Программа 22. Школьный автобус

На примере программы рисования школьного автобуса легко увидеть, что есть много способов решить эту задачу. Сначала мы могли нарисовать корпус автобуса, а потом колёса, мы также могли нарисовать сначала переднюю часть, а затем заднюю. . . Мы можем назвать десятки различных способов решения этой простой задачи.

Таблица 4.3. Основные элементы блок-схем

Элемент	Имя и описание
	Поток — стрелка представляет движение от одного элемента или шага в процессе к другому. Вы должны следовать направлению стрелки.
	Пуск-останов — этот элемент подскажет вам где начало и конец блок-схемы. Каждая блок-схема должна иметь начало и конец.
	Процесс — этот элемент представляет собой деятельность или действия, которые должны произойти в программе. Только один поток (стрелка) может выходить из процесса.
	Ввод-вывод (I/O) — этот элемент представляет данные, которые читаются или записываются в системе. Примером может служить сохранение или загрузка файлов.
	Решение — элемент в форме ромба содержит простой (да/нет, верно/неверно) вопрос. Должно быть два потока (стрелки), которые выходят из элемента "решение". В зависимости от ответа мы будем следовать одним из двух путей.

Запомните одну очень важную вещь, **не существует единственно верного способа решения задачи**. Некоторые способы лучше других (меньше инструкций, проще читать...), но главное, чтобы задача была вами решена.

Попробуйте свои силы в написании псевдокода. Как бы вы попросили BASIC-256 нарисовать посох деда Мороза или жезл волшебника?

4.2 Блок-схемы

Другой метод, который используют программисты, чтобы понять задачу, называется блок-схема. Следуя старой поговорке «лучше один раз увидеть, чем сто раз услышать», программисты порой чертят диаграмму, представляющую логику работы программы. Блок-схема является одним из старейших и широко используемых методов изображения такой логики.

Это краткое введение в блок-схемы охватит только небольшую часть того, что можно с ними делать, однако уже с несколькими простыми элементами и соединителями вы сможете моделировать очень сложные процессы. Эта технология может хорошо послужить не только в программировании, но и в решении многих других задач, с которыми вы столкнётесь. Вот несколько основных элементов:

Лучший способ научиться блок-схемам — это взглянуть на примеры и попробовать изобразить что-то своими руками.

4.2.1 Блок-схема, пример первый

Вы только что выбрались из постели, и ваша мама предлагает два варианта завтрака. Вы можете выбрать свою любимую холодную овсянку или яичницу. Если вы не выберете ни один из этих вариантов, то пойдёте в школу голодным.

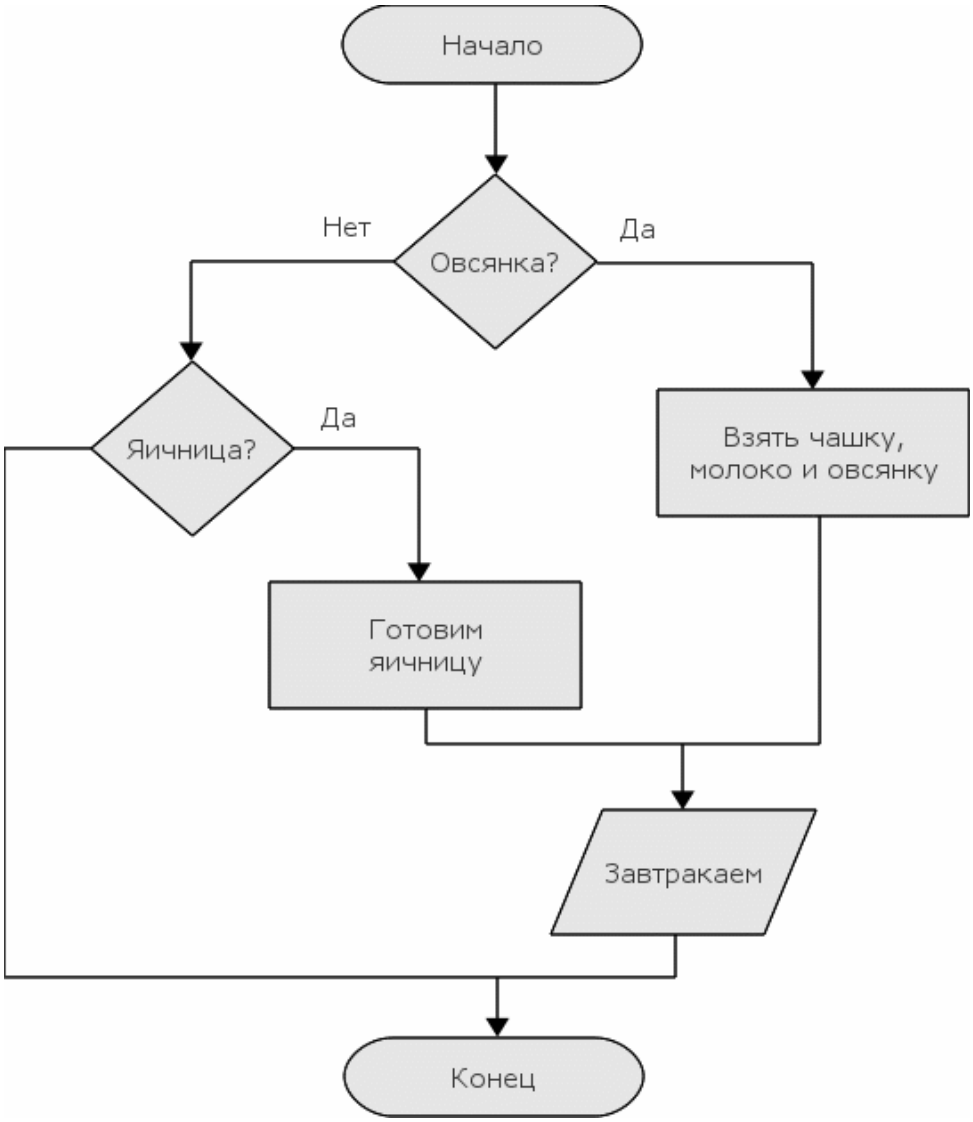


Рис. 4.2. Завтрак — блок-схема

Взгляните на рисунок 4.2 и проследите за всеми стрелками. Вы видите, как эта схема представляет собой сценарий?

4.2.2 Блок-схема, пример второй

Ещё один пример, связанный с едой. Вы очень хотите пить и намереваетесь купить газировку в торговом автомате. Посмотрите на рисунок 4.3.

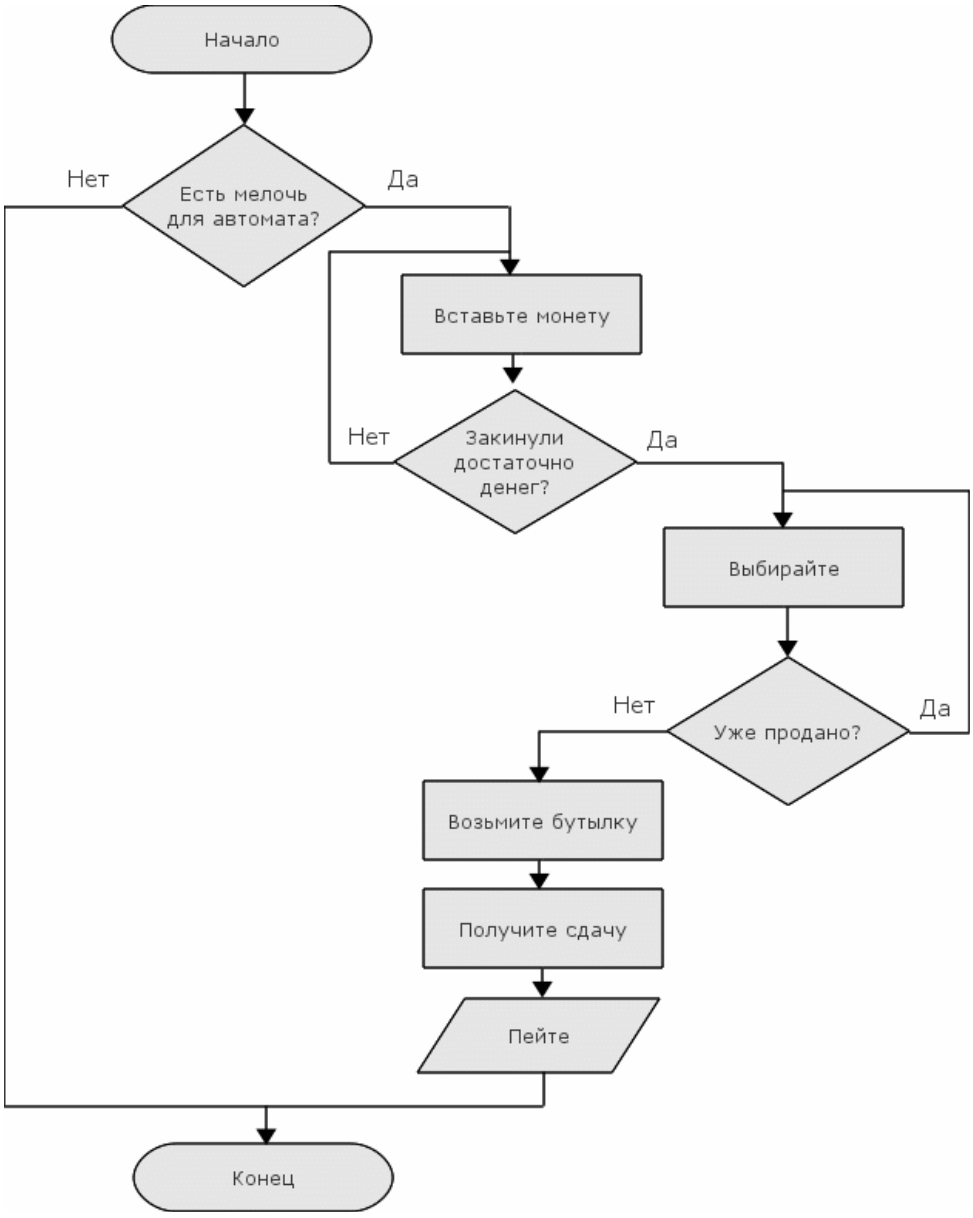
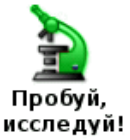


Рис. 4.3. Автомат с газировкой — блок-схема

Обратите внимание во второй блок-схеме на то, что нам может понадобиться несколько раз повторить процесс. Вы не видели, как это делается в BASIC-256, но это будет рассмотрено в последующих главах.



Испытайте себя в составлении простых блок-схем. Попробуйте составить блок-схему, описывающую как чистить зубы или как переходить улицу.

Глава 5

Программа задаёт вам вопросы

Эта глава представляет новый тип переменных (строковые переменные) и рассказывает о том, как получать числовые и текстовые ответы от пользователей.

5.1 Новый тип переменной — строковая переменная

В главе 3 вы познакомились с числовыми переменными, которые могут хранить только целые или десятичные числа. Иногда вам может понадобиться сохранить в компьютерной памяти строку, — текст, заключённый в двойные кавычки (). Для этого мы будем использовать новый тип переменной, который называется строковая переменная. Строковая переменная обозначается добавлением знака доллара $ в конце её имени.

Вы можете сохранять и извлекать значения из строковой переменной так же, как и при использовании числовых переменных. Помните, правила назначения имён, чувствительность к регистру и правила зарезервированных слов одинаковы как для строковых, так и для числовых переменных.

```
1 # ilikejim.kbs
2 name$ = "Серёжа"
3 firstmessage$ = name$ + " мой друг."
4 secondmessage$ = "Мне нравится " + name$ + "."
5 print firstmessage$
6 say firstmessage$
7 print secondmessage$
8 say secondmessage$
```

Программа 23. Мне нравится Серёжа

```
Серёжа мой друг.
Мне нравится Серёжа.
```

Вывод программы 23. Мне нравится Серёжа

Строковые переменные
Строковая переменная позволяет дать название области в оперативной памяти компьютера. Вы можете хранить и извлекать текст и символьные значения из строковой переменной в вашей программе. Имя строковой переменной должно начинаться с буквы, может содержать буквы и цифры, чувствительно к регистру, должно заканчиваться знаком доллара — **$**.
Вы также не можете использовать зарезервированные BASIC-256 слова (смотри Приложение I). Примеры правильных имён строковых переменных: **d$**, **c7$**, **book$**, **X$** и **barnYard$**.

Нельзя сохранять число в строковую переменную или строку в числовую переменную. Если вы это сделаете, то получите сообщение о синтаксической ошибке.

5.2 Input — получение текста или чисел от пользователя

До сих пор в коде самой программы содержалась вся необходимая информация для её выполнения. А теперь представляем следующий оператор — **input**. Он запоминает строку или число, которое пользователь набирает в окне ввода-вывод текста и сохраняет это значение в переменной.

Давайте обратимся к программе 23 и изменим её таким образом, что она сначала спросит ваше имя, а потом скажет, что вы её друг и нравитесь ей.

```
1 # ilikeinput.kbs
2 input "Как вас зовут? ", name$
3 firstmessage$ = name$ + " мой друг."
4 secondmessage$ = "Мне нравится " + name$ + "."
5 print firstmessage$
6 say firstmessage$
7 print secondmessage$
8 say secondmessage$
```

Программа 24. Мне нравится — кто?

```
Как вас зовут? Володя
Володя мой друг.
Мне нравится Володя.
```

Вывод программы 24. Мне нравится — кто?

input *"подсказка", имя_строковой_переменной$*
input *"подсказка", имя_числовой_переменной*
input *имя_строковой_переменной$*
input *имя_числовой_переменной*

Оператор **input** получает строку или число, которое пользователь вводит в окне ввода-вывода текста. Результат сохраняется в переменной (*имя_строковой_переменной$* или *имя_числовой_переменной*), которая затем может быть использована в программе. *Подсказка*, если она указана, будет отображаться в окне вывода текста, а курсор будет стоять непосредственно после неё. Если нужно получить число (в операторе указано имя числовой переменной), а пользователь вводит строку, то есть символы, которые не могут быть преобразованы в число, **input** устанавливает значение переменной равное нулю (0).

Программа «Волшебная математика» показывает как использовать оператор input с числовыми переменными.

```
1 # mathwiz.kbs
2 input "a? ", a
3 input "b? ", b
4 print a + "+" + b + "=" + (a+b)
5 print a + "-" + b + "=" + (a-b)
6 print b + "-" + a + "=" + (b-a)
7 print a + "*" + b + "=" + (a*b)
8 print a + "/" + b + "=" + (a/b)
9 print b + "/" + a + "=" + (b/a)
```

Программа 25. Волшебная математика

```
a? 7
b? 56
7+56=63
7-56=-49
56-7=49
7*56=392
7/56=0.125
56/7=8
```

Пример вывода программы 25. Волшебная математика

В этой главе две «Большие программы». Первая — причудливая программа, которая называет ваше имя и сообщает, каким будет ваш возраст спустя 8 лет, а вторая представляет собой генератор глупых историй.

```
1 # sayname.kbs
2 input "Как вас зовут? ", name$
3 input "Сколько вам лет? ", age
4 greeting$ = "Рад познакомиться, "+name$+"."
5 print greeting$
6 say greeting$
7 greeting$ = "Через 8 лет вам будет "+(age+8)+". Однако, немало!"
8 print greeting$
9 say greeting$
```

Программа 26. Причуда — назови имя

```
Как вас зовут? Оля
Сколько вам лет? 13
Рад познакомиться, Оля.
Через 8 лет вам будет 21. Однако, немало!
```

Пример вывода программы 26. Причуда — назови имя

```
 1 # sillystory.kbs
 2
 3 print "Глупая история."
 4
 5 input "Существительное? ", noun1$
 6 input "Глагол? ", verb1$
 7 input "Название комнаты в доме? ", room1$
 8 input "Глагол? ", verb2$
 9 input "Существительное? ", noun2$
10 input "Прилагательное? ", adj1$
11 input "Глагол? ", verb3$
12 input "Существительное? ", noun3$
13 input "Как вас зовут? ", name$
14
15 sentence$ = "Глупая история, автор " + name$ + "."
16 print sentence$
17 say sentence$
18
19 sentence$ = "Как-то, совсем недавно, я увидел, что "
   + noun1$ + " собирается " + verb1$ + " вниз по лестнице."
20 print sentence$
21 say sentence$
22
23 sentence$ = "Я подумал, что " + room1$
   + " станет мне укрытием, чтобы " + verb2$
   + " там, пока не получится " + noun2$ + "."
24 print sentence$
25 say sentence$
26
27 sentence$ = Вдруг" " + noun1$ + " сделался " + adj1$
   + ", из-за того, что я начал " + verb3$ + " в "
   + noun3$ + "."
28 print sentence$
29 say sentence$
30
31 sentence$ = "Конец."
32 print sentence$
33 say sentence$
```

Программа 27. Генератор глупых историй

```
Глупая история.
Существительное? кот
Глагол? идти
Название комнаты в доме? кухня
Глагол? рисовать
Существительное? колбаса
Прилагательное? зелёный
```



```
Глагол? играть
Существительное? карандаш
Как вас зовут? Сергей
Глупая история, автор Сергей.
Как-то, совсем недавно, я увидел,
что кот собирается идти вниз по лестнице.
Я понял, что его целью стала моя кухня,
чтобы рисовать там до тех пор, пока не
получится колбаса.
Вдруг кот сделался зелёный, из-за того,
что я начал играть в карандаш.
Конец.
```

Пример вывода программы 27. Генератор глупых историй

Глава 6

Сравнения, сравнения, сравнения

Компьютер великолепно сравнивает выражения. В этой главе мы изучим как сравнивать два выражения, как работать с комбинированными условиями и как выполнять те или иные операторы в зависимости от результатов сравнения. Мы также научимся генерировать случайные числа.

6.1 Истина и ложь

Язык BASIC-256 имеет ещё один специальный вид данных, который может храниться в числовых переменных. Это булевый (логический) тип. Результатом сравнения или логических операций являются логические значения *истина* или *ложь*. Для облегчения работы, в выражениях можно использовать две логические константы: *true* (истина) и *false* (ложь).

true
false
Две булевы (логические) константы *true* и *false* могут быть использованы в любом числовом или логическом выражении, но, как правило, являются результатом сравнения или логических операций. В действительности, константа *true* хранится в виде числа один (1), а *false* — в виде числа ноль (0).

6.2 Операторы сравнения

Ранее мы обсудили основные арифметические операции, а сейчас пришло время взглянуть на некоторые дополнительные операторы. Нам часто требуется сравнить два значения в программе, чтобы решить что делать дальше. Оператор сравнения работает с двумя значениями и возвращает истину или ложь в зависимости от результата сравнения (см. таблицу 6.1).

Таблица 6.1: Операторы сравнения

Оператор	**Операция**
<	Меньше выражение1 < выражение2 Возвращает истину, если *выражение1* меньше *выражения2* и ложь в противном случае

Таблица 6.1 — продолжение

Оператор	Операция
<=	Меньше или равно *выражение1* <= *выражение2* Возвращает истину, если *выражение1* меньше или равно *выражению2* и ложь в противном случае
>	Больше *выражение1* > *выражение2* Возвращает истину, если *выражение1* больше *выражения2* и ложь в противном случае
>=	Больше или равно *выражение1* >= *выражение2* Возвращает истину, если *выражение1* больше или равно *выражению2* и ложь в противном случае
=	Равно *выражение1* = *выражение2* Возвращает истину, если *выражение1* равно *выражению2* и ложь в противном случае
<>	Не равно *выражение1* <> *выражение2* Возвращает истину, если выражение1 не равно выражению2 и ложь в противном случае

< <= > >= <>
Шесть операций сравнения, это: меньше (<), меньше или равно (<=), больше (>), больше или равна (>=), равно (=), не равно (<>). Они используются для сравнения чисел и строк. Строки сравниваются по алфавиту слева направо. Вы также можете использовать скобки для группировки операций.

6.3 Простой выбор — оператор if

Оператор **if** (если) может использовать результат сравнения для выборочного выполнения оператора или блока операторов. Первая программа этой главы (программа 28) использует три оператора **if**, чтобы показать являетесь ли вы старше, одного возраста или моложе своего друга (см. блок-схему на рисунке 6.1).

```
1 # compareages.kbs - сравним два возраста
2 input "сколько вам лет? ", yourage
3 input "сколько лет вашему другу? ", friendage
4
5 print "Вы ";
6 if yourage < friendage then print "младше своего друга."
7 if yourage = friendage then print "одного с ним возраста."
8 if yourage > friendage then print "старше своего друга."
```

Программа 28. Сравним два возраста.

```
сколько вам лет? 13
сколько лет вашему другу? 12
Вы старше своего друга.
```

Пример вывода программы 28. Сравним два возраста.

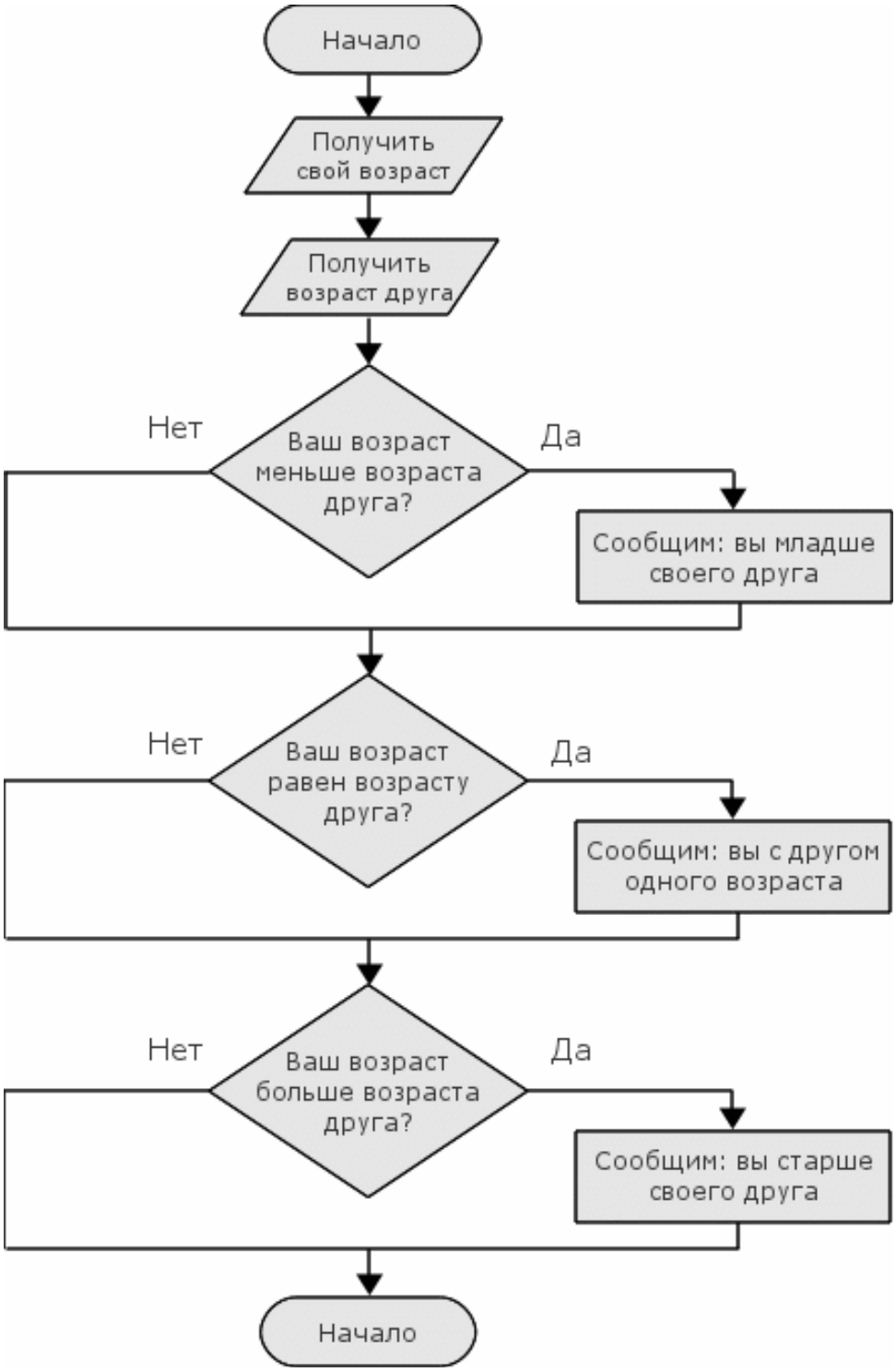
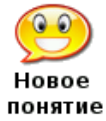


Рис. 6.1. Сравним два возраста — блок-схема

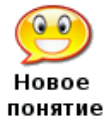


if *условие* **then** *оператор*

Если *условие* истинно, то выполняется *оператор*, следующий за словом **then**.

6.4 Случайные числа

Когда мы разрабатываем игры и симуляторы, может возникнуть необходимость имитировать игральные кости, работу рулетки в казино и другие случайные события. BASIC-256 имеет встроенный генератор случайных чисел, который сделает это для нас.



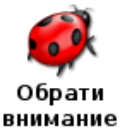
rand
Оператор **rand** возвращает случайное число, которое можно использовать в выражении. Полученное число будет в диапазоне от нуля до единицы, но никогда не будет равным единице ($0 \leq n < 1$).
Если вы хотите генерировать случайные целые числа от 1 до r, используйте следующее выражение: **n = int(rand * r) + 1**

```
1 # coinflip.kbs - бросаем монетку
2 coin = rand
3 if coin < .5 then print "Орёл."
4 if coin >= .5 then print "Решка."
```

Программа 29. Бросаем монетку.

```
Решка.
```

Пример вывода программы 29. Бросаем монетку.



В программе 29 у вас, возможно, был соблазн использовать **rand** дважды, внутри каждого **if**. Это создало бы то, что называют «логическая ошибка». Запомните, что каждый раз, когда выполняется **rand**, он возвращает разные случайные числа.

6.5 Логические операторы

Иногда необходимо соединить вместе несколько простых условий. Это может быть сделано с помощью четырёх логических операторов: **and** (и), **or** (или), **xor** (исключающее или) и **not** (отрицание). Логические операторы действуют подобно союзам в естественном языке, только **or** (или) используется в значении «одно событие, или другое, или оба вместе» (см. таблицу 6.2 Логические операторы).

Таблица 6.2: Логические операторы

<table>
<tr><th>Оператор</th><th>Операция</th></tr>
<tr><td>AND</td><td>Логическое И
выражение1 AND выражение2
Если оба выражения истинны, тогда и результат будет истинным, в противном случае он будет ложным
<table><tr><th colspan="2" rowspan="2">AND</th><th colspan="2">выражение1</th></tr><tr><th>ИСТИНА</th><th>ЛОЖЬ</th></tr><tr><td rowspan="2">выражение2</td><td>ИСТИНА</td><td>ИСТИНА</td><td>ЛОЖЬ</td></tr><tr><td>ЛОЖЬ</td><td>ЛОЖЬ</td><td>ЛОЖЬ</td></tr></table></td></tr>
<tr><td>OR</td><td>Логическое Или
выражение1 OR выражение2
Если выражение1 или выражение2 истинны, то результат будет истинным, если оба ложны — ложным.
<table><tr><th colspan="2" rowspan="2">OR</th><th colspan="2">выражение1</th></tr><tr><th>ИСТИНА</th><th>ЛОЖЬ</th></tr><tr><td rowspan="2">выражение2</td><td>ИСТИНА</td><td>ИСТИНА</td><td>ИСТИНА</td></tr><tr><td>ЛОЖЬ</td><td>ИСТИНА</td><td>ЛОЖЬ</td></tr></table></td></tr>
<tr><td>XOR</td><td>Логическое исключающее Или
выражение1 XOR выражение2
Только если оба выражения истинны, результат будет истинным, в противном случае — ложь. Оператор XOR действует подобно союзу «или» в естественном языке — «нельзя иметь и то, и другое».
<table><tr><th colspan="2" rowspan="2">XOR</th><th colspan="2">выражение1</th></tr><tr><th>ИСТИНА</th><th>ЛОЖЬ</th></tr><tr><td rowspan="2">выражение2</td><td>ИСТИНА</td><td>ЛОЖЬ</td><td>ИСТИНА</td></tr><tr><td>ЛОЖЬ</td><td>ИСТИНА</td><td>ЛОЖЬ</td></tr></table></td></tr>
<tr><td>NOT</td><td>Логическое отрицание
NOT выражение1
Возвращает противоположное значение. Если выражение1 истинно, результат будет ложь, если ложно — истина.
<table><tr><th colspan="2">NOT</th><th></th></tr><tr><td rowspan="2">выражение1</td><td>ИСТИНА</td><td>ЛОЖЬ</td></tr><tr><td>ЛОЖЬ</td><td>ИСТИНА</td></tr></table></td></tr>
</table>

and or xor not

Четыре логических оператора: логические и (and), логическое или (or), исключающее или (xor) и и логическое отрицание (not) позволяют соединять или изменять выражения сравнения. Вы также можете использовать скобки для группировки операций.

6.6 Оператор выбора в более сложной форме — If/End If

Когда мы пишем программы, иногда возникает необходимость выполнить несколько операторов, если условие истинно. Это можно сделать, используя альтернативный формат оператора **if**. В этом случае вы не размещаете операторы на одной строке с **if ... then**, а помещаете их ниже — на следующей, один или несколько (по одному в каждой строке), закрывая блок конструкцией **end if**.

if *условие* **then**
    оператор(ы) # выполняем, если *условие* истинно
end if

Оператор **if/end if** позволяет создать блок программного кода, который выполняется, если условие истинно. Как правило, операторы внутри такого блока пишутся с некоторым отступом, чтобы было удобнее читать код.

```
 1 # dice.kbs - бросаем (игральные) кости
 2 die1 = int(rand * 6) + 1
 3 die2 = int(rand * 6) + 1
 4 total = die1 + die2
 5
 6 print "кость 1 = " + die1
 7 print " кость 2 = " + die2
 8 print "у вас выпало " + total
 9 say "у вас выпало " + total
10
11 if total = 2 then
12     print "глаза змеи!"
13      say "глаза змеи!"
14 end if
15 if total = 12 then
16     print "полночь!"
17     say "полночь!"
18 end if
19 if die1 = die2 then
20     print "дубль - бросай снова!"
21     say "дубль - бросай снова!"
22 end if
```

Программа 30. Бросаем (игральные) кости.

```
кость 1 = 6
кость 2 = 6
у вас выпало 12
полночь!
дубль - бросай снова!
```

Пример вывода программы 30. Бросаем (игральные) кости.

«Правка» → «Красивый код» в меню
Пункт «Красивый код» в меню «Правка» отформатирует вашу программу, сделав её более удобной для чтения. Будут удалены пробелы в начале и конце строк и сделаны отступы для блоков кода (как в операторе **if/end if**).

6.7 Оператор выбора в полной форме — If/Else/End If

Третья и последняя форма оператора **if** — это **if/else/end if**. Эта полная форма, которая позволяет создать один блок кода, который будет выполнен, если условие истинно, и другой блок кода, который будет выполнен, если условие ложно.

if *условие* **then**
    оператор(ы) # выполняем, если условие истинно
else
    оператор(ы) # выполняем, если условие ложно
end if

Операторы **If**, **else** и **end if** позволяют определить два блока программного кода. Первый блок, после слова **then**, выполняется, если условие истинно, а второй блок, после слова **else**, выполняется, если условие ложно.

Программа 31 — переписанная программа 29 с использованием else.

```
1 # coinflip2 - бросаем монетку, используя else
2 coin = rand
3 if coin < .5 then
4     print "Орёл."
5     say "Орёл."
6 else
7     print "Решка."
8     say "Решка."
9 end if
```

Программа 31. Бросаем монетку с использованием else.

```
Орёл.
```

Пример вывода программы 31 Бросаем монетку с использованием else.

6.8 Вложенные операторы выбора

И последнее. Операторы **if/end if** и **if/else/end if** можно вкладывать друг в друга. Возможно, звучит сложновато, но в последующих главах вы увидите как это происходит.

«Большая программа» этой главы — программа, «бросающая» шестигранный кубик и рисующая его «выпавшую» сторону с определённым количеством точек.

```
 1 # dieroll.kbs - "бросаем кубик"и рисуем его
 2 # hw - высота и ширина точки
 3 hw = 70
 4 # расстояние между точками
 5 # 1/4 оставшегося места, после того, как нарисуем 3 точки
 6 margin = (300 - (3 * hw)) / 4
 7 # z1 - x,y левой верхней точки
 8 z1 = margin
 9 # z2 - x,y средней точки
10 z2 = z1 + hw + margin
11 # z3 - x,y правой нижней точки
```



```
12 z3 = z2 + hw + margin
13
14 # получаем значение
15 roll = int(rand * 6) + 1
16 print roll
17
18 color black
19 rect 0,0,300,300
20
21 color white
22 # верхний ряд точек
23 if roll <> 1 then rect z1,z1,hw,hw
24 if roll = 6 then rect z2,z1,hw,hw
25 if roll >= 4 and roll <= 6 then rect z3,z1,hw,hw
26 # средний ряд
27 if roll = 1 or roll = 3 or roll = 5 then rect z2,z2,hw,hw
28 # нижний ряд
29 if roll >= 4 and roll <= 6 then rect z1,z3,hw,hw
30 if roll = 6 then rect z2,z3,hw,hw
31 if roll <> 1 then rect z3,z3,hw,hw
32
33 say "у вас выпало " + roll
```

Программа 32. Большая программа — «Бросаем» кубик и рисуем его.

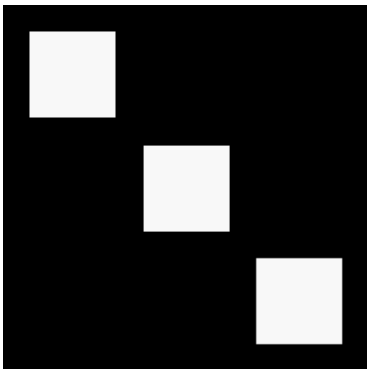


Рис. 6.2. Пример вывода программы 32. «Бросаем» кубик и рисуем его.

Глава 7

Циклы и счётчики — повторяем снова и снова

До сих пор наши программы после старта выполняли шаг за шагом инструкции и завершали свою работу. Пока это хорошо работало для простых программ, однако по большей части при программировании возникают задачи, требующие повторений или работы со счётчиком, или и то и другое вместе. В этой главе мы рассмотрим три вида оператора цикла, освоим режим «быстрой графики», и научимся замедлять выполнение программы.

7.1 Цикл For

Наиболее распространённый вид оператора цикла — это **for**. Цикл **for** повторяет блок операторов указанное число раз, отслеживая значение счётчика. Счётчик цикла может начинаться с любого значения и завершаться любым значением, шаг приращения счётчика также может быть любым. Программа 33 — простой пример использования цикла **for**, печатает и проговаривает цифры от 1 до 10 (включительно). Программа 34 считает по 2, начиная с нуля и заканчивая 10.

```
1 # for.kbs
2 for t = 1 to 10
3     print t
4     say t
5 next t
```

Программа 33. Оператор for.

```
1
2
3
4
5
6
7
8
9
10
```

Вывод программы 33. Оператор for.

```
1 # forstep2.kbs
```

```
2 for t = 1 to 10 step 2
3     print t
4     say t
5 next t
```

Программа 34. Оператор for вместе с step.

```
0
2
4
6
8
10
```

Вывод программы 34 Оператор for вместе с step.

for *счётчик* = *выражение1* **to** *выражение2* [**step** *выражение3*]
оператор(ы)
next *счётчик*

Выполняет определённый блок кода указанное число раз. Переменная *счётчик* начинает со значения *выражение1*. После каждого выполнения блока *операторов* *счётчик* увеличивается на значение *выражение3* (или на единицу, если шаг — так переводится английское слово **step** — не указан). Цикл прекращается, когда значение *счётчика* превысит значение *выражение2*.

Используя цикл, можно легко рисовать очень интересные картинки. Программа 35 рисует муаровый узор[1]. Это действительно интересное изображение возникает потому, что компьютер не в состоянии нарисовать абсолютно прямую линию[2]. Поскольку рисование происходит по точкам (пикселям), линия под наклоном выглядит как лесенка. И если вы внимательно посмотрите на линии, которые мы нарисовали, то увидите, что они на самом деле зубчатые.

```
1 # moire.kbs
2 clg
3 color black
4 for t = 1 to 300 step 3
5     line 0,0,300,t
6     line 0,0,t,300
7 next t
```

Программа 35. Муаровый узор.

Какие ещё муаровые узоры вы можете нарисовать? Начните с центра, используйте различные значения шага, накладывайте одно на другое, пробуйте разные цвета — развлекайтесь!

Оператор **for** может быть также использован и для счёта в обратном направлении. Для этого укажите после **step** отрицательное число.

[1] Узор, возникающий при наложении двух периодических сетчатых рисунков. Явление обусловлено тем, что повторяющиеся элементы двух рисунков следуют с немного разной частотой и то накладываются друг на друга, то образуют промежутки (*прим. разработчика*).

[2] Разумеется, если только она не строго вертикальная или горизонтальная (*прим. разработчика*).

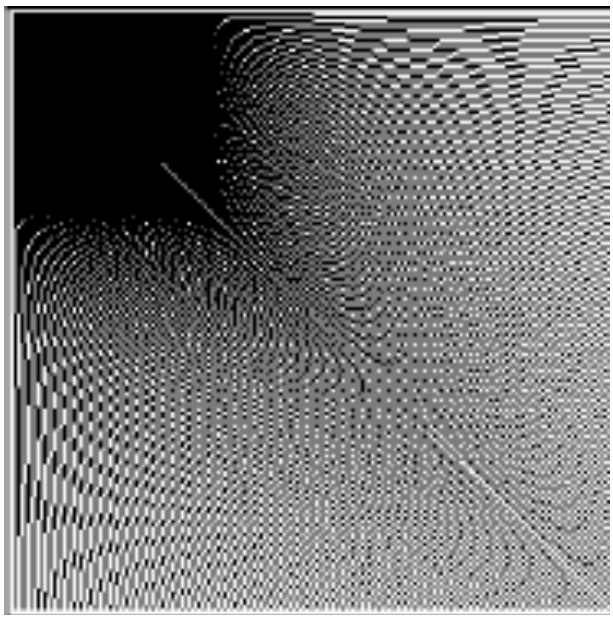


Рис. 7.1. Вывод программы 35 Муаровый узор.

```
1 # forstepneg1.kbs
2 for t = 10 to 0 step -1
3     print t
4     pause 1.0
5 next t
```

Программа 36. Оператор for — счёт в обратном направлении.

```
10
9
8
7
5
4
3
2
1
0
```

Вывод программы 36. Оператор for — счёт в обратном направлении.

pause *секунды*
Оператор pause говорит BASIC-256 приостановить выполнение текущей программы на указанное число *секунд*. Количество секунд может быть задано десятичной дробью, если необходима пауза меньше секунды.

7.2 Делай, пока я не скажу остановиться

Следующий тип цикла — **do**/**until** — повторяет блок кода один или несколько раз. После каждого повтора проверяется логическое условие. Цикл повторяется до тех пор, пока условие *ложно (false)*.

Программа 37 использует цикл **do**/**until**, для проверки корректности ввода пользователем данных. Цикл будет повторяться до тех пор, пока пользователь не введёт число от 1 до 10.

```
1 # dountil.kbs
2 do
3     input "введите число от 1 до 10?",n
4 until n>=1 and n<=10
5 print "вы ввели " + n
```

Программа 37. Введите число от 1 до 10.

```
введите число от 1 до 10?66
введите число от 1 до 10?-56
введите число от 1 до 10?3
вы ввели 3
```

Пример вывода программы 37. Введите число от 1 до 10.

do
оператор(ы)
until *условие*
Выполняет блок *операторов* снова и снова, пока *условие* ложно. *Оператор(ы)* будут выполнены один или несколько раз.

Программа 38 использует **do**/**until** для счёта от 1 до 10 подобно программе 33, которая использовала для того же цикл **for**.

```
1 # dountilfor.kbs
2 t = 1
3 do
4     print t
5     t = t + 1
6 until t >= 11
```

Программа 38. Do/Until считает до 10.

```
1
2
3
4
5
6
7
8
9
10
```

Вывод программы 38. Do/Until считает до 10.

7.3 Делай, пока я говорю делать

Третий тип цикла — **while**/**end while** — проверяет условие перед выполнением каждого повтора, и если результат проверки условия принимает значение *истина (true)*, то выполняется код в цикле. Цикл **while**/**end while** может выполнить внутренний код несколько раз или не выполнить ни разу.

Иногда нам нужна программа с бесконечным циклом, пока сам пользователь не остановит её. Это может быть легко сделано с помощью логической константы *true (истина)* (см. программу 39).

```
1 # whiletrue.kbs
2 while true
3     print "больше никогда";
4 end while
```

Программа 39. Вечный цикл.

```
больше никогда
больше никогда
больше никогда
больше никогда
больше никогда
... выполняется пока вы её не остановите
```

Пример вывода программы 39. Вечный цикл.[3]

while *условие*
оператор(ы)
end while
Выполняет *оператор(ы)* в теле цикла снова и снова, пока *условие* истинно. *Оператор(ы)* будет выполнены нуль (0) или несколько раз.

Программа 40 использует **while**/**end while** для счёта от 1 до 10 подобно программе 33, которая использовала для того же цикл **for**.

```
1 # whilefor.kbs
2 t = 1
3 while t <= 10
4     print t
5     t = t + 1
6 end while
```

Программа 40. While считает до 10.

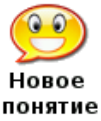
```
1
2
3
4
5
6
7
8
9
10
```

[3] У автора используется слово "nevermore— намёк на стихотворение Эдгара Алана По "Ворон в котором есть строчка: Quoth the Raven, “Nevermore.” — Каркнул: "Больше никогда!"(*прим. редактора*).

Вывод программы 40. While считает до 10

7.4 Быстрая графика

Когда нам нужно быстро выполнить много графических операций, например, при создании анимации или в играх, BASIC-256 предлагает режим «быстрой графики». Такой режим включается командой **fastgraphics**. В этом режиме окно для вывода графики будет обновляться только при выполнении команды **refresh**.



fastgraphics
refresh
Команда **fastgraphics** включает режим «быстрой графики». В этом режиме окно для вывода графики будет обновляться только по команде **refresh**. Включив «быструю графику» во время выполнения программы, вы не сможете вернуться в стандартный (медленный) режим[4].

```
 1 # kalidescope.kbs
 2 clg
 3 fastgraphics
 4 for t = 1 to 100
 5     r = int(rand * 256)
 6     g = int(rand * 256)
 7     b = int(rand * 256)
 8     x = int(rand * 300)
 9     y = int(rand * 300)
10     h = int(rand * 100)
11     w = int(rand * 100)
12     color rgb(r,g,b)
13     rect x,y,w,h
14     rect 300-x-w,y,w,h
15     rect x,300-y-h,w,h
16     rect 300-x-w,300-y-h,w,h
17 next t
18 refresh
```

Программа 41. Калейдоскоп.

Запустите программу 41, предварительно удалив или закомментировав строку номер 3, с командой **fastgraphics**. Вы видите разницу?

В «Большой программе» этой главы мы будем использовать цикл **while** для анимации подпрыгивающего мяча в окне для вывода графики.

```
1 # bouncingball.kbs
2 fastgraphics
3 clg
4
```

[4]После завершения выполнения программы режим «быстрой графики» выключается автоматически (*прим. разработчика*).

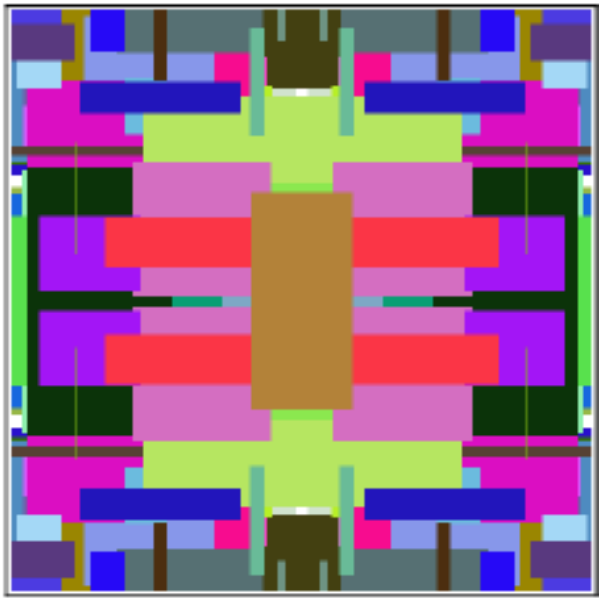


Рис. 7.2. Пример вывода программы 41 Калейдоскоп.

```
 5 # начальная позиция мяча
 6 x = rand * 300
 7 y = rand * 300
 8 # размер мяча
 9 r = 10
10 # скорость по направлениям x и y
11 dx = rand * r + 2
12 dy = rand * r + 2
13
14 color green
15 rect 0,0,300,300
16
17 while true
18 # стираем старый мяч
19     color white
20     circle x,y,r
21 # вычисляем новую позицию
22     x = x + dx
23     y = y + dy
24 # столкновение с левой или правой границей
25     if x < 0 or x > 300 then
26         dx = dx * -1
27         sound 1000,50
28     end if
29 # столкновение с верхней или нижней границей
30     if y < 0 or y > 300 then
31         dy = dy * -1
32         sound 1500,50
33     end if
34 # рисуем новый мяч
35     color red
```



```
36      circle x,y,r
37  # обновляем окно графики
38      refresh
39  end while
```

Программа 42. Большая программа — прыгающий мяч.

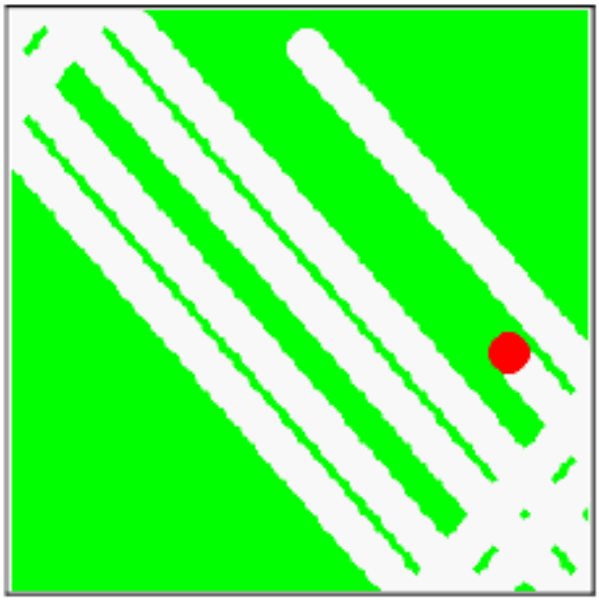


Рис. 7.3. Пример вывода программы 42. Большая программа — прыгающий мяч.

Глава 8

Графика на заказ — создание фигур своими руками

В этой главе мы покажем вам, как рисовать красочные слова и специальные фигуры в окне для вывода графики. Будут рассмотрены следующие моменты: форматированный текст, рисование многоугольников, оператор **stamp**, изменяющий расположение, размеры и даже способный вращать нарисованные фигуры. А ещё вы научитесь измерять углы в радианах.

8.1 Форматированный текст в окне для вывода графики

В первой главе вы познакомились с оператором **print** и смогли выводить строки и числа в окно ввода-вывода текста. Команды **text** и **font** позволяют размещать цифры и текст в окне для вывода графики.

```
1 # graphichello.kbs
2 clg
3 color red
4 font "Tahoma",33,100
5 text 50,50,"Привет."
5 font "Impact",33,50
6 text 50,100,"Привет."
7 font "Courier New",33,50
8 text 50,150,"Привет."
```

Программа 43. «Привет» в окне для вывода графики.

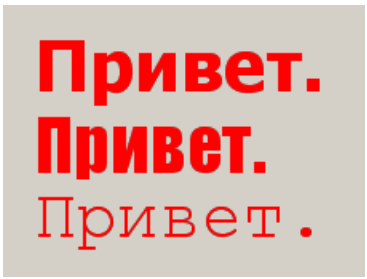


Рис. 8.1. Вывод программы 43. «Привет» в окне для вывода графики.

font *имя_шрифта, размер, насыщенность*

Устанавливает *имя шрифта*, его *размер* и *насыщенность* для использования в последующем отображении текста в окне вывода графики.

Аргумент	Описание
имя_шрифта	Строка, содержащая имя системного шрифта. Шрифт должен быть предварительно установлен в системе. Общеупотребительные имена шрифтов Windows это: «Verdana», «Courier New», «Tahoma», «Arial», и «Times New Roman»[1].
размер	Высота текста в единицах изменения, известных как «точка». Стандартное разрешение монитора — 72 точки на дюйм.
насыщенность	Число от 1 до 100, определяющими, насколько «тёмными» должны быть знаки. Используйте 25 для светлых, 50 для нормальных и 75 для «жирных» символов.

text *x, y, выражение*

Рисует строку символов, представленную *выражением*, в окне для вывода графики, при этом верхний левый угол позиционируется по координатам x и y. Использует шрифт, его размер и насыщенность, согласно последнему оператору **font**.

Рис. 8.2. Общеупотребительные Windows шрифты

[1] В Linux также можно использовать стандартные MS Windows™шрифты. Пользователям ALT Linux Достаточно поставить пакет fonts-ttf-ms из репозитория вашего дистрибутива (см http://sisyphus.ru/ru/srpm/Sisyphus/fonts-ttf-ms) или скопировать ttf файлы из вашей легальной установки Windows в Linux (*прим. редактора*).

8.2 Изменения размеров окна для вывода графики

По умолчанию размер окна для вывода графики — 300x300 пикселей. Хотя этого достаточно для большинства программ, но для некоторых может оказаться слишком много или слишком мало. Оператор **graphsize** позволяет изменить размеры окна для вывода графики, установив требуемые именно вам ширину и высоту. Ваша программа также может использовать функции **graphwidth** и **graphheight**, чтобы получить текущие размеры этого окна.

```
 1 # resizegraphics.kbs
 2 graphsize 400,200
 3 xcenter = graphwidth/2
 4 ycenter = graphheight/2
 5
 6 color black
 7 line xcenter, ycenter - 10, xcenter, ycenter + 10
 8 line xcenter - 10, ycenter, xcenter + 10, ycenter
 9
10 font "Tahoma",12,50
11 text xcenter + 10, ycenter + 10, "Центр в
   точке (" + xcenter + "," + ycenter + ")"
```

Программа 44. Изменение размера окна для графики.

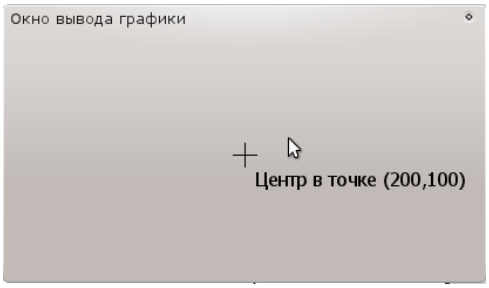


Рис. 8.3. Вывод программы 44. Изменение размера окна для графики (фрагмент).

graphsize *ширина, высота*

Устанавливает размер окна для вывода графики, согласно указанным в параметрах *ширине* и *высоте*.

graphwidth или **graphwidth()**
graphheight или **graphheight()**

Функции возвращают текущую ширину и высоту окна графики для использования в вашей программе.

8.3 Многоугольник на заказ

В предыдущих главах мы научились рисовать прямоугольники и круги. Нередко нам требуется рисовать и другие фигуры. Оператор **poly** позволяет изобразить произвольный многоугольник в любом месте экрана.

Давайте нарисуем большую красную стрелу в центре окна для вывода графики. Для начала изобразим её на листе бумаги, чтобы мы смогли отчётливо представлять себе координаты вершин этой стрелы (см. Рис. 8.4).

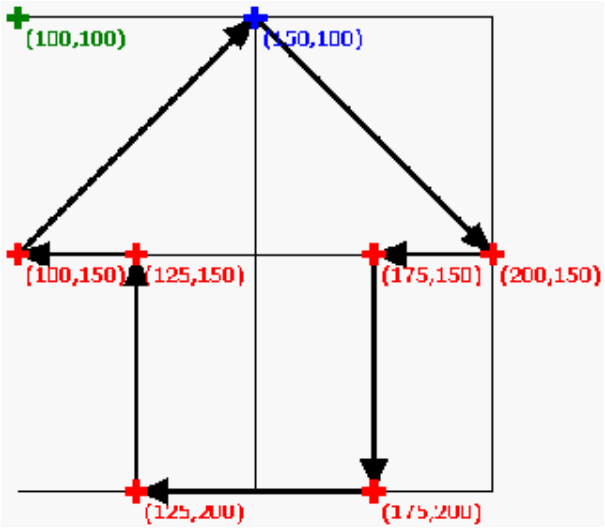


Рис. 8.4. Большая красная стрела

А теперь запишем координаты x и y точек, начиная с вершины и двигаясь по часовой стрелке.

```
1 # bigredarrow.kbs
2 clg
3 color red
4 poly {150, 100, 200, 150, 175, 150, 175, 200, 125, 200, 125, 150, 100,
    150}
```

Программа 45. Большая красная стрела.

poly x1, y1, x2, y2 ...
poly числовой_массив

Рисует многоугольник, где пары координат (x1, y1), (x2, y2) ... являются его вершинами.

8.4 Штампуем многоугольники

Оператор **poly** помещает многоугольник в заданное место на экране, однако будет сложно перемещать его или подогнать по размеру. Эти проблемы решаются с помощью оператора **stamp**. Оператор **stamp** позволяет задать позицию многоугольника в любом месте экрана и, если необходимо, указать масштабирование и поворот .

Рис. 8.5. Вывод программы 45. Большая красная стрела.

Давайте нарисуем равносторонний треугольник (все стороны одинаковой длины) на листе бумаги. Установим координаты вершины (0,0) и сделаем каждую сторону длиной 10 (смотрите рисунок 8.6).

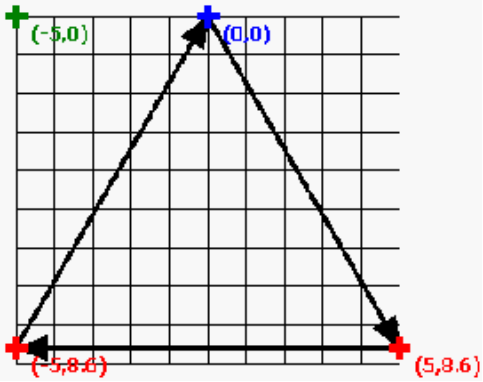


Рис. 8.6. Равносторонний треугольник

Теперь мы создадим программу, используя простую форму оператора **stamp**, чтобы заполнить экран треугольниками. Программа 46 будет делать именно это. Она использует **stamp**, рисующий треугольник, внутри двух вложенных циклов, чтобы заполнить экран.

```
1 # stamptri.kbs
2 clg
3 color black
4 for x = 25 to 200 step 25
5     for y = 25 to 200 step 25
6         stamp x, y, {0, 0, 5, 8.6, -5, 8.6}
7     next y
8 next x
```

Программа 46. Заполнение экрана треугольниками.

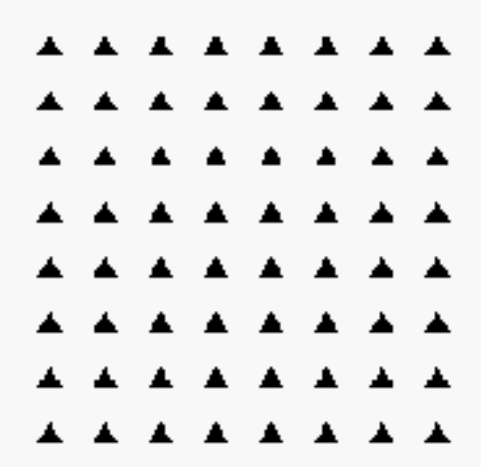


Рис. 8.7. Вывод программы 46. Заполнение экрана треугольниками.

stamp x, y, x1, y1, x2, y2 ...
stamp x, y, *числовой_массив*
stamp x, y, масштаб, x1, y1, x2, y2 ...
stamp x, y, масштаб, угол_поворота, x1, y1, x2, y2 ...
stamp x, y, масштаб, угол_поворота, *числовой_массив*

Рисует многоугольник относительно точки (0,0), заданной координатами x и y. Дополнительно можно задать *масштаб*, где число 1 обозначает реальный размер (100%). Многоугольник также можно «повернуть», задав *угол поворота* по часовой стрелке в радианах (от 0 до 2π). Пары координат вершин, как и в операторе **poly**, можно задать числовым массивом.

Радианы от 0 до 2π

Углы в BASIC-256 выражаются в единицах измерения, известных как радиан. Радианы измеряются в диапазоне от 0 до 2π. Прямой угол (90°) составляет $\frac{\pi}{2}$ радиан, а развёрнутый угол (180°) — π радиан. Вы можете перевести радианы в градусы по формуле $\textit{радианы} = \frac{\textit{градусы}}{180} \times \pi$

Давайте рассмотрим другой пример, использующий **stamp**. Программа 47 использует тот же равносторонний треугольник, что и предыдущая программа, но размещает его 100 раз в произвольных местах произвольно масштабируя и вращая.

```
1 # stamptri2.kbs
2 clg
3 color black
4 for t = 1 to 100
5     x = rand * graphwidth
6     y = rand * graphheight
7     s = rand * 7
8     r = rand * 2 * pi
```

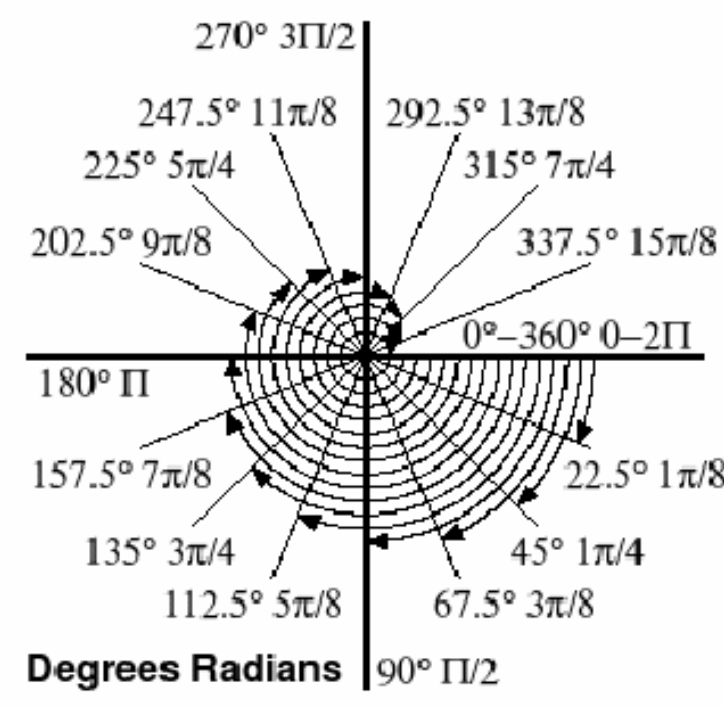


Рис. 8.8. Градусы и радианы.

```
 9     stamp x, y, s, r, {0, 0, 5, 8.6, -5, 8.6}
10 next t
```

Программа 47. Сотня произвольных треугольников.

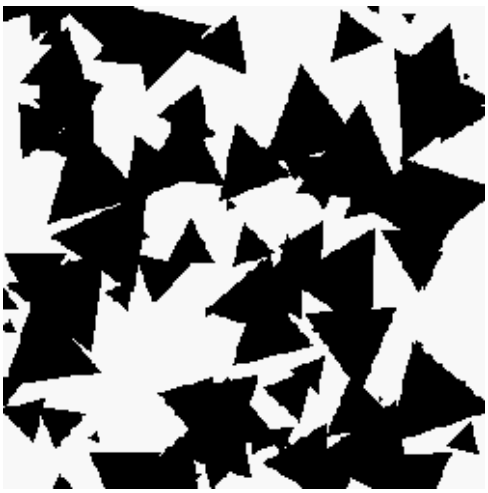


Рис. 8.9. Пример вывода программы 47. Сотня произвольных треугольников.

pi

Константа **pi** может использоваться в выражениях, для того, чтобы вам не пришлось запоминать значение π. Это значение примерно равно 3,1415[2].

В программе 47 добавьте команды так, чтобы выбирался случайный цвет для треугольников. А также создайте свой многоугольник для **stamp**.

Давайте пошлём какой-нибудь вашей знакомой цветы. Следующая программа рисует цветок, используя вращение и **stamp**.

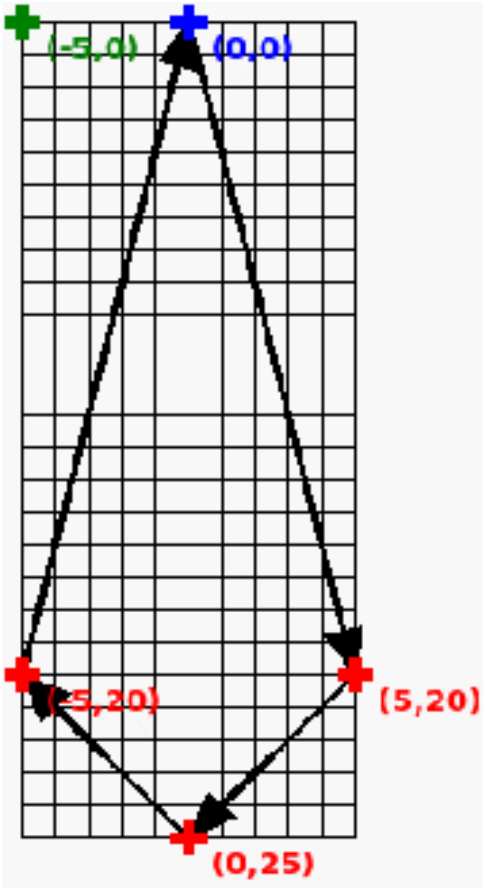


Рис. 8.10. Большая программа — цветы для тебя, заготовка для рисования лепестка цветка

[2]BASIC-256 использует значение $\pi = 3,141593$. Первые 1000 знаков числа π можно найти в Википедии: http://ru.wikipedia.org/wiki/\T2A\CYRP \T2A\cyri _(\T2A\cyrch \T2A\cyri \T2A\cyrs \T2A\cyrl \T2A\cyro). Первые 6 знаков $\pi = 3,14159$ можно запомнить по фразе «Это я знаю и помню прекрасно», где количество букв в слове соответствует цифре в записи числа π (*прим. редактора*).

```
1 # aflowerforyou.kbs
2 clg
3
4 color green
5 rect 148,150,4,150
6
7 color 255,128,128
8 for r = 0 to 2*pi step pi/4
9     stamp graphwidth/2, graphheight/2, 2, r, {0, 0, 5, 20, 0, 25, -5,
   20}
10 next r
11
12 color 128,128,255
13 for = 0 to 2*pi step pi/5
14     stamp graphwidth/2, graphheight/2, 1, r, {0, 0, 5, 20, 0, 25, -5,
    20}
15 next r
16
17 message$ = "Цветы для тебя."
18
19 color darkyellow
20 font "Tahoma", 14, 50
21 text 10, 10, message$
22 say message$
```

Программа 48. Большая программа — цветы для тебя.

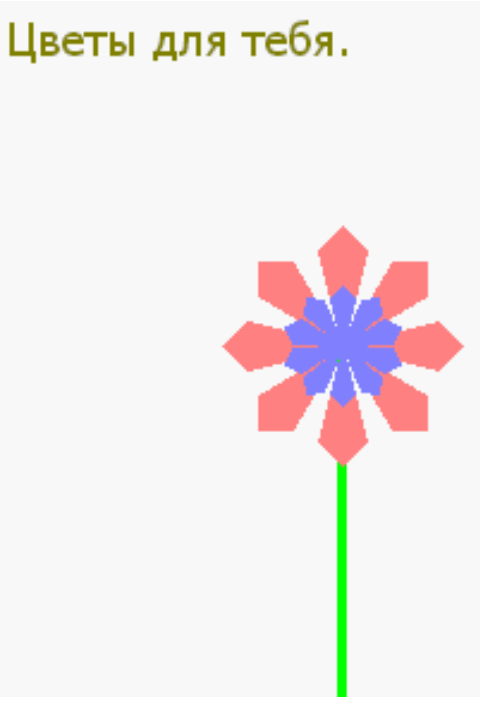


Рис. 8.11. Вывод программы 48 Большая программа — цветы для тебя.

Глава 9

Подпрограммы — повторное использование кода

В это граве мы рассмотрим создание специальных меток в коде и переход к этим меткам. Это позволит программе выполнять код в более сложном порядке. Мы также познакомимся с подпрограммами. Обращение к подпрограмме, **gosub**, действует подобно переходу к метке, но при этом с возможностью вернуться назад.

9.1 Метки и оператор goto

В седьмой главе мы видели, как использовать конструкции языка BASIC-256 для организации циклов. В программе 49 показан пример бесконечного цикла, с использованием метки и оператора **goto**.

```
1 # gotodemo.kbs
2 top:
3 print "Ку-ку"
4 goto top
```

Программа 49. Goto с меткой.

```
Ку-ку
Ку-ку
Ку-ку
... повторяет бесконечно
```

Пример вывода программы 49. Goto с меткой.

метка:

Метка позволяет вам дать название определённому месту в программе для того, чтобы вы смогли туда перейти во время выполнения программы. Меток в программе может быть много.
Имя метки должно заканчиваться двоеточием (:), в строке с меткой не должно быть других операторов, имя метки должно начинаться с буквы и содержать латинские буквы и цифры с учётом регистра букв. Кроме того, нельзя использовать зарезервированные слова BASIC-256 или используемые имена переменных (см. приложение I).
Примеры правильных имён меток: *top:, far999:, About:*.

goto *метка*

Оператор **goto** осуществляет переход к оператору, следующему непосредственно за *меткой*.

Некоторые программисты используют метки и оператор **goto** во всех своих программах. Хотя порой программировать с **goto** легче, но такой стиль добавляет сложности в большие программы и их труднее отлаживать и поддерживать. Поэтому рекомендуем вам использовать **goto** как можно реже.

Давайте рассмотрим другой пример использования метки и оператора **goto**. В программе 50 мы создадим красочные часы.

```
 1 # textclock.kbs
 2 fastgraphics
 3 font "Tahoma", 20, 100
 4 color blue
 5 rect 0, 0, 300, 300
 6 color yellow
 7 text 0, 0, "Мои часы:"
 8 showtime:
 9 color blue
10 rect 100, 100, 200, 100
11 color yellow
12 text 100, 100, hour + ":" + minute + ":" + second
13 refresh
14 pause 1.0
15 goto showtime
```

Программа 50. Цифровые часы.

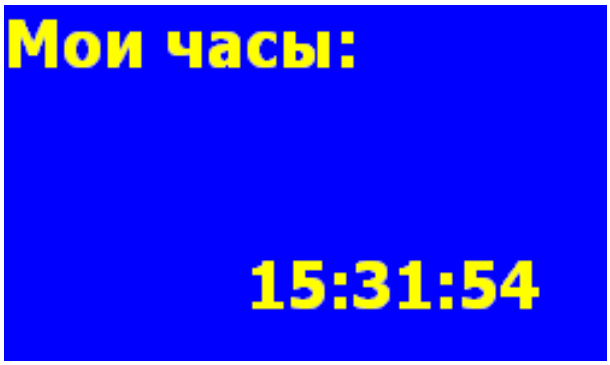
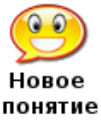


Рис. 9.1. Пример вывода программы 50. Цифровые часы (фрагмент).

hour или **hour()**
minute или **minute()**
second или **second()**
month или **month()**
day или **day()**
year или **year()**



Функции **year**, **month**, **day**, **hour**, **minute** и **second** возвращают соответствующие значения системных часов. Они позволят вашей программе сказать который час.

year	Возвращает 4 цифры текущего года.
month	Возвращает номер месяца от 0 до 11. 0 — январь, 1 — февраль, ... 11 — декабрь.
day	Возвращает текущий день от 1 до 28, 29, 30 или 31.
hour	Возвращает текущий час от 0 до 24 в 24 часовом формате
minute	Возвращает минуты от 0 до 59 текущего часа.
second	Возвращает секунды от 0 до 59 текущей минуты.

9.2 Повторное использование кода — оператор Gosub

Во многих программах мы найдём строки или даже целые блоки кода, которые требуются снова и снова. Чтобы разрешить эту проблему, BASIC-256 использует понятие «подпрограммы». Подпрограмма — это блок кода, который может быть вызван из разных мест программы, чтобы выполнить определённую задачу или часть задачи. Когда подпрограмма завершается, управление возвращается туда, откуда она была вызвана.

Программа 51 показывает пример подпрограммы, которая вызывается три раза.

```
 1 # gosubdemo.kbs
 2 gosub showline
 3 print "Привет"
 4 gosub showline
 5 print "всем"
 6 gosub showline
 7 end
 8
 9 showline:
10 print "------------------"
11 return
```

Программа 51. Подпрограмма.

```
------------------
Привет
------------------
всем
------------------
```

Вывод программы 51. Подпрограмма.

gosub *метка*

Оператор **gosub** обеспечивает переход к подпрограмме, обозначенной *меткой*.

return

Выполнение оператора **return** в подпрограмме передаёт управление обратно — туда, откуда была вызвана подпрограмма.

end

Прекращение выполнения программы (стоп).

Теперь, когда мы увидели подпрограммы в действии, давайте напишем новую программу цифровых часов, используя подпрограмму для лучшего форматирования времени и даты (программа 52).

```
 1 # textclockimproved.kbs
 2
 3 fastgraphics
 4
 5 while true
 6 color blue
 7 rect 0, 0, graphwidth, graphheight
 8 color white
 9 font "Times New Roman", 40, 100
10
11 line$ = ""
12 n = month + 1
13 gosub addtoline
14 line$ = line$ + "/"
15 n = day
16 gosub addtoline
17 line$ = line$ + "/"
18 line$ = line$ + year
19 text 50,100, line$
20
21 line$ = ""
22 n = hour
23 gosub addtoline
24 line$ = line$ + ":"
25 n = minute
26 gosub addtoline
27 line$ = line$ + ":"
28 n = second
29 gosub addtoline
30 text 50,150, line$
31 refresh
32 end while
33
34 addtoline:
```



```
35 # добавляем две цифры в строку line$
36 if n < 10 then line$ = line$ + "0"
37 line$ = line$ + n
38 return
```

Программа 52. Цифровые часы — усовершенствованные.

Рис. 9.2. Пример вывода программы 52. Цифровые часы — усовершенствованные (фрагмент).

В качестве «Большой программы» этой главы, давайте сделаем программу «бросающую кости» (два кубика), рисующую их на экране и подсчитывающую общую сумму выпавших очков.
Будем использовать **gosub** для рисования этих двух кубиков так, чтобы пришлось описать это только один раз.

```
 1 # roll2dice.kbs
 2 clg
 3 total = 0
 4
 5 x = 30
 6 y = 30
 7 roll = int(rand * 6) + 1
 8 total = total + roll
 9 gosub drawdie
10
11 x = 130
12 y = 130
13 roll = int(rand * 6) + 1
14 total = total + roll
15 gosub drawdie
16
17 print "вы выбросили " + total + " очков."
18 end
19
20 drawdie:
21 # используем: x,y -- левый верхний угол, roll -- число очков
22 # рисуем куб 70x70 с точками 10x10 пикселей
23 color black
24 rect x,y,70,70
25 color white
26 # верхний ряд
```

```
27 if roll <> 1 then rect x + 10, y + 10, 10, 10
28 if roll = 6 then rect x + 30, y + 10, 10, 10
29 if roll >= 4 and roll <= 6 then rect x + 50, y + 10, 10, 10
30 # средний ряд
31 if roll = 1 or roll = 3 or roll = 5 then rect x + 30, y + 30, 10, 10
32 # нижний ряд
33 if roll >= 4 and roll <= 6 then rect x + 10, y + 50, 10, 10
34 if roll = 6 then rect x + 30, y + 50, 10, 10
35 if roll <> 1 then rect x + 50, y + 50, 10, 10
36 return
```

Программа 53. Большая программа — бросаем два кубика.

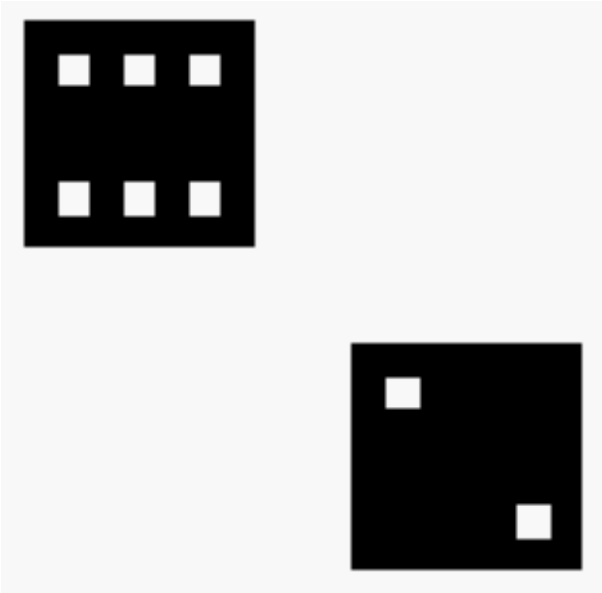


Рис. 9.3. Пример вывода программы 53 Большая программа — Бросаем два кубика.

Глава 10

Управляем мышкой, перемещаем объекты

В этой главе рассмотрим как можно заставить программу реагировать на мышку. Есть два различных способа использовать мышь: двигать её и нажимть на кнопки. Обе эти возможности обсуждаются на простых примерах.

10.1 Режим перемещения

Во время перемещения мышки можно использовать три числовые функции (**mousex**, **mousey** и **mouseb**), которые возвращают координаты указателя мыши в окне графического вывода. Если мышка вышла за пределы окна графического вывода, её перемещение не регистрируется, а функции возвращают последнее сохранённое значение.

```
 1 # mousetrack.kbs
 2 print "Подвигай мышкой по окну графического вывода."
 3 print "Щёлкни левой кнопкой, чтобы завершить работу."
 4
 5 fastgraphics
 6
 7 # Повторять, пока пользователь не нажмёт левую кнопку мышки
 8 while mouseb <> 1
 9     # очищаем экран
10     color white
11     rect 0, 0, graphwidth, graphheight
12     # рисуем мячик
13     color red
14     circle mousex, mousey, 10
15     refresh
16 end while
17
18 print "Завершено."
19 end
```

Программа 54. Мышиный след.

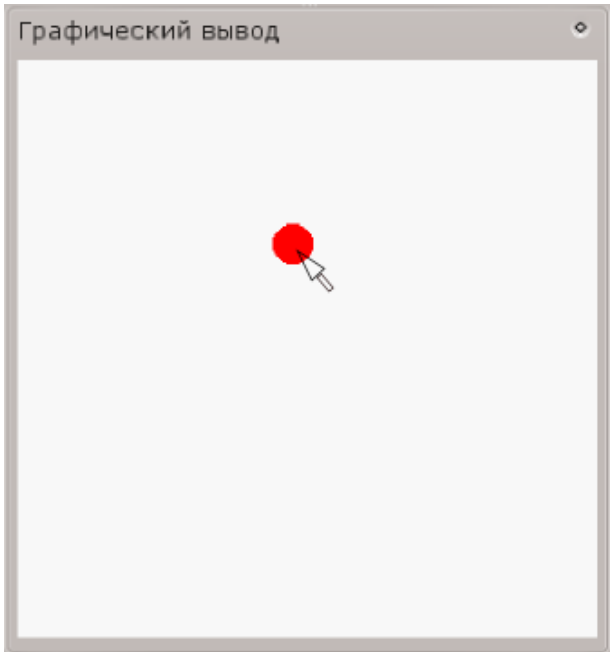


Рис. 10.1. Пример экрана программы 54. Мышиный след.

mousex or mousex()
mousey or mousey()
mouseb or mouseb()

Три функции для работы с манипулятором мышь возвращают текущее положение указателя мышки, когда он находится над областью графического отображения. Положение мышки вне этого экрана не фиксируется, но последние записанные данные возвращаются в программу.

mousex	Возвращает x-координату указателя мышки. Диапазон: от 0 до $graphwidth - 1$
mousey	Возвращает y-координату указателя мышки. Диапазон: от 0 до $graphheight - 1$
mouseb	<table><tr><td>0</td><td>Возвращает это значение, когда ни одна кнопка не нажата</td></tr><tr><td>1</td><td>Возвращает это значение, когда левая кнопка нажата</td></tr><tr><td>2</td><td>Возвращает это значение, когда правая кнопка нажата</td></tr><tr><td>4</td><td>Возвращает это значение, когда средняя кнопка нажата</td></tr></table> Если несколько кнопок нажаты одновременно, результатом будет сумма значений по каждой кнопке.

10.2 Режим щелчков

Второй режим для управления мышкой называется «режим щелчков». В этом режиме положение мышиного курсора и нажатых кнопок (или комбинации кнопок) запоминаются, когда происходит щелчок. Как только щелчок обработан программой, надо вызвать команду **clecklear**, чтобы программа могла регистрировать следующее нажатие на кнопку.

```
 1 # mouseclick.kbs
 2 # X отмечает место где вы щёлкнули мышкой
 3 print "Подвигайте мышкой в окне графического вывода"
 4 print "Щёлкните левой кнопкой, чтобы отметить точку"
 5 print "Щёлкните правой кнопкой, чтобы закончить."
 6 clg
 7 clickclear
 8 while clickb <> 2
 9     # стираем значения предыдущего нажатия на кнопку мыши
10     # и ждём следующего
11     clickclear
12     while clickb = 0
13         pause .01
14     end while
15     #
16     color blue
17     stamp clickx, clicky, 5, {-1, -2, 0, -1, 1, -2, 2, -1, 1, 0, 2,
   1, 1, 2, 0, 1, -1, 2, -2, 1, -1, 0, -2, -1}
18 end while
19 print "Завершено."
20 end
```

Программа 55. Щелчки мышкой.

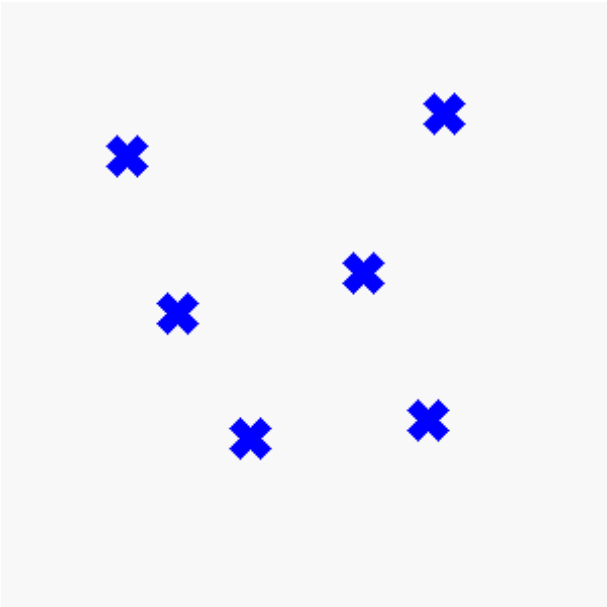


Рис. 10.2. Пример вывода программы 55. Щелчки мышкой


Новое понятие

clickx или **clickx()**
clicky или **clicky()**
clickb или **clickb()**

Значения этих трёх функций обновляются каждый раз, когда какая-нибудь кнопка мыши нажата и при этом указатель мыши находится внутри области графического вывода. Последнее местоположение мыши во время последнего нажатия кнопки доступно из этих трёх функций.


Новое понятие

clickclear

Функция **clickclear** сбрасывает (устанавливает в 0) значения функций **clickx**, **clicky** и **clickb** . Новый щелчок можно зарегистрировать по условию: *clickb* <> 0


Большая программа

Большая программа в этой главе показывает как с помощью перемещения ползунков на цветных шкалах можно выбрать любой цвет из 16777216 различных цветов экрана.

```
 1 # colorchooser.kbs
 2 fastgraphics
 3
 4 print "colorchooser - выбери цвет"
 5 print "нажми и протащи мышкой красный, зелёный и синий ползунок"
 6
 7 # Переменные для хранения красной (r), зелёной (g) и синей (b) составляющих цвета
 8 r = 128
 9 g = 128
10 b = 128
11
12 gosub display
13
14 while true
15 # ждём нажатия на кнопку мыши
16     while mouseb = 0
17         pause .01
18     end while
19 # изменяем положение ползунка
20     if mousey < 75 then
21         r = mousex
22         if r > 255 then r = 255
23     end if
24     if mousey >= 75 and mousey < 150 then
25         g = mousex
26         if g > 255 then g = 255
27     end if
28     if mousey >= 150 and mousey < 225 then
29         b = mousex
30         if b > 255 then b = 255
31     end if
32     gosub display
```



```
33 end while
34 end
35
36 display:
37 clg
38 # рисуем красным
39 color 255, 0, 0
40 font "Tahoma", 30, 100
41 text 260, 10, "r"
42 for t = 0 to 255
43     color t, 0, 0
44     line t,0,t,37
45     color t, g, b
46     line t, 38, t, 75
47 next t
48 color black
49 rect r-1, 0, 3, 75
50 # рисуем зелёным
51 color 0, 255, 0
52 font "Tahoma", 30, 100
53 text 260, 85, "g"
54 for t = 0 to 255
55     color 0, t, 0
56     line t,75,t, 75 + 37
57     color r, t, b
58     line t, 75 + 38, t, 75 + 75
59 next t
60 color black
61 rect g-1, 75, 3, 75
62 # рисуем синим
63 color 0, 0, 255
64 font "Tahoma", 30, 100
65 text 260, 160, "b"
66 for t = 0 to 255
67     color 0, 0, t
68     line t, 150, t, 150 + 37
69     color r, g, t
70     line t, 150 + 38, t, 150 + 75
71 next t
72 color black
73 rect b-1, 150, 3, 75
74 # рисуем получившийся образец цвета
75 color black
76 font "Tahoma", 15, 100
77 text 5, 235, "(" + r + "," + g + "," + b + ")"
78 color r,g,b
79 rect 151,226,150,75
80 refresh
81 return
```

Программа 56. Большая программа — Выбор цвета.

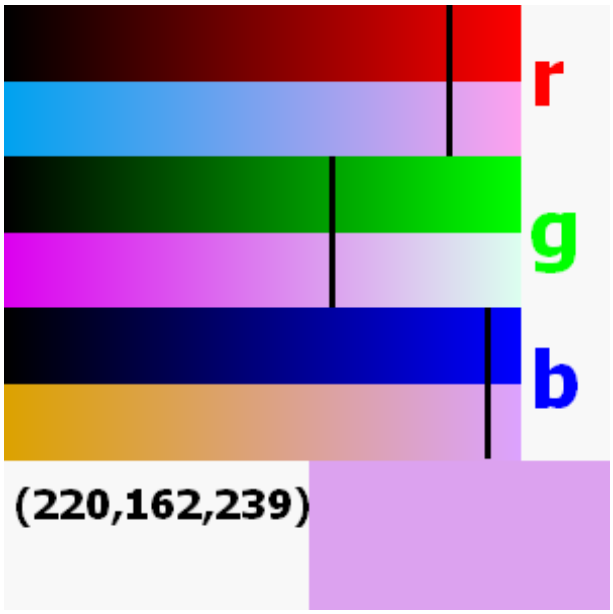


Рис. 10.3. Пример экрана программы 56. Большая программа — Выбор цвета.

Глава 11

Использование клавиатуры для управления программой

Эта глава показывает как заставить программу реагировать на нажатия клавиш (стрелок, букв и специальных символов) на клавиатуре.

11.1 Какая клавиша нажата последней?

Функция **key** возвращает последний сгенерированный системой код нажатой клавиши. Некоторые сигналы (такие как код сгенерированный комбинацией **CTRL+C** или функциональной клавишей **F1**), перехватываются самой средой BASIC256 и не возвращаются в программу. После передачи кода последнего нажатого символа функцией в программу её значение обнуляется (становится равным нулю), пока не будет нажата ещё какая-нибудь клавиша.

Значения функции **key** для печатных символов (0-9, знаки, буквы) равно юникод-значению символа в верхнем регистре независимо от состояния клавиш **CupsLock** и **Shift**.

```
 1 # readkey.kbs
 2 print "Нажми Q, чтобы закончить"
 3 do
 4     k = key
 5     if k <> 0 then
 6         if k >=32 and k <= 127 then
 7             print chr(k) + "=";
 8         endif
 9         print k
10     endif
11 until k = asc("Q")
12 end
```

Программа 57. Чтение символов клавиатуры.

```
Нажми Q, чтобы закончить
-1,
A=65,
S=83,
D=68,
F=70,
G=71,
16777248,
```

```
@=64,
16777248,
#=35,
3=51,
```

Пример вывода программы 57. Чтение символов клавиатуры.

key
key()

Функция **key** возвращает значение кода последней нажатой пользователем клавиши. Как только функция считала значение нажатой клавиши, оно становится равным нулю, чтобы обозначить тот факт, что никакая клавиша больше не нажата.

Unicode (юникод)

Стандарт Юникод придуман для того, чтобы присвоить числовые значения буквам или символам применяемым в мировых системах письменности. Стандарт Unicode 5.0 содержит более 107 000 различных символов. Смотри: http://www.unicode.org

asc(*выражение*)

Функция **asc** возвращает целое число, равное юникод-значению первого символа строки «*выражение*».

chr(*выражение*)

Функция **chr** возвращает строку, содержащую единственный символ, код которого в Юникод-таблице равен целому числу, которому равно «*выражение*»

Давайте рассмотрим более сложный пример. Программа 58 рисует красный мяч (круг) на экране, которым пользователь может управлять, используя клавиатуру.

```
 1 # moveball.kbs
 2 print "используй: i - вверх, j - влево, k - вправо, m - вниз, q - закончить"
 3
 4 fastgraphics
 5 clg
 6 ballradius = 20
 7
 8 # позиция мяча
 9 # начинаем в центре экрана
10 x = graphwidth /2
11 y = graphheight / 2
12
13 # рисуем мяч в начальном положении
14 gosub drawball
15
16 # цикл ожидания нажатий на клавиши
```



```
17 while true
18     k = key
19     if k = asc("I") then
20         y = y - ballradius
21         if y < ballradius then y = graphheight - ballradius
22         gosub drawball
23     end if
24     if k = asc("J") then
25         x = x - ballradius
26         if x < ballradius then x = graphwidth - ballradius
27         gosub drawball
28     end if
29     if k = asc("K") then
30         x = x + ballradius
31         if x > graphwidth - ballradius then x = ballradius
32         gosub drawball
33     end if
34     if k = asc("M") then
35         y = y + ballradius
36         if y > graphheight - ballradius then y = ballradius
37         gosub drawball
38     end if
39     if k = asc("Q") then end
40 end while
41
42 drawball:
43 color white
44 rect 0, 0, graphwidth, graphheight
45 color red
46 circle x, y, ballradius
47 refresh
48 return
```

Программа 58. Двигай мяч.[1]

Большая программа этой главы — игра, использующая клавиатуру. Буквы случайным образом падают по экрану, а вы набираете очки нажимая эти буквы так быстро, как только сможете.

```
 1 # fallinglettergame.kbs
 2 # скорость падения - чем число меньше, тем быстрее
 3 speed = .25
 4 nletters = 10 # количество букв в игре
 5
 6 score = 0
 7 misses = 0
 8 color black
 9
10 fastgraphics
11
```

[1] В данной программе поле представляет собой сферу: когда вы подводите мячик к верхнему краю и нажимаете на «I» он появляется снизу, аналогично при движении мячика вправо или влево или вниз. Хорошее упражнение — исправить программу так, чтобы поле было прямоугольником со стенками. (*прим. переводчика*).

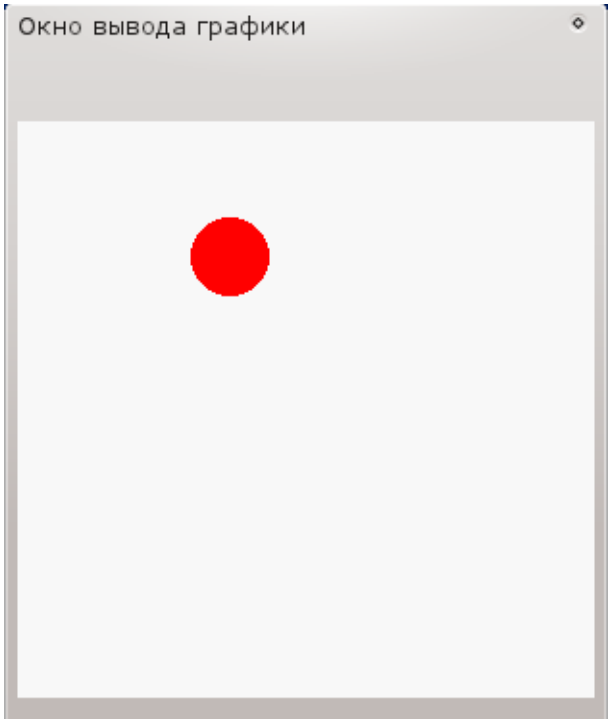


Рис. 11.1. Примерный вывод программы 58 Двигай мяч.

```
12 clg
13 font "Tahoma", 20, 50
14 text 20, 60, "Падающие буквы"
15 font "Tahoma", 18, 50
16 text 20, 100, "Нажми любую клавишу"
17 text 20, 140, "чтобы начать игру"
18 font "Tahoma", 16, 50
19 text 20, 180, "Включите английскую"
20 text 20, 220, "раскладку клавиатуры!"
21 refresh
22 # очищаем буфер клавиатуры и ждём нажатия на клавишу
23 k = key
24 while key = 0
25     pause speed
26 end while
27
28 for n = 1 to nletters
29     letter = int((rand * 26)) + asc("A")
30     x = 10 + rand * 225
31     for y = 0 to 250 step 20
32         clg
33         # показываем букву
34         font "Tahoma", 20, 50
35         text x, y, chr(letter)
36         # показываем счёт очков (score)
37         # и величину возможного выигрыша (value)
38         font "Tahoma", 12, 50
```



```
39          value = (250 - y)
40          text 10, 270, "Стоимость "+ value
41          text 200, 270, "Очки "+ score
42          refresh
43          k = key
44          if k <> 0 then
45              if k = letter then
46                  score = score + value
47              else
48                  score = score - value
49                  misses = misses + 1
50              end if
51              goto nextletter
52          end if
53          pause speed
54      next y
55      misses = misses + 1
56  nextletter:
57  next n
58
59  clg
60  font "Tahoma", 20, 50
61  text 20, 40, "Падающие буквы"
62  text 20, 80, "Игра окончена"
63  text 20, 120, "Очки: " + score
64  text 20, 160, "Промахи: " + misses
65  refresh
66  end
```

Программа 59. Большая программа — Падающие буквы.[2]

[2]В данную программу внесены изменения:

- строка 3: было `speed = .15` у меня, на процессоре 2 ггц, с таким значением буквы мелькают как сумасшедшие.
- строки 18-20 добавлены мной, поскольку программа выдаёт только латинские буквы. Сделать буквы русскими — неплохое упражнение!
- строки 13-17 подкорректированы, чтобы русский текст умещался в окне графического вывода
- строка 49 добавлена мной, поскольку без неё любое нажатие на клавишу (даже неверную) считается попаданием.

(*прим. переводчика*).

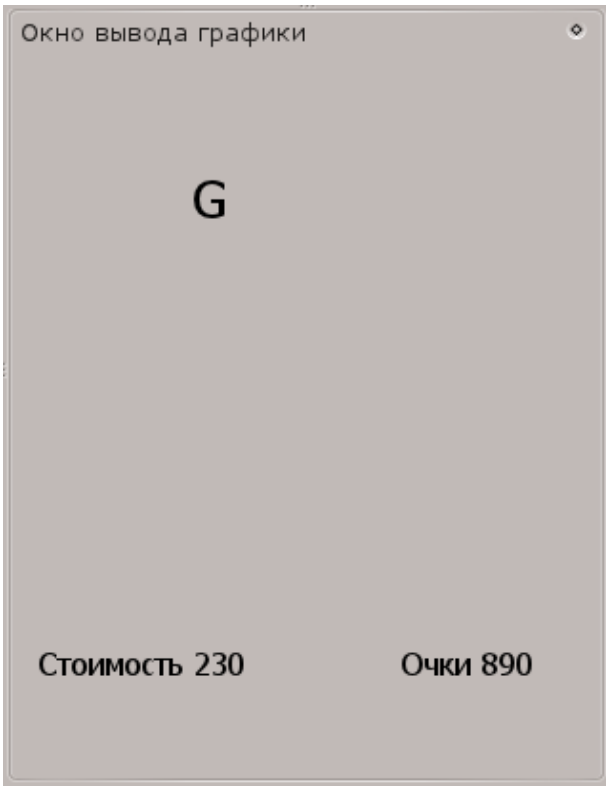


Рис. 11.2. Пример экрана програмы 59. Большая программа — Падающие буквы.

Глава 12

Картинки, музыка и спрайты

В этой главе мы начинаем изучать управление мультмедийным и графическим содержанием программ. Научимся загружать графические образы (картинки) из файлов, проигрывать асинхронно звуковые файлы типа WAV, а также создавать анимацию (мультфильмы) с использованием спрайтов.

12.1 Образы из файла

Ранее мы видели как создавать графические формы используя встроенные инструменты рисования. Функция imgload позволит вам загружать картинки прямо из файла и отображать их в ваших программах на BASIC-256.

```
1 # imgload_ball.kbs - Демонстрация функции Imgload
2 clg
3 for i = 1 to 50
4       imgload rand * graphwidth, rand * graphheight, "greenball.png"
5 next i
```

Программа 60. Загрузка графического файла.

Програма 60 показывает действие **imgload**. Последний аргумент этой функции — имя графического файла на вашем компьютере. Необходимо, чтобы это файл был в одном и том же каталоге с программой, иначе надо указать полный путь к файлу. Обратите внимание, что координаты (x,y) — это центр загруженной картинки, а не верхний левый угол.

В большинстве случаев удобно сохранять файлы с картинками и звуками в том же каталоге, что и программа **до того**, как вы её запустите. Это задаёт текущий рабочий каталог программы так, что BASIC-256 сможет эти файлы найти и загрузить.

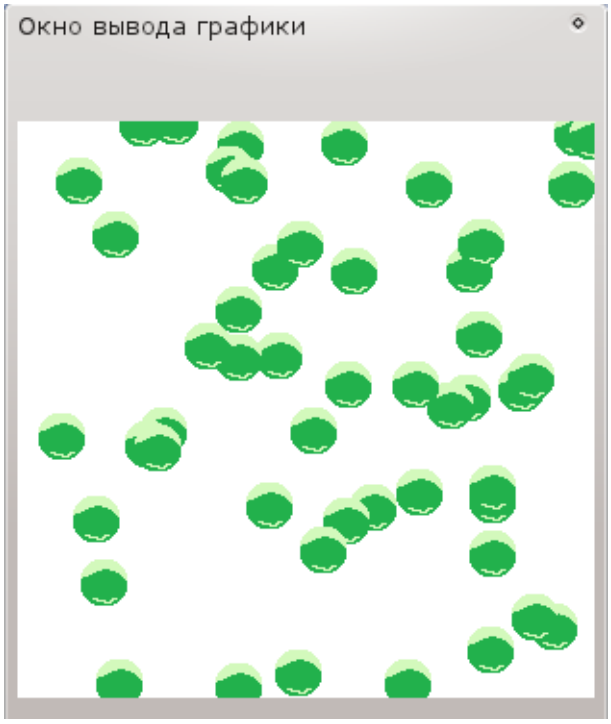


Рис. 12.1. Пример вывода программы 60. Загрузка графического файла.

imgload x, y, имя_файла
imgload x, y, коэфф_искажения, имя_файла
imgload x, y, коэфф_искажения, угол, имя_файла

Функция читает картинку из файла и показывает её в области графического вывода. Значения x и y соответствуют **центру** образа.
Образы могут быть различных форматов: *BMP*, *PNG*, *GIF*, *JPG*, и*JPEG*
Дополнительно можно изменить размер картинки задав десятичное число *коэфф_искажения*. Коэффициент равный 1 означает полный размер. Если вы хотите предварительно повернуть картинку, задайте *угол* поворота по часовой стрелке в радианах (от 0 до 2π)

Функция **impload** также позволяет масштабировать картинку и поворачивать её, как это делает **stamp**. В качестве примера посмотрите программу 61.

```
1 # imgload_picasso.kbs
2 # Загрузка картинки с вращением и масштабированием
3 graphsize 500,500
4 clg
5 for i = 1 to 50
6     imgload graphwidth/2, graphheight/2, i/50, 2*pi*i/50, "picasso_
   selfport1907.jpg"
7 next i
8 say "Здравствуй, Пикассо."
```

Программа 61. Загрузка картинки с масштабированием и вращением.

Рис. 12.2. Примерный вывод программы 61. Загрузка картинки с масштабированием и вращением.

12.2 Воспроизведение звуков из файлов типа WAV

До сих пор для извлечения звуков мы использовали команду **sound** и команду **say** преобразования текста в звук. BASIC-256 также умеет воспроизводить звуки, записанные в WAV-файл. Воспроизведение звука из WAV-файла происходит в фоновом режиме. Программа продолжает выполнять команды идущие после команды запуска звука, при этом звук также продолжает воспроизводиться.

```
 1 # spinner.kbs
 2 fastgraphics
 3 wavplay "roll.wav"
 4
 5 # задаём параметры вращающегося луча
 6 angle = rand * 2 * pi
 7 speed = rand * 2
 8 color darkred
 9 rect 0,0,300,300
10
11 for t = 1 to 100
12 # рисуем вращающийся луч
13     color white
14     circle 150,150,150
15     color black
16     line 150,300,150,0
17     line 300,150,0,150
18     text 100,100,"A"
19     text 200,100,"B"
20     text 200,200,"C"
21     text 100,200,"D"
22     color darkgreen
```

```
23      line 150,150,150 + cos(angle)*150, 150 + sin(angle)*150
24      refresh
25  # обновляем угол поворота для следующего положения луча
26      angle = angle + speed
27      speed = speed * .9
28      pause .05
29  next t
30
31  # ждём, пока перестанет звучать воспроизводимый звуковой файл.
32  wavwait
```

Программа 62. Вращающийся луч (радар) со звуковым сопровождением.

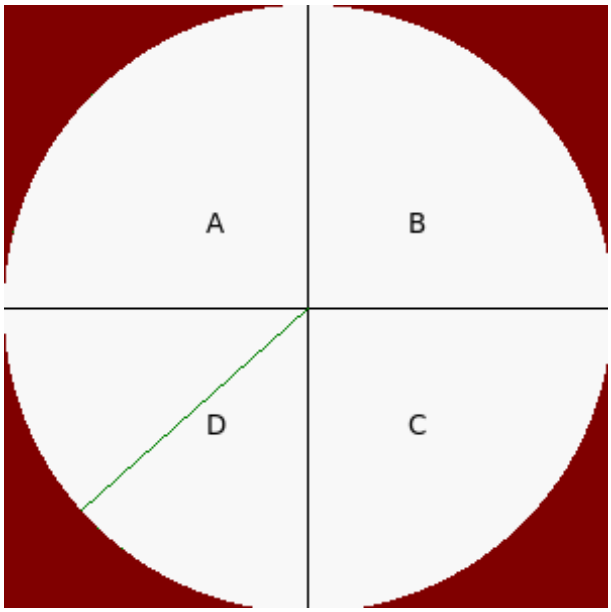


Рис. 12.3. Примерный вывод программы 62. Вращающийся луч (радар) со звуковым сопровождением.

wavplay имя_файла
wavwait
wavstop


Новое понятие

Функция **wavplay** загружает файл типа *.wav* из текущего каталога и воспроизводит его. Следующая инструкция программы выполняется сразу же как начинается воспроизведение звука.

wavstop — останавливает воспроизведение текущего файла.
wavwait — останавливает исполнение дальнейших операторов программы, пока не закончится воспроизведение текущего звукового файла.

12.3 Перемещение картинок — спрайты

Спрайты — специальные графические объекты, которые можно перемещать по экрану без необходимости перерисовки экрана. Кроме этого можно определить когда два спрайта перекрываются (сталкиваются). Спрайты дают возможность более лёгкого программирования сложных игр и анимаций.

```
 1 # sprite_1ball.kbs
 2
 3 color white
 4 rect 0, 0, graphwidth, graphheight
 5
 6 spritedim 1
 7
 8 spriteload 0, "blueball.png"
 9 spriteplace 0, 100,100
10 spriteshow 0
11
12 dx = rand * 10
13 dy = rand * 10
14
15 while true
16   if spritex(0) <=0 or spritex(0) >= graphwidth -1 then
17       dx = dx * -1
18       wavplay "4359__NoiseCollector__PongBlipF4.wav"
19   end if
20   if spritey(0) <= 0 or spritey(0) >= graphheight -1 then
21       dy = dy * -1
22       wavplay "4361__NoiseCollector__pongblipA_3.wav"
23   endif
24   spritemove 0, dx, dy
25   pause .05
26 end while
```

Программа 63. Прыгающий мяч с использованием спрайтов и звуковых эффектов.[1]

.

По коду программы 63 видно, что достижение анимационного эффекта прыгающего мяча со звуком проще, чем это можно было бы реализовать ранее. Для использования спрайтов, необходимо сообщить программе на BASIC-256 сколько их будет (функция **spritedim**), определить спрайт (командой **spriteload** или **spriteplace**), затем сделать его видимым (командой **spriteshow**), и затем только перемещать (командой **spritemove**). Кроме этого, есть ещё функции, позволяющие определить положение спрайта на экране (команды **spritex** и **spritey**), его размеры (команды **spritew** и **spritey**) и его видимость (команда **spritev**).

spritedim количество_спрайтов

Функция **spritedim** резервирует в памяти место для указанного количества спрайтов. Вы можете создать столько спрайтов, сколько нужно, однако, если вы создадите слишком много спрайтов, программа будет работать очень медленно.

[1] Автор не учитывает размер мяча, что приводит к кажущемуся запаздыванию звука. Столкновение мяча со стенкой стоит определять не по центру шара (спрайта) а по его краю, а именно в строках 16 и 20 соответственно вычитать или прибавлять радиус шара (spritew(0)/2) (*прим. переводчика*).

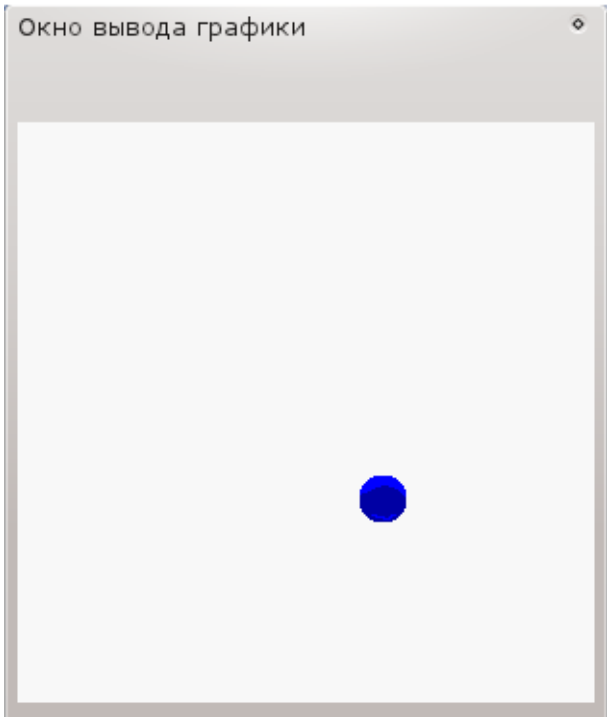


Рис. 12.4. Примерный вывод программы 63 Прыгающий мяч с использованием спрайтов и звуковых эффектов.

spriteload номер_спрайта, имя_файла
Эта команда загружает графический файл (форматов GIF, BMP, PNG, JPG, или JPEG) с учётом пути (если указан) к файлу, и создаёт спрайт.
По умолчанию спрайт помещается на экран так, что его центр находится в точке (0,0) и спрайт скрыт. Необходимо его переместить в нужную позицию экрана (используя **spritemove** или **spriteplace**) и затем показать его (используя **spriteshow**)

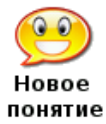
spritehide номер_спрайта
spriteshow номер_спрайта

Команда **spriteshow** показывает на экране графического вывода предварительно загруженный, созданный или спрятанный спрайт.
Команда **spritehide** делает указанный (по номеру) спрайт невидимым на экране. Этот спрайт по прежнему существует и его можно будет показать позже.

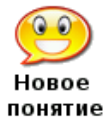
spriteplace номер_спрайта, x, y

Эта команда помещает центр спрайта в указанную координатами x и y точку экрана.

spritemove номер_спрайта, dx, dy

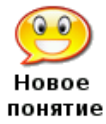


Эта команда перемещает спрайт на dx пикселей вправо и на dy пикселей вниз. Отрицательные значения также можно использовать для обозначения перемещения спрайта влево и вверх.
Центр спрайта не может выйти за границы окна графического вывода т.е. всегда в пределах от (0,0) до (graphwidth-1, graphheight-1).
Можно перемещать скрытый спрайт, но его не будет видно, пока вы не покажете его используя команду **showsprite**.



spritev(номер_спрайта)

Эта функция возвращает **true** (истина), если спрайт виден в графической области экрана и **false** (ложь), когда спрайт не виден.



spriteh(номер_спрайта)
spritew(номер_спрайта)
spritex(номер_спрайта)
spritey(номер_спрайта)

Эти функции возвращают различную информацию о загруженном спрайте **spriteh** — возвращает высоту спрайта в пикселях **spritew** — возвращает ширину спрайта в пикселях **spritex** — возвращает координату x центра спрайта **spritey** — возвращает координату y центра спрайта

Во втором примере (Программа 64) у нас будет два спрайта. Первый спрайт (номер нуль) — неподвижный, а второй (номер один) будет прыгать, отражаясь от стенок и неподвижного спрайта.

```
 1 # sprite_bumper.kbs
 2
 3 color white
 4 rect 0, 0, graphwidth, graphheight
 5
 6 spritedim 2
 7
 8 # спрайт-отбойник
 9 spriteload 0, "paddle.png"
10 spriteplace 0,graphwidth/2,graphheight/2
11 spriteshow 0
12
13 # подвижный мяч
14 spriteload 1, "blueball.png"
15 spriteplace 1, 50, 50
16 spriteshow 1
17 dx = rand * 5 + 5
18 dy = rand * 5 + 5
19
20 while true
21       if spritex(1) <=0 or spritex(1) >= graphwidth -1 then
```

```
22          dx = dx * -1
23      end if
24      if spritey(1) <= 0 or spritey(1) >= graphheight -1 then
25          dy = dy * -1
26      end if
27      if spritecollide(0,1) then
28          dy = dy * -1
29          print Бабах"-!"
30      end if
31      spritemove 1, dx, dy
32      pause .05
33 end while
```

Программа 64. Столкновение спрайтов.[2]

.

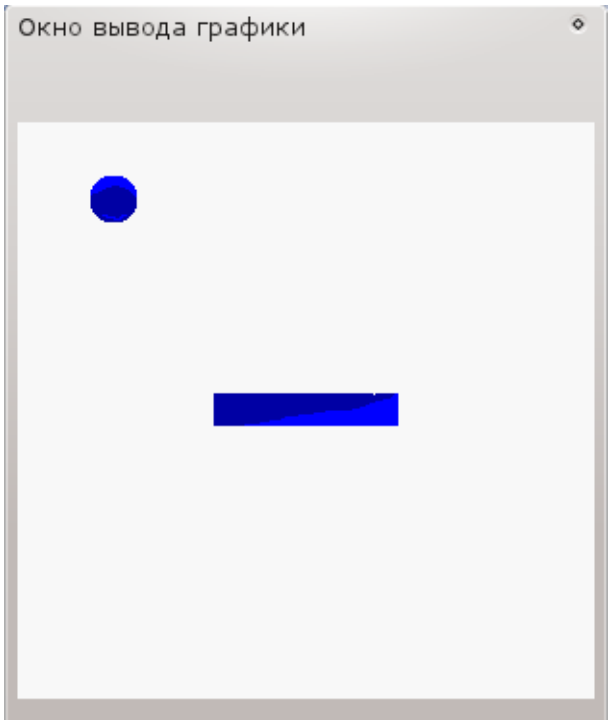


Рис. 12.5. Примерный экран программы 64 Столкновение спрайтов.

spritecollide(номер_первого, номер_второго)

Эта функция возвращает **true** (истину), если два спрайта столкнулись или один перекрывает другой[3].

[2] Автор опять, видимо в целях упрощения программы, не учитывает соударение неподвижного спрайта с подвижным, когда удар приходится в левую или правую стенку прямоугольника, изображающего неподвижный спрайт. В этот момент, подвижный спрайт начинает скакать вдоль неподвижного. В качестве упражнения, предлагаем поправить код (*прим. переводчика*).

Большая программа данной главы использует спрайты и звуки для создания игры типа пинг-понг.

```
 1 # sprite_paddleball.kbs
 2
 3 color white
 4 rect 0, 0, graphwidth, graphheight
 5
 6 spritedim 2
 7
 8 spriteload 1, "greenball.png"
 9 spriteplace 1, 100,100
10 spriteshow 1
11 spriteload 0, "paddle.png"
12 spriteplace 0, 100,270
13 spriteshow 0
14
15 dx = rand * .5 + .25
16 dy = rand * .5 + .25
17
18 bounces = 0
19
20 while spritey(1) < graphheight -1
21      k = key
22      if chr(k) = "K" then
23          spritemove 0, 20, 0
24      end if
25      if chr(k) = "J" then
26          spritemove 0, -20, 0
27      end if
28      if spritecollide(0,1) then
29          # отскок назад с увеличением скорости
30          dy = dy * -1
31          dx = dx * 1.1
32          bounces = bounces + 1
33          wavstop
34          wavplay "96633__CGEffex__Ricochet_metal5.wav"
35          # отражение спрайта мяча от ракетки
36          while spritecollide(0,1)
37              spritemove 1, dx, dy
38          end while
39      end if
40      if spritex(1) <=0 or spritex(1) >= graphwidth -1 then
41          dx = dx * -1
42          wavstop
43          wavplay "4359__NoiseCollector__PongBlipF4.wav"
44      end if
45      if spritey(1) <= 0 then
46          dy = dy * -1
47          wavstop
```

[3]Функция слишком примитивна. На самом дела она вычисляет столкнулись ли два прямоугольника, описанных около спрайта. Если в примере 64 неподвижный спрайт будет в виде тонкой палочки, идущей по диагонали, вы наглядно убедитесь в бесполезности этой функции.(*прим. переводчика*).

```
48          wavplay "4361__NoiseCollector__pongblipA_3.wav"
49      end if
50      spritemove 1, dx, dy
51 end while
52
53 print "Вы отбили мяч " + bounces + " раз(а)."
```

Программа 65. Пинг-понг.

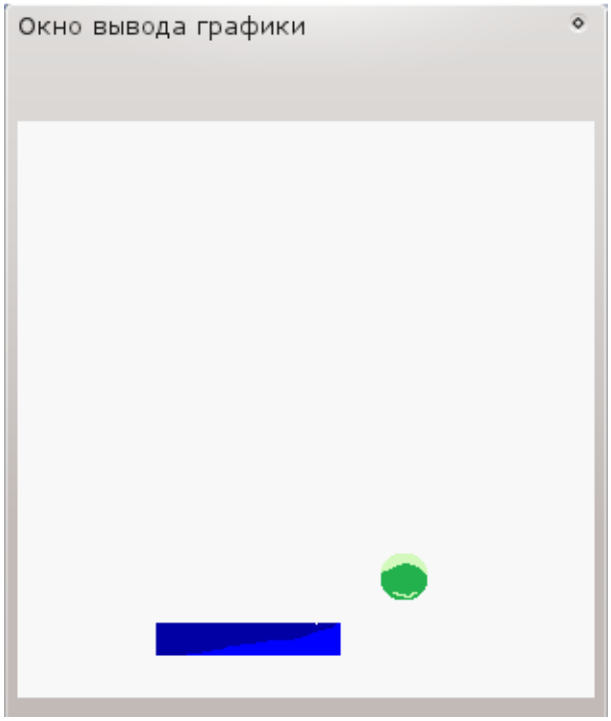


Рис. 12.6. Примерный вывод программы 65 Пинг-понг.

Глава 13

Массивы — коллекции данных

Мы использовали простые строковые и числовые переменные во многих программах, но они могут содержать только одно значение за раз. Очень часто приходится иметь дело с набором или списком значений. Для работы с такими объектами придуманы линейные (одномерные) и двухмерные массивы. В этой главе вы узнаете как создавать, инициализировать (задавать начальные значения), использовать массивы, а также как изменять их размеры.

13.1 Одномерный числовой массив.

Одномерный массив создаёт список в памяти так, что вы можете обращаться к его элементам по числовому адресу, называемому индексом. Массивы могут быть как числовыми, так и строковыми в зависимости от типа переменной, использованной в операторе **dim**.

```
 1 # numeric1d.kbs
 2
 3 dim a(10)
 4
 5 a[0] = 100
 6 a[1] = 200
 7 a[3] = a[1] + a[2]
 8
 9 input "Введите число: ", a[9]
10 a[8] = a[9] - a[3]
11
12 for t = 0 to 9
13     print "a[" + t + "] = " + a[t]
14 next t
```

Программа 66. Одномерный числовой массив.

```
Введите число: 63
a[0] = 100
a[1] = 200
a[2] = 0
a[3] = 200
a[4] = 0
a[5] = 0
a[6] = 0
a[7] = 0
```

```
a[8] = -137
a[9] = 63
```

Примерный вывод программы 66. Одномерный числовой массив.

dim variable(количество_элементов)
dim variable$(количество_элементов)
dim variable(количество_рядов, количество_столбцов)
dim variable$(количество_рядов, количество_столбцов)

Оператор **dim** создаёт массив в оперативной памяти компьютера размером равном указанному в скобках и именем variable или variable$ (можно задать своё). Размер массива (количество элементов, строк и рядов) должно быть целым положительным числом.
Оператор **dim** автоматически назначает значение элементам массива равное нулю (0), если массив числовой или равное пустой строке (""), если массив строковый.

variable[номер_элемента]
variable[номер_строки, номер_столбца]
variable$[номер_элемента]
variable$[номер_строки, номер_столбца]
где **variable** или **variable$** — имя массива.

Вы можете делать ссылки на элементы массива с помощью его имени и индекса внутри квадратных скобок в любом месте, где можно использовать простые переменные. Индексы должны быть целым числом от 0 до уменьшенного на один значения указанного в операторе **dim**
Вас может смутить тот факт, что BASIC-256 использует нуль (0) для первого элемента массива и число на единицу меньшее, чем его размер для последнего, но так принято в программировании. Массивы с отсчётом от нуля — так называют такие объекты программисты.

Мы можем использовать числовые массивы для рисования большого количества одновременно прыгающих на экране мячиков. Программа 67 использует 5 массивов для хранения местоположения каждого из мячей, его направления движения и цвета. С помощью циклов мы задаём начальные значения этим массивам, а также заставляем мячи двигаться. Эта программа также использует функцию **rgb()**, которая определяет и сохраняет цвет каждого мяча.

```
 1 # manyballbounce.kbs
 2 fastgraphics
 3
 4 r = 10  # размер мяча
 5 balls = 50  # количество мячей
 6
 7 dim x(balls)
 8 dim y(balls)
 9 dim dx(balls)
10 dim dy(balls)
11 dim colors(balls)
```



```
12
13 for b = 0 to balls-1
14       # начальная позиция мяча
15       x[b] = 0
16       y[b] = 0
17       # скорость по осям x и y
18       dx[b] = rand * r + 2
19       dy[b] = rand * r + 2
20       # каждому мячу - свой цвет!
21       colors[b] = rgb(rand*256, rand*256, rand*256)
22 next b
23
24 color green
25 rect 0,0,300,300
26
27 while true
28       # очистим экран
29       clg
30       # позиционируем и рисуем мячики
31       for b = 0 to balls -1
32            x[b] = x[b] + dx[b]
33            y[b] = y[b] + dy[b]
34 # если мяч выходит за боковые границы поля - поворачиваем его
35            if x[b] < 0 or x[b] > 300 then
36                 dx[b] = dx[b] * -1
37            end if
38 # если мяч выходит за верхнюю/нижнюю границу - поворачиваем его
39            if y[b] < 0 or y[b] > 300 then
40                 dy[b] = dy[b] * -1
41            end if
42            # рисуем новый мяч
43            color colors[b]
44            circle x[b],y[b],r
45       next b
46       # обновляем экран
47       refresh
48       pause .05
49 end while
```

Программа 67. Много прыгающих мячиков.

rgb (красный, зелёный, синий)

Функция **rgb** возвращает одно число, которое представляет цвет, вычисленный из трёх значений (красного, зелёного и синего). Аргументами этой функции могут быть арифметические выражения, но с одним условием — их значение должно быть в интервале от 0 до 255.

Ещё один вариант реализации прыгающих мячей приведён в программе 68. В этом примере используются спрайты и два массива для хранения их траекторий.

```
1 #manyballsprite.kbs
2
3 # Ещё один способ заставить прыгать мячи с использованием спрайтов
4
```

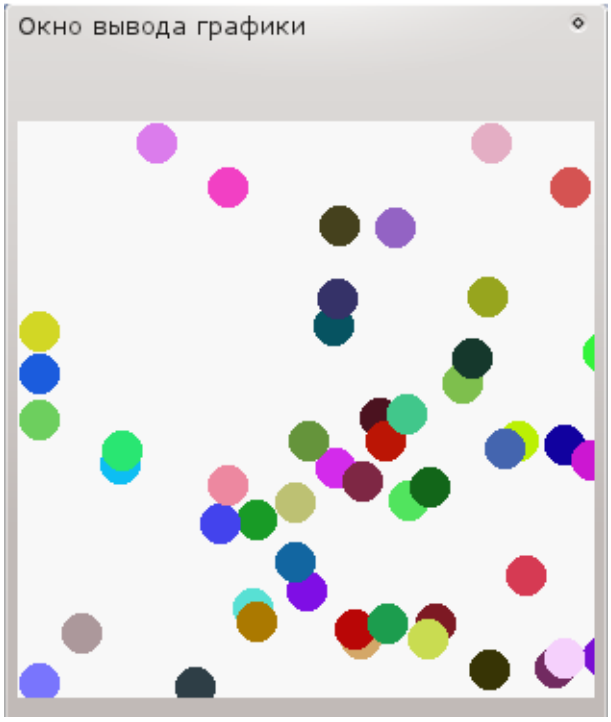


Рис. 13.1. Пример экрана программы 67. Много прыгающих мячиков.

```
 5 fastgraphics
 6 color white
 7 rect 0, 0, graphwidth, graphheight
 8
 9 n = 20
10 spritedim n
11
12 dim dx(n)
13 dim dy(n)
14
15 for b = 0 to n-1
16      spriteload b, "greenball.png"
17      spriteplace b,graphwidth/2,graphheight/2
18      spriteshow b
19      dx[b] = rand * 5 + 2
20      dy[b] = rand * 5 + 2
21 next b
22
23 while true
24      for b = 0 to n-1
25          if spritex(b) <=0 or spritex(b) >= graphwidth -1 then
26              dx[b] = dx[b] * -1
27          end if
28          if spritey(b) <= 0 or spritey(b) >= graphheight -1 then
29              dy[b] = dy[b] * -1
30          end if
31          spritemove b, dx[b], dy[b]
```



```
32      next b
33      refresh
34 end while
```

Программа 68. Много прыгающих мячиков с использованием спрайтов.

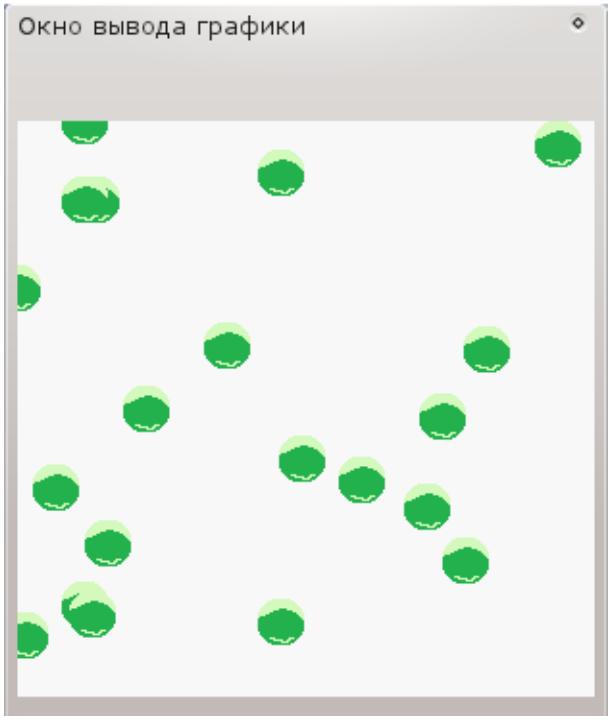


Рис. 13.2. Пример вывода программы 68. Много прыгающих мячиков с использованием спрайтов.

13.2 Массивы строк

Массивы могут хранить не только числа, но и строки. Чтобы открыть строковый массив необходимо использовать идентификатор строковой переменной в операторе **dim**. Всё, что мы знаем про числовые массивы применимо и к строковым с учётом различия в типах данных. Использование строковых массивов показано в программе 69.

```
 1 # listoffriends.kbs
 2 print "Составь список друзей"
 3 input "Сколько всего друзей?", n
 4
 5 dim names$(n)
 6
 7 for i = 0 to n-1
 8     input "Напиши имя друга ?", names$[i]
 9 next i
10
11 cls
```

```
12 print "Мои друзья"
13 for i = 0 to n-1
14     print "Друг номер ";
15     print i + 1;
16     print " это " + names$[i]
17 next i
```

Программа 69. Список друзей.

```
Составь список друзей
Сколько всего друзей?3
Напиши имя друга ?Иван
Напиши имя друга ?Артём
Напиши имя друга ?Денис
-- очистка экрана --
Мои друзья
Друг номер 1, это Иван
Друг номер 2, это Артём
Друг номер 3, это Денис
```

Пример вывода программы 69. Список друзей.

13.3 Присваивание значений массивам

Мы использовали фигурные скобки ({}) для воспроизведения музыки, рисования многоугольников и штампов. Фигурные скобки можно также использовать для задания конкретных значений массивам.

```
 1 # arrayassign.kbs
 2 dim number(3)
 3 dim name$(3)
 4
 5 number = {1, 2, 3}
 6 name$ = {"Юра", "Аня", "Петя"}
 7
 8 for i = 0 to 2
 9     print number[i] + " " + name$[i]
10 next i
```

Программа 70. Задание значений массива списком.

```
1 Юра
2 Аня
3 Петя
```

Пример вывода программы 70. Задание значений массива списком.

Новое понятие

array = {знач0, знач1, ...}
array$ = {текст0, текст1, ...}

Массиву можно задать значения (начиная с индекса 0) с помощью списка значений, заключённых в фигурные скобки. Это применимо как к числовым, так и строковым массивам.

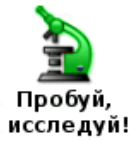
13.4 Звуки и массивы

В главе 3 мы видели как использовать список частот и длительностей (заключённых в фигурные скобки) для проигрывания нескольких звуков одновременно. Команда **sound** также может принять список частот и длительностей из массива. Массив должен иметь чётное число элементов, причём частоты хранятся в элементах с чётными индексами (0, 2, 4, ...), а длительности в элементах с нечётными индексами (1, 3, 5, ...)

Следующий пример (программа 71) ниже использует простую линейную зависимость для создания фонового звука типа чириканья.

```
 1 # spacechirp.kbs
 2
 3 # чётные значения 0,2,4,... - частота звука
 4 # нечётные значения 1,3,5,... - длительность
 5
 6 # чириканье начинается со 100гц и увеличивается на 40
 7 # для каждого из 50 звуков в списке, длительность всегда равна 10
 8 dim a(100)
 9 for i = 0 to 98 step 2
10     a[i] = i * 40 + 100
11     a[i+1] = 10
12 next i
13 sound a
```

Программа 71. Космическое чириканье.



Как много разных фантастических звуков ты сможешь запрограммировать? Поэкспериментируй с предложенными формулами для изменения частот и длительностей.

13.5 Графики и массивы

В главе 8 мы использовали списки для создания многоугольников и штампов. Массивы также применимы для этой цели, что приведёт к упрощению кода. Достаточно один раз определить штамп или многоугольник, сохраняя данные в массиве, и использовать этот массив в разных местах программы.

В массиве, используемом для штампов и многоугольников чётные элементы (0, 2, 4,...) содержат абсциссы (x), а нечётные элементы (1, 3, 5, ...) содержат ординату (y) точек, образующих фигуру, по два значения на каждую точку.

```
 1 # shadowstamp.kbs
 2
 3 dim xmark(24)
 4 xmark = {-1,-2,0,-1,1,-2,2,-1,1,0,2,1,1,2,0,1,-1,2,-2,1,-1,0,-2,-1}
 5
 6 clg
 7 color grey
 8 stamp 160,165,50,xmark
 9 color black
10 stamp 150,150,50,xmark
```

Программа 72. Штамп с тенью.

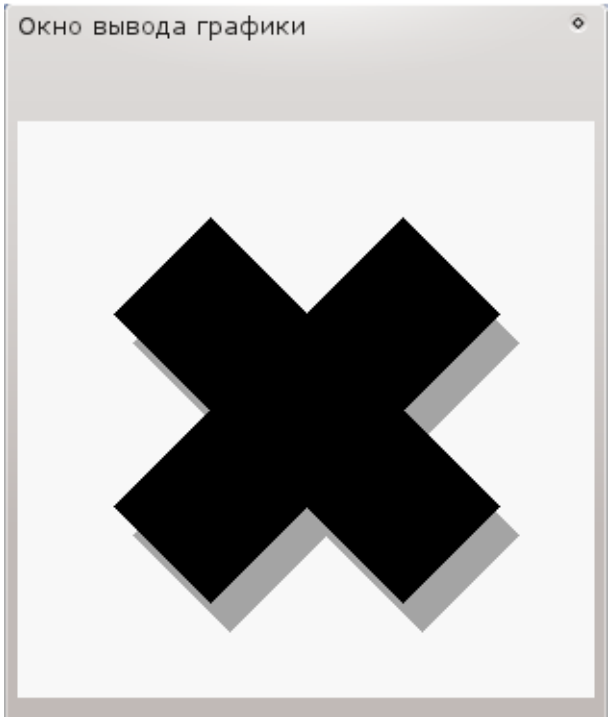


Рис. 13.3. Пример вывода программы 72. Штамп с тенью.

Массивы также можно использовать, чтобы создать многоугольник математически (по формулам). В программе 73 мы создаём массив из 10 элементов (5 точек) придавая им случайные значения. BASIC-256 заполняет эти формы как умеет, но если линии сторон пересекаются (невыпуклый многоугольник), заполнение может оставлять пустоты и дырки.

```
 1 # mathpoly.kbs
 2
 3 dim shape(10)
 4
 5 for t = 0 to 8 step 2
 6     x = 300 * rand
 7     y = 300 * rand
 8     shape[t] = x
 9     shape[t+1] = y
10 next t
11
12 clg
13 color black
14 poly shape
```

Программа 73. Создание случайного прямоугольника.

13.6 Для продвинутых: Двумерные массивы

До сих пор в этой главе мы использовали массивы, как простые списки чисел или строк. Такие массивы называются одномерными, поскольку они похожи на линейку значений.

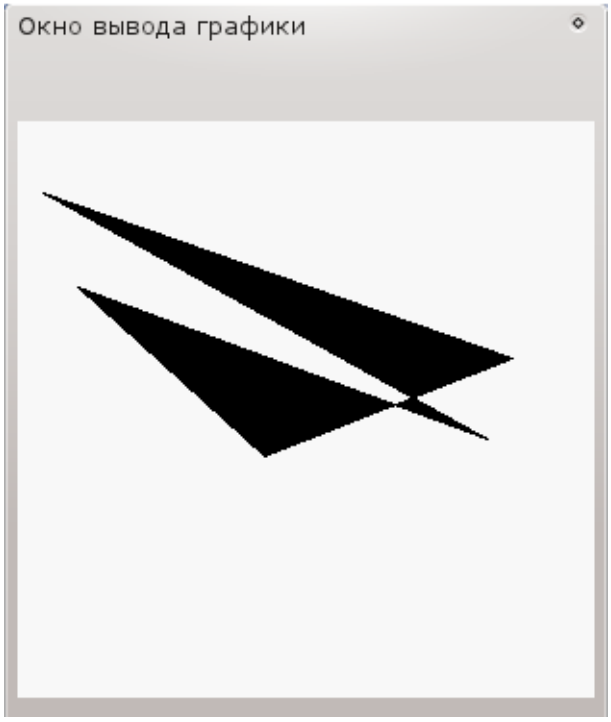


Рис. 13.4. Пример вывода программы 73. Создание случайного прямоугольника.

Таблица 13.1. Таблица к программе 74[1]

	Оценка 1	Оценка 2	Оценка 3	Оценка 4	Среднее
Джим	90	92	81	55	?
Сью	66	99	98	88	?
Тони	79	81	87	73	?

Массивы также могут быть созданы с использованием строк и столбцов данных — такие массивы называют двумерными. Программа 74 использует одномерные и двумерные массивы для вычисления средней оценки студента. (см таблицу 13.1 к программе 74)

```
 1 # grades.kbs
 2 # вычисление средней оценки студента
 3 # и для всего класса
 4
 5 nstudents = 3 # количество студентов
 6 nscores = 4 # количество оценок у студента
 7
 8 dim students$(nstudents)
 9
10 dim grades(nstudents, nscores)
11 # сохраняем оценки студента в столбцах одного ряда
12 # первый студент
13 students$[0] = "Джим"
```

[1]В школах многих зарубежных стран используется 100-бальная система оценок (*прим. переводчика*)

```
14 grades[0,0] = 90
15 grades[0,1] = 92
16 grades[0,2] = 81
17 grades[0,3] = 55
18 # второй студент
19 students$[1] = "Сью"
20 grades[1,0] = 66
21 grades[1,1] = 99
22 grades[1,2] = 98
23 grades[1,3] = 88
24 # третий студент
25 students$[2] = "Тони"
26 grades[2,0] = 79
27 grades[2,1] = 81
28 grades[2,2] = 87
29 grades[2,3] = 73
30
31 total = 0
32 for row = 0 to nstudents-1
33     studenttotal = 0
34     for column = 0 to nscores-1
35         studenttotal = studenttotal + grades[row, column]
36         total = total + grades[row, column]
37     next column
38     print students$[row] + " - среднееравно  ";
39     print studenttotal / nscores
40 next row
41 print "Среднее по классу ";
42 print total / (nscores * nstudents)
43
44 end
```

Программа 74. Вычисление успеваемости.

```
Джим - среднее равно 79,5
Сью - среднее равно 87,75
Тони - среднее равно 80,
Среднее по классу 82,416667
```

Пример вывода программы 74. Вычисление успеваемости.

13.7 Для действительно продвинутых — Размеры массива

Иногда необходимо в программе использовать массив, размеры которого мы не знаем. Если поставить вместо индекса внутри квадратных скобок знак вопроса, BASIC-256 вернёт размер этого массива. В программе 75 мы распечатываем массив независимо от размера. В строке 16 обратите внимание на специальный знак **[?]** использованный для получения размера массива.

```
 1 # size.kbs
 2 dim number(3)
 3 number = {77, 55, 33}
 4 print "до"
 5 gosub shownumberarray
```



```
 6
 7 # добавляем новый элемент в конец
 8 redim number(4)
 9 number[3] = 22
10 print "после"
11 gosub shownumberarray
12 #
13 end
14 #
15 shownumberarray:
16 for i = 0 to number[?] - 1
17     print i + " " + number[i]
18 next i
19 return
```

Программа 75. Получение размера массива

```
до
0,
77,
1,
55,
2,
33,
после
0,
77,
1,
55,
2,
33,
3,
22,
```

Пример вывода программы 75 Получение размера массива

array[?]
array$[?]
array[?,]
array$[?,]
array[,?]
array$[,?]

Ссылка на [?] элемент массива возвращает длину одномерного массива или полное число элементов (количество строк умноженное на количество столбцов) двумерного массива. [?,] — возвращает количество строк, а [,?] — возвращает количество столбцов двумерного массива.

13.8 Для очень продвинутых — Динамические массивы

BASIC-256 позволяет изменять размер существующего массива. Оператор **redim** предназначен для изменения массива с сохранением существующих данных. Если новый массив больше старого, новые элементы заполняются нулями (0) или пустыми строками (""). Если

новый массив меньше старого, значения, которые выходят за рамки нового массива обрезаются. Массивы, размеры которых можно менять во время работы программы называются динамическими[2].

```
 1 # redim.kbs
 2 dim number(3)
 3 number = {77, 55, 33}
 4 # создаём новый элемент в конце массива
 5 redim number(4)
 6 number[3] = 22
 7 #
 8 for i = 0 to 3
 9       print i + " " + number[i]
10 next i
```

Программа 76. Изменение размера массива.

```
0  77
1  55
2  33
3  22
```

Пример вывода программы 76. Изменение размера массива.

Новое понятие

redim variable(количество_элементов)
redim variable$(количество_элементов)
redim variable(число_рядов, число_колонок)
redim variable$(число_рядов, число_колонок)

Оператор **redim** изменяет размер массива в памяти компьютера. Данные, хранившиеся в массиве сохраняются, если они подходят по размеру к новому массиву.
Когда изменяется размер двухмерного массива данные копируются линейно, и могут нежелательно сдвинутся, если вы измените количество колонок.

Большая программа

Большая программа в этой главе использует три числовых массива для хранения позиций и скоростей падающего космического мусора. Тут вы не играете в пинг-понг, а наоборот, стараетесь увернуться от падающих объектов, чтобы заработать очки.

```
 1 # spacewarp.kbs
 2 # Игра Космический мусор
 3
 4 balln = 5  # количество мячей
 5 dim ballx(balln)  # массивы для хранения положения и скоростей мячей
 6 dim bally(balln)
 7 dim ballspeed(balln)
 8 ballr = 10 # радиус мяча
 9
10 minx = ballr # минимальное значение координаты x мяча
11 maxx = graphwidth - ballr # максимальное значение координаты x мяча
```

[2]Программа 75 также является демонстрацией динамических массивов. Обратите внимание на оператор **redim** (*прим. переводчика*).


```
miny = ballr # минимальное значение координаты y мяча
maxy = graphheight - ballr # максимальное значение координаты y мяча
score = 0 # в начале счёт равен 0
playerw = 30  # ширина игрока
playerm = 10  # шаг игрока
playerh = 10  # высота игрока
playerx = (graphwidth - playerw)/2  # начальное значение координаты x игрока - середина поля
keyj = asc("J") # значение для клавиши 'j'
keyk = asc("K") # значение для клавиши 'k'
keyq = asc("Q") # значение для клавиши 'q'
growpercent = .20  # случайное расширение игрока - чем больше, тем быстрее
speed = .15   # скорость - чем меньше, тем быстрее

print "Космический мусор - использовать j и k клавиши чтобы избежать столкновения с космическим мусором"
print "q - закончить"

fastgraphics

# установка начальных значений скорости и положения мячей
for n = 0 to balln-1
    gosub setupball
next n

more = true
while more
    pause speed
    score = score + 1

    # очистка экрана
    color black
    rect 0, 0, graphwidth, graphheight

    # рисуем мячи и проверяем на наличие столкновений
    color white
    for n = 0 to balln-1
        bally[n] = bally[n] + ballspeed[n]
        if bally[n] > maxy then gosub setupball
        circle ballx[n], bally[n], ballr
        if ((bally[n]) >= (maxy-playerh-ballr)) and ((ballx[n]+ballr) >= playerx) and ((ballx[n]-ballr) <= (playerx+playerw)) then more = false
    next n

    # рисуем игрока
    color red
    rect playerx, maxy - playerh, playerw, playerh
    refresh

    # игрок растёт в ширину
    if (rand<growpercent) then playerw = playerw + 1

    # проверяем нажата ли клавиша и перемещаем игрока, если нажата
    k = key
    if k = keyj then playerx = playerx - playerm
```

```
64      if k = keyk then playerx = playerx + playerm
65      if k = keyq then more = false
66
67      # не вышел ли игрок за границы поля?
68      if playerx < 0 then playerx = 0
69      if playerx > graphwidth - playerw then playerx = graphwidth -
    playerw
70
71  end while
72
73  print "счёт " + string(score)
74  print "вы погибли."
75  end
76
77  setupball:
78  bally[n] = miny
79  ballx[n] = int(rand * (maxx-minx)) + minx
80  ballspeed[n] = int(rand * (2*ballr)) + 1
81  return
```

Программа 77. Большая программа — Игра Космический мусор.

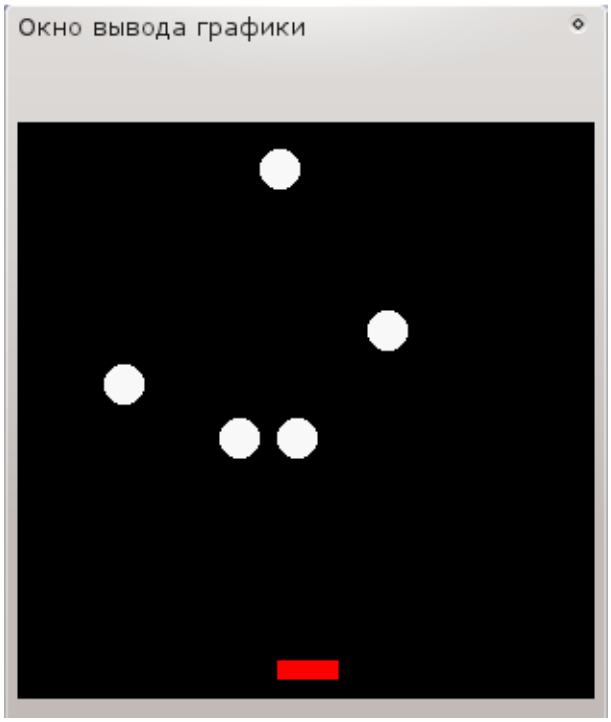


Рис. 13.5. Примерный экран программы 77. Большая программа — Игра Космический мусор.

Глава 14

Математика — развлечёмся с числами

В этой главе мы рассмотрим некоторые дополнительные математические операторы и функции. Глава разбита на четыре части: 1) новые операторы 2) новые целые функции 3) новые функции для работы с числами с плавающей точкой (действительные числа), и 4) тригонометрические функции.

14.1 Новые операторы

В дополнение к базовым математическим операциям, которые мы использовали с первой главы книги, есть ещё три оператора в BASIC-256. Операции, аналогичные этим есть в большинстве языков программирования. Это операция деления по модулю, целочисленного деления и возведения в степень.

Операция	Оператор	Описание
Деление по модулю	%	Возвращает остаток от деления на целое число
Целочисленное деление	\	Возвращает (целое) частное от деления одного целого на другое
Степень	^	Возводит одно (целое) число в степень другого (целого)

14.2 Деление по модулю

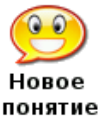
Операция деления по модулю возвращает остаток от деления нацело. Когда вы делите два целых числа «уголком», остаток, полученный в результате и есть модуль.

```
1 # c12_mod.kbs
2 input "Введите целое число ", n
3 if n % 2 = 0 then print "делится на 2"
4 if n % 3 = 0 then print "делится на 3"
5 if n % 5 = 0 then print "делится на 5"
6 if n % 7 = 0 then print "делится на 7"
7 end
```

Программа 78. Деление по модулю.

```
Введите целое число 10
делится на 2
делится на 5
```

Пример вывода программы 78. Деление по модулю.



выражение1 % *выражение2*

Оператор деления по модулю (%) возвращает остаток от деления числа *выражение1* на число *выражение2*.
Если одно или оба выражения не целые, то сначала они округляются до целого отбрасыванием дробной части (как это делает функция **int()**) до того, как операция будет произведена.

Вы может и не задумывались, но операция деления по модулю используется программистами довольно часто. Основные два применения это:

1. проверить делится ли одно число на другое (программа 78)
2. получить числа из ограниченного диапазона (программа 79)

```
 1 # moveballmod.kbs
 2 # Переписанная программа moveball.kbs с использованием оператора
 3 # деления по модулю для ограничения движения мяча экраном
 4
 5 print "клавиша i - вверх, j - влево, k - вправо, m - вниз, q - закончить"
 6
 7 fastgraphics
 8 clg
 9 ballradius = 20
10
11 # позиция мяча
12 # начинаем с центра экрана
13 x = graphwidth / 2
14 y = graphheight / 2
15
16 # рисуем мяч в начальном положении на экране
17 gosub drawball
18
19 # цикл ожидания ввода пользователем нажатий на клавиши
20 while true
21    k = key
22    if k = asc("I") then
23    # y может стать отрицательным, + graphheight делает это число
   положительным
24        y = (y - ballradius + graphheight) % graphheight
25        gosub drawball
26    end if
27    if k = asc("J") then
28        x = (x - ballradius + graphwidth) % graphwidth
29        gosub drawball
30    end if
31    if k = asc("K") then
32        x = (x + ballradius) % graphwidth
33        gosub drawball
34    end if
35    if k = asc("M") then
36        y = (y + ballradius) % graphheight
37        gosub drawball
38    end if
39    if k = asc("Q") then end
```



```
40 end while
41
42 drawball:
43 color white
44 rect 0, 0, graphwidth, graphheight
45 color red
46 circle x, y, ballradius
47 refresh
48 return
```

Программа 79. Двигаем мяч с использованием деления по модулю.

14.3 Целочисленное деление

Операция целочисленного деления (\) производит обычное деление, но работает только с целыми числами и возвращает целое число. Например 13 разделить на 4 будет 3 и в остатке 1. Результатом операции целочисленного деления будет 3.

```
1 # c12_integerdivision.kbs
2 input "Делимое ", dividend
3 input "Делитель ", divisor
4 print dividend + " / " + divisor + " = частное ";
5 print dividend \ divisor;
6 print " и остаток (r) ";
7 print dividend % divisor;
```

Программа 80. Проверь, как ты делишь уголком.

```
Делимое 43
Делитель 6
43 / 6 = частное 7 и остаток (r) 1
```

Пример вывода программы 80. Проверь, как ты делишь уголком.

выражение1 \ *выражение2*

Целочисленное деление (\) делит *выражение1* на *выражение2* и в качестве результата возвращает целое число, которое показывает во сколько раз *выражение1* больше *выражение2* (частное от деления)
Если одно или оба числа не целые, они округляются до целого, отбрасыванием десятичной части (как это делает **int()**) до того, как операция будет произведена.

14.4 Возведение в степень

Оператор возведения в степень возвращает степень первого числа относительно второго.

```
1 # c12_power.kbs
2 for t = 0 to 16
3   print "2 ^ " + t + " = ";
4   print 2 ^ t
5 next t
```

Программа 81. Степени числа 2.

```
2 ^ 0 = 1
2 ^ 1 = 2
2 ^ 2 = 4
2 ^ 3 = 8
2 ^ 4 = 16
2 ^ 5 = 32
2 ^ 6 = 64
2 ^ 7 = 128
2 ^ 8 = 256
2 ^ 9 = 512
2 ^ 10 = 1024
2 ^ 11 = 2048
2 ^ 12 = 4096
2 ^ 13 = 8192
2 ^ 14 = 16384
2 ^ 15 = 32768
2 ^ 16 = 65536
```

Примерный вывод программы 81. Степени числа 2.

выражение1 ^ *выражение2*

Оператор степень (^) возводит *выражение1* в степень заданную числом *выражение2*. Математическая запись степени $a = b^c$ в BASIC-256 записывается как `a = b^c`

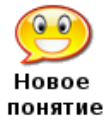
14.5 Новые целочисленные функции

Три новые функции этой главы связаны с преобразованием строк и чисел с плавающей точкой (действительных чисел) в целые числа. Все три эти функции по разному обрабатывают десятичную часть числа.

Функция **int**() просто отбрасывает десятичную часть числа, что равносильно вычитанию дробной части от (положительного) числа или прибавления десятичной части к отрицательному числу. Это может привести к проблемам, если мы пытаемся округлить числа меньшие нуля (0).

Функции **ceil**() и **floor**() по своему решают проблему **int**(). Функция **ceil**() дополняет дробное число до ближайшего целого, большего, чем исходное число, в то время как **floor**() всегда уменьшает число до ближайшего целого, меньшего данного числа.

Всех нас учили округлять, прибавляя 0.5 и отбрасывать дробную часть. Если мы используем **int**(), то она работает с положительными числами и не работает с отрицательными. В BASIC-256 для округления («как привычно») следует использовать формулу типа *a=floor(b+0.5)* .



Функция	Описание
int(*выражение*)	Преобразует *выражение* (строку, целое или десятичную дробь) в целое число. При этом дробная часть просто отбрасывается. Если строка не содержит числа возвращается нуль.
ceil(*выражение*)	Преобразует *выражение* в ближайшее большее целое
floor(*выражение*)	Преобразует *выражение* в ближайшее меньшее целое. Для округления следует использовать формулу *a=floor(b+0.5)*

```
1 # c12_intceilfloor.kbs
2 for t = 1 to 10
3     n = rand * 100 - 50
4     print n;
5     print " int=" + int(n);
6     print " ceil=" + ceil(n);
7     print " floor=" + floor(n)
8 next t
```

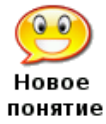
Программа 82. Различие между *int*, *ceil* и *floor*.

```
-34,342984 int=-34 ceil=-34 floor=-35
23,295121 int=23 ceil=24 floor=23
-25,956241 int=-25 ceil=-25 floor=-26
10,697896 int=10 ceil=11 floor=10
-16,144697 int=-16 ceil=-16 floor=-17
44,576658 int=44 ceil=45 floor=44
35,383576 int=35 ceil=36 floor=35
4,651446 int=4 ceil=5 floor=4
-39,022827 int=-39 ceil=-39 floor=-40
47,965051 int=47 ceil=48 floor=47
```

Примерный вывод программы 82. Различие между *int*, *ceil* и *floor*.

14.6 Новые функции для дробных чисел

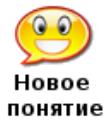
Математические функции, в которые погрузила нас эта глава могут пригодиться при написании некоторых специальных программ. В подавляющем большинстве программ они не нужны.



Функция	Описание
float(*выражение*)	Преобразует *выражение* (строку, целое или десятичную дробь) в дробное число. Обычно используется для преобразования строк в числа. Если строка не содержит числа возвращается нуль.
abs(*выражение*)	Возвращает беззнаковое значение выражения (абсолютное значение)
log(*выражение*)	Возвращает натуральный логарифм числа (логарифм по основанию e)
log10(*выражение*)	Возвращает десятичный логарифм числа (логарифм по основанию 10)

14.7 Тригонометрические функции (для знакомых с ними)

Тригонометрия — наука об измерении углов и сторон (и их соотношений) в треугольнике. BASIC-256 поддерживает основные тригонометрические функции. Углы измеряются в радианах ($0 - 2\pi$). Если вы хотите использовать градусы ($0° - 360°$) в своей программе, то необходимо значения в градусах предварительно преобразовать в радианы, а потом использовать в тригонометрических функциях.



Функция[1]	Описание
cos(*выражение*)	Возвращает значение косинуса угла
sin(*выражение*)	Возвращает значение синуса угла
tan(*выражение*)	Возвращает значение тангенса угла
degrees(*выражение*)	Преобразует число из радиан ($0 - 2\pi$) в градусы ($0° - 360°$)
radians(*выражение*)	Преобразует число градусов ($0° - 360°$) в радианы ($0 - 2\pi$)
acos(*выражение*)	Функция, обратная косинусу (арккосинус)
asin(*выражение*)	Функция, обратная синусу (арксинус)
atan(*выражение*)	Функция, обратная тангенсу (арктангенс)

Первые три функции имеют прямое отношение к сторонам прямоугольного треугольника. Рисунок 14.1 демонстрирует прямоугольный треугольник, его стороны и углы.

14.8 Косинус

Косинус угла равен отношению прилежащего катета к гипотенузе: $\cos A = \frac{b}{c}$. Косинус повторяет свои значения в диапазоне от -1 до 1 на каждом интервале длиной 2π радиан. Рисунок 14.2 показывает волновой график косинуса в диапазоне от 0 до 2π.

[1] В российской математике приняты обозначения для тангенса — tg, для арккосинуса — arccos, для арксинуса — arcsin, для арктангенса — arctg.

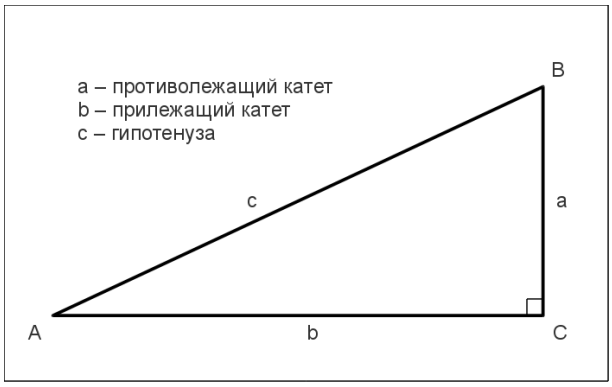


Рис. 14.1. Прямоугольный треугольник.

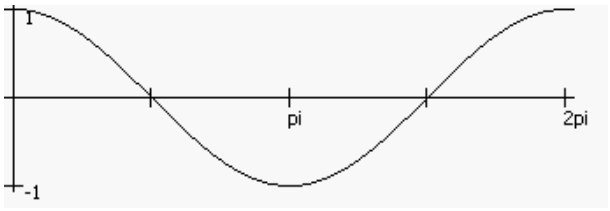


Рис. 14.2. Функция cos()

14.9 Синус

Синус равен отношению прилежащего катета к гипотенузе: $\sin A = \frac{a}{c}$. Синус также, как и косинус повторяет свои значения из диапазона от -1 до 1 на каждом интервале длиной 2π. Рисунок 14.3 показывает волновой график синуса в диапазоне от 0 до 2π.

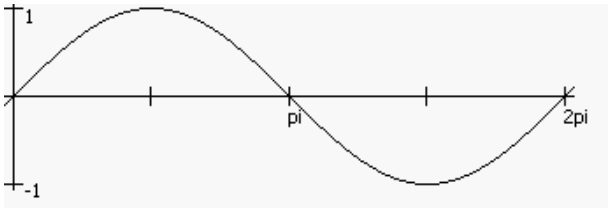


Рис. 14.3. Функция sin()

14.10 Тангенс

Тангенс равен отношению противолежащего катета к прилежащему: $\mathrm{tg} = \frac{a}{b}$. Тангенс повторяет свои значения лежащие в диапазоне от $-\infty$ до ∞ на каждом отрезке в π радиан. Такой диапазон тангенса объясняется тем, что когда угол треугольника становится очень маленьким, противолежащий катет также становится очень маленьким.

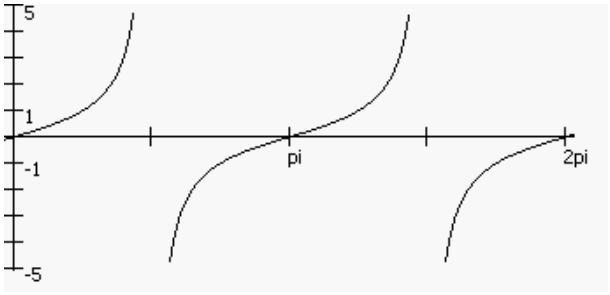


Рис. 14.4. Функция tan()

14.11 Функция degrees

Функция **degrees**() преобразует значения угла из радиан в градусы по формуле: $\textit{градусы} = \frac{\textit{радианы}}{2\pi} \times 360$.

14.12 Функция radians

Функция **radians**() преобразует значения угла из градусов в радианы по формуле $\textit{радианы} = \frac{\textit{градусы}}{360} \times 2\pi$. Запомните, все тригонометрические функции BASIC-256 используют в качестве аргумента радианы, а не градусы.

14.13 Арккосинус

Арккосинус (**acos**()) — функция обратная косинусу. Её значением является величина угла, косинус которого равен заданному числу.

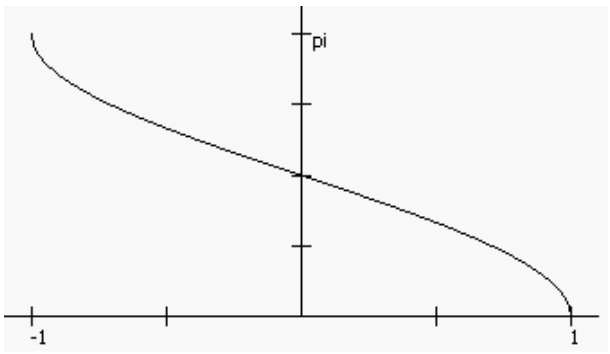


Рис. 14.5. Функция acos()

14.14 Арксинус

Арксинус (**asin**()) — функция, обратная синусу. Её значением является величина, синус которого равен заданному числу.

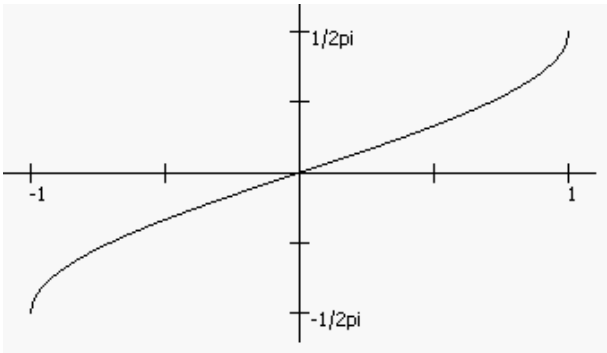


Рис. 14.6. Функция asin()

14.15 Арктангенс

Арктангенс (**atan**()) — функция, обратная тангенсу. Её значением является угол, тангенс которого равен заданному числу.

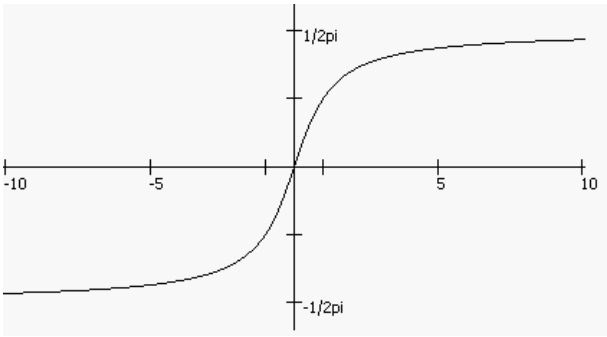


Рис. 14.7. Функция atan()

Большая программа этой главы воспроизводит процесс деления целых чисел уголком. Программа использует логарифм, чтобы вычислить количество знаков в числах, деление по модулю и целочисленное деление для определения цифр, и в целом очень сложная программа. Не расстраивайтесь и не опускайте руки, если сразу не поймёте полностью, как она работает.

```
 1 # longdivision.kbs
 2 # Показывает графически процесс деления уголком
 3 # двух положительных целых чисел.
 4
 5 input "Делимое? ", b
 6 input "Делитель? ", a
 7
 8 originx = 100
 9 originy = 20
10 height = 12
11 width = 9
```

```
12 margin = 2
13
14 b = int(abs(b))
15 a = int(abs(a))
16
17 color white
18 rect 0,0,graphwidth, graphheight
19 color black
20
21 # Отображение задания на деление
22 row = 0
23 col = -1
24 number = a
25 underline = false
26 gosub drawrightnumber
27 row = 0
28 col = 0
29 number = b
30 gosub drawleftnumber
31 line originx - margin, originy, originx + (width * 11), originy
32 line originx - margin, originy, originx - margin, originy + height
33
34 # Сколько цифр в делимом?
35 lb = ceil(log10(abs(b)))
36
37 r = 0
38 bottomrow = 0     ## номер последнего ряда вычислений
39
40 # Цикл по всем цифрам слева направо
41 for tb = lb-1 to 0 step -1
42    # получаем следующюю цифру текущего остатка и убираем её из делимого
43    r = r * 10
44    r = r + (b \ (10 ^ tb))
45    b = b % (10 ^ tb)
46    # отображаем текущий остаток
47    row = bottomrow
48    bottomrow = bottomrow + 1
49    col = lb - tb - 1
50    number = r
51    underline = false
52    gosub drawrightnumber
53    # вычисляем следующую цифру ответа и отображаем её
54    digit = r \ a
55    row = -1
56    col = lb - tb - 1
57    gosub drawdigit
58    # вычисляем количество которое надо удалить из текущего остатка и отображаем
59    number = digit * a
60    r = r - number
61    col = lb - tb - 1
62    row = bottomrow
63    bottomrow = bottomrow + 1
64    underline = true
65    gosub drawrightnumber
66 next tb
```



```
 67 #
 68 # печатаем остаток в нижнем ряду
 69 row = bottomrow
 70 col = lb - 1
 71 number = r
 72 underline = false
 73 gosub drawrightnumber
 74 end
 75
 76 drawdigit:
 77 # вычисляем строку и столбец для отображения
 78 text col * width + originx, row * height + originy, digit
 79 if underline then
 80    line col * width + originx - margin, (row + 1) * height + originy
    , (col + 1) * width + originx - margin, (row + 1) * height + originy
 81 end if
 82 return
 83
 84 drawleftnumber:
 85 # передаём начальный ряд и колонку, а также число для левой колонки
 86 if number < 10 then
 87    digit = number
 88    gosub drawdigit
 89 else
 90    lnumber = ceil(log10(abs(number)))
 91    for tnumber = lnumber-1 to 0 step -1
 92        digit = (number \ (10 ^ tnumber)) % 10
 93        gosub drawdigit
 94        col = col + 1
 95    next tnumber
 96 endif
 97 return
 98
 99 drawrightnumber:
100 # передаём начальный ряд, колонку и число для отображения в правой колонке
101 if number < 10 then
102    digit = number
103    gosub drawdigit
104 else
105    lnumber = ceil(log10(abs(number)))
106    for tnumber = 0 to lnumber - 1
107        digit = (number \ (10 ^ tnumber)) % 10
108        gosub drawdigit
109        col = col - 1
110    next tnumber
111 endif
112 return
```

Программа 83. Большая программа — Деление уголком.

```
Делимое? 123456
Делитель? 78
```

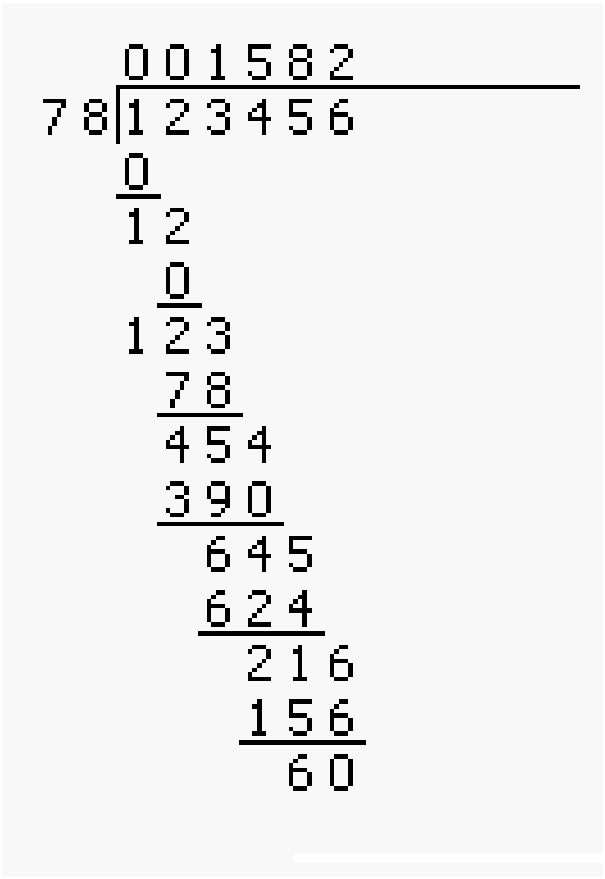


Рис. 14.8. Пример вывода программы 83. Большая программа — Деление уголком.

Глава 15

Работаем со строками

Мы использовали строки, чтобы хранить нечисловую информацию, формировать вывод на экран и получать вводимые пользователем символы. В главе 11 мы использовали Unicode значения отдельных символов для построения строк.

В этой главе мы узнаем несколько новых функций для действий со строками.

15.1 Строковые функции

BASIC-256 имеет восемь стандартных функций для преобразования строк, которые собраны в таблице 15.1

15.2 Функция string()

Функция **string**() берёт выражение в любом формате и преобразовывает его в строку. Она очень удобна для преобразования целых или дробных чисел в символы так, что с ними можно работать как со строками.

```
1 # string.kbs
```

Таблица 15.1. Строковые функции

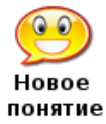
Функция	**Описание**
string(*выражение*)	приводит *выражение* (строковое или целое или дробь) к строке
length(*строка*)	возвращает длину строки
left(*строка, длина*)	возвращает часть строки заданной длины, отсчитывая слева
right(*строка, длина*)	возвращает часть строки заданной длины, отсчитывая справа
mid(*строка, начало, длина*)	возвращает среднюю часть строки заданной длины от позиции начало
upper(*выражение*)	Возвращает строку в верхнем регистре (заглавные буквы)
lower(*выражение*)	Возвращает строку в нижнем регистре (строчные буквы)
instr(*стог, иголка*)	Ищет *иголку* в *стоге* (сена) и возвращает её позицию.

```
2 a$ = string(10 + 13)
3 print a$
4 b$ = string(2 * pi)
5 print b$
```

Программа 84. Функция **string**().

```
23
6,283185
```

Пример вывода программы 84. Функция **string**().



string(*выражение*)

Преобразует *выражение* (строку, целое или дробное число) в строку

15.3 Функция length()

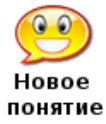
Функция length() возвращает количество символов (букв, знаков) входящих в строку.

```
1 # length.kbs
2 # печатает 7, 0, и 22
3 print length("Привет!")
4 print length("")
5 print length("Программировать круто!")
```

Программа 85. Функция **length**().

```
7
0
22
```

Вывод программы 85. Функция **length**().



length(*выражение*)

Принимает строковое *выражение* и возвращает его длину. Для пустой строки ("") возвращает нуль (0).

15.4 Функции left(), right() и mid()

Функции left(), right() и mid() извлекают из строки определённую подстроку (часть строки).

```
1 # leftrightmid.kbs
2 a$ = "автомашина"
3 # печатает "авто"
4 print left(a$,4)
5 # печатает "на"
6 print right(a$,2)
7 # печатает "маш" и "шина"
8 print mid(a$,5,3)
9 print mid(a$,7,9)
```


Программа 86. Функции **left**(), **right**() и **mid**().

```
авто
на
маш
шина
```

Пример вывода программы 86. Функции **left**(), **right**() и **mid**().

left(*строка, длина*)

Возвращает подстроку заданной длины, начиная слева. Если *длина* равна или больше длины переменной *строка*, то только *строка* и будет возвращена.

right(*строка, длина*)

Возвращает подстроку заданной длины, начиная справа. Если *длина* равна или больше длины переменной *строка*, то только *строка* и будет возвращена.

mid(*строка, начало, длина*)

Возвращает подстроку заданной длины переменной *строка* из её середины. Параметр *начало* указывает, где эта подстрока должна начаться (1 = начало строки)[1]

15.5 Функции upper() и lower()

Функции **upper**() и **lower**() просто возвращают строку состоящую из заглавных (upper) или строчных (lower) букв. Эти функции наиболее полезны, когда надо сравнить значения двух строк и вам не важно в каком регистре они написаны.

```
1 # upperlower.kbs
2 a$ = "Привет!"
3 # печатает "привет!"
4 print lower(a$)
5 # печатает "ПРИВЕТ!"
6 print upper(a$)
```

Программа 87. Функции **upper**() и **lower**().

```
привет!
ПРИВЕТ!
```

Пример вывода программы 87. Функции **upper**() и **lower**().

[1]Если сумма параметров *начало* + *длина* больше, чем длина строки, то вернётся только часть строки от места *начало* до конца строки, что и показано в последнем выводе программы 86. (*Прим. редактора*)

lower(*строка*)
upper(*строка*)

Возвращает копию переменной *строка*, в которой все символы заменены на строчные (lower) или заглавные (upper) буквы. Остальные (не буквы) символы возвращаются без изменений.

15.6 Функция instr()

Функция **instr()** ищет первое вхождение подстроки в заданную строку и возвращает местоположение первого символа подстроки в строке. Если подстрока отсутствует в строке, возвращается нуль (0).

```
1 # instr.kbs
2 a$ = "автомашина"
3 # ищем подстроку "шина"
4 print instr(a$,"шина")
5 # ищем подстроку "колесо"
6 print instr(a$,"колесо")
```

Программа 88. Функция **instr**().

```
7
0
```

Пример вывода программы 88. Функция **instr**().

instr(*стог_сена, иголка*)

Ищет подстроку (*иголка*) в другой строке (*стог_сена*). Возвращает позицию первого символа подстроки. Если подстрока не найдена, возвращает нуль (0).

Десятичная система счисления (по основанию 10), используемая наиболее часто, использует цифры от 0 до 9 для записи чисел.
А теперь, давайте представим, что случится, если у нас будет только 5 цифр от 0 до 4. В этом случае число 23 ($23 = 2{\cdot}10^1{+}3{\cdot}10^0$) станет 43 ($23 = 4{\cdot}5^1{+}3{\cdot}5^0$), но будет представлять одно и то же количество предметов. Такое преобразование чисел называется сменой основания системы счисления.
Компьютер, в своём внутреннем устройстве, не использует и не понимает числа по основанию 10, он преобразует все числа в двоичную систему (по основанию 2) и их он хранит в памяти и с ними только оперирует.
«Большая программа» этой главы преобразует (конвертирует) положительные целые числа из одной системы счисления в интервале от 2 до 36 в любую другую. В качестве цифр от 11-ой до 36-ой используются заглавные буквы английского алфавита[2].

[2]Как известно, в английском алфавите ровно 26 букв. A означает «цифру» 10, B – 12 и т. д. Наиболее известна в среде программистов шестнадцатеричная система счисления, где цифрами от 10 до 15 являются буквы от A до F (*Прим. переводчика*).


```
 1 # radix.kbs
 2 # преобразование чисел из одного основания (2-36) в другое
 3
 4 digits$ = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 5
 6 message$ = "из основания"
 7 gosub getbase
 8 frombase = base
 9
10 input "число по основанию " + frombase + " >", number$
11 number$ = upper(number$)
12
13 # преобразуем число в десятичную систему (основание 10) и сохраняем его в
   переменной n
14 n = 0
15 for i = 1 to length(number$)
16     n = n * frombase
17     n = n + instr(digits$, mid(number$, i, 1)) - 1
18 next i
19
20 message$ = "в основание"
21 gosub getbase
22 tobase = base
23
24 # строим строку в tobase
25 result$ = ""
26 while n <> 0
27     result$ = mid(digits$, n % tobase + 1, 1) + result$
28     n = n \ tobase
29 end while
30
31 print "по основанию " + tobase + " это число равно " + result$
32 end
33
34 getbase: # получение основания от 2 до 36
35 do
36     input message$+"> ", base
37 until base >= 2 and base <= 36
38 return
```

Программа 89. Большая программа — Смена основания системы счисления.

```
из основания> 10
число по основанию 10, >999
в основание> 16
по основанию 16, это число равно 3E7
```

Пример вывода программы 89. Большая программа — Смена основания системы счисления.

Глава 16

Файлы. Сохраним информацию на будущее

Мы исследовали оперативную память компьютера с помощью переменных и массивов, но как сохранить данные надолго? Существует много разных способов долгосрочного хранения данных.

BASIC-256 поддерживает запись и чтение информации из файлов на вашем жёстком диске. Этот процесс ввода и вывода информации часто обозначают I/O[1].

Эта глава покажет как читать данные из файла и как их потом туда записывать для долгосрочного хранения.

16.1 Чтение строк из файла

В первой программе этой главы вы увидите много новых операторов и констант (постоянных) которые нужны для управления данными.

```
 1 # readfile.kbs
 2 input "имя файла>", fn$
 3 if not exists(fn$) then
 4     print fn$ + " не существует."
 5     end
 6 end if
 7 #
 8 n = 0
 9 open fn$
10 while not eof
11     l$ = readline
12     n = n + 1
13     print n + " " + l$
14 end while
15 #
16 print "Файл " + fn$ + " содержит " + size + " байт."
17 close
```

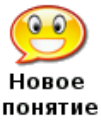
Программа 90. Чтение строк из файла.

```
имя файла>test_ru.txt
1 Настало время испытать
2 силу человеческой души!
3 - Томас Пейн
```

[1] I/O — от английских слов Input (ввод) и Output (вывод) (*Прим. переводчика*).

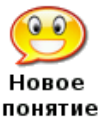
```
Файл test_ru.txt содержит 110 байт.
```

Пример вывода программы 90. Чтение строк из файла.



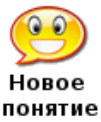
exist(*выражение*)

Ищет на компьютере файл, имя которого определено переменной *выражение*. Имя диска[2] и путь должны быть указаны, как часть имени файла. Если они опущены, происходит поиск по текущему рабочему каталогу.
Возвращает *true* (истина или 1) если файл существует и *false* (ложь или 0) в противном случае.



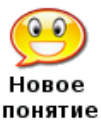
open*выражение*
open (*выражение*)
open *номер_ файла, выражение*
open (*номер_ файла, выражение*)

Открывает указанный переменной *выражение* файл, задавая ему указанный *номер_ файла*. Если файл не существует, создаётся новый файл так, что в него можно добавлять информацию (см. операторы **write** и **writeln**). Не забудьте выполнить команду **close**, когда ваша программа закончит работать с файлом. BASIC-256 может открыть одновременно не более восьми (8) файлов с номерами от 0 до 7. Если номер файла не указан, он открывается с номером нуль (0).



eof
eof ()
eof(*номер_ файла*)

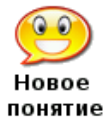
Функция **eof** возвращает истину (*true*) если достигнут конец файла в процессе чтения и ложь (*false*) в если ещё есть не прочтённые данные.
Если *номер_ файла* не указан используется значение нуль (0).



readline
readline ()
readline (*номер_ файла*)

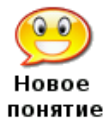
Возвращает прочитанную из файла строку. Если достигнут конец файла [**eof**(*номер_ файла*)=*true*], тогда функция возвращает пустую строку ("")
Если *номер_ файла* не указан используется значение нуль (0).

[2]Имя диска имеет смысл только на операционных системах Windows™(*Прим. редактора*).



size
size ()
size(*номер_файла*)

Возвращает длину файла в байтах.
Если *номер_файла* не указан используется значение нуль (0).



close
close ()
close(*номер_файла*)

Функция **close** закрывает ввод/вывод в файл и позволяет другому файлу использовать тот же номер.
Если *номер_файла* не указан используется значение нуль (0).

16.2 Запись строк в файл

В программе 90 мы видели, как читать информацию из файла. Следующие две программы демонстрируют различные варианты записи информации в файл. В программе 91 мы открываем файл для записи стирая всю имеющуюся там ранее информацию, а в программе 92 добавляем новые строчки в конец файла, сохраняя предыдущую информацию.

```
 1 # resetwrite.kbs
 2 open "resetwrite.dat"
 3
 4 print "введите пустую строку для завершения записи"
 5
 6 # стираем данные из файла (reset) и начинаем туда писать сначала
 7 reset
 8 repeat:
 9 input ">", l$
10 if l$ <> "" then
11     writeline l$
12     goto repeat
13 end if
14
15 # переходим к началу файла и отображаем содержимое
16 seek 0
17 k = 0
18 while not eof()
19     k = k + 1
20     print k + " " + readline()
21 end while
22
23 close
24 end
```

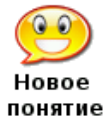
Программа 91. Очищение файла и запись.

```
введите пустую строку для завершения записи
>Вот простая информация
>которую я записал в файл
```



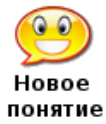
```
>раз, два, три, четыре пять
>Я иду искать!
>
1 Вот простая информация
2 которую я записал в файл
3 раз, два, три, четыре пять
4 Я иду искать!
```

Пример вывода программы 91. Очищение файла и запись.



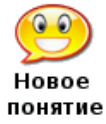
reset
reset ()
reset *номер_файла*
reset (*номер_файла*)

Стирает все данные в открытом файле, и перемещает указатель файла на начало.
Если *номер_файла* не указан, используется нулевой номер (0).



seek*выражение*
seek (*выражение*)
seek *номер_файла, выражение*
seek (*номер_файла, выражение*)

Перемещает указатель чтения или записи файла в указанную позицию. Для перемещения указателя в начало файла используйте значение нуль (0). Для перемещения указателя в конец файла используйте значение функции **size**().
Если *номер_файла* не указан, используется нулевой номер (0).



writeline*выражение*
writeline (*выражение*)
writeline *номер_файла, выражение*
writeline (*номер_файла, выражение*)

Записывает значение аргумента *выражение* в открытый файл и добавляет символ конца строки. Указатель файла перемещается в конец записи так, что следующая команда записи запишет данные непосредственно за этими.
Если *номер_файла* не указан, используется нулевой номер (0).

```
 1 # appendwrite.kbs
 2 open "appendwrite.dat"
 3
 4 print "введите пустую строку для завершения записи"
 5
 6 # перемещаем указатель файла в конец и добавляем записи
 7 seek size
 8 repeat:
 9 input ">", l$
10 if l$ <> "" then
11     writeline l$
```

```
12      goto repeat
13  end if
14
15  # перемещаем указательк началу файла, чтобы вывести его содержимое
16  seek 0
17  k = 0
18  while not eof()
19      k = k + 1
20      print k + " " + readline()
21  end while
22
23  close
24  end
```

Программа 92. Дописываем строчки в файл.

```
введите пустую строку для завершения записи
>sed sed sed
>vim vim vim
>

1 bar bar bar
2 foo foo foo
3 grap grap grap
4 sed sed sed
5 vim vim vim
```

Пример вывода программы 92. Дописываем строчки в файл.

16.3 Функция read() и оператор write

В первых трёх программах этой главы мы обсуждали функцию **readline**() и оператор **writeline**. Вот ещё два других способа читать из файла и писать в файл: функция **read**() и оператор **write**.

read
read ()
read (*номер_ файла*)

Читает следующее слово или число (токен) из файла. Токены отделяются друг от друга пробелом, табуляцией или символом конца строки. Множественные пробельные элементы воспринимаются как один.
Если *номер_ файла* не указан, используется нулевой номер (0).

write *выражение*
write (*выражение*)
write *номер_ файла, выражение*
write (*номер_ файла, выражение*)

Записывает значение аргумента *выражение* в открытый файл и не добавляет символ конца строки.
Если *номер_ файла* не указан, используется нулевой номер (0).

Эта программа использует простой текстовый файл имитируя телефонную записную книжку.

```
 1 # phonelist.kbs
 2 # Добавляем номер телефона в список и показываем
 3 filename$ = "phonelist.txt"
 4
 5 print "phonelist.kbs - Управление списком телефонов."
 6 do
 7 input "Добавить, Посмотреть, Закончить (a/l/q)?",action$
 8 if left(lower(action$),1) = "a" then gosub addrecord
 9 if left(lower(action$),1) = "l" then gosub listfile
10 until left(lower(action$),1) = "q"
11 end
12
13 listfile:
14 if exists(filename$) then
15     # отображаем список имён и телефонов из файла
16     open filename$
17     print "Файл имеет размер " + size + " байт"
18     while not eof
19         # читаем следующую строки из файла и печатаем её
20         print readline
21     end while
22     close
23 else
24     print "В файле нет записей. Добавьте первую."
25 end if
26 return
27
28 addrecord:
29 input "Имя?", name$
30 input "Номер телефона?", phone$
31 open filename$
32 # ищем конец файла
33 seek size()
34 # мы достигли конца файла, - добавляем строку
35 writeline name$ + ", " + phone$
36 close
37 return
```

Программа 93. Большая программа — Телефонная книжка.

```
phonelist.kbs - Управление списком телефонов.
Добавить, Посмотреть, Закончить (a/l/q)?l
Файл имеет размер 76 байт
Аня, 555-5555
Валя, 555-7777
Дима, 555-3333
Паша, 555-0987
Петя, 234-3445
Добавить, Посмотреть, Закончить (a/l/q)?a
Имя?Вася
Номер телефона?345-4556
```

```
Добавить, Посмотреть, Закончить (a/l/q)?l
Файл имеет размер 95 байт
Аня, 555-5555
Валя, 555-7777
Дима, 555-3333
Паша, 555-0987
Петя, 234-3445
Вася, 345-4556
Добавить, Посмотреть, Закончить (a/l/q)?q
```

Пример вывода программы 93. Большая программа — Телефонная книжка.

Глава 17

Стеки, очереди, списки и сортировка

Эта глава посвящена введению в достаточно сложные понятия, которые рассматриваются на первых курсах факультетов информатики университетов. Первые три темы (стеки, очереди и связанные списки) описывают наиболее общие способы хранения информации в компьютерных системах, последние две описывают алгоритмы сортировки информации.

17.1 Стек

Стек, наиболее общая структура данных, используемая программистами во многих задачах. Стек работает как колода карт в игре «сумасшедшая восьмёрка»[1]. Когда вы добавляете данные в стек, они добавляются сверху (называется «*push*»[2]) и эти данные расположены в стеке над другими. Когда вы хотите получить информацию из стека, то можете взять только самый верхний элемент, открывая доступ к нижележащему (называется «*pop*»[3]). Рисунок 17.1 демонстрирует эту схему.

Операции со стеком также можно описать как «последним пришёл, первым ушёл», или как принято в англоязычных странах «last in, first out» сокращённо *LIFO*. Последнее добавленное в стек значение, будет следующим, которое из него извлечено. Программа 94 организует стек используя массив и указатель на последний добавленный элемент. В подпрограмме «pushstack» вы увидите как с помощью изменения размеров массива реализуется теоретически произвольный размер стека.

```
 1 # stack.kbs
 2 # реализация стека, используя массив
 3
 4 dim stack(1) # массив для хранения стека в начале
 5 nstack = 0  # количество элементов стека
 6
 7 value = 1
 8 gosub pushstack
 9 value = 2
10 gosub pushstack
11 value = 3
12 gosub pushstack
```

[1] Можно сравнить стек с магазином патронов у автомата или автоматического пистолета (*прим. переводчика*).

[2] В данном случае слово «push» можно перевести как «продавить», что очевидно, если сравнивать стек с магазином, а добавление информации с помещением патрона в магазин. (*прим. переводчика*).

[3] вытолкнуть (*прим. переводчика*).

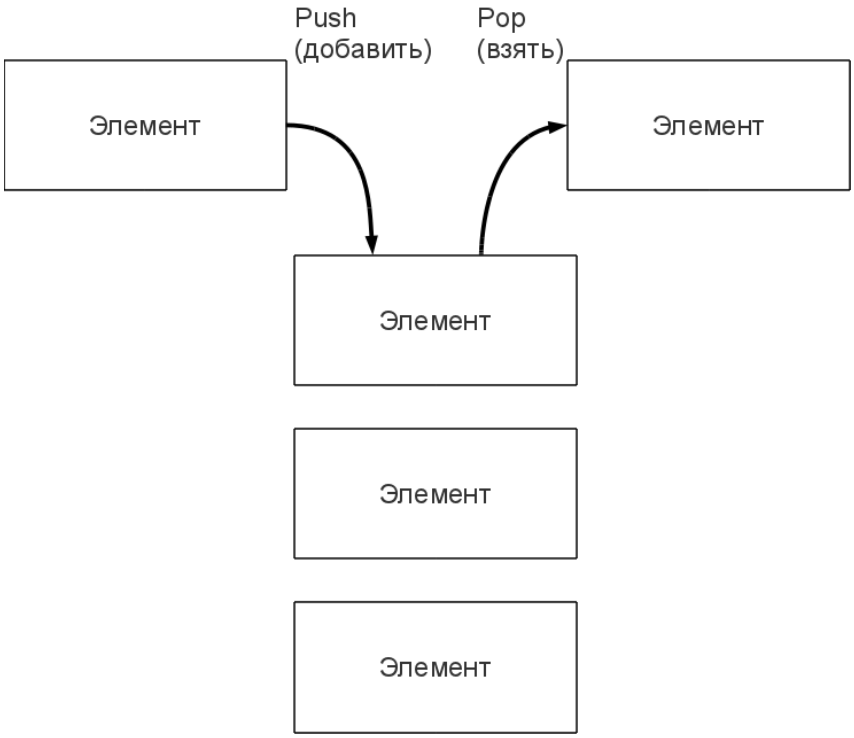


Рис. 17.1. Что такое стек

```
13 value = 4
14 gosub pushstack
15 value = 5
16 gosub pushstack
17
18 while nstack > 0
19     gosub popstack
20     print value
21 end while
22
23 end
24
25 popstack: #
26 # получаем значение верхнего элемента стека и помещаем его в переменную value
27 if nstack = 0 then
28     print "стек пуст"
29 else
30     nstack = nstack - 1
31     value = stack[nstack]
32 end if
33 return
34
35 pushstack: # помещает число в переменную стека
36 # увеличивает размер стека, если он уже полон
37 if nstack = stack[?] then redim stack(stack[?] + 5)
38 stack[nstack] = value
39 nstack = nstack + 1
```



```
40 return
```

Программа 94. Стек.

```
5
4
3
2
1
```

Пример вывода программы 94. Стек.

17.2 Очередь

Очередь — ещё одна из распространённых форм представления данных. Очередь, в простейшем виде, напоминает цепочку выстроившуюся у линии раздачи завтрака в школе. Первый в очереди, первым получает еду. Рисунок 17.2 демонстрирует блочную. диаграмму очереди.

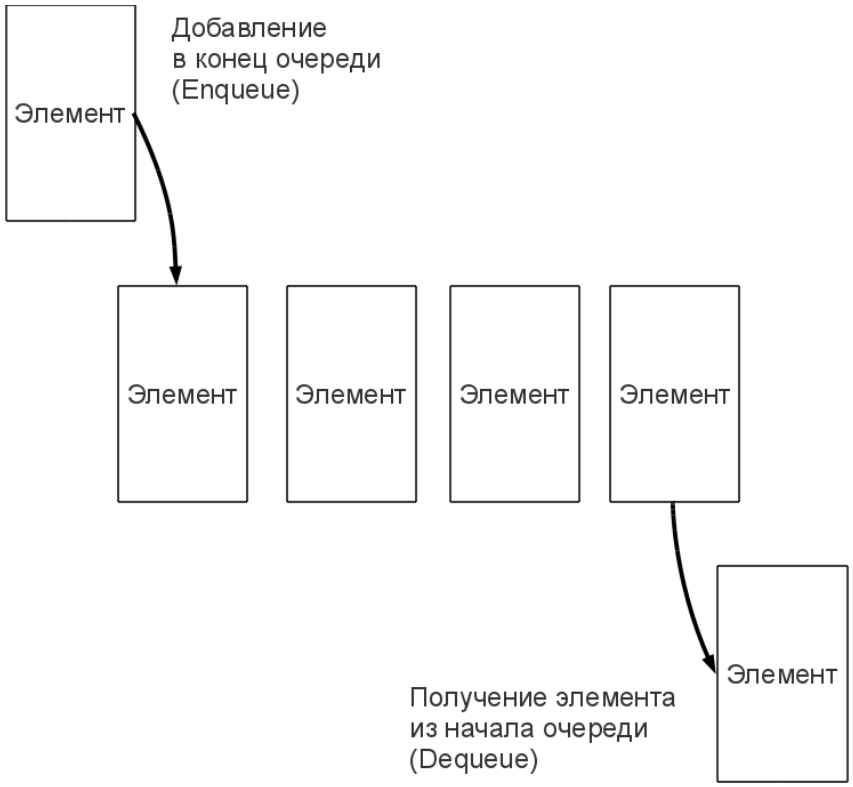


Рис. 17.2. Что такое очередь

Термины *enqueue* и *dequeue* используются для обозначения процессов добавления нового элемента в хвост очереди и удаления элемента из головы очереди соответственно. Иногда очередь описывают, как «первым пришёл, первым ушёл» или на английском языке *FIFO* («first in, first out»). Пример в программе 95 использует массив и два указателя, для отслеживания «головы» и «хвоста» очереди.

```
 1 # queue.kbs
 2 # реализация очереди через массив
 3
 4 # размер очереди в любой момент времени
 5 queuesize = 4
 6 # массив для элементов очереди
 7 dim queue(queuesize)
 8 # Положение следующего входящего элемента (указатель на хвост)
 9 tail = 0
10 # Положение в очереди возвращаемого элемента (указатель на голову)
11 head = 0
12 # количество элементов в очереди
13 inqueue = 0
14
15 value = 1
16 gosub enqueue
17 value = 2
18 gosub enqueue
19
20 gosub dequeue
21 print value
22
23 value = 3
24 gosub enqueue
25 value = 4
26 gosub enqueue
27
28 gosub dequeue
29 print value
30 gosub dequeue
31 print value
32
33 value = 5
34 gosub enqueue
35 value = 6
36 gosub enqueue
37 value = 7
38 gosub enqueue
39
40 # очищение очереди
41 while inqueue > 0
42     gosub dequeue
43     print value
44 end while
45
46 end
47
48 dequeue: #
49 if inqueue = 0 then
50     print "очередь пуста"
51 else
52     inqueue = inqueue - 1
53     value = queue[head]
54     print "удаление элемента  value=" + value + " из
   положения=" + head + " в очереди=" + inqueue +" элементов"
```



```
55      # перемещение указателя на начало. Если мы в конце массива, переходим к
    началу.
56      head = head + 1
57      if head = queuesize then head = 0
58  end if
59  return
60
61  enqueue: #
62  if inqueue = queuesize then
63      print "очередь заполнена"
64  else
65      inqueue = inqueue + 1
66      queue[tail] = value
67      print "добавляемый элемент value=" + value + " в
    положение=" + tail + " в очереди=" + inqueue +" элементов"
68      # перемещаем указатель конца очереди, если достигли конца, переходим к
    началу.
69      tail = tail + 1
70      if tail = queuesize then tail = 0
71  end if
72  return
```

Программа 95. Очередь.

```
добавляемый элемент value=1 в положение=0 в очереди=1 элементов
добавляемый элемент value=2 в положение=1 в очереди=2 элементов
удаление элемента value=1 из положения=0 в очереди=1 элементов
1
добавляемый элемент value=3 в положение=2 в очереди=2 элементов
добавляемый элемент value=4 в положение=3 в очереди=3 элементов
удаление элемента value=2 из положения=1 в очереди=2 элементов
2
удаление элемента value=3 из положения=2 в очереди=1 элементов
3
добавляемый элемент value=5 в положение=0 в очереди=2 элементов
добавляемый элемент value=6 в положение=1 в очереди=3 элементов
добавляемый элемент value=7 в положение=2 в очереди=4 элементов
удаление элемента value=4 из положения=3 в очереди=3 элементов
4
удаление элемента value=5 из положения=0 в очереди=2 элементов
5
удаление элемента value=6 из положения=1 в очереди=1 элементов
6
удаление элемента value=7 из положения=2 в очереди=0 элементов
7
```

Пример вывода программы 95. Очередь.

17.3 Связный список

В большинстве книг обсуждение материала этой главы начинается со связных списков. Однако BASIC-256 по другому организует память, чем другие языки, поэтому мы обсуждаем связные списки после стеков и очередей.

Связный список, это последовательность элементов, содержащих данные и указатель на следующий элемент в списке. Кроме этих элементов нам необходимо иметь указатель на

первый объект списка. Мы назовём первый элемент списка «голова». Взгляните на рисунок 17.3, где видно, как каждый объект списка указывает на другой.

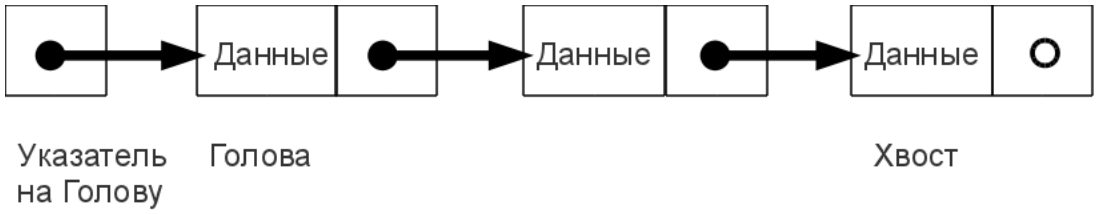


Рис. 17.3. Связный список

Преимущество связных списков над массивом в том, что можно с лёгкостью добавлять или удалять его элементы. Чтобы стереть элемент списка достаточно поменять указатель у предыдущего элемента на последующий (см. рисунок 17.4) и освобождение элемента так, чтобы его можно было использовать.

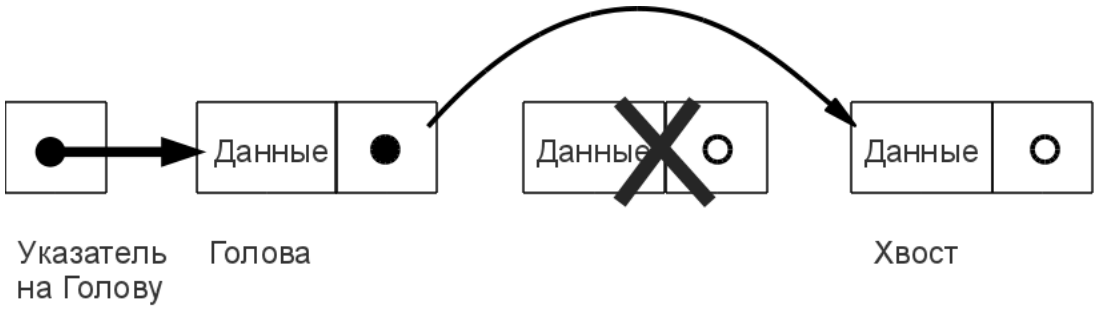


Рис. 17.4. Удаление элемента списка.

Добавление нового элемента тоже просто. Надо создать элемент и связать его с предыдущим и последующим элементом в списке. Рисунок 17.5 иллюстрирует этот процесс.

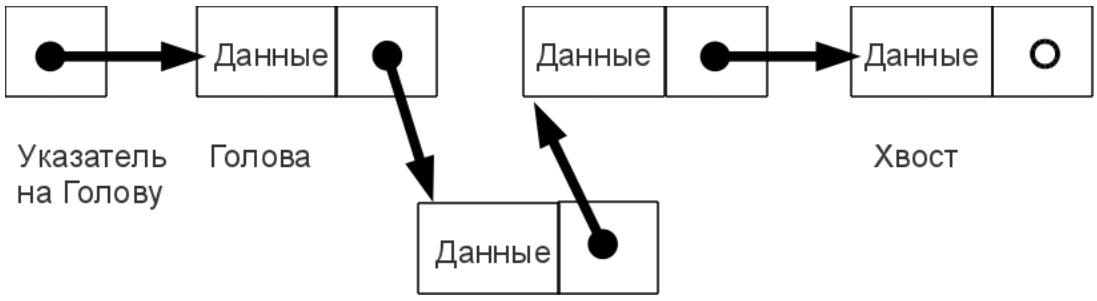


Рис. 17.5. Добавление элемента в связный список.

Связные списки обычно рассматривают как простейшие структуры данных. В языке BASIC мы не можем выделять память так как это делается в большинстве языков, поэтому мы имитируем поведение списка через массивы. В программе 96 мы используем массив

`data$` для хранения текста в списке, массив `nextitem` для хранения указателей на следующий элемент списка и `freeitem` массив для хранения стека свободных (неиспользуемых) индексов.

```
 1 # linkedlist.kbs
 2 # Связный список через массивы
 3 # наибольшее количество элементов списка
 4 n = 8
 5 # данные для элементов списка
 6 dim data$(n)
 7 # указатели на следующий элемент списка
 8 dim nextitem(n)
 9 # список (стек) свободных элементов
10 dim freeitem(n)
11
12 # начальный стек свободных элементов
13 for t = 0 to n-1
14     freeitem[t] = t
15 next t
16 lastfree = n-1
17
18 # начало списка -  -1 = указатель "в никуда"
19 head = -1
20
21 # список из 3 элементов
22 text$ = "Голова"
23 gosub append
24 text$ = "ещё"
25 gosub append
26 text$ = "нечто"
27 gosub append
28 gosub displaylist
29 gosub displayarrays
30 gosub wait
31
32 print "удаляем второй элемент"
33 r = 2
34 gosub delete
35 gosub displaylist
36 gosub displayarrays
37 gosub wait
38
39 print "вставляем элемент 1"
40 r = 1
41 text$ = "что-то"
42 gosub insert
43 gosub displaylist
44 gosub displayarrays
45 gosub wait
46
47 print "вставляем элемент 2"
48 r = 2
49 text$ = "кое-что"
50 gosub insert
51 gosub displaylist
```

```
 52 gosub displayarrays
 53 gosub wait
 54
 55 print "удаляем элемент 1"
 56 r = 1
 57 gosub delete
 58 gosub displaylist
 59 gosub displayarrays
 60 gosub wait
 61
 62 end
 63
 64 wait: ## ждём ввода
 65 input "нажмите ввод? ", garbage$
 66 print
 67 return
 68
 69 # показываем список следуя связям
 70 displaylist:
 71 print "Список..."
 72 k = 0
 73 i = head
 74 do
 75     k = k + 1
 76     print k + " ";
 77     print data$[i]
 78     i = nextitem[i]
 79 until i = -1
 80 return
 81
 82 # Показываем сохранённые в массивах данные: что и как
 83 displayarrays:
 84 print "массивы..."
 85 for i = 0 to n-1
 86     print i + " " + data$[i] + " ->" + nextitem[i] ;
 87     for k = 0 to lastfree
 88         if freeitem[k] = i then print " <<свободно";
 89     next k
 90     if head = i then print " <<голова";
 91     print
 92 next i
 93 return
 94
 95 # вставить в text$ в позицию r
 96 insert:
 97 if r = 1 then
 98     gosub createitem
 99     nextitem[index] = head
100     head = index
101 else
102     k = 2
103     i = head
104     while i <> -1 and k <> r
105         k = k + 1
106         i = nextitem[i]
```



```
107     end while
108     if i <> -1 then
109         gosub createitem
110         nextitem[index] = nextitem[i]
111         nextitem[i] = index
112     else
113         print "Нельзя вставить после конца списка"
114     end if
115 end if
116 return
117
118 # удалить элемент r из связного списка
119 delete:
120 if r = 1 then
121     index = head
122     head = nextitem[index]
123     gosub freeitem
124 else
125     k = 2
126     i = head
127     while i <> -1 and k <> r
128         k = k + 1
129         i = nextitem[i]
130     end while
131     if i <> -1 then
132         index = nextitem[i]
133         nextitem[i] = nextitem[nextitem[i]]
134         gosub freeitem
135     else
136         print "нельзя удалить после конца списка"
137     end if
138 end if
139 return
140
141 # добавляем  text$ в конец связного списка
142 append:
143 if head = -1 then
144     gosub createitem
145     head = index
146 else
147     i = head
148     while nextitem[i] <> -1
149         i = nextitem[i]
150     end while
151     gosub createitem
152     nextitem[i] = index
153 endif
154 return
155
156 # освобождённый элемент из index добавляем к стеку свободных элементов
157 freeitem:
158 lastfree = lastfree + 1
159 freeitem[lastfree] = index
160 return
161
```

```
162 # сохраняем значение text$ в data$ и возвращаем указатель на новое положение
163 createitem:
164 if lastfree < 0 then
165     print "нет свободного места"
166     end
167 end if
168 index = freeitem[lastfree]
169 data$[index] = text$
170 nextitem[index] = -1
171 lastfree = lastfree - 1
172 return
```

Программа 96. Связный список.

Перепишите программу 96 чтобы реализовать стек и очередь через связный список.

17.4 Медленно и не эффективно — сортировка пузырьком

Пузырьковая сортировка, возможно, самый плохой алгоритм, придуманный для сортировки данных. Он очень медлителен и не эффективен за исключением небольших объёмов данных. Это классический пример плохого алгоритма.

Единственная польза этого алгоритма в том, что его легко объяснить и реализовать. Рисунок 17.6 показывает блок-схему этого алгоритма. Суть пузырьковой сортировки в том, что мы многократно просматриваем массив переставляя смежные элементы до тех пор, пока всё не отсортируем.

```
 1 # bubblesort.kbs
 2 # реализация простой сортировки
 3 #
 4 # Пузырьковая сортировка один из самых медлительных
 5 # алгоритмов для сортировки, но он прост для понимания
 6 # и реализации.
 7 #
 8 # Алгоритм Пузырьковой сортировки состоит из
 9 # 1. Просмотра всех элементов массива с перестановкой
10 # соседних значений так, что меньшее значение
11 # перемещается вперёд.
12 # 2. Шаг 1 повторяется многократно, пока не прекратятся
13 # перестановки (массив отсортирован)
14 #
15
16 dim d(20)
17
18 # Заполняем массив случайными
19 # неотсортированными данными
20 for i = 0 to d[?]-1
21     d[i] = rand * 1000
22 next i
23
24 print "*** До сортировки ***"
25 gosub displayarray
26
```

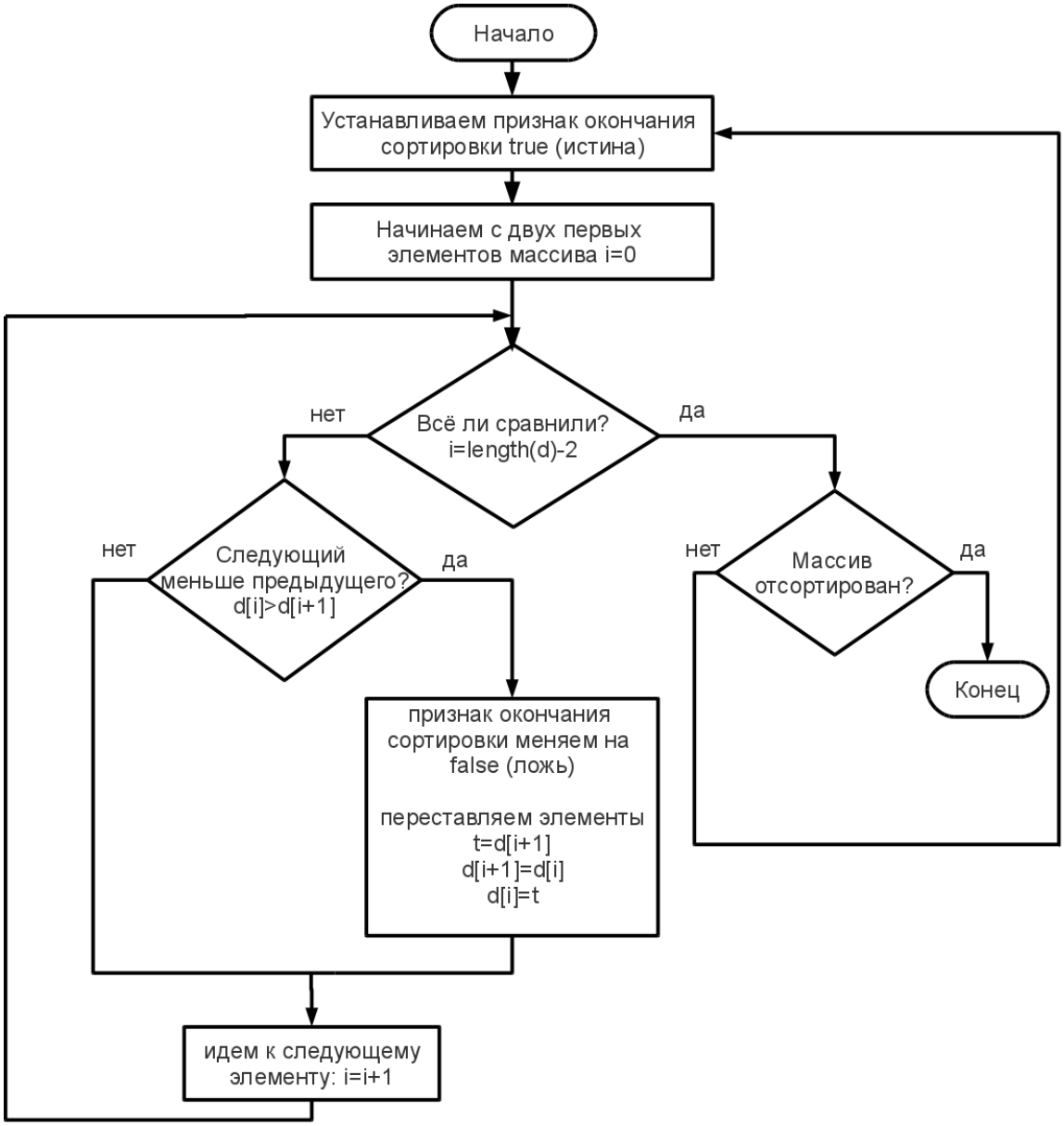



Рис. 17.6. Блок-схема пузырьковой сортировки

```
27 gosub bubblesort
28
29 print "*** Отсортировано ***"
30 gosub displayarray
31 end
32
33 displayarray:
34 # распечатка значений массива
35 for i = 0 to d[?]-1
36     print d[i] + " ";
37 next i
38 print
```

```
39 return
40
41 bubblesort:
42 do
43     sorted = true
44     for i = 0 to d[?] - 2
45         if d[i] > d[i+1] then
46             sorted = false
47             temp = d[i+1]
48             d[i+1] = d[i]
49             d[i] = temp
50         end if
51     next i
52 until sorted
53 return
```

Программа 97. Пузырьковая сортировка.

17.5 Лучшая сортировка — сортировка вставками

Сортировка вставками — ещё один алгоритм сортировки данных. Обычно он быстрее чем пузырьковая сортировка, но в худшем случае может занять столько же времени.

Сортировка вставками получила своё название из-за принципа работы. Во время сортировки просматривает массив элементов (от `index=1` до `length-1`) и вставляет значение в правильное положение, по отношению к предыдущим уже отсортированным элементам массива. Рисунок 17.7 демонстрирует этот принцип.

```
 1 # insertionsort.kbs
 2 # реализация эффективной сортировки
 3
 4 dim d(20)
 5
 6 # заполнение массива случайными данными
 7 for i = 0 to d[?]-1
 8     d[i] = rand * 1000
 9 next i
10
11 print "*** Без сортировки ***"
12 gosub displayarray
13
14 gosub insertionsort
15
16 print "*** Отсортировано ***"
17 gosub displayarray
18 end
19
20 displayarray:
21 # Печать данных массива
22 for i = 0 to d[?]-1
23     print d[i] + " ";
24 next i
25 print
26 return
27
```

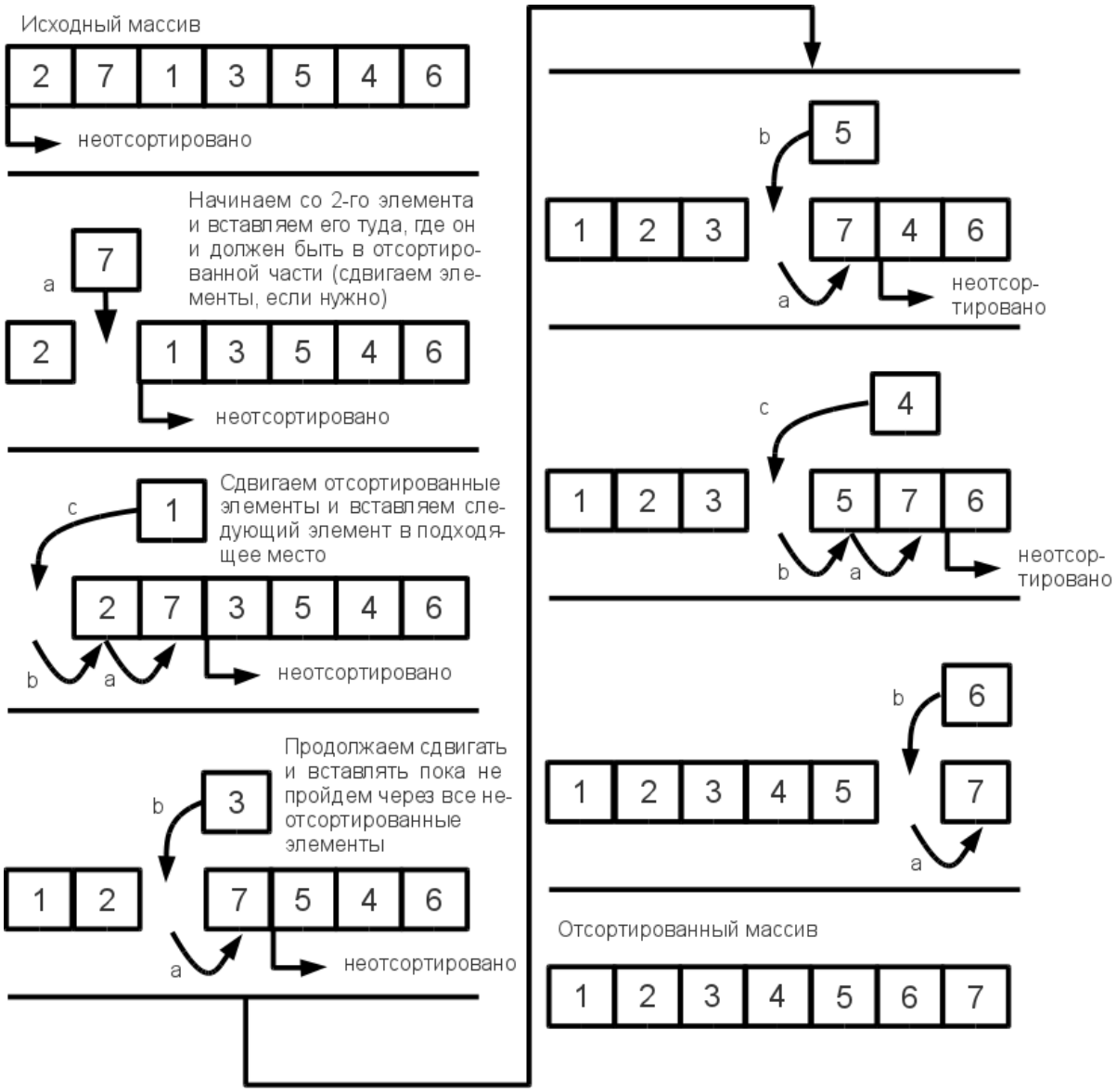



Рис. 17.7. Сортировка вставкой, шаг за шагом.

```
28 insertionsort:
29 # цикл по массиву начиная со второго элемента
30 # берём текущий элемент и вставляем его
31 # в правильное (по сортировке) место
32 # среди цепочки предыдущих отсортированных элементов
33 # двигаясь в обратном направлении от текущего
34 # положения и сдвигая элементы с большим
35 # значением для получения места для текущего
36 # элемента в правильном месте
37 # (в уже частично отсортированном массиве)
38
39 for i = 1 to d[?] - 1
40     currentvalue = d[i]
```

```
41      j = i - 1
42      done = false
43      do
44          if d[j] > currentvalue then
45              d[j+1] = d[j]
46              j = j - 1
47              if j < 0 then done = true
48          else
49              done = true
50          endif
51      until done
52      d[j+1] = currentvalue
53  next i
54  return
```

Программа 98. Сортировка вставкой.

Перепишите программу 98, используя связные списки, как в программе 96.

Узнайте про другие алгоритмы сортировки и реализуйте их на BASIC-256.

Глава 18

Ловушки для ошибок времени исполнения

Если вы работали над примерами и создавали свои программы, то видели ошибки, случающиеся, когда программа выполняется. Такие ошибки называются «ошибки времени исполнения» (по английски: «runtime errors»). BASIC-256 включает группу специальных команд, которые позволяют запрограммировать восстановление после ошибки или их обработку.

Перехват ошибок, даже если вы об этом не думаете, может вызвать проблемы. Его следует использовать только если это нужно и отключать такие ловушки, когда в этом нет необходимости.

18.1 Перехват ошибок

Когда режим перехвата ошибок включён с помощью оператора **onerror**, программа переходит на указанную подпрограмму в момент возникновения ошибки. Если мы посмотрим в программу 99, то увидим, что подпрограмма обработки ошибки вызывается в момент, когда программа пытается прочитать значение неопределённой переменной `z`. Если закомментировать первую строку этой программы и попробовать её запустить, она просто завершится, когда произойдёт ошибка.

```
 1 onerror errortrap
 2
 3 print "z = " + z
 4 print "всё ещё работает после ошибки"
 5 end
 6
 7 errortrap:
 8 print "Я поймал ошибку!"
 9 return
```

Программа 99. Простой пример перехвата ошибок.

```
Я поймал ошибку!
z = 0
всё ещё работает после ошибки
```

Пример вывода программы 99. Простой пример перехвата ошибок.

onerror метка

Создаёт ловушку для ошибки, программа автоматически переходит к подпрограмме, определённой меткой, когда произойдёт ошибка.

18.2 Какая ошибка произошла?

Иногда знание того, что произошла ошибка не достаточно. Есть функции, которые возвращают номер ошибки (**lasterror**), строку программы, где произошла ошибка (**lasterrorline**), текстовое сообщение, поясняющее суть ошибки (**lasterrormessage**) и дополнительные сообщения об ошибке (**lasterrorextra**).

Программа 100 модифицирует предыдущую программу и печатает деталировку произошедшей ошибки.

```
 1 onerror errortrap
 2
 3 print "z = " + z
 4 print "всё ещё работает после ошибки"
 5 end
 6
 7 errortrap:
 8 print "Ловушка активирована!"
 9 print "  Ошибка = " + lasterror
10 print "  На строке = " + lasterrorline
11 print "  Сообщение = " + lasterrormessage
12 return
```

Программа 100. Ловушка для ошибок с комментариями.

```
Ловушка активирована!
Ошибка = 12
На строке = 3
Сообщение = Неизвестная переменная
z = 0
всё ещё работает после ошибки
```

Пример вывода программы 100. Ловушка для ошибок с комментариями.

lasterror или **lasterror()**
lasterrorline или **lasterrorline()**
lasterrormessage или **lasterrormessage()**
lasterrorextra или **lasterrorextra()**


Новое понятие

Четыре функции возвращают информацию о последней перехваченной ошибке. Значения остаются неизменными, пока не произойдёт ещё одна ошибка.

lasterror	Возвращает номер последней перехваченной ошибки. Если ошибок не было, возвращает нуль. См. Дополнение J Коды ошибок.
lasterrorline	Возвращает номер строки, где произошла ошибка
lasterrormessage	Возвращает строку описания ошибки
lasterrorextra	Возвращает строку с дополнительным описанием ошибки. Для большинства ошибок эта функция ничего не выводит.

18.3 Отключение режима перехвата ошибок

Иногда нужно, чтобы программа перехватывала ошибки в одной своей части и не перехватывала в другой. Вы увидите такие примеры в следующих главах.

Оператор **offerror** отключает режим перехвата ошибок. После него, любая ошибка приведёт к остановке выполнения программы.

```
 1 onerror errortrap
 2 print "z = " + z
 3 print "Продолжаю работать после первой ошибки"
 4
 5 offerror
 6 print "z = " + z
 7 print "Продолжаю работать после второй ошибки"
 8
 9 end
10
11 errortrap:
12 print "Перехват ошибок активирован"
13 return
```

Программа 101. Отключение перехвата ошибок.

```
Перехват ошибок активирован
z = 0
Продолжаю работать после первой ошибки
ОШИБКА в строке 6: Неизвестная переменная
```

Пример вывода программы 101. Отключение перехвата ошибок.

Глава 19

Программирование баз данных

В этой главе вы увидите, как BASIC-256 может установить соединение с простой реляционной базой данных для хранения в ней и получения из неё полезных данных.

19.1 Что такое база данных

База данных, говоря упрощённо, — это организованная коллекция чисел, строк и информации другого типа. Наиболее известны базы данных реляционного типа. Реляционные базы данных состоят из четырёх главных составляющих: таблиц, строк, колонок и отношений между ними (связей, см табл 19.1).

19.2 Язык SQL

Большинство реляционных баз данных сегодня используют язык, называемый SQL для выбора и управления данными. SQL это акроним от Structured Query Language — Структурированный язык запросов. Первый SQL язык был разработан IBM в 1970 году и стал главным языком используемым в реляционных базах.

SQL достаточно мощный язык и был реализован многими компаниями за все годы с момента появления. В результате такого многообразия появилось и используется множество различных диалектов языка SQL. BASIC-256 использует SQLite в качестве системы управления базой данных. Пожалуйста ознакомьтесь с диалектом этого SQL языка на вебстранице SQLite: http://www.sqlite.org, поскольку данный диалект используется в приведённых ниже примерах.

Таблица 19.1. Главные компоненты реляционной базы данных

Таблица	Таблица состоит из заранее определённого числа колонок и любого количества строк содержащих информацию об объектах или субъектах. Таблицу также называют отношением.
Строка	Также называемая кортеж
Колонка	Также называют атрибут
Связи	Ссылка ключевого столбца одной таблицы на колонку другой таблицы, создающее взаимосвязь таблиц.

19.3 Создание базы и добавление данных в неё

SQLite база данных не требует установки сервера или создания какой-то сложной системы. База данных и все её части хранятся в простом файле на вашем компьютере. Этот файл можно скопировать на другой компьютер и использовать там без проблем. Первая программа (программа 102. Создаём базу данных) создаёт новый файл базы данных и таблицы. Таблицы представлены на ER-диаграмме[1] (см. рис. 19.1)

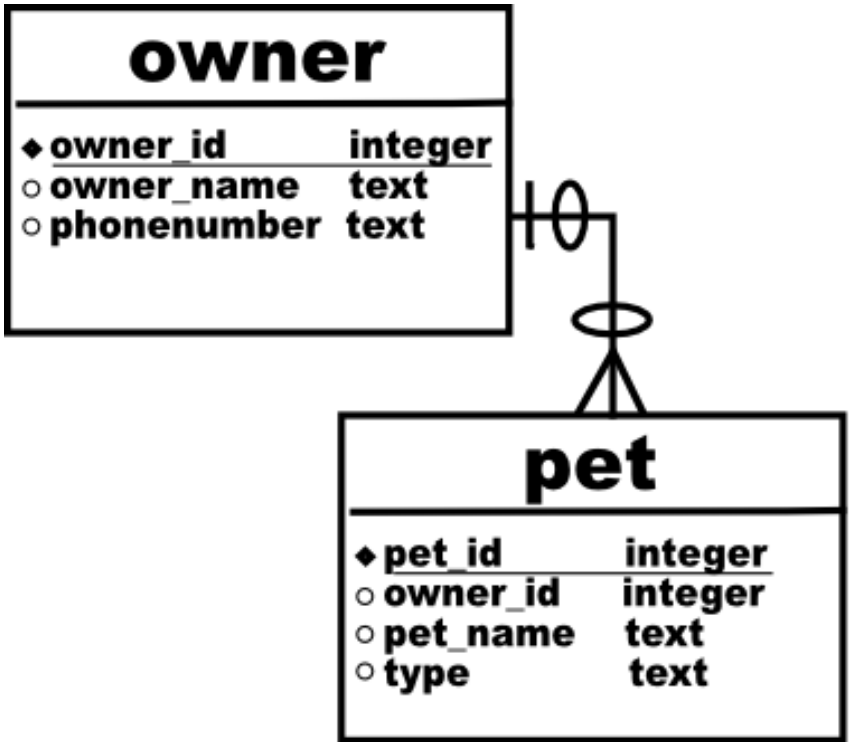


Рис. 19.1. ER-диаграмма базы данных

```
 1 # chapter 19 dbcreate.kbs
 2 # Удаляем прежний файл базы данных и создаём базу с двумя таблицами
 3 errors = 0
 4 file$ = "pets.sqlite3"
 5 if exists(file$) then kill(file$)
 6 dbopen file$
 7
 8 stmt$ = "CREATE TABLE owner (owner_id INTEGER, ownername TEXT,
    phonenumber TEXT, PRIMARY KEY (owner_id));"
 9 gosub execute
10
11 stmt$ =  "CREATE TABLE pet (pet_id INTEGER, owner_id INTEGER,
   petname TEXT, type TEXT, PRIMARY KEY (pet_id), FOREIGN KEY (owner_id
   ) REFERENCES owner (owner_id));"
12 gosub execute
13
```

[1]Entity Relationship Diagram (англ.) — диаграмма отношений сущностей (*прим. переводчика*).

```
14 # Закрываем всё.
15 dbclose
16 print file$ + " создан. " + errors + " ошибок."
17 end
18
19 execute:
20 print stmt$
21 onerror executeerror
22 dbexecute stmt$
23 offerror
24 return
25
26 executeerror:
27 errors = errors + 1
28 print "ERROR: " + lasterror + " " + lasterrormessage + " " +
   lasterrorextra
29 return
```

Программа 102. Создаём базу данных.

```
CREATE TABLE owner (owner_id INTEGER, ownername TEXT, phonenumber TEXT,
   PRIMARY KEY (owner_id));
CREATE TABLE pet (pet_id INTEGER, owner_id INTEGER, petname TEXT, type
   TEXT, PRIMARY KEY (pet_id), FOREIGN KEY (owner_id) REFERENCES owner (
   owner_id));
pets.sqlite3 создан. 0 ошибок.
```

Пример вывода программы 102. Создаём базу данных.

В приведённой выше программе вы видели три новых функции для работы с базами данных: **dbopen** — открывает файл базы данных или создаёт новый, если файл не существует, **dbexecute** — выполняет SQL команду на открытой базе данных, и **dbclose** — закрывает открытый файл базы данных.

dbopen *имя_файла*

Открывает файл SQLite базы данных. Если база не существует, создаётся пустой файл базы данных.

dbexecute *sql-запрос*

Выполняет *sql-запрос* на текущей открытой базе данных. Эта функция не возвращает данных, но возможно перехватить ошибку, если таковая возникнет в процессе выполнения запроса.

dbclose

Закрывает текущий открытый файл SQLite базы данных. Эта функция гарантирует, что все данные будут записаны в файл базы.

Эти же три функции можно использовать и для выполнения других SQL запросов к базе. Например, вставка новых строк в таблицы (см программу 103) с помощью `INSERT INTO` и обновление имеющихся строк таблицы с помощью SQL-запроса `UPDATE` (см. программу 104).


```
 1 # chapter19 dbinsert.kbs
 2 # Добавление строк в базу данных
 3
 4 file$ = "pets.sqlite3"
 5 dbopen file$
 6
 7 owner_id = 0
 8 pet_id = 0
 9
10 ownername$ = "Джим": phonenumber$ = "555-3434"
11 gosub addowner
12 petname$ = "Пятнышко": type$ = "Кот"
13 gosub addpet
14 petname$ = "Фред": type$ = "Кот"
15 gosub addpet
16 petname$ = "Элвис": type$ = "Кот"
17 gosub addpet
18
19 ownername$ = "Сью": phonenumber$ = "555-8764"
20 gosub addowner
21 petname$ = "Альфред": type$ = "Кот"
22 gosub addpet
23 petname$ = "Фидо": type$ = "Собака"
24 gosub addpet
25
26 ownername$ = "Эми": phonenumber$ = "555-9932"
27 gosub addowner
28 petname$ = "Домино": type$ = "Собака"
29 gosub addpet
30
31 ownername$ = "Ди": phonenumber$ = "555-4433"
32 gosub addowner
33 petname$ = "Сэм": type$ = "Козёл"
34 gosub addpet
35
36 # закрываем всё
37 dbclose
38 end
39
40 addowner:
41 owner_id = owner_id + 1
42 stmt$ = "INSERT INTO owner (owner_id, ownername, phonenumber) VALUES
    (" + owner_id + "," + chr(34) + ownername$ + chr(34) + "," + chr
   (34) + phonenumbe    r$ + chr(34) + ");"
43 print stmt$
44 onerror adderror
45 dbexecute stmt$
46 offerror
47 return
48
49 addpet:
50 pet_id = pet_id + 1
```

```
51 stmt$ = "INSERT INTO pet (pet_id, owner_id, petname, type) VALUES ("
    + pet_id + "," + owner_id + "," + chr(34) + petname$ + chr(34) +
   "," + chr(34) + type$ + chr(34) + ");"
52 print stmt$
53 onerror adderror
54 dbexecute stmt$
55 offerror
56 return
57
58 adderror:
59 print "ERROR: " + lasterror + " " + lasterrormessage + " " +
   lasterrorextra
60 return
```

Программа 103. Вставка строк в базу данных.

```
INSERT INTO owner (owner_id, ownername, phonenumber) VALUES (1,"
   Джим","555-3434");
INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (1,1,"
   Пятнышко","Кот");
INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (2,1,"Фред","
   Кот");
INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (3,1,"Элвис","
   Кот");
INSERT INTO owner (owner_id, ownername, phonenumber) VALUES (2,"
   Сью","555-8764");
INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (4,2,"Альфред","
   Кот");
INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (5,2,"Фидо","
   Собака");
INSERT INTO owner (owner_id, ownername, phonenumber) VALUES (3,"
   Эми","555-9932");
INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (6,3,"Домино","
   Собака");
INSERT INTO owner (owner_id, ownername, phonenumber) VALUES (4,"
   Ди","555-4433");
INSERT INTO pet (pet_id, owner_id, petname, type) VALUES (7,4,"Сэм","
   Козёл");
```

Пример вывода программы 103. Вставка строк в базу данных.

```
 1 # chapter 19 dbupdate.kbs
 2 # Обновление данных
 3
 4 dbopen "pets.sqlite3"
 5
 6 # обновляем содержимое строки
 7 s$ = "UPDATE owner SET phonenumber = " + chr(34) + "555-5555" + chr
    (34) + " where owner_id = 1;"
 8 print s$
 9 dbexecute s$
10 dbclose
```

Программа 104. Обновление данных строки.

```
UPDATE owner SET phonenumber = "555-5555" where owner_id = 1;
```

Пример вывода программы 104. Обновление данных строки.

19.4 Получение информации из базы данных

Итак, мы умеем открывать и закрывать файл базы данных, а также исполнять SQL-запрос, который не возвращает данных. База данных, из которой нельзя получить информацию достаточно бесполезна.

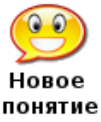
Оператор `SELECT` (выбор) языка SQL позволяет нам получать необходимые данные. После выполнения SQL-запроса содержащего оператор `SELECT` создаётся набор записей, содержащих информацию из строк и столбцов таблиц базы данных. Программа 105 демонстрирует три различных запроса на выборку данных и как BASIC-256 программа может их использовать.

```
 1 # chapter 19 dbselect.kbs
 2 # Get data from the pets database
 3
 4 dbopen "pets.sqlite3"
 5
 6 # Показываем владельцев и номера их телефонов
 7 print "Владельцы и номера телефонов"
 8 dbopenset "SELECT ownername, phonenumber FROM owner ORDER BY
    ownername;"
 9 while dbrow()
10     print dbstring(0) + " " + dbstring(1)
11 end while
12 dbcloseset
13
14 print
15
16 # Показываем вадельцев и их питомцев
17 print "Владельцы и питомцы"
18 dbopenset "SELECT owner.ownername, pet.pet_id, pet.petname, pet.type
    FROM owner JOIN pet ON pet.owner_id = owner.owner_id ORDER BY
   ownername, petname;"
19 while dbrow()
20     print dbstring(0) + " " + dbint(1) + " " + dbstring(2) + " " +
   dbstring(3)
21 end while
22 dbcloseset
23
24 print
25
26 # Показываем среднее значение количества питомцев
27 print "Среднее количество питомцев"
28 dbopenset "SELECT AVG(c) FROM (SELECT COUNT(*) AS c FROM owner JOIN
   pet ON pet.owner_id = owner.owner_id GROUP BY owner.owner_id) AS
   numpets;"
29 while dbrow()
30     print dbfloat(0)
31 end while
32 dbcloseset
33
34 # Закрываем всё
35 dbclose
```

Программа 105. Получение данных из базы.

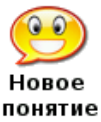
```
Владельцы и номера телефонов
Джим 555-5555
Ди 555-4433
Сью 555-8764
Эми 555-9932

Владельцы и питомцы
Джим 1 Пятнышко Кот
Джим 2 Фред Кот
Джим 3 Элвис Кот
Ди 7 Сэм Козёл
Сью 4 Альфред Кот
Сью 5 Фидо Собака
Эми 6 Домино Собака

Среднее количество питомцев
1.75
```

Пример вывода программы 105. Получение данных из базы.



dbopenset *sql-запрос*

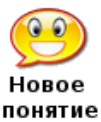
Выполняет запрос на выборку данных (`SELECT`) из базы данных создавая набор строк как результат, так что программно можно их читать. Результатом запроса может быть 0 или более строк в зависимости от запроса.



dbrow
dbrow()

Читает очередную строку из набора строк сформированного функцией **dbopenset**. Если строчки кончились, возвращает **false** (ложь)
Для чтения первой строки из набора необходимо использовать **dbrow** сразу после **dbopenset.**

dbint (*колонка*)
dbfloat (*колонка*)
dbstring (*колонка*)



Эти функции возвращают данные из текущей строки запроса. Необходимо знать цифровой номер (начинающийся с нуля) колонки с необходимыми данными.

dbint	Возвращает содержимое ячейки как целое.
dbfloat	Возвращает содержимое ячейки как десятичную дробь.
dbstring	Возвращает содержимое ячейки как строку.

dbcloseset

Закрывает и сбрасывает результат предыдущего запроса функции **dbopenset**

Глава 20

Сетевые соединения

В этой главе мы обсудим как использовать сетевые возможности BASIC-256. Сетевые функции в BASIC-256 реализованы как простое соединение через сокет с использованием TCP протокола (Transmission Control Protocol — протокол управления передачей). Здесь вы не найдёте полного введения в программирование для TCP/IP сокетов.

20.1 Соединение с сокетом

Сетевые сокеты создают соединение между двумя компьютерами или программами. Пакеты с информацией могут быть отправлены или получены в двух направлениях через такое соединение.

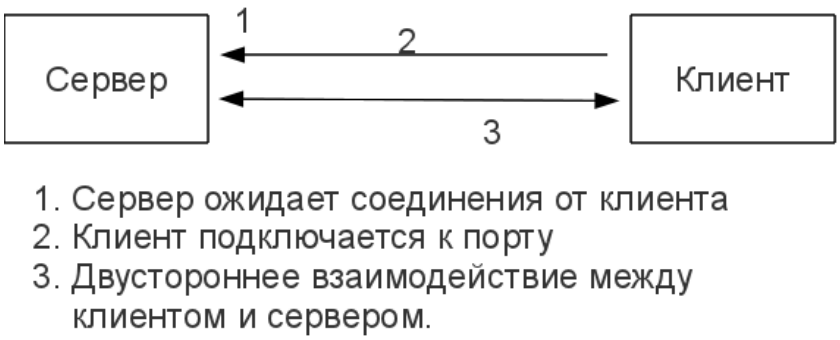


Рис. 20.1. Взаимодействие между сокетами.

Когда вы звоните по телефону, то вы (как клиент) должны знать номер телефона абонента, кому вы звоните (он — сервер). Мы называем такие номера — IP-адрес. BASIC-256 использует IP-адреса версии 4, которые обычно записывают четырьмя десятичными числами, разделёнными точкой (A.B.C.D, где A,B,C и D в диапазоне от 0 до 255).

Дополнительно к IP-адресу сервера, клиент и сервер должны общаться через определённый порт. Можно представлять порт, как дополнительный номер к многоканальному телефонному номеру. Если вам (как клиенту) нужен определённый человек в большой компании (имеющей один общий входящий номер), то вам необходимо набрать дополнительно внутренний номер (порт) в ответ на сообщение (телефонного сервера) о соединении с общим номером компании (IP-адресом).

Номера портов могут быть любыми в диапазоне от 0 до 65535, но обычно порты в диапазоне от 0 до 1023 заняты различными системными службами (приложениями), поэтому такие порты не следует использовать.

Пример простого сервера и клиента.

```
1 # simple_server.kbs пример сервера
2 print "Приём запросов по порту 9999 на IP:" + netaddress()
3 NetListen 9999
4 NetWrite "Тестовое сообщение простого сервера."
5 NetClose
```

Программа 106. Простой сервер.

```
1 # simple_client.kbs пример клиента
2 input "Введите IP-адрес сервера: ", addr$
3 if addr$ = "" then addr$ = "127.0.0.1"
4 #
5 NetConnect addr$, 9999
6 print NetRead
7 NetClose
```

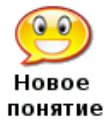
Программа 107. Простой клиент.

```
Приём запросов по порту 9999 на IP:10.1.2.103
```

Пример вывода программы 106. Простой сервер.

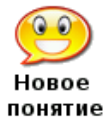
```
Введите IP-адрес сервера:
Тестовое сообщение простого сервера.
```

Пример вывода программы 107. Простой клиент.



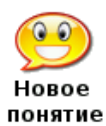
netaddress
netaddress()

Эта функция возвращает числовой Ipv4 адрес сетевого компьютера



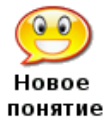
netlisten *номер_порта*
netlisten (*номер_порта*)
netlisten *номер_сокета, номер_порта*
netlisten (*номер_сокета, номер_порта*)

Открывает сетевое соединение (со стороны сервера) на указанном порту и ожидает соединений от других программ. Если *номер_сокета* не указан, используется нулевой (0) номер.



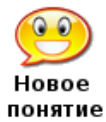
netclose
netclose()
netclose *номер_сокета*
netclose (*номер_сокета*)

Завершает указанное сетевое соединение (сокет). Если *номер_сокета* не указан, сокет с номером ноль (0) будет закрыт.



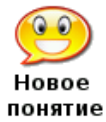
netwrite*строка*
netwrite (*строка*)
netwrite *номер_сокета, строка*
netwrite (*номер_сокета, строка*)

Посылает текстовое сообщение в указанное сетевое соединение. Если *номер_сокета* не указан, используется нулевой (0) номер.



netconnect *имя_сервера, номер_порта*
netconnect (*имя_сервера, номер_порта*)
netconnect *номер_сокета, имя_сервера, номер_порта*
netconnect (*номер_сокета, имя_сервера, номер_порта*)

Открывает соединение с сервером со стороны клиента по указанному порту. В параметре *имя_сервера* указывается IP адрес или сетевое имя сервера. Если *номер_сокета* не указан, используется нулевой (0) номер.



<td>**netread**
netread()
netread (*номер_сокета*)

Читает данные из указанного сетевого соединения и возвращает в виде текстовой строки. Эта функция является блокирующей — она ждёт, пока данные не будут получены. Если *номер_сокета* не указан, используется нулевой (0) номер.

20.2 Сетевой чат

Следующий пример использует ещё одну сетевую функцию (**netdata**). Использование этой новой функции позволит нашему сетевому клиенту обрабатывать такие события, как нажатия клавиш и чтение сетевых данных только тогда, когда они действительно есть.

Программа сетевого чата (программа 108) комбинирует в себе и свойства сервера и свойства клиента. Если при запуске программы она не сможет соединиться с сервером, это событие будет перехвачено и, программа сама станет сервером. Это один из многих способов дать возможность одной программе выполнять как клиентскую, так и серверную роли.

```
 1 # chat.kbs чат
 2 # использует порт 9999 когда работает сервером
 3
 4 input "Адрес сервера чата (ввод для локальной машины)?", addr$
 5 if addr$ = "" then addr$ = "127.0.0.1"
 6 #
 7 # Пробуем соединиться с сервером, если неудачно - становимся сервером сами
 8 OnError startserver
 9 NetConnect addr$, 9999
10 OffError
11 print "Соединение с сервером установлено"
12
```

```
13 chatloop:
14 while true
15     # получаем код нажатой клавиши и отсылаем его
16     k = key
17     if k <> 0 then
18         gosub show
19         netwrite string(k)
20     end if
21     # получаем данные из сети и показываем их
22     if NetData() then
23         k = int(NetRead())
24         gosub show
25     end if
26     pause .01
27 end while
28 end
29
30 show:
31 if k=16777220 then
32     print
33 else
34     print chr(k);
35 end if
36 return
37
38 startserver:
39 OffError
40 print "Сервер запущен, ждём чат-клиента"
41 NetListen 9999
42 print "Клиент установил соединение"
43 goto chatloop
44 return
```

Программа 108. Сетевой чат.

Запустите две копии BASIC-256 с программой 108 Сетевой чат. На первой запущенной копии заработает сервер, а на второй клиент. На клиенте напишите «Привет сервер!», а на сервере напишите в ответ «Привет клиент!». Вывод программы для сервера и для клиента приведены ниже.

```
Адрес сервера чата (ввод для локальной машины)?
Сервер запущен, ждём чат-клиента
Клиент установил соединение
ПРИВЕТ СЕРВЕР !
ПРИВЕТ КЛИЕНТ !
```

Пример вывода программы 108.1. Сетевой чат со стороны сервера.

```
Адрес сервера чата (ввод для локальной машины)?
Соединение с сервером установлено
ПРИВЕТ СЕРВЕР !
ПРИВЕТ КЛИЕНТ !
```

Пример вывода программы 108.2. Сетевой чат со стороны клиента.

Новое понятие

netdata
netdata()

Возвращает истину (**true**), если есть сетевые данные ожидающие чтения. Это позволяет программе продолжать работать без ожидания прибытия сетевого пакета.

Большая программа

Большая программа этой главы создаёт сетевую аркадную игру в танки для двух игроков.
У каждого из игроков на экране имеется белый танк (свой) и чёрный танк (чужой). Используя стрелки можно управлять движением танка. Стреляем пробелом.

```
 1 # battle.kbs танковое сражение
 2 # используем порт 9998 когда сервер
 3
 4 kspace = 32
 5 kleft = 16777234
 6 kright = 16777236
 7 kup = 16777235
 8 kdown = 16777237
 9 dr = pi / 16     # смена направления
10 dxy = 2.5        # скорость передвижения
11 scale = 20       # размер танка
12 shotscale = 4    # размер снаряда
13 shotdxy = 5      # скорость снаряда
14 port = 9998      # номер порта для связи с сервером
15
16 dim tank(30)
17 tank = {-1,-.66, -.66,-.66, -.66,-.33, -.33,-.33, 0,-1, .33,-.33,
   .66,-.33, .66,-.66, 1,-.66, 1,1, .66,1, .66,.66, -.66,.66, -.66,1,
   -1,1}
18 dim shot(14)
19 shot = {0,-1, .5,-.5, .25,0, .5,.75, -.25,.75, -.25,0, -.5,-.5}
20
21 print "Танковое сражение - вы на белом танке."
22 print "Ваша задача поразить выстрелом чёрный"
23 print "танк. Используйте стрелки для перемещения"
24 print "и пробел для стрельбы."
25 print
26 input "Адрес сервера (ввод для локальной машины)?", addr$
27 if addr$ = "" then addr$ = "127.0.0.1"
28
29 # Пробуем соединиться с сервером, если неудачно - становимся сервером сами
30 OnError startserver
31 NetConnect addr$, port
32 OffError
33 print "Соединение с сервером установлено"
34
35 playgame:
36
37 myx = 100
38 myy = 100
```



```
39 myr = 0
40 # Позиция снаряда (для себя): направление и оставшееся расстояние
41 # (снарядов нет, если mypl=0)
42 mypx = 0
43 mypy = 0
44 mypr = 0
45 mypl = 0
46 yourx = 200
47 youry = 200
48 yourr = pi
49 # Позиция снаряда (для противника): направление и оставшееся расстояние
50 # (снарядов нет, если yourpl=0)
51 yourpx = 0
52 yourpy = 0
53 yourpr = 0
54 yourpl = 0
55 gosub writeposition
56
57 fastgraphics
58 while true
59     # Получаем значение нажатой клавиши и перемещаем танк на экране
60     k = key
61     if k <> 0 then
62         if k = kup then
63             myx = myx + sin(myr) * dxy
64             myy = myy - cos(myr) * dxy
65         end if
66         if k = kdown then
67             myx = myx - sin(myr) * dxy
68             myy = myy + cos(myr) * dxy
69         end if
70         if k = kspace then
71             mypr = myr
72             mypx = myx + sin(mypr) * scale
73             mypy = myy - cos(mypr) * scale
74             mypl = 100
75         end if
76         if myx < scale then myx = graphwidth - scale
77         if myx > graphwidth-scale then myx = scale
78         if myy < scale then myy = graphheight - scale
79         if myy > graphheight-scale then myy = scale
80         if k = kleft then myr = myr - dr
81         if k = kright then myr = myr + dr
82         gosub writeposition
83     end if
84     # перемещаем свой снаряд (если есть снаряды)
85     if mypl > 0 then
86         mypx = mypx + sin(mypr) * shotdxy
87         mypy = mypy - cos(mypr) * shotdxy
88         if mypx < shotscale then mypx = graphwidth - shotscale
89         if mypx > graphwidth-shotscale then mypx = shotscale
90         if mypy < shotscale then mypy = graphheight - shotscale
91         if mypy > graphheight-shotscale then mypy = shotscale
92         if (mypx-yourx)^2 + (mypy-youry)^2 < scale^2 then
93             NetWrite "!"
```

```
 94              print "Противник убит, Игра закончена."
 95              end
 96          end if
 97          mypl = mypl - 1
 98          gosub writeposition
 99      end if
100      # Получаем из сети позицию
101      gosub getposition
102      #
103      gosub draw
104      #
105      pause .1
106  end while
107
108  writeposition: ###
109  # 10 символов для x, 10 символов для y, 10 символов для r (поворот)
110  position$ =  left(myx + "          ",10) + left(myy + "
    ",10) + left(myr + "          ",10) + left(mypx + "          ",10) +
    left(mypy + "          ",10)+ left(mypr + "          ",10)+ left(mypl
     + "          ",10)
111  NetWrite position$
112  return
113
114  getposition: ###
115  # Получаем из сети позицию и устанавливаем переменные для противника
116  while NetData()
117      position$ = NetRead()
118      if position$ = "!" then
119          print "Вы убиты. Игра окончена."
120          end
121      end if
122      yourx = 300 - float(mid(position$,1,10))
123      youry = 300 - float(mid(position$,11,10))
124      yourr = pi + float(mid(position$,21,10))
125      yourpx = 300 - float(mid(position$,31,10))
126      yourpy = 300 - float(mid(position$,41,10))
127      yourpr = pi + float(mid(position$,51,10))
128      yourpl = pi + float(mid(position$,61,10))
129  end while
130  return
131
132  draw: ###
133  clg
134  color green
135  rect 0,0,graphwidth,graphheight
136  color white
137  stamp myx, myy, scale, myr, tank
138  if mypl > 0 then
139      stamp mypx, mypy, shotscale, mypr, shot
140  end if
141  color black
142  stamp yourx, youry, scale, yourr, tank
143  if yourpl > 0 then
144      color red
145      stamp yourpx, yourpy, shotscale, yourpr, shot
```



```
146 end if
147 refresh
148 return
149
150 startserver:
151 OffError
152 print "Сервер запущен, ожидаем присоединения клиента"
153 NetListen port
154 print "Клиент соединился с сервером"
155 goto playgame
156 return
```

Программа 109. Танковое сражение по сети[1].

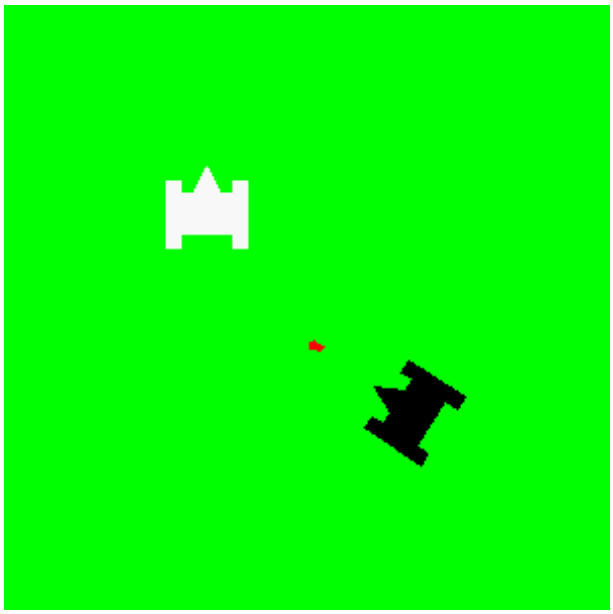


Рис. 20.2. Пример графического экрана программы 109. Танковое сражение по сети.

[1]Программа нуждается в доработке: а) Выстрел в сторону может попасть в противника, т. к. поле рассматривается как сферическое, при этом снаряд может пройти и через ваш танк, не причинив ему вреда, что довольно странно; б) Если первый выстрел ещё не долетел, а вы выстрелили вновь — первый выстрел пропадёт (*прим. редактора*).

Приложение A

Установка BASIC-256 на компьютер или переносной USB Flash

Эта глава описывает процесс загрузки и установки BASIC-256 на компьютер с операционной системой Microsoft Windows™. Эти инструкции написаны для Windows XP и Firefox 3.x в качестве интернет-браузера. Если у вас другая система, отдельные детали могут отличаться, но основные шаги будут аналогичны.

A.1 Загрузка

Подключитесь к сети Интернет, зайдите на сайт http://www.basic256.org и проследуйте по ссылке «download»[1], которая приведёт вас на сайт проекта на Sourceforge[2]. Найдите на странице зелёную кнопку «Download Now!»[3], чтобы начать загрузку.

Во время загрузки вас могут спросить, что делать с загружаемым файлом — Щёлкните на кнопке «Сохранить файл» (Рис. A.2).

Firefox должен показать окно «Загрузки» и загрузить установочный файл BASIC-256. Когда процесс загрузки завершится, у вас будет окошко, аналогичное картинке Рис. A.3. Не закрывайте это окошко, оно пригодится, чтобы начать установку.

A.2 Установка

После того, как файл загружен (см. Рис.A.3), щёлкните мышкой по загруженному файлу из списка загрузки. Вы увидете одно или два диалоговых окна, уточняющих ваше намерение запустить файл установщика (см. Рис. A.4 и A.5). Щёлкните на «OK» и «Run» соответственно в этих окошках.

После того, как все предупреждающие окна будут закрыты вы увидите окно установщика BASIC-256. Щёлкните по кнопке «Next» на первом же экране (см. Рис.A.6).

Прочтите и согласитесь с лицензией GNU GPL и щёлкните по кнопочке «I Agree»[4] (см. Рис. A.7). Лицензия GNU GPL одна из наиболее распространённых лицензий для свободного программного обеспечения. Она даёт вам возможность запускать, изучать, изменять

[1] Ищите на страничке ссылку с текстом «Download Windows Installer or Source Code (LINUX and Mac OS X») (Прим. переводчика).

[2] http://sourceforge.net/projects/kidbasic/ Sourceforge — громадное хранилище свободных проектов. (Прим. переводчика).

[3] Загрузить сейчас!

[4] «Я согласен»

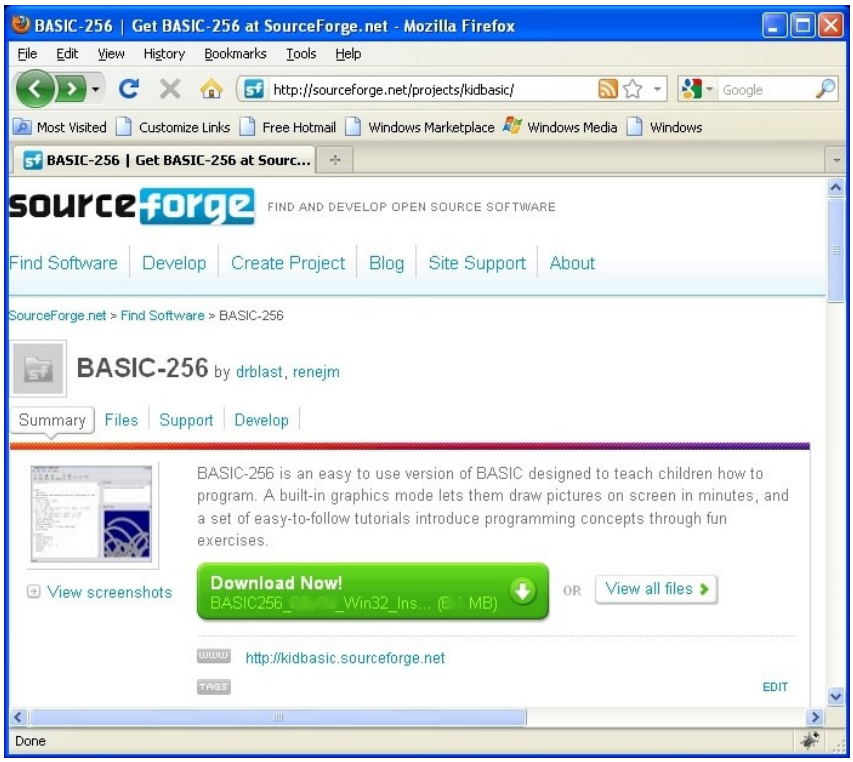


Рис. А.1. BASIC-256 на Sourceforge.

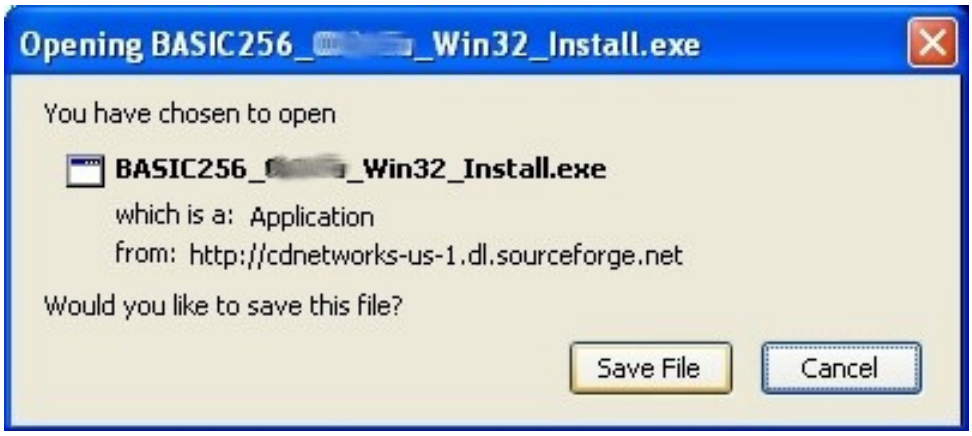


Рис. А.2. Сохранение установочного файла.

и распространять изменённые копии программы. Эта лицензия относится только к самой программе BASIC-256, но не к содержимому этой книги.

Следующий экран установщика (см. Рис. А.8) уточняет у вас, что вы хотите установить. Если вы устанавливаете BASIC-256 на USB-флешку или другой переносной носитель, рекомендуется снять галочку «Start Menu Shortcuts»[5]. Для большинства пользователей,

[5] Добавить пункт в стартовое меню.

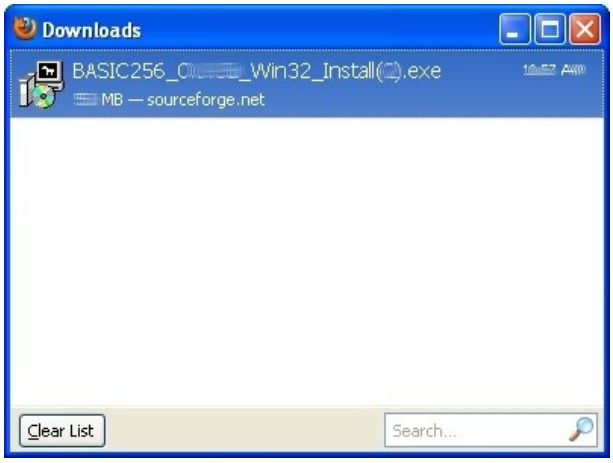


Рис. A.3. Файл загружен.

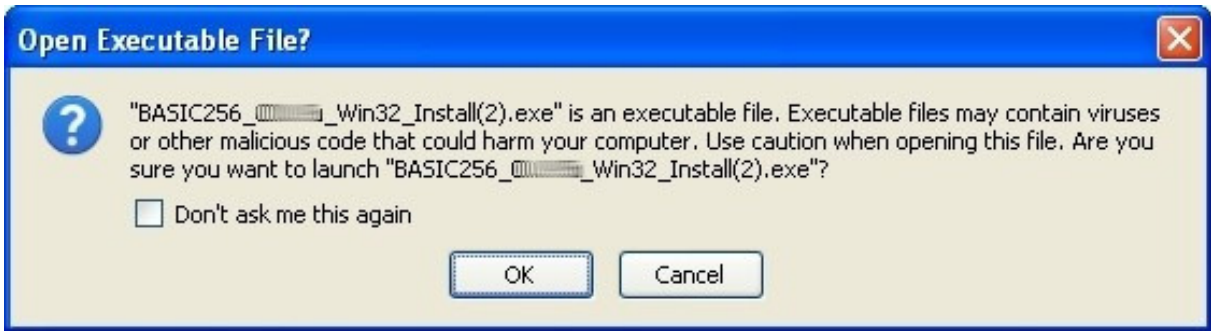


Рис. A.4. Предупреждение перед открытием выполняемого файла.

устанавливающих программу на жёсткий диск, рекомендуем полную установку. Нажмите «Next»[6].

Последний экран перед началом установки просит указать каталог, куда следует установить исполняемые файлы BASIC-256 (см. Рис A.9). Если вы устанавливаете программу на жёсткий диск, рекомендуем оставить значение по умолчанию. Нажмите «Install»[7].

Установка завершена, если вы увидели экран как на Рис. A.10. Нажмите «Close»[8].

A.3 Запуск BASIC-256

Установка завершена. Чтобы запустить программу нажмите кнопку «Start» («Пуск») меню Windows и затем «All Programs» («Все программы») (см. Рис. A.11).

В общем меню, найдите подменю BASIC-256. Вы можете открыть программу щелчком по ней, или удалить программу или посмотреть документацию к программе из этого меню ((см. Рис. A.12).

[6] Далее
[7] Установить
[8] Закрыть

Рис. A.5. Предупреждение проверки безопасности перед запуском файла.

A.4 Установка и запуск BASIC-256 в Linux[9]

Один из переводчиков данной книги (Сергей Ирюпин) является мантейнером пакета basic256 для дистрибутивов ALT Linux, поэтому для дистрибутивов ALT Linux смотрите подходящий пакет на http://sisyphus.ru/ru/srpm/Sisyphus/basic256. На момент подготовки книги последними пакетами были:
ветка **4.0** http://ftp.altlinux.org/pub/distributions/ALTLinux/4.0/branch/files/i586/RPMS/basic25
ветка **4.1** http://ftp.altlinux.org/pub/distributions/ALTLinux/4.1/branch/files/i586/RPMS/basic25
ветка **p5** http://ftp.altlinux.org/pub/distributions/ALTLinux/p5/branch/files/i586/RPMS/basic256
ветка **5.1** http://ftp.altlinux.org/pub/distributions/ALTLinux/5.1/branch/files/i586/RPMS/basic25

A.4.1 Как установить BASIC-256 в Linux

Для дистрибутивов ALT Linux необходимо настроить репозиторий и обновить или установить пакет через synaptic или apt.

```
apt-get install basic256
```

Для rpm-based дистрибутивов можно поставить консольной командой:

```
rpm -Uvh <имя_пакета>.rpm
```

[9]Этот раздел добавлен переводчиком.

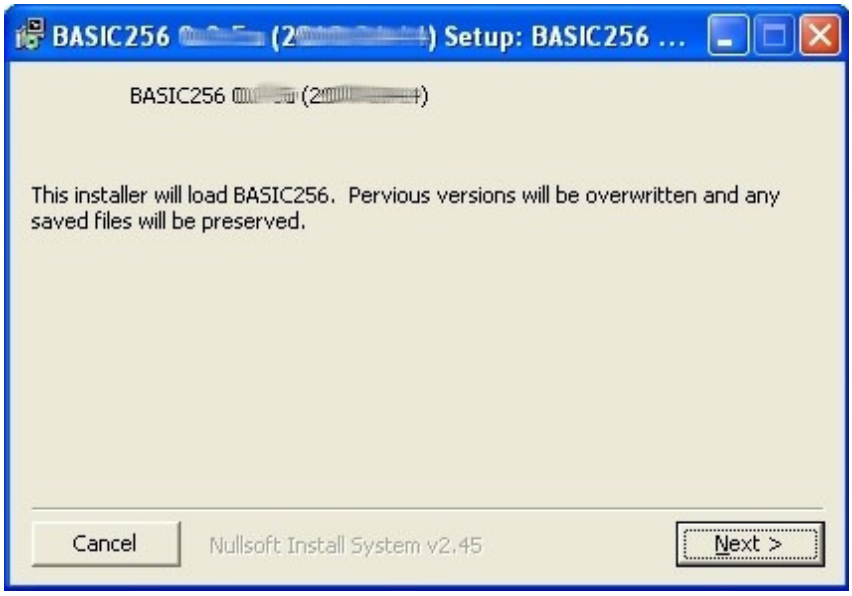


Рис. А.6. Приветственное окно установщика.

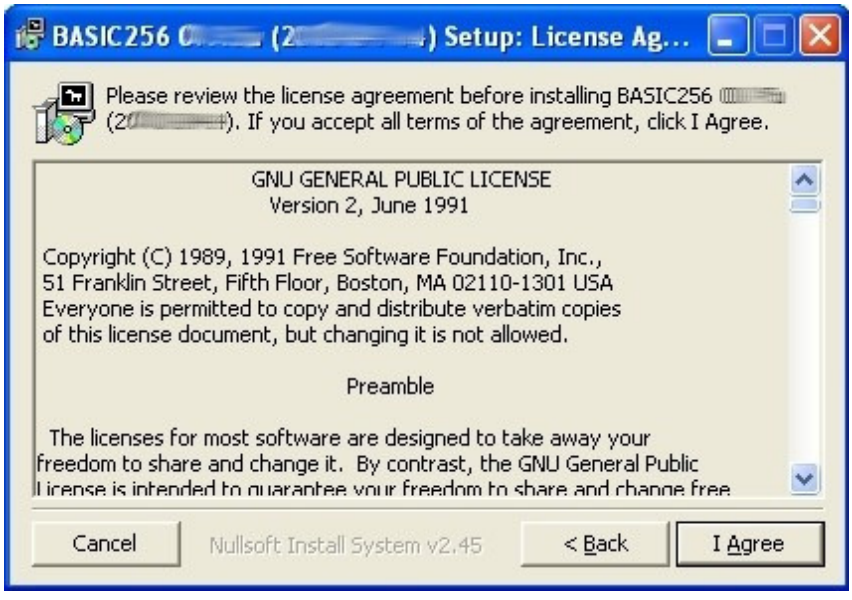


Рис. А.7. Установщик, экран GPL лицензии.

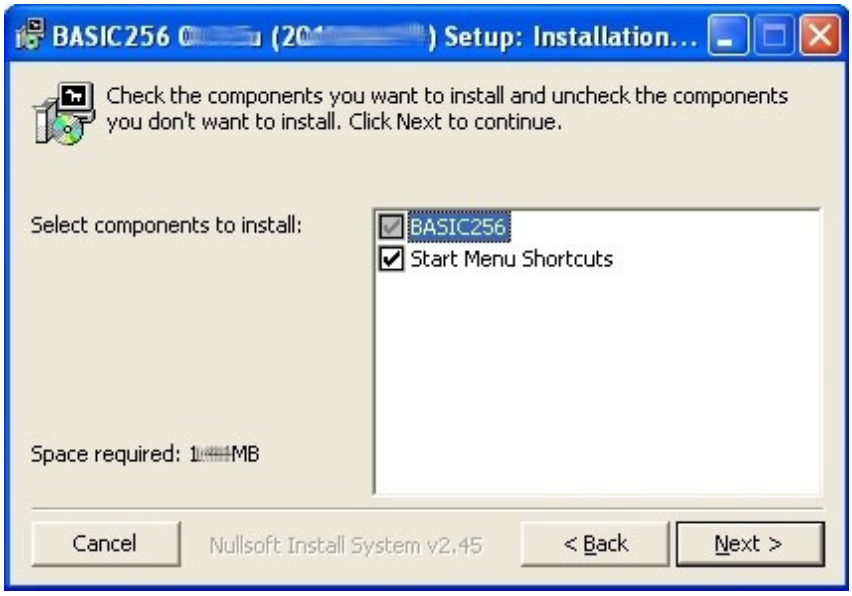


Рис. А.8. Установщик — что устанавливать.

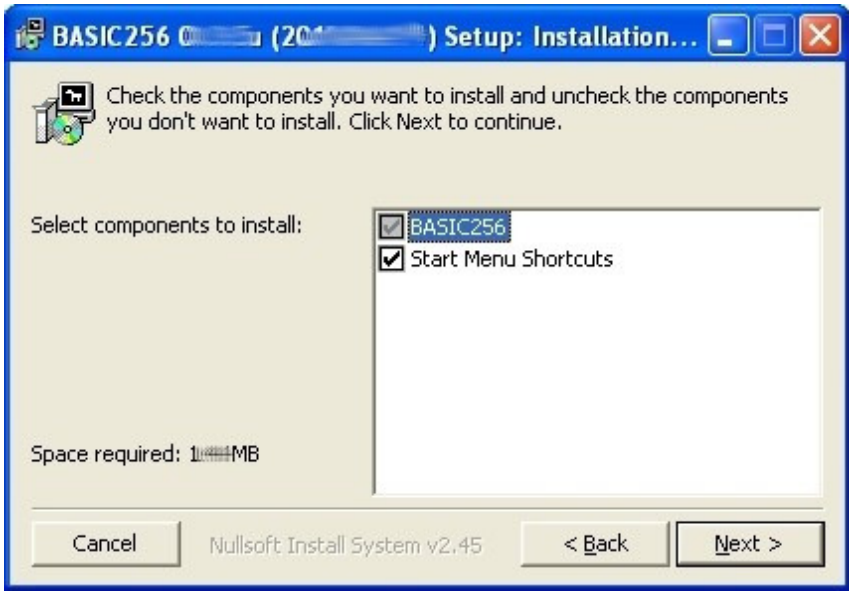


Рис. А.9. Установщик — куда устанавливать.

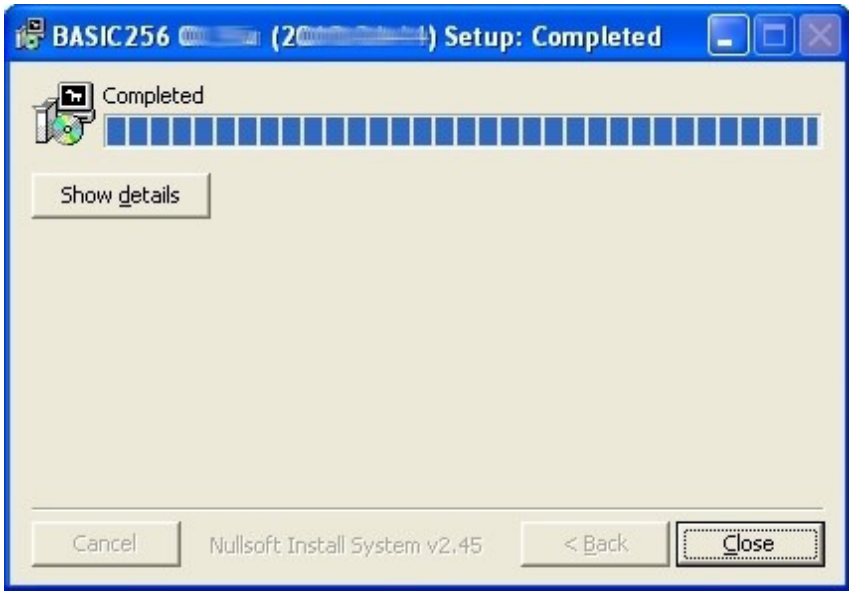


Рис. А.10. Установка завершена.

Рис. А.11. Меню Windows.

Рис. А.12. Меню BASIC-256.

Приложение B

Справочник по языку. Операторы

В скобках указан номер главы, где впервые появился данный оператор.

circle — Рисуем окружность в окне графического вывода (2)

circle *x,y, radius*

Оператор **circle** рисует текущим цветом круг в окне графического вывода. Центр круга, точка (x,y), определяется параметрами x и y, а радиус определяется параметром *radius*.

Пример

```
clg
color 255,128,128
circle 150,150,150
color red
circle 150,150,100
```

changedir — Смена текущего рабочего каталога (16)

changedir *путь*

Оператор **changedir** позволяет сменить текущий рабочий каталог вашего приложения. Когда вы используете файл без указания полного пути к нему (в операторах **imgload**, **open**, **spriteload** или подобных, использующих в качестве параметра имя файла), — приложение использует текущий рабочий каталог. Проверить какой у вас в данный момент текущий рабочий каталог можно с помощью функции **currentdir**.
Для всех систем, (включая Windows™) только прямой слеш (/) должен использоваться как разделитель каталогов внутри полного пути.
Смотри также: Close, Currentdir, Eof, Exists, Kill, Open, Read, Readline, Reset, Seek, Size, Write, Writeline

clg — Очистка окна вывода графики (2)

clg

Очищает окно графического вывода. Окно не очищается автоматически во время работы программы, что приводит к появлению нежелательных наложений

графических элементов. Если вы планируете использовать окно графического вывода, то лучше предварительно очистить его.

clickclear — Очистить последнее нажатие клавиши мыши (10)

clickclear

Устанавливает значения **clickb**, **clickx**, и **clicky** равным нулю (0), чтобы было удобно отследить следующее нажатие кнопки мыши.
Смотри также: Clickb, Clickx, Clicky, Mouseb, Mousex, Mousey
Пример. Смотри пример к Clickb.

close — Закрыть текущий открытый файл (16)

close
close()
close *номер_файла*
close (*номер_файла*)

Закрывает текущий открытый файл. Если открытых файлов нет — ничего не делает. Если *номер_файла* не указан, используется значение нуль (0).
Смотри также: Changedir, Currentdir, Eof, Exists, Kill, Open, Read, Readline, Reset, Seek, Size, Write, Writeline

cls — Очищение текстового окна

cls

Этот оператор автоматически очищает окно текстового вывода во время работы программы.

color или colour — Установка цвета для рисования

color *имя_цвета*
color *rgb-значение*
color *красный, зелёный, синий*

Оператор **color** устанавливает цвет для всех графических комманд. Цвет может быть определён по имени (см. Приложение Е), или как целое число представляющее RGB значение, или как три раздельные компоненты цвета.
Специальный цвет CLEAR или в числовом варианте -1 указывает командам рисования стереть пикселы и сделать их прозрачными.
Смотри также: Rgb, GetColor

Пример

```
clg
color black
rect 100,100,100,100
color 255,128,128
circle 150,150,75
```


dbclose (19) — Закрывает базу данных

dbclose

Закрывает текущую открытую SQLite базу данных. Больше информации о базах данных и, в частности, об SQLite можно найти на домашней странице SQLite http://sqlite.org и странице SQL на Wikipedia http://ru.wikipedia.org/wiki/SQL.
Смотри также: DBCloseSet, DBExecute, DBFloat, DBInt, DBOpen, DBOpenSet, DBRow, DBString
Пример. Смотри пример к DBOpen.

dbcloseset (19) — закрыть выборку

dbcloseset

Закрывает текущую открытую SQL выборку, созданную оператором **DBOpenSet**.
Смотри также: DBClose, DBExecute, DBFloat, DBInt, DBOpen, DBOpenSet, DBRow, DBString.
Пример. Смотри пример к DBOpen.

dbexecute (19) — Выполнить SQL запрос

dbexecute *sql_ запрос*
dbexecute (*sql_ запрос*)

Выполняет указанный sql-запрос на открытой SQLite базе данных. Этот оператор не порождает набора записей, соответствующих запросу, но вернёт сообщение об ошибке, если выполнение запроса невозможно.
Смотри также: DBClose, DBCloseSet, DBFloat, DBInt, DBOpen, DBOpenSet, DBRow, DBString.
Пример. Смотри пример к DBOpen.

dbopen (19)

dbopen *имя_ файла*
dbopen *(имя_ файла)*

Открывает указанный параметром *имя_ файла* файл SQLite базы данных. Если файл не существует, — создаётся новый. Больше информации о базах данных и, в частности, об SQLite можно найти на домашней странице SQLite http://sqlite.org и странице SQL на Wikipedia http://ru.wikipedia.org/wiki/SQL.
Смотри также: DBClose, DBCloseSet, DBExecute, DBFloat, DBInt, DBOpenSet, DBRow, DBString

Пример

```
# создаем базу и в ней таблицу foo,
# заполняем её данными, затем делаем запрос
# к базе и читаем данные из таблицы
# создаем новую базу данных или открываем существующую
dbopen "dbtest.sqlite3"
# удаляем старую таблицу foo,
```

```
# перехватываем ошибку, если база новая
onerror errortrap
dbexecute "drop table foo;"
offerror
# создаём и заполняем таблицу
dbexecute "create table foo (id integer, words text,
           value decimal);"
dbexecute "insert into foo values (1,'one',3.14);"
dbexecute "insert into foo values (2,'two',6.28);"
dbexecute "insert into foo values (3,'three',9.43);"
# формируем запрос к базе и выдаём записи в цикле
dbopenset "select * from foo order by words;"
while dbrow()
    print dbint(0) + dbstring(1) + dbfloat(2)
end while
dbcloseset
# закрываем базу
dbclose
end
errortrap:
# обрабатываем ошибку. ничего не отображаем,
# просто переходим к следующей строке программы
return
```

Будет напечатано

```
1one3.14
3three9.43
2two6.28
```

dbopenset (19)

dbopenset *sql_запрос*
dbopenset (*sql_запрос*)

Выполняет указанный *sql_запрос* и порождает набор записей, соответствующих запросу, так что из программы можно в цикле получить результаты запроса.
Смотри также: DBClose, DBCloseSet, DBExecute, DBFloat, DBInt, DBOpen, DBRow, DBString.
Пример. Смотри пример к DBOpen.

decimal — Количество десятичных знаков

decimal *выражение*
decimal(*выражение*)

Определяет максимальное количество десятичных знаков (0-16) после запятой, при преобразовании числа с плавающей точкой в строку. Это не влияет на точность числовых расчётов, а только изменяет отображение числа в виде строки. По умолчанию отображается максимум 6 цифр после запятой.
Смотри также: Print, String

Пример

```
print 2/3
decimal 10
```



```
print 2/3
decimal 15
print 2/3
```

Будет напечатано
```
0.666667
0.6666666667
0.666666666666667
```

dim — Размер нового массива (13)

dim *числовая_переменная(целое)*
dim *строковая_переменная$(целое)*
dim *числовая_переменная(ряд, колонка)*
dim *строковая_переменная$(ряд, колонка)*

Создаёт одномерный массив заданной длины или двумерный массив заданного размера, адресуемый через номер ряда и колонки. В зависимости от типа переменной, создаётся числовой или строковый массив. Первый элемент массива имеет номер 0 (нуль). Индекс лежит в диапазоне от 0 до length-1.
Оператор **dim** устанавлвает начальные значения элементам массива равными нулю (0), если массив числовой или равными пустой строке (""), если массив строковый.
Смотри также: Redim

Пример

```
dim z(5)
z = {1, 2, 3, 4, 5}
print z[0] + " " + z[4]
```

Будет напечатано
```
1 5
```

Ещё пример
```
dim c$(4)
c$ = {"корова", "коричневая", "как", "поживает"}
print c$[2] + " " + c$[3] + " ";
print c$[1] + " " + c$[0] + "?"
```

Будет напечатано
```
как поживает коричневая корова?
```

do / until — Цикл do / until (7)

do
оператор(ы)
until *логическое_выражение*

Выполняет операторы внутри цикла пока *логическое_выражение* ложно. **Do / Until** выполняется один или несколько раз. Проверка условия выполняется после каждого прохода по (всем) операторам внутри цикла.
Смотри также: For / Next, While / End While

Пример

```
t = 1
do
    print t
    t = t + 1
until t> 5
```

Будет напечатано
```
1
2
3
4
5
```

end — Завершение выполнения программы(9)

end

Завершает выполнение программы.

Пример

```
print "Выполнение окончено."
end
print "или нет?"
```

Будет напечатано
```
Выполнение окончено.
```

fastgraphics (8) — Включение режима быстрой графики

fastgraphics

Включает режим быстрой графики (fastgraphics mode), который действует до завершения программы. Режим быстрой графики означает, что экран графического вывода не обновляется, пока не будет вызвана команда **refresh**. Это существенно ускоряет вывод сложной анимации и предотвращает мерцание. Рекомендуем сделать все подготовительные к анимации команды в подпрограмме и использовать одну команду **refresh** для каждого фрейма. Войдя однажды в режим быстрой графики, из программы нельзя вернуться к обычной медленной графике.
Смотри также: Refresh

font — Установка параметров шрифта

font *имя_шрифта, размер, вес*

Устанавливает шрифт, используемый командой **text** в значение *имя_шрифта*. На каждом компьютере может быть несколько различных шрифтов, но «Гельветика» («Helvetica»), «Таймс» («Times»), «Системный» («System») и «Символьный» («Symbol») шрифты доступны на большинстве компьютеров. Размер определяется параметром *размер* в пойнтах (1 дюйм = 72 пойнта), *вес* является числом от 1 до 100 и определяет жирность начертания. Light=25 (тонкое

начертание), Normal=50 (стандартное начертание), и Bold=75 (жирное, привычнее полужирное, начертание).
Смотри также: Text

Пример

```
clg
color black
n = 5
dim fonts$(n)
fonts$ = {"Helvetica", "Times", "Courier", "System",
          "Symbol"}
for t = 0 to n-1
    font fonts$[t], 32, 50
    text 10, t*50, fonts$[t]
next t
```

Будет изображено (см. Рис. В.1)

Рис. В.1. Функция **font**

for / next — Цикл со счётчиком

for *переменная* = *выражение1* **to** *выражение2* [**step** *выражение3*]
оператор(ы)
next *переменная*

Команды **for** и **next** используются только совместно для выполнения команды или группы команд фиксированное число раз. Когда команда **for** выполняется первый раз, параметру *переменная* устанавливается значение равное *выражение1*. После каждой команды **next**, *переменная* увеличивается на 1 (по умолчанию) или на величину шага, равному значению *выражение3*, пока *переменная* не станет больше или равной значению *выражение2* для положительных значений шага или меньше или равно чем *выражение2* для отрицательных значений шага.
Смотри также: Do / Until, While / End While

goto — Переход к метке

goto *метка*

Выполнение программы переходит на точку, определённую параметром *метка*
Смотри также:Gosub / Return

Пример

```
print "Я";
goto skipit
print " не";
skipit: #
print " хочу печенье."
```

Будет напечатано
```
Я хочу печенье.
```

gosub / return — Уход в подпрограмму и возврат (9)

gosub *метка*
. . .
метка:
оператор(ы)
return

Выполнение программы переходит на точку, определённую параметром *метка*. Пока не встретится команда **return**, программа выполняет *оператор(ы)*. Команда **return** возвращает управление в точку вызова и выполнение продолжается со строчки следующей за **gosub**.
Команды **gosub** могут быть вложены друг в друга.
Смотри также: Goto

graphsize — Установить размер графического экрана (8)

graphsize *ширина, высота*

Изменяет размер окна вывода графики в соответствии с параметрами *ширина* и *высота*, и перерисовывает графический вывод приложения BASIC-256.
Смотри также: Graphheight, Graphwidth

if / then — Проверка условия (6)

if *логическое_выражение* **then** *оператор*
или
if *логическое_выражение* **then**
    оператор(ы)
end if
или
if *логическое_выражение* **then**
    оператор(ы)
else
    оператор(ы)
end if

Оператор **if**, записанный в одну строку, вычисляет *логическое_выражение* и, когда оно истинно (true), оператор(ы) следующие за **then** выполняются, в противном случае выполнение программы продолжается со следующей строки. Существуют также две формы многострочного **if** оператора, один с блоком, выполняемым, когда *логическое_выражение* истинно и другой с двумя блоками для случая когда *логическое_выражение* истинно и когда оно ложно.

Пример

```
print "Какую букву я загадал? - нажми клавишу"
# ждём, пока пользователь нажмёт клавишу
do
    a = key
    pause .01
until a<>0
if chr(a) = "Z" then
    print "Ура!!! Ты нажал клавишу Z!"
else
    print "Засада! Ты нажал что-то не то."
end if
end
```

imgload — Загрузка картинки из файла (12)

imgload *x, y, имя_файла*
imgload *x, y, масштаб, имя_файла*
imgload *x, y, масштаб, угол_поворота, имя_файла*

Загружает изображение из файла и отображает его в окне графического вывода. Параметры *x* и *y* определяют положение центра загруженной фигуры. Такое поведение отличает эту функцию от других графических утилит. Угол поворота также отсчитывается от этой центральной точки.
Функция **imgload** позволяет загружать большинство известных форматов графических файлов включая: BMP (Windows Bitmap), GIF (Graphic Interchange Format), JPG/JPEG (Joint Photographic Experts Group), и PNG (Portable Network Graphics).
Дополнительно, функция может изменить размер изображения в соответствии с заданным масштабом (1=полный размер). Изображение также может быть повёрнуто на указанный в радианах (от 0 до 2π) угол поворота по часовой стрелке вокруг своего центра.
Смотри также: ImgSave

imgsave — Сохранить образ на диск

imgsave *имя_файла*
imgsave *имя_файла, тип_файла*
imgsave(*имя_файла*)
imgsave (*имя_файла, тип_файла*)

Сохраняет текущее состояние окна вывода графики в графический файл. По умолчанию сохранение происходит в формате Portable Networks Graphics (PNG). Вы можете дополнительно указать *тип_файла* как «BMP», «JPG», «JPEG» или «PNG». Иногда в сохранённом изображении «неиспользованные» области могут

быть заполнены чёрным цветом. Это вызвано использованием оператора **clg** с указанием цвета -1 (прозрачный). Когда изображение сохраняется, «прозрачность» меняется на чёрный цвет.
Смотри также: ImgLoad

Пример

```
color white
rect 0, 0, graphwidth, graphheight
for t = 0 to 100
    color rand()*256, rand()* 256, rand()*256
    rect rand()*graphwidth, rand()*graphheight,
            rand()*graphwidth, rand()*graphheight
next t
imgsave "testimgsave1.png"
imgsave "testimgsave2.jpg", "jpg"
```

input — Получение строки от пользователя (7)

input *выражение, строковая_переменная$*
input *выражение, числовая_переменная*
input *строковая_переменная$*
input *числовая_переменная*

Функция ждёт от пользователя ввода строки в окне текстового вывода. Когда пользователь нажимает клавишу «Ввод», строка считывается в строковую или числовую переменную. Можно дополнительно проинформировать пользователя, что требуется ввести, используя *выражение*. Если ожидается число, а пользователь ввёл строку, не являющуюся числом, то значением числовой переменной будет нуль (0). Можно также использовать ссылки на элементы массива.

kill — Стереть файл

kill *имя_файла*
kill(*имя_файла*)

Удаляет файл, имя которого задано параметром *имя_файла*.
Смотри также: Changedir, Close, Currentdir, Eof, Open, Read, Readline, Reset, Write, Writeline, Exists, Seek, Size

line — Рисует прямую (2)

line *x0, y0, x1, y1*

Рисует линию от точки (x0, y0) до точки (x1, y1) толщиной в один пиксель, используя текущий цвет.

Пример

```
color white
rect 0,0,300,300
color black
line 50,50,200,200
line 100,200,200,200
line 100,200,50,50
```


netclose — Закрывает сетевое соединение (20)

netclose
netclose( )
netclose *номер_сокета*
netclose(*номер_сокета*)

Закрывает текущее сетевое соединение (сокет). Если *номер_сокета* не указан, используется нулевой (0) номер.
Смотри также: NetAddress, NetConnect, NetData, NetListen, NetRead, NetWrite
Пример. Смотри пример к NetConnect.

netconnect — Создание сетевого соединения (20)

netconnect *имя_сервера, номер_порта*
netconnect(*имя_сервера, номер_порта*)
netconnect *номер_сокета, имя_сервера, номер_порта*
netconnect(emphномер_сокета, имя_сервера, номер_порта)

Открывает клиентское сетевое соединение с сервером. IP адрес или имя хоста указывается в параметре *имя_сервера*, а порт в параметре *номер_порта*.
Если *номер_сокета* не указан, используется нулевой (0) номер.
Смотри также: NetAddress, NetClose, NetData, NetListen, NetRead, NetWrite

Пример

Откройте два экземпляра BASIC-256 на одном компьютере. Скопируйте «код сервера» в один экземпляр и «код клиента» в другой. Запустите сначала сервер, затем клиент. Вы сможете увидеть, как два различных процесса обмениваются сообщениями.
Код для сервера

```
# Получаем сообщение и посылаем ответ
# об успешном соединении
print "wait for connection on " + netaddress()
netlisten 9997
print "got connection"
do
    while not netdata
        pause .1
        print ".";
    end while
    n$ = netread
    print n$
    netwrite "I got '" + n$ + "'."
until n$ = "end"
netclose
```

Будет напечатано (где xxx.xxx.xxx.xxx IPv4 адрес вашего компьютера)

```
wait for connection on xxx.xxx.xxx.xxx
got connection
.1 Hi There
....2 Hi There
........3 Hi There
```

```
.........4 Hi There
.....5 Hi There
.......6 Hi There
....7 Hi There
.........8 Hi There
....9 Hi There
.....10 Hi There
.end
```

Код для клиента

```
# Ожидаем ввода сообщения от пользователя
# и посылаем его на сервер
input "enter message?", m$
netconnect "127.0.0.1", 9997
for t = 1 to 10
    pause rand
    netwrite t + " " + m$
    print netread
next t
netwrite "end"
print netread
netclose
```

Будет напечатано

```
enter message?Hi There
I got '1 Hi There'.
I got '2 Hi There'.
I got '3 Hi There'.
I got '4 Hi There'.
I got '5 Hi There'.
I got '6 Hi There'.
I got '7 Hi There'.
I got '8 Hi There'.
I got '9 Hi There'.
I got '10 Hi There'.
I got 'end'.
```

netlisten — Ожидание соединения с портом (20)

netlisten *номер_порта*
netlisten (*номер_порта*)
netlisten *номер_сокета, номер_порта*
netlisten (*номер_сокета, номер_порта*)

Открывает сетевое соединение (сервер) по указанному номеру порта и ждёт подключения. Если *номер_сокета* не указан, используется нулевой (0) номер.
Смотри также: NetAddress, NetClose, NetConnect, NetData, NetRead, NetWrite
Пример. Смотри пример к NetConnect.

netwrite — Передача данных по сети (20)

netwrite *строка*
netwrite (*строка*)
netwrite *номер_сокета, строка*
netwrite (*номер_сокета, строка*)

Посылает строку, заданную параметром *строка* в открытое сетевое соединение. Если *номер_сокета* не указан, используется нулевой (0) номер.
Смотри также: NetAddress, NetClose, NetConnect, NetData, NetListen, NetRead
Пример. Смотри пример к NetConnect.

offerror — Отключение режима перехвата ошибок (18)

offerror

Отключает режим перехвата ошибок и восстанавливает обычное поведение при возникновении ошибки.
Смотри также: Приложение J «Коды ошибок», Lasterror, Lasterrorextra, Lasterrorline, Lasterrormessage, Onerror
Пример. Пример смотри в Приложении J «Коды ошибок».

onerror — Обработка ошибки(18)

onerror *метка*

Вызывает исполнение подпрограммы, обозначенной меткой *метка*, когда происходит ошибка времени исполнения. Управление передаётся на следующую строчку программы после оператора Return в подпрограмме.
Смотри также: Приложение J «Коды ошибок», Lasterror, Lasterrorextra, Lasterrorline, Lasterrormessage, Offerror
Пример. Пример смотри в Приложении J «Коды ошибок».

open — Открыть файл на чтение и запись (16)

open *имя_файла*
open (*имя_файла*)
open *номер_файла, имя_файла*
open (*номер_файла, имя_файла*)

Открывает файл на чтение и запись. Параметр *имя_файла* представляет собой абсолютный или относительный путь. Если номер файла не указан, используется номер нуль (0). В BASIC-256 может быть открыто не более 8 файлов одновременно. Номера файлов лежат в диапазоне от 0 до 7. Если открыть новый файл с номером уже открытого файла, ранее открытый файл автоматически закрывается.
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Read, Readline, Reset, Seek, Size, Write, Writeline

pause — Приостановка выполнения программы (7)

pause *секунд*
pause (*секунд*)

Приостанавливает выполнение программы на указанное количество *секунд*. Число секунд может быть десятичной дробью, поэтому длина паузы может быть в долях секунды.

plot — Рисует точку в окне графического вывода (7)

plot *x, y*

Меняет цвет пикселя с координатами (*x,y*) в окне графического вывода на текущий.

poly — Рисует многоугольник (8)

poly *числовой_массив*
poly *{ x1, y1, x2, y2, x3, y3 ... }*

Рисует многоугольник. Стороны многоугольника определяются координатами вершин, хранящихся в массиве *числовой_массив* последовательно в виде пар x,y. Количество вершин равно половине длины массива. Многоугольник может также быть определён прямым указанием списка координат вершин, заключённых в фигурные скобки {}. Замечание: Число вершин больше не используется как параметр с версии 0.9.4.
Смотри также: Stamp

Пример

```
color blue
rect 0,0,300,300
color green
dim tri(6)
tri = {100, 100, 200, 200, 100, 200}
poly tri
```

Ещё пример:

```
color blue
rect 0,0,300,300
color green
poly {100, 100, 200, 200, 100, 200}
```

Обе программы рисуют зелёный треугольник на синем фоне.

portout — Запись данных в системный порт

portout (*номер_порта_вв/выв, значение*)

Записывает *значение* в диапазоне (0-255) в системный порт ввода/вывода (*номер_порта_вв/выв*). Чтение и запись в системные порты ввода/вывода является опасной операцией и может привести к непредсказуемым последствиям.
Эта функция может быть отключена по соображениям безопасности. Включить или выключить эту функцию можно в меню Правка → Настройки в панели

меню программы. Эта функция реализована только в сборке для Windows™. В Vista и Windows 7 необходимо запустить BASIC-256 один раз с правами администратора и установить драйвер inpout32. После установки драйвера, функции **portin/portout** будут работать без необходимости повышения прав пользователя.
Смотри также: PortIn

Пример

```
for y = 0 to 255
  portout 0x378,y
  print y + " " + portin(0x379)
  pause .1
next
```

Будет выведены числа от 0 до 255 в стандартный порт принтера и напечатаны соответствующее значения статусного регистра.

print — Отображает строку в текстовом окне (1)

print *выражение*
print *выражение*;

Печатает текст в окне текстового вывода, добавляя перевод строки. Если в конце указан знак точка с запятой, символ новой строки не добавляется.
Смотри также: Decimal

Пример

```
print "2 x 2 = ";
print 2*2
print "2 + 2 = "
print 2+2
```

Будет напечатано

```
2 x 2 = 4
2 + 2 =
4
```

putslice — Отображение области графического окна

putslice *x, y, имя_объекта*
putslice *x, y, имя_объекта, прозрачный_цвет*

Помещает графический объект с именем *имя_объекта* на экран в точку *(x,y)*. Если указан параметр *прозрачный_цвет*, то точки с таким цветом не рисуются на экране.
Смотри также: GetSlice

rect — Рисует прямоугольник в графическом окне (2)

rect *x, y, ширина, высота*
rect (*x, y, ширина, высота*)

Рисует прямоугольник размером *ширина* x *высота*, заполненный текущим цветом. Левый верхний угол прямоугольника имеет координаты *(x,y)*.

Пример

```
color white
rect 0,0,300,300
color red
rect 50,50,150,150
color blue
rect 100,100,150,150
color green
rect 10,140,280,20
```

redim — Изменяет размер массива (12)

redim *числовая_переменная(целое)*
redim *строковая_переменная$ (целое)*
redim *числовая_переменная(ряды, колонки)*
redim *строковая_переменная$ (ряды, колонки)*

Изменяет размер ранее созданного массива с сохранением данных. Если массив увеличивается, новые элементы получают значение нуль (0) или пустая строка (""). Если массив уменьшается, данные во всех избыточных элементах теряются.
Смотри также: Dim

refresh — Обновление окна графического вывода (8)

refresh

Обновляет окно графического вывода, показывая все графические объекты, изменившиеся с последнего запуска **refresh**. Эта команда работает только в режиме **fastgraphics**.
Смотри также: Fastgraphics

rem — Комментарий (2)

rem *комментарий*
 # комментарий

Строка комментария. Строка, начинающаяся с **rem** (или в сокращённом варианте — #), игнорируется интерпретатором.
Комментарий используется программистами для описания что делает программа а также кто её написал и какие изменения внёс.

reset — Удаление данных из открытого файла (16)

reset
reset ()
reset (*номер_файла*)

Удаляет все данные из открытого файла и перемещает указатель чтения/записи в начало файла.

Если *номер_файла* не указан, используется номер нуль (0).
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Open, Read, Readline, Seek, Size, Write, Writeline

say — Произносит текст (1)

say *выражение*

Произносит *выражение*, используя системный движок преобразования текста в речь. В Linux нужны библиотеки FLite или eSpeak. В Windows™ используется текущий SAPI (Speech Application Programming Interface) голос.

seek — Перемещает указатель чтения/записи файла (16)

seek *место*
seek (*место*)
seek (*номер_файла, место*)

Перемещает указатель чтения/записи файла в указанное *место* — сдвиг в байтах от начала файла — внутри открытого файла.
Если *номер_файла* не указан, используется номер нуль (0).
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Open, Read, Readline, Reset, Size, Write, Writeline

setsetting — Сохранение переменной в постоянном хранилище

setsetting *имя_программы, имя_ключа, значение*
setsetting (*имя_программы, имя_ключа, значение*)

Сохраняет *значение* в системном регистре (или другом постоянном хрнилище). Параметры *имя_программы* и *имя_ключа* используются для классификации и для того, чтобы можно было воспользоваться данными, когда это необходимо, предотвращая случайное изменения этих значений другой программой.
Смотри также: GetSetting

Пример

```
setsetting "thisprogram", "testsetting",
           "значение для сохранения"
print getsetting("thisprogram", "testsetting")
```

Будет напечатано
```
значение для сохранения
```

spritedim — Резервирует память для спрайтов (12)

spritedim *число_спрайтов*

Создаёт заготовку заданного числа спрайтов в памяти. Каждый спрайт нужно загрузить с помощью **spliteload** или создать с помощью **spriteslice** до того, как он будет отображён на экране. Вы можете создать столько спрайтов, сколько вам нужно, но учтите, большое число спрайтов сильно замедляет выполнение программы.

Спрайты отображаются в окне графического вывода по порядку их номеров. Спрайт с большим номером может быть отображён поверх всех спрайтов с меньшими номерами.
Спрайты доступны из программы по номерам от 0 до *число_спрайтов*-1.
Смотри также: Spritecollide, Spriteh, Spritehide, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritex, Spritey

spritehide — Скрывает спрайт (12)

spritehide *номер_спрайта*

Скрывает спрайт, убирая его изображение с экрана. Сам спрайт, его образ и положение при этом сохраняются в памяти и он может быть снова показан с помощью команды **spriteshow**.
Смотри также: Spritecollide, Spritedim, Spriteh, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritex, Spritey

spriteload — Загружает образ из фала в спрайт (12)

spriteload *номер_спрайта, имя_файла*
spriteload (*номер_спрайта, имя_файла*)

Загружает образ из файла и сохраняет его в указанном спрайте. Спрайт будет активным и его можно перемещать, но будет невидим, пока к нему не будет применена команда **spriteshow**. Оператор **spriteload** умеет читать большинство основных форматов графических файлов включая: BMP (Windows Bitmap), GIF (Graphic Interchange Format), JPG/JPEG (Joint Photographic Experts Group), и PNG (Portable Network Graphics).
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritex, Spritey

spritemove — Перемещает спрайт (12)

spritemove *номер_спрайта, dx, dy*
spritemove (*номер_спрайта, dx, dy*)

Перемещает спрайт из текущей позиции на указанное число пикселей. Перемещение ограничено текущими размерами экрана.
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spriteload, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritex, Spritey

spriteplace — Помещает спрайт на экран (12)

spriteplace *номер_спрайта, x, y*

Помещает центр спрайта в указанную позицию экрана. Также, как и в операторе **imgload**, положение спрайта отсчитывается от его центра, а не от верхнего левого угла, как в большинстве графических функций.
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spriteload, Spritemove, Spriteshow, Spriteslice, Spritev, Spritew, Spritex, Spritey

spriteshow — Показывает спрайт

spriteshow *номер_спрайта*

Показывает на экране предварительно загруженный, созданный или скрытый спрайт.
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spriteload, Spritemove, Spriteplace, Spriteslice, Spritev, Spritew, Spritex, Spritey

spriteslice — Копирует спрайт (12)

spriteslice *номер_спрайта, x, y, ширина, высота*

Копирует прямоугольную область с левым верхним углом в точке x, y и заданными шириной и высотой и создаёт спрайт. Этот спрайт будет активным и перемещаемым, но не будет видимым, пока не будет выполнена команда **spriteshow**. Рекомендуется выполнить команду **clg** перед рисованием и разрезанием спрайта. Все не закрашенные точки будут прозрачными при отображении спрайта на экране. Прозрачные точки можно также задать с помощью указания цвета **CLEAR** для рисования.
Спрайт будет помещён в точку (0,0) и скрыт. Вы можете его переместить в нужную точку операторами **spritemove** и **spriteplace**, а затем показать его оператором **spriteshow**.
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spritemove, Spriteplace, Spriteshow, Spritev, Spritew, Spritex, Spritey

sound — Воспроизводит звук через динамики (3)

sound *частота, длительность*
sound (*частота, длительность*)
sound (*массив*)
sound *массив*
sound {*частота1, длительность1, частота2, длительность2, ...*}

Воспроизводит звук через динамики компьютера. Частота задаётся в герцах (Hz), длительность — в миллисекундах (1/1000 секунды). Оператору можно передать массив или список состоящий из пар *частота*, *длительность*. Передача массива или списка помогает избежать щелчков между двумя соседними звуками, воспроизводимыми последовательно.
Смотри также: Volume

stamp — Помещает многоугольник на экран (8)

stamp *x, y, массив*
stamp *x, y, {x1, y1, x2, y2, x3, y3 ...}*
stamp *x, y, масштаб, массив*
stamp *x, y, масштаб, {x1, y1, x2, y2, x3, y3 ...}*
stamp *x, y, масштаб, угол, массив*
stamp *x, y, масштаб, угол, {x1, y1, x2, y2, x3, y3 ...}*

Рисует многоугольник с верхним левым углом (базой) в точке *(x, y)*. Можно изменить размер, если указать *масштаб* (1 = исходный размер). Также мож-

но поворачивать многоугольник на указанный в радианах *угол* вокруг базовой точки (по часовой стрелке). Стороны многоугольника определяются координатами вершин, хранящихся в массиве последовательно в виде пар *(x,y)*. Количество вершин равно половине длины массива. Многоугольник может также быть определён прямым указанием списка координат вершин, заключённых в фигурные скобки {}.
Смотри также: Poly

Пример

```
clg
color green
dim tri(6)
tri = {0, 0, 100, 100, 0, 100}
# Рисует треугольник в точке 0,0 (полноразмерный)
stamp 0, 0, tri
# Рисует треугольник в точке 200,100 (в половину размера)
stamp 200, 100, .5, tri
```

Ещё пример

```
clg
color green
# Рисует треугольник в точке 0,0 (полноразмерный)
stamp 0, 0, {0, 0, 100, 100, 0, 100}
# Рисует треугольник в точке 200,100 (в половину размера)
stamp 200, 100, .5, {0, 0, 100, 100, 0, 100}
```

Оба примера кода рисуют пару зелёных треугольников в окне графического вывода.

system

system *выражение*
system (*выражение*)

Выполняет системную команду в терминальном окне.
Предупреждение: Эта команда может быть опасна. Используйте её, если точно знаете, что вы делаете. Эта функция может быть отключена по соображениям безопасности. Включить или выключить эту функцию можно в меню Правка → Настройки в панели меню программы.

text — Текст в графическом окне

text *x, y, строка*

Рисует текстовую строку в окне графического вывода от точки *(x,y)* используя текущий цвет и шрифт.
Смотри также: Color, Font

Пример

```
color grey
rect 0,0,graphwidth,graphheight
color red
font "Times New Roman",18,50
```



```
text 10,100,"This is Times New Roman"
color darkgreen
font "Tahoma",28,100
text 10,200,"This is BOLD!"
```

volume — Уровень громкости звука

volume *уровень*
volume (*уровень*)

Устанавливает уровень громкости звуков, воспроизводимых командой **sound**. Значением параметра *уровень* должно быть число в диапазоне от 0 до 10. По умолчанию уровень равен 5.
Смотри также: Sound

wavplay — Воспроизведение WAV-файлов (12)

wavplay *имя_файла*
wavplay (*имя_файла*)

Воспроизводит звуковые файлы типа WAV асинхронно в фоновом режиме.
Смотри также: WAVstop, WAVwait

wavstop — Останавливает воспроизведение WAV-файла (12)

wavstop

Останавливает воспроизведение асинхронного (в фоновом режиме) звукового WAV файла.
Смотри также: WAVplay, WAVwait

wavwait — Ожидает окончания воспроизведения WAV-файла (12)

wavwait

Ждёт, пока закончится текущий воспроизводимый WAV файл.
Смотри также: WAVplay, WAVstop

while / end while — Цикл типа пока (7)

while *логическое_выражение*
    оператор(ы)
end while

Выполняет *оператор(ы)* внутри цикла пока *логическое_выражение* ложно. **while / end while** исполняет входящие внутрь операторы нуль или более раз. Проверка условия производится до исполнения кода внутри цикла.
Смотри также: Do / Until, For / Next

Пример

```
r = 1
while r < 6
    print r
    r = r + 1
end while
```

Будет напечатано

```
1
2
3
4
5
```

write — Запись строки в файл без символа перевода строки (16)

write *строка*
write (*строка*)
write *номер_файла, строка*
write (*номер_файла, строка*)

Записывает строку в конец открытого файла не добавляя при этом символа конца строки.
Если *номер_файла* не указан, используется номер нуль (0).
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Open, Read, Readline, Reset, Seek, Size, Writeline

writeline — Запись строки в файл с добавлением символа новой строки (16)

writeline *строка*
writeline (*строка*)
writeline *номер_файла, строка*
writeline (*номер_файла, строка*)

Записывает строку с добавлением символа новой строки в конец открытого файла.
Если *номер_файла* не указан, используется номер нуль (0).
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Open, Read, Readline, Reset, Seek, Size, Write

Приложение C

Справочник по языку. Функции

Функции производят вычисления используя переданные им значения переменных и возвращают результат в программу.

Каждая функция возвращает значения определенного типа (целые, булевы, дробные или строковые) и, возможно, в определенном диапазоне. Номер главы, где функция определена впервые указан в скобках.

abs — Абсолютное значение (14)

abs(*выражение*)

Возвращает абсолютное значение числового выражения.

Пример

```
print abs(-45)
print abs(6.45)
```

будет напечатано

```
45
6.45
```

acos — Арккосинус числа (14)

acos(*выражение*)

Вычисляет арккосинус выражения. Углы выражены в радианах (0 до 2π).
Смотри также: Asin, Atan, Cos, Degrees, Radians, Sin, Tan

asc — Unicode значение символа

asc (*выражение*)

Выдает Unicode значение (целое число от 0 до 65535) первого символа строки, заданного *выражением*
Смотри также: Chr

Пример

```
# Английская A
print asc("A")
# Русская Ы
print abs("Ы")
```

Будет напечатано
```
65
1067
```

asin — Арксинус (14)

asin (*выражение*)

Вычисляет арксинус заданного выражения — величину, обратную синусу. Значения арксинуса лежат в диапазоне от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$
Углы выражены в радианах (0 to 2π).
В российской математике принято обозначение arcsin.
Смотри также: Acos, Atan, Cos, Degrees, Radians, Sin, Tan

atan — Арктангенс (14)

atan (*выражение*)

Вычисляет арктангенс заданного *выражения*. Арктангенс — функция, обратная тангенсу, поэтому её значения лежат в диапазоне от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$
Углы выражены в радианах (0 to 2π).
В русской математической науке принято обозначение arctg.
Смотри также: Acos, Asin, Cos, Degrees, Radians, Sin, Tan

ceil — Округление (14)

ceil (*выражение*)

Возвращает наименьшее целое, большее или равное заданному *выражению*.
Смотри также: Floor

Пример

```
print ceil(9.1)
print ceil(-5.4)
```

Будет напечатано
```
10
-5
```

chr — Возвращает Unicode символ (11)

chr (*выражение*)

Преобразует число в символ, Unicode код которого равен (целому) значению заданного выражения.
Смотри также: Asc

Пример


```
print chr(1052) +chr(1099) + chr(1096) +chr(1100)
```

Будет напечатано
Мышь

clickb — Возвращает номер нажатой кнопки мыши (10)

clickb
clickb()

Возвращает номер кнопки мыши, которую пользователь нажал последний раз (в то время как курсор мыши был над областью графического вывода). Возвращает 0, если нажатие не было зафиксировано. Если было нажато несколько кнопок одновременно, то вернётся сумма значений нажатых кнопок.

Значение	Описание
0	Ни одна кнопка не нажата
1	Левая кнопка нажата
2	Правая кнопка нажата
4	Средняя кнопка нажата

Смотри также: Clickclear, Clickx, Clicky, Mouseb, Mousex, Mousey

Пример

```
# очищаем историю нажатий
clickclear
# ждём, пока пользователь нажмёт на кнопку мыши
print "Щёлкни мышкой в окне графического вывода"
while clickb = 0
    pause .01
endwhile
# Показываем, где пользователь щёлкнул
print "Вы нажали в точке (" + clickx + "," + clicky + ")"
```

clickx — Возвращает координату x мыши (10)

clickx
clickx()

Возвращает координату x указателя мыши над окном графического вывода в момент последнего нажатия на кнопку мыши.
Смотри также: Clickclear, Clickb, Clicky, Mouseb, Mousex, Mousey
Пример. Смотри пример к Clickb.

clicky — Возвращает координату y мыши (10)

clicky
clicky()

Возвращает координату y указателя мыши над окном графического вывода в момент последнего нажатия на кнопку мыши.
Смотри также: Clickclear, Clickb, Clickx, Mouseb, Mousex, Mousey
Пример. Смотри пример к Clickb.

cos — Косинус угла (14)

cos (*выражение*)

Вычисляет косинус *выражения*. Выражение (угол) должно быть в радианах.
Возвращаемые значения лежат в диапазоне от -1 до 1.
Замечание: Функция косинус не даёт точного результата.
Смотри также: Acos, Asin, Atan, Degrees, Radians, Sin, Tan

Пример

```
clg
color black
# рисует горизонтальную линию
# поперёк окна графического вывода
line 0,150,300,150
# начало графика
lastx = 0
lasty = cos(0) * 50 + 150
# шаг вдоль осевой линии и рисуем
for x = 0 to 300 step 5
    angle = x / 300 * 2 * pi
    y = cos(angle) * 50 + 150
    line lastx, lasty, x, y
    lastx = x
    lasty = y
next x
```

count — Подсчёт количества вхождений подстроки в строку*[1]

count (*строка, искомая_строка*)
count (*строка, искомое, регистр_не_важен*)

Возвращает количество вхождений строки *искомая_строка* в переменной или константе *строка*. Вы также можете указать дополнительное булево значение *регистр_не_важен* = true, если не хотите учитывать регистр букв.
Смотри также: Countx

Пример

```
print count("Привет", "ве")
print count("Буйвол, буйвол, БуйвОл.","БУЙВОЛ",true)
```

Будет напечатано

```
1
3
```

countx — Количество вхождений подстроки, заданной регулярным выражением*

countx (*строка, рег_выражение*)

[1]Здесь и далее звёздочкой помечены функции, добавленные в версии BASIC-256, собранные после написания автором книги (*прим. переводчика*).

Возвращает количество вхождений строк, заданных регулярным выражением *рег_выражение*, в строку *строка*.
Смотри также: Count

Пример

```
print countx("Привет", "[пП]")
print countx("Буйвол, буйвол, буйвол.","[Бб]уйвол")
```

Будет напечатано
```
1
3
```

currentdir — Возвращает текущий рабочий каталог (16)

currentdir
currentdir ()

Возвращает полный путь к рабочему каталогу программы BASIC-256. Для всех систем (включая Windows™) прямой слеш (/) используется для разделения имён каталогов в полном пути.
Смотри также: Changedir, Close, Eof, Exists, Kill, Open, Read, Readline, Reset, Seek, Size, Write, Writeline

day — Возвращает день месяца (9)

day
day()

Возвращает день месяца (1-31) из текущей системной даты.
Смотри также: Hour, Minute, Month, Second, Year

Пример

```
print "Сегодня ";
print (month + 1) + "/" + day + "/" + year
```

Будет напечатано (для 30 ноября 2010 года):
```
Сегодня 11/30/2010
```

dir — Получение списка файлов каталога*

dir()
dir (*каталог*)

Открывает *каталог* (как список) для извлечения имён файлов или каталогов, которые содержатся в нём и возвращает первое найденное имя файла или каталога. Без указания каталога функция выдаёт следующее имя в списке.
Смотри также: Currentdir

Пример

Для Windows систем:

```
f$ = dir("c:\")
while f$<>""
  print f$
  f$ = dir()
end while
```

Будет напечатано что-то вроде

```
$Recycle.Bin
autoexec.bat
Backup
Boot
bootmgr
Documents and Settings
IO.SYS
MSDOS.SYS
MSOCache
pagefile.sys
Program Files
ProgramData
System Volume Information
temp
Users
Windows
```

Для систем на базе Linux замените:

```
f$ = dir("c:\")
```

например, на корневой каталог:

```
f$ = dir("/")
```

dbfloat — Возвращает дробное значение из таблицы (19)

dbfloat (*номер_колонки*)

Возвращает число с плавающей точкой (десятичную дробь) из указанной колонки результата SQL-запроса к базе.
Больше информации о базах данных и, в частности, об SQLite можно найти на домашней странице SQLite http://sqlite.org и странице SQL на Wikipedia http://ru.wikipedia.org/wiki/SQL.
Смотри также: DBClose, DBCloseSet, DBExecute, DBInt, DBOpen, DBOpenSet, DBRow, DBString
Пример. Смотри пример к DBOpen.

dbint — Возвращает целое значение из таблицы (19)

dbint (*номер_колонки*)

Возвращает целое число из указанной колонки результата SQL запроса к базе.
Больше информации о базах данных и, в частности, об SQLite можно найти на домашней странице SQLite http://sqlite.org и странице SQL на Wikipedia http://ru.wikipedia.org/wiki/SQL.
Смотри также: DBClose, DBCloseSet, DBExecute, DBFloat, DBOpen, DBOpenSet, DBRow, DBString
Пример. Смотри пример к DBOpen.

dbrow — Переход к следующей строке выборки из базы (19)

dbrow
dbrow ()

Эта функция осуществляет переход к следующей строке в множестве строк SQL-запроса. Возвращает **true** (истину) если следующий ряд есть и **false** (ложь) если больше строк в выборке нет.
Больше информации о базах данных и, в частности, об SQLite можно найти на домашней странице SQLite http://sqlite.org и странице SQL на Wikipedia http://ru.wikipedia.org/wiki/SQL.
Смотри также: DBClose, DBCloseSet, DBExecute, DBFloat, DBInt, DBOpen, DBOpenSet, DBString
Пример. Смотри пример к DBOpen.

dbstring — Возвращает строковое значение из таблицы (19)

dbstring (*номер_колонки*)

Возвращает строку из указанной колонки результата SQL-запроса к базе.
Больше информации о базах данных и, в частности, об SQLite можно найти на домашней странице SQLite http://sqlite.org и странице SQL на Wikipedia http://ru.wikipedia.org/wiki/SQL.
Смотри также: DBClose, DBCloseSet, DBExecute, DBFloat, DBInt, DBOpen, DBOpenSet, DBRow
Пример. Смотри пример к DBOpen.

degrees – Преобразует радианы в градусы (14)

degrees (*выражение*)

Преобразует угол из радиан в градусы по формуле: $градусы = \frac{радианы}{2\pi} \times 360$
Смотри также: Acos, Asin, Atan, Cos, Radians, Sin, Tan

eof — Обнаружение конца файла (16)

eof
eof()
eof (*номер_файла*)

Возвращает логическое значение (истина/ложь) в качестве признака конца чтения файла EOF (на англ. End Of File - конец файла).
Если *номер_файла* не указан, используется значение нуль (0).
Смотри также: Changedir, Close, Currentdir, Exists, Kill, Open, Read, Readline, Reset, Seek, Size, Write, Writeline

exists — Возвращает признак наличия файла (16)

exists (*выражение*)

Возвращает логическое значение (истина/ложь) в качестве признака существования файла, путь к которому определяет *выражение*.

Смотри также: Changedir, Close, Currentdir, Eof, Kill, Open, Read, Readline, Reset, Seek, Size, Write, Writeline

Пример

```
if not exists("myfile.dat") then goto fileerror
```

exp — Экспоненциальная функция*

exp (*выражение*)

Вычисляет экспоненциальную функцию — число *e* (2,718281828459045... — число Эйлера), возведённое в степень, заданную *выражением*.
Смотри также: Sqr, Log, Log10

Пример

```
print exp(1)
print exp(0)
```

Будет напечатано
```
2.718282
1
```

explode — Разделение строки на подстроки*

строковый_массив$ = **explode** (*строка, разделитель*)
строковый_массив$ = **explode** (*строка, разделитель, регистр_не_важен*)
числовой_массив = **explode** (*строка, разделитель*)
числовой_массив = **explode** (*строка, разделитель, регистр_не_важен*)

Разделяет *строку* на подстроки, используя *разделитель*. Подстроки сохраняются в строковом или числовом массиве, определённом в операторе присваивания. Размер массива будет изменён, согласно количеству полученных подстрок.
Если указать дополнительное булево значение *регистр_не_важен* = true, то регистр символов не будет учитываться.
Смотри также: Explodex, Implode

Пример

```
# разделение пробелами
a$ = "В лесу родилась ёлочка."
print a$
w$ = explode(a$," ")
for t = 0 to w$[?]-1
  print "w$["+t+"]=" + w$[t]
next t
# разделение по буквам A или a
a$ = "klj;lkjalkjAlkj;"
print a$
w$ = explode(a$,"A",true)
for t = 0 to w$[?]-1
  print "w$["+t+"]=" + w$[t]
next t
# разделение цифр по запятой
```



```
a$="1,2,3,77,ничего,9.987,6.45"
print a$
n = explode(a$,",")
for t = 0 to n[?]-1
  print "n["+t+"]=" + n[t]
next t
```

Будет напечатано

```
В лесу родилась ёлочка.
w$[0]=В
w$[1]=лесу
w$[2]=родилась
w$[3]=ёлочка.
klj;lkjalkjAlkj;
w$[0]=klj;lkj
w$[1]=lkj
w$[2]=lkj;
1,2,3,77,ничего,9.987,6.45
n[0]=1
n[1]=2
n[2]=3
n[3]=77
n[4]=0
n[5]=9.987
n[6]=6.45
```

explodex — Разделение строки на подстроки с использованием регулярных выражений*

строковый_массив$ = **explodex** (*строка, рег_выражение*)
числовой_массив = **explodex** (*строка, рег_выражение*)

Разделяет *строку* на подстроки, где разделитель определяет регулярное выражение *рег_выражение*. Подстроки сохраняются в строковом или числовом массиве, определённом в операторе присваивания. Размер массива будет изменён, согласно количеству полученных подстрок.
Смотри также: Explode, Implode

Пример

```
# разделение по выражению //[,]* //
a$ = "В лесу родилась ёлочка, в лесу она росла."
w$ = explodex(a$,"[,]* ")
for t = 0 to w$[?]-1
  print "w$["+t+"]=" + w$[t]
next t
# разделение по выражению //[Ии][Лл][Ии]//
a$="1 или 2 ИЛИ 3 Или 5 иЛИ 99 ИЛи 8.88 иЛи 6.45"
n = explodex(a$,"[Ии][Лл][Ии]")
for t = 0 to n[?]-1
  print "n["+t+"]=" + n[t]
next t
```

Будет напечатано

```
w$[0]=В
w$[1]=лесу
w$[2]=родилась
w$[3]=ёлочка
w$[4]=в
w$[5]=лесу
w$[6]=она
w$[7]=росла.
n[0]=1
n[1]=2
n[2]=3
n[3]=5
n[4]=99
n[5]=8.88
n[6]=6.45
```

float — Возвращает десятичную дробь (14)

float (*выражение*)

Преобразует *выражение* в десятичную дробь (число с плавающей точкой). **float** преобразует строку или целое число в десятичную дробь. Если выражение не может быть преобразовано, то возвращается нуль.
Смотри также: Int

Пример

```
a$ = "1.234"
b = float(a$)
print a$
print b
```

Будет напечатано

```
1.234
1.234
```

floor — Округление «вниз» (14)

floor (*выражение*)

Возвращает наибольшее целое, не превосходящее заданное *выражение*.
Смотри также: Ceil

Пример

```
print floor(5)
print floor(12.8)
print floor(-3.2)
```

Будет напечатано

```
5
12
-4
```


getcolor

getcolor
getcolor()

Возвращает RGB значение текущего установленного цвета (того, который был последний раз установлен командой **color**) в диапазоне от 0 до 16777215. Если цвет был установлен как **CLEAR**, возвращаемое значение равно -1.
RGB значение вычисляется как $((red \times 256) + green) \times 256 + blue$ где red, green, и blue целые числа в диапазоне от 0 до 255.
Смотри также: Color, Rgb

Пример

```
color red
print getcolor

Будет напечатано
16711680
```

getsetting — Получение данных из хранилища

getsetting (*имя_ программы, имя_ ключа*)

Получает установки из системного регистра (или другого постоянного хранилища). Параметры *имя_ программы* и *имя_ ключа* необходимы для доступа к ранее сохранённым данным. Если параметр не был ранее установлен, будет возвращена пустая строка (""). Сохранённое значение доступно из другой програмы на BASIC-256 и остаётся доступным на длительный период.
Эта функция может быть отключена по соображениям безопасности. Включить или выключить эту функцию можно в меню Правка → Настройки в панели меню программы.
Смотри также: SetSetting

Пример

```
setsetting "thisprogram", "testsetting",
           "сохраняемое значение"
print getsetting("thisprogram", "testsetting")

Будет напечатано
сохраняемое значение
```

getslice — Захват части графического окна

getslice(*x, y, ширина, высота*)

Возвращает строку содержащую шестнадцатеричное представление прямоугольника, определяемого параметрами. Строка имеет следующий формат: первые 4 байта *ширина*, затем 4 байта *высота* и потом по 6 байт на каждый пиксель (*ширина* × *высота*).
Смотри также: PutSlice

graphheight — Высота графического окна (8)

graphheight
graphheight()

Возвращает высоту (размер по y) текущего окна вывода графики.
Смотри также: Graphsize, Graphwidth

graphwidth — Ширина графического окна (8)

graphwidth
graphwidth()

Возвращает ширину (размер по x) текущего окна вывода графики.
Смотри также: Graphheight, Graphsize

hour — Возвращает час из системной даты (9)

hour
hour()

Возвращает количество часов (0-23) текущей системной даты.
Смотри также: Day, Hour, Minute, Month, Second, Year

Пример

```
# показать чудесную дату
dim months$(12)
months$ = {"Января", "Февраля", "Марта", "Апреля",
           "Мая", "Июня", "Июля", "Августа",
   "Сентября", "Октября", "Ноября", "Декабря"}
print right("0" + day, 2) + "-" + months$[month] +
           "-" + year
# display pretty time
h = hour
if h> 12 then
    h = h - 12
    ampm$ = "PM"
else
    ampm$ = "AM"
end if
if h = 0 then h = 12
print right("0" + h, 2) + ":" + right("0" + minute, 2) +
      ":" + right("0" + second, 2) + " " + ampm$
```

Будет напечатано (для 22 часов 15 июля 2010 г.)

```
15-Июля-2010
10:12:02 PM
```

implode — Объединение массива в строку*

implode (*массив*)
implode (*массив, разделитель*)

Объединяет элементы массива в строку. При необходимости можно указать *разделитель*, который будет помещён между элементами массива. Противоположна

функции **explode**.
Смотри также: Explode, Explodex

Пример

```
dim a$(1)
dim b(1)
a$ = Explode("Как поживает коричневая корова?"," ")
print implode(a$,"-")
print implode(a$)
b = Explode("1,2,3.33,4.44,5.55",",")
print implode(b,", ")
print implode(b)
```

Будет напечатано

```
Как-поживает-коричневая-корова?
Какпоживаеткоричневаякорова?
1, 2, 3.33, 4.44, 5.55
123.334.445.55
```

instr — Позиция подстроки (15)

instr (*стог_сена, иголка*)
instr (*стог_сена, иголка, старт_символ*)
instr (*стог_сена, иголка, старт_символ, регистр_не_важен*)

Проверяет, содержится ли *иголка* в *стог_сена*. Если да, то функция возвращает номер позиции первого вхождения подстроки *иголка*. Если нет, возвращается нуль (0). Вы можете дополнительно указать число *старт_символ*, с которого нужно начинать проверку строки, а также булево значение *регистр_не_важен* = true, если не хотите учитывать регистр символов. Нумерация символов в строке начинается с 1.
Смотри также: Instrx

Пример

```
print instr("Привет", "вет")
print instr("101,222,333",",",5)
```

Будет напечатано

```
4
8
```

instrx — Поиск подстроки через регулярное выражение

instrx (*строка, рег_выражение*)
instrx (*строка, рег_выражение, старт_символ*)

Проверяет, содержится ли в *строке* подстрока, определяемая регулярным выражением *рег_выражение*. Если да, то функция возвращает номер позиции первого вхождения подстроки. Если нет, возвращается нуль (0). Вы можете дополнительно указать число *старт_символ*, с которого нужно начинать проверку строки. Нумерация символов в строке начинается с 1.
Смотри также: Instr

Пример

```
print instrx("Привет", "[Ии]в")
print instrx("Проверка, проверка","[еЕ][рР]",6)
```

Будет напечатано
```
3
15
```

int — Получение целого значения (14)

int (*выражение*)

Преобразует строку или десятичную дробь содержащуюся в параметре *выражение* в целое число. При преобразовании рационального числа просто отбрасывается дробная часть.
При преобразовании строки возвращается целое, находящееся в начале строки. Если выражение не может быть преобразовано, возвращается нуль.
Смотри также: Float

Пример

```
print int(9)
print int(9.9999)
print int(-8.765)
print int(" 321 555 foo")
print int("У меня 42 бананы.")
```

Будет напечатано
```
9
9
-8
321
0
```

key — Получение кода нажатой клавиши (11)

key
key()

Немедленно возвращает целое число, соответствующее нажатой на клавиатуре клавише. Если ни одна клавиша не была нажата с времени последнего вызова функции **key**, возвращается нуль. Каждая клавиша на клавиатуре имеет уникальный код, который обычно совпадает с Unicode значением латинской буквы в верхнем регистре, изображённой на этой клавише.

Пример

Код
```
if key = 47 then print key
```
не будет работать, поскольку функция key вызывается дважды и каждый раз будет давать разные значения.
Вот правильное использование функции key:
```
a = key
if a = 47 then print a
```


lasterror — Код последней ошибки (18)

lasterror
lasterror()

Возвращает номер последней ошибки.
Смотри также: Приложение J «Коды ошибок», Lasterrorextra, Lasterrorline, Lasterrormessage, Offerror, Onerror
Пример. Смотри пример в Приложении J «Коды ошибок».

lasterrorextra — Дополнительная информация об ошибке (18)

lasterrorextra
lasterrorextra()

Возвращает дополнительную информацию об ошибке.
Смотри также: Приложение J «Коды ошибок», Lasterror, Lasterrorline, Lasterrormessage, Offerror, Onerror

Пример. Смотри пример в Приложении J «Коды ошибок».

lasterrorline — Возвращает номер строки с ошибкой (18)

lasterrorline
lasterrorline()

Возвращает номер строки программы, где произошла ошибка.
Смотри также: Приложение J «Коды ошибок», Lasterror, Lasterrorextra, Lasterrormessage, Offerror, Onerror
Пример. Смотри пример в Приложении J «Коды ошибок».

lasterrormessage — Текстовое сообщение об ошибке (18)

lasterrormessage
lasterrormessage ()

Возвращает строку сообщения последней ошибки.
Смотри также: Приложение J «Коды ошибок», Lasterror, Lasterrorextra, Lasterrorline, Offerror, Onerror
Пример. Смотри пример в Приложении J «Коды ошибок».

left — Возвращает подстроку с началом слева (15)

left (*строка, длина_подстроки*)

Возвращает подстроку из строки, заданной параметром *строка* начиная с первого символа и длиной, заданной параметром *длина_подстроки*. Если длина подстроки превышает размер строки, то возвращена будет вся строка.
Смотри также: Mid, Right

Пример

```
print left("Привет", 3)
```

Будет напечатано
```
При
```

length — Длина строки (15)

length (*строка*)

Возвращает количество символов в строке, заданной параметром *строка* (длину строки).

Пример

```
print length("Компьютер")
```

Будет напечатано
```
9
```

log — Натуральный логарифм*

log (*выражение*)

Возвращает натуральный логарифм (логарифм по основанию *e*) от *выражения*. В российской математике эта функция обычно обозначается ln.
Смотри также: Log10

log10 — Десятичный логарифм*

log10 (*выражение*)

Возвращает десятичный логарифм (логарифм по основанию 10) от *выражения*. В российской математике принято обозначать эту функцию lg.
Смотри также: Log

lower — Заменяет все буквы на строчные (15)

lower (*строка*)

Возвращает строку, в которой все прописные буквы, из переменной (или константы) *строка* заменены на строчные.
Смотри также: Upper

Пример

```
print lower("ПРИвет!")
```

Будет напечатано
```
привет!
```

md5 — Возвращает md5 хэш строки

md5 (*строка*)

Возвращает MD5 хэш строки, заданной аргументом *строка*. Эта функция была получена из RSA Data Security, Inc MD5 Message-Digest Algorithm.
md5 обычно используется для создания контрольной суммы строк при проверке корректности передачи данных.

Пример


```
print MD5("Something")
print MD5("something")
```

Будет напечатано

```
73f9977556584a369800e775b48f3dbe
437b930db84b8079c2dd804a71936b5f
```

mid — Возвращает подстроку из середины строки (15)

mid (*строка, номер_начала, длина*)

Возвращает подстроку строки заданной параметром *строка* начиная с позиции *номер_начала* и длиной *длина*.
Смотри также: Left, Right

Пример

```
print mid("Привет", 2, 3)
```

Будет напечатано

```
рив
```

minute — Возвращает минуты из системной даты (9)

minute
minute()

Возвращает количество минут (0-59) текущей системной даты.
Смотри также: Day, Hour, Month, Second, Year
Пример. Смотри пример к Hour

month — Возвращает месяц из системной даты

month
month()

Возвращает (как число) значение месяца текущей системной даты. Январю соответствует 0, февралю — 1, ... декабрю — 11.
Смотри также: Day, Hour, Minute, Second, Year

Пример

```
cls
dim n$(12)
n$ = {"Янв", "Фев", "Мар", "Апр", "Май", "Июн",
      "Июл", "Авг", "Сен", "Окт", "Ноя", "Дек"}
print day + "-" + n$[month] + "-" + year
```

В Новый (2011) Год будет напечатано

```
1-Янв-2011
```

mouseb — Возвращает состояние кнопок мыши (10)

mouseb
mouseb()

Возвращает код нажатой кнопки мыши (курсор должен быть над окном для вывода графики). Возвращает 0, если нажатий не было. Если несколько кнопок нажато одновременно, возвращается сумма значений нажатых кнопок.

Значение	Описание
0	Ни одна кнопка не нажата
1	Левая кнопка нажата
2	Правая кнопка нажата
4	Средняя кнопка нажата

Смотри также: Clickb, Clickclear, Clickx, Clicky, Mousex, Mousey

mousex — Позиция курсора мыши по оси x (10)

mousex
mousex()

Возвращает текущее или последнее значение координаты x указателя мыши над окном для вывода графики.
Смотри также: Clickb, Clickclear, Clickx, Clicky, Mouseb, Mousey

mousey — Позиция курсора мыши по оси y (10)

mousey
mousey()

Возвращает текущее или последнее значение координаты y указателя мыши над окном для вывода графики.
Смотри также: Clickb, Clickclear, Clickx, Clicky, Mouseb, Mousex

netaddress — Какой у меня IP адрес? (20)

netaddress
netaddress()

Возвращает строку с IPv4 адресом этого компьютера в сети.
Смотри также: NetClose, NetConnect, NetData, NetListen, NetRead, NetWrite
Пример. Смотри пример к NetConnect.

netdata — Есть ли стетевые данные для чтения? (20)

netdata
netdata ()
netdata *номер_сокета*
netdata (*номер_сокета*)

Возвращает значение *true* (истина, числовое значение 1), если имеются данные, которые можно прочитать функцией **netread**, в противном случае возвращает *false* (ложь, значение 0). Если *номер_сокета* не указан, используется нулевой

(0) номер.
Смотри также: NetAddress, NetClose, NetConnect, NetListen, NetRead, NetWrite
Пример. Смотри пример к NetConnect.

netread — Чтение сетевых данных (20)

netread
netread ()
netread (*номер_сокета*)

Читает данные из сетевого соединения и возвращает их в виде строки. Эта функция ждёт, пока происходит получение данных.
Если *номер_сокета* не указан, используется нулевой (0) номер.
Смотри также: NetAddress, NetClose, NetConnect, NetData, NetListen, NetWrite
Пример. Смотри пример к NetConnect.

ostype — Определение типа операционной системы*

ostype()

Возвращает число, представляющее операционную систему, в которой запущен BASIC-256. Полезно при написании кросплатформенных программ.

Число	ОС
0	Windows
1	Linux
2	Macintosh

Пример

```
print "Вы используете операционную систему ";
os = ostype()
if os = 0 then  print "Windows."
if os = 1 then  print "Linux."
if os = 2 then  print "Macintosh."
```

Будет напечатано (для Linux системы)

```
Вы используете операционную систему Linux.
```

pixel — RGB-значение цвета точки

pixel (*x, y*)

Возвращает значение RGB пикселя с координатами x и y. Если точка не была определена с момента последнего исполнения команды **clg** или была нарисована цветом Clear, то возвращаемое значение равно -1.
Смотри также: Rgb

portin — Чтение данных из системного порта

portin (*номер_порта_вв/выв*)

Читает системный порт ввода/вывода (используя *номер_порта_ввода/вывода*), возвращая значение в диапазоне (0-255).
Чтение и запись в системные порты ввода/вывода является опасной операцией и может привести к непредсказуемым последствиям. Эта функция может быть отключена по соображениям безопасности. Включить или выключить эту функцию можно в меню Правка → Настройки в панели меню программы. Эта функция реализована только в сборке для Windows.
Смотри также: PortOut
Пример. Смотри пример к PortOut.

radians — Преобразует угол в радианы (8)

radians (*выражение*)

Преобразует угол из градусов в радианы по формуле $радианы = \frac{градусы}{360} \times 2\pi$
Смотри также: Acos, Asin, Atan, Cos, Degrees, Sin, Tan

rand — Случайное число (6)

rand
rand()

Возвращает случайное число между 0 и 1. Функция выдаёт равномерно распределённый набор случайных чисел. Чтобы получить случайные числа в заданном диапазоне используйте умножение на нужное число с добавлением подходящего сдвига. Например, чтобы получить случайное целое число от 0 до 10 используйте `int(rand * 10)`.

read — Читает токен из открытого файла (16)

read
read()
read (*номер_файла*)

Читает и возвращает "токен"из открытого файла. Токен — это строка символов, разделённая пробелами или табуляцией или символом новая строка.
Если *номер_файла* не указан, используется номер нуль (0).
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Open, Readline, Reset, Seek, Size, Write, Writeline

readline — Читает строку из открытого файла (16)

readline
readline ()
readline (*номер_файла*)

Читает и возвращает строку из открытого файла. Если достигнут конец файла (**eof** = **true**), функция вернёт пустую строку.
Если *номер_файла* не указан, используется номер нуль (0).
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Open, Read, Reset, Seek, Size, Write, Writeline

replace — Замена подстроки*

replace (*строка, что_меняем, на_что_меняем*)
replace (*строка, что_меняем, на_что_меняем, регистр_не_важен*)

Возвращает новую строку, где все подстроки *что_меняем* заменены на подстроки *на_что_меняем*. Вы можете дополнительно указать булево значение *регистр_не_важен* = true, если не хотите учитывать регистр символов.
Смотри также: Replacex

Пример

```
a$ = "В лесу родилась ёлочка, в лесу она росла."
print Replace(a$,"лесу","горах")
print Replace(a$, "ЕлОчКа", "белочка", true)
```

Будет напечатано

```
В горах родилась ёлочка, в горах она росла.
В лесу родилась белочка, в лесу она росла.
```

replacex — Замена подстроки с использованием регулярных выражений*

replacex (*строка, рег_выражение, на_что_меняем*)

Возвращает новую строку, где все вхождения в *строку*, согласно регулярному выражению *рег_выражение*, заменены на подстроку *на_что_меняем*.
Смотри также: Replace

Пример

```
a$ = "В лесу родилась ёлочка, в лесу она росла."
print Replacex(a$,"ле.у","горах")
print Replacex(a$,"[Её]лочка","белочка")
```

Будет напечатано

```
В горах родилась ёлочка, в горах она росла.
В лесу родилась белочка, в лесу она росла.
```

rgb — Преобразование цветного триплета в одно числовое значение (12)

rgb (*красный, зелёный, синий*)

Возвращает RGB значение цвета, состоящего из красной, зелёной и синей компонент. Значения *красный*, *зелёный*, *синий* — целые числа в диапазоне от 0 до 255, могут быть заданы переменными или константами. RGB значение для цвета формируется по формуле: $RGB = красный \times 256^2 + зелёный \times 256 + синий$.
Смотри также: Color, GetColor, Pixel

right — Возвращает подстроку с началом справа (15)

right (*строка, длина*)

Возвращает подстроку из строки, заданной параметром *строка*, указанной длины, начиная с правого края. Если параметр *длина* больше длины строки, будет возвращена вся строка.
Смотри также: Mid, Left

Пример

```
print right("Привет", 2)
```

Будет напечатано
```
ет
```

second — Возвращает секунды из системной даты (9)

second
second()

Возвращает количество секунд (0-59) текущей системной даты.
Смотри также: Day, Hour, Minute, Month, Year
Пример. Смотри пример к Hour.

sin — Синус (14)

sin (*выражение*)

Вычисляет синус выражения. Параметр *выражение* должен быть числом в радианах. Результат лежит в диапазоне от -1 до 1. Функция синус не даёт точного результата.
Смотри также: Acos, Asin, Atan, Cos, Degrees, Radians, Tan

Пример

```
clg
color black
# Рисует ось поперёк окна графического вывода
line 0,150,300,150
# Начало графика
lastx = 0
lasty = sin(0) * 50 + 150
# Идём вдоль оси заданным шагом и рисуем график
for x = 0 to 300 step 5
  angle = x / 300 * 2 * pi
  y = sin(angle) * 50 + 150
  line lastx, lasty, x, y
  lastx = x
  lasty = y
next x
```

size — Возвращает размер открытого файла (15)

size
size ()
size (*номер_ файла*)

Возвращает размер в байтах открытого файла.
Если *номер_файла* не указан, используется номер нуль (0).
Смотри также: Changedir, Close, Currentdir, Eof, Exists, Kill, Open, Read, Readline, Reset, Seek, Write, Writeline

spritecollide — Определяет факт столкновения спрайтов (12)

spritecollide (*спрайт1, спрайт2*)

Функция возвращает истину, если два спрайта сталкиваются. Функция **spritecollide** предполагает, что спрайты помещены в прямоугольники размером охватывающим загруженный образ. Факт столкновения выясняется с использованием этих прямоугольников. Для круглых или странной формы спрайтов функция может определять столкновения там, где их нет.
Смотри также: Spritedim, Spriteh, Spritehide, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritex, Spritey

spriteh — Высота спрайта (12)

spriteh (*номер_спрайта*)

Возвращает высоту загруженного спрайта в пикселях.
Смотри также: Spritecollide, Spritedim, Spritehide, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritex, Spritey

spritev — Виден ли спрайт? (12)

spritev (*номер_спрайта*)

Возвращает истину, если спрайт виден.
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritex, Spritey

spritew — Ширина спрайта (12)

spritew (*номер_спрайта*)

Возвращает ширину (в пикселях) загруженного спрайта.
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritex, Spritey

spritex — Координата x спрайта (12)

spritex (*номер_спрайта*)

Возвращает координату x центра загруженного спрайта.
Смотри также: Spritecollide, Spritedim, Spriteh, Spritehide, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritey

spritey — Координата у спрайта (12)

spritey (*номер_спрайта*)

Возвращает координату y центра загруженного спрайта.
Смотри также: Spritecollide, Spritedim, Spriteh, Spriteload, Spritemove, Spriteplace, Spriteshow, Spriteslice, Spritev, Spritew, Spritex

sqr — Квадратный корень*

sqr (*выражение*)
sqrt (*выражение*)

Вычисляет квадратный корень *выражения*. Аналогичный результат можно получить, возведя *выражение* в степень $\frac{1}{2}$. Функция введена для совместимости с другими версиями BASIC.
Смотри также: Exp, Log

Пример

```
print sqr(4)
print sqr(2)
print 2^(1/2)
```

Будет напечатано
```
2
1.414214
1.414214
```

string — Представление числа в виде строки (14)

string (*выражение*)

Возвращает в виде строки число, определяемое параметром *выражение*.

Пример

```
print 12+2
print string(12)+2
```

Будет напечатано
```
14
122
```

tan — Тангенс (14)

tan (*выражение*)

Вычисляет тангенс выражения. Параметр *выражение* должен быть задан в радианах.
Эта функция не даёт точного результата.
В российской математике принято обозначение tg.
Смотри также: Acos, Asin, Atan, Sin, Cos, Degrees, Radians

Пример


```
clg
color black
# draw a line across the graphic output
line 0,150,300,150
# where do we start
lastx = 0
lasty = tan(0) * 50 + 150
# now step across the line and draw
for x = 0 to 300 step 5
  angle = x / 300 * 2 * pi
  y = tan(angle) * 50 + 150
  line lastx, lasty, x, y
  lastx = x
  lasty = y
next x
```

upper — Заменяет все буквы на прописные (15)

upper (*строка*)

Возвращает строку, в которой все строчные буквы, в переменной (или константе), заданной параметром *строка*, заменены на прописные.
Смотри также: Lower

Пример

```
print upper("ПРИвет!")
```

Будет напечатано
`ПРИВЕТ!`

year — Возвращает год из системной даты (9)

year
year()

Возвращает 4 цифры года из текущей системной даты.
Смотри также: Day, Hour, Minute, Month, Second

Пример

```
print "Сегодня ";
print (month + 1) + "/" + day + "/" + year
```

Будет напечатано (для 30-го ноября 2010 года):
`Сегодня 11/30/2010`

Приложение D

Справочник по языку. Операторы и константы

Математические операторы

Математические операторы применяются к одному или двум числам и, в результате определённых этим оператором действий, возвращают число[1].

\+ — Сложение двух чисел или соединение двух строк (1)[2]
\− — Вычитание двух чисел (1)
* — Умножение двух чисел (1)
/ — Деление двух чисел (1)
% — Остаток от целочисленного деления двух чисел (14)
\ — Целочисленное деление двух чисел (14)
^ — Возведение в степень (14)
() — Оператор группировки (1)
= — Присвоение значений переменным (3)

Числовые константы

Числовая константа — это набор цифр, в начале которого может стоять знак минус, указывающий на отрицательное число, и, возможно, десятичная точка, после которой стоят цифры для задания десятичной дроби (числа с плавающей точкой).
Положительные целые числа могут также быть двоичными (по основанию 2) или восьмеричными (по основанию 8) или шестнадцатеричными (по основанию 16). Для задания двоичного числа начинайте его запись с 0b (0b1110 = 14), восьмеричного — с 0o (0o177 = 127), и шестнадцатеричного — с 0x (0xff = 255).

Математические константы

Математические константы похожы на переменные, но возвращают всегда одно предварительно определённое значение, запоминать которое нет необходимости.

[1] В математике такие операторы называют унарными, если применяются к одному числу и бинарными, если к двум. Например, знак «−» (минус) можно применить к одному числу для смены его знака, и тогда это будет унарный оператор (*прим. редактора*).

[2] Здесь и далее в скобках указан номер главы, когда термин появился впервые (*прим. переводчика*).

Константа	Значение
pi	3.141593

Строковые константы

Строковой константой является любой набор символов, включая нулевой, заключённый внутри двойных кавычек (").

Цветовые константы

BASIC-256 содержит список предопределённых цветов. Цветовая константа представляет собой целое число являющееся RGB-значением данного цвета. Указания этого значения достаточно для отображения цвета на экране.

Таблица D.1: Числовые значения цветов

Константа	Значение	код (R,G,B)
black	0	(0,0,0)
white	16 316 664	(248,248,248)
red	16 711 680	(255,0,0)
darkred	8 388 608	(128,0,0)
green	65 280	(0,255,0)
darkgreen	32 768	(0,128,0)
blue	255	(0,0,255)
darkblue	128	(0,0,128)
cyan	65 535	(0,255,255)
darkcyan	32 896	(0,128,128)
purple	16 711 935	(255,0,255)
darkpurple	8 388 736	(128,0,128)
yellow	16 776 960	(255,255,0)
darkyellow	8 421 376	(128,128,0)
orange	16 737 792	(255,102,0)
darkorange	11 154 176	(170,51,0)
gray или grey	10 790 052	(164,164,164)
darkgray или darkgrey	8 421 504	(128,128,128)
clear	-1	

Логические операторы

Логические операторы возвращают два значение истина или ложь (**true**/**false**) и используются для сравнения значений переменных или сообещния о состоянии некоего условия в программе. Они могут использоваться в операторе **if** или для провекрки условий в различных циклах.

= — Проверка равенства двух значений (6)
<> — Проверка неравенства двух значений (6)
< — Проверка что левое значение меньше правого (6)
<= — Проверка что левое значение меньше или равно правому (6)
> — Проверка что левое значение больше правого (6)
>= — Проверка что левое значение больше или равно правому (6)
and — Возвращает истину, если оба условия истинны (6)
not — Меняет значение истинности на противоположное (6)
or — Возвращает истину, если хотя бы одно условие истнно (6)

Логические константы

Логические операторы похожи на константы, они возвращают предопределённое значение, так что нет необходимости их запоминать. Значения логических констант переопределить в программе нельзя.

Константа	Значение	Примечание
true	1	Представляет истинное событие числом один
false	0	Представляет ложное событие числом нуль

Порядок операций

Уровень	Операторы	Категория/описание
1	()	Группировка
2	^	Степень
3	−	Унарный минус (смена знака)
4	*/ \	Умножение и деление
5	%	Деление по модулю (Mod)
6	+−	Сложение, соединение и вычитание
7	< <= > >= = <>	Сравнение строк
8	NOT	Логическое отрицание
9	AND	Логическое И
10	OR	Логическое ИЛИ
11	XOR	Логическое исключающее ИЛИ

Битовые операторы

Битовые операторы производят действия с числами на уровне битов (термин bit произошёл от binary digit или двоичная цифра), т.е двоичных знаков представления числа. Эти операции применимы только к целым числам.

& — Битовое И (**and**)

Строка `print 11 & 7` напечатает на экране 3 в следствии следующих действий с битами:

```
    1011
&   0111
-------
    0011
```

| — Битовое ИЛИ (**or**)

Строка `print 10 | 6` напечатает на экране 14 в следствии следующих действий с битами:


```
   1010
|  0110
------
   1110
```

~ — Битовое отрицание (**not**)

Строка `print ~12` напечатает на экране -13 в следствии следующих действий с битами:

```
~  00000000000000000000000000001100
-----------------------------------
   11111111111111111111111111110011
```

Замечание. Целые числа в BASIC-256 хранятся как 32-битные целые со знаком. Отрицательные числа являются бинарным дополнением положительных[3].

Переменные

Имена переменных, хранящих числовые значения должны начинаться с буквы и могут содержать любое количество алфавитно-цифровых символов. Имена переменных регистрозависимые. Такие переменные могут использоваться наравне с числовыми константами.
Переменные, хранящие строковые значения именуются по аналогичным правилам, но имя должно заканчиваться знаком доллара ($). Такие переменные могут использоваться наравне со строковыми константами.

Массивы

Оперативная память под массив выделяется с помощью команды **dim** и может быть переопределена с помощью команды **redim**. Массивы могут хранить числовые или строковые данные. Доступ к элементу массива осуществляется указанием номер элемента (индекса) отсчитанного от нуля, заключённого в квадратные скобки.
Например:

```
print myarray[4]
```

напечатает пятый элемент в массиве 'myarray'.
Массивы также могут быть определены как двумерные.
Размер массива можно получить используя комбинации [?] [?,] и [,?] приписанные к концу имени переменной массива.

Безымянные массивы

Безымянный массив — это набор числовых или строковых значений, разделённых запятыми и заключённый в фигурные скобки {}
Безымянный массив может использоваться как переменная массива или его можно присвоить переменной массива в качестве её определения таким образом:

```
dim myarray(4)
myarray = {1, 2, 3, 4}
```

[3]Представление числа в виде целого со знаком означает, что первый бит это знак числа. У положительного числа он равен 0. Таким образом максимальное положительное число представляемое 32-я битами равно $2^{31} - 1$. У отрицательных чисел первый бит равен 1. Получить представление числа -13 можно так: $13 = 00...1101_2$ в двоичной системе (впереди 27 нулей, поскольку знаковый бит не рассмативаем). Заменяем все 1 на 0 и наоборот, получаем 11...10010 (число начинается с 27-и единиц), прибавляем к результату 1 и получаем 11...10011 (тут 31 знак!), и последний шаг — добавляем 32-ой знаковый бит равный 1. Получаем ровно число из примера автора (*прим. редактора*).

Синтаксис программы

Программа на BASIC-256 состоит из списка операторов (команд), записанных по одному на строке и выполняющихся по очереди.
Например:

```
print "1я строка"
print "2я строка"
```

выведет

```
1я строка
2я строка
```

Кроме того, существуют операторы, позволяющие изменять последовательный порядок выполнения команд:

- Goto (переход по метке);
- If / Then (проверка условия);
- For / Next, Do / Until, While / End While (циклы);
- Gosub / Return (подпрограммы);

.

Приложение E

Стандартные цвета

Список стандартных наименований цвета и соответствующих им комбинаций красной (R), зелёной (G) и синей (B) составляющих.

Таблица E.1: Числовые значения цветов

Образец цвета	**Константа**	**код (R,G,B)**	**название цвета**
	black	(0,0,0)	чёрный
	white	(248,248,248)	белый
	red	(255,0,0)	красный
	darkred	(128,0,0)	тёмно-красный
	green	(0,255,0)	зелёный
	darkgreen	(0,128,0)	тёмно-зелёный
	blue	(0,0,255)	синий
	darkblue	(0,0,128)	темносиний
	cyan	(0,255,255)	голубой
	darkcyan	(0,128,128)	тёмно-голубой
	purple	(255,0,255)	пурпурный
	darkpurple	(128,0,128)	тёмно-пурпурный
	yellow	(255,255,0)	жёлтый
	darkyellow	(128,128,0)	тёмно-жёлтый
	orange	(255,102,0)	оранжевый
	darkorange	(170,51,0)	тёмно-оранжевый

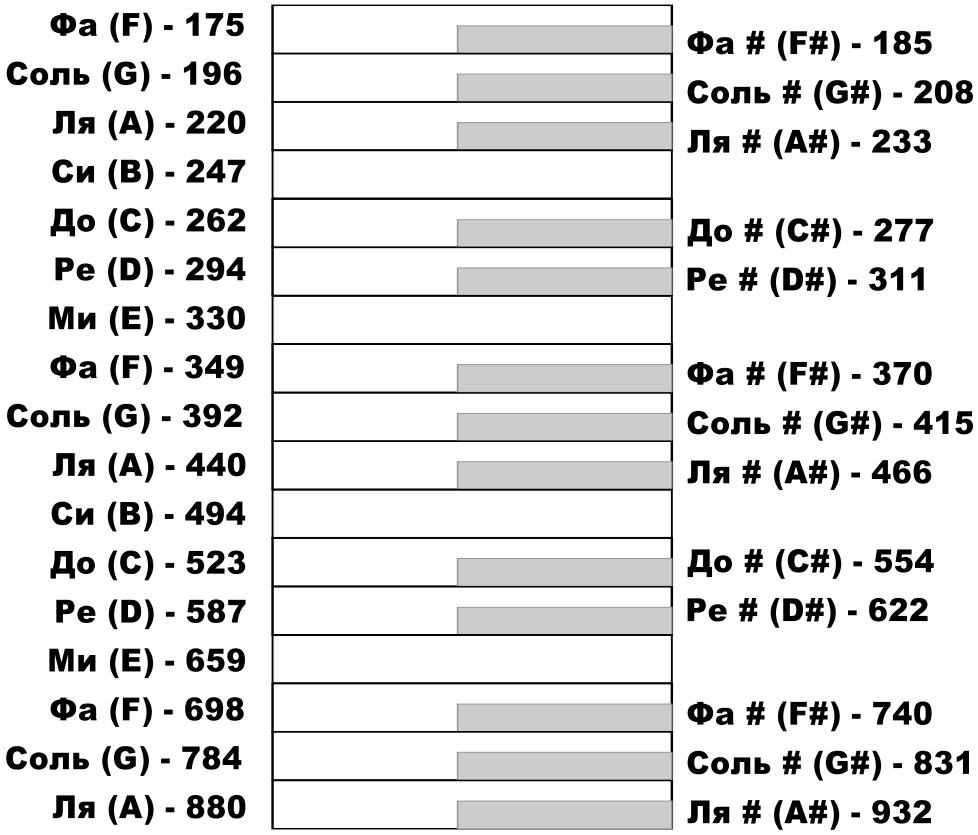
Таблица E.1 — продолжение

Образец цвета	**Константа**	**код (R,G,B)**	**название цвета**
	gray или grey	(164,164,164)	серый
	darkgray или darkgrey	(128,128,128)	тёмно-серый
	clear (-1)		прозрачный

Приложение F

Ноты

Этот рисунок поможет вам перевести клавиши пианино в частоты, используемые оператором **sound**[1].



[1] Частота Ля первой октавы равна 440Гц (*прим. редактора*).

Приложение G

Коды клавиш

Коды клавиш, возвращаемых функцией **key()** представляют собой код последней нажатой клавиши с момента последнего чтения этого кода. Таблица ниже показывает наиболее употребимые коды клавиш стандартной английской и русских[1] раскладок. В таблице представлены далеко не все значения.

Коды английской раскладки									
Клавиша	**Код**		**Клавиша**	**Код**		**Клавиша**	**Код**		**Клавиша**	**Код**
Пробел	32		A	65		L	76		W	87
0	48		B	66		M	77		X	88
1	49		C	67		N	78		Y	89
2	50		D	68		O	79		Z	90
3	51		E	69		P	80		ESC	16777216
4	52		F	70		Q	81		Забой	16777219
5	53		G	71		R	82		Ввод	16777220
6	54		H	72		S	83		←	16777234
7	55		I	73		T	84		↑	16777235
8	56		J	74		U	85		→	16777236
9	57		K	75		V	86		↓	16777237
Коды русской раскладки										
А	1040		Б	1041		В	1042		Г	1043
Д	1044		Е	1045		Ж	1046		З	1047
И	1048		Й	1049		К	1050		Л	1051
М	1052		Н	1053		О	1054		П	1055
Р	1056		С	1057		Т	1058		У	1059
Ф	1060		Х	1061		Ц	1062		Ч	1063
Ш	1064		Щ	1065		Ъ	1066		Ы	1067
Ь	1068		Э	1069		Ю	1070		Я	1071
Ё	1025									

[1]Эта таблица добавлена переводчиком (*прим. редактора*).

Приложение H

Unicode значения символов

В таблице ниже отображены Unicode значения символов стандартной раскладки Latin (английской). Эти значения соответствуют ASCII кодам, разработанным ещё в далёких 1960-х годах. Другие кодовые таблицы можно найти на http://www.unicode.org.

CHR	#		**CHR**	#		**CHR**	#		**CHR**	#		**CHR**	#		**CHR**	#
NUL	0		SYN	22		,	44		B	66		X	88		n	110
SOH	1		ETB	23		-	45		C	67		Y	89		o	111
STX	2		CAN	24		.	46		D	68		Z	90		p	112
ETX	3		EM	25		/	47		E	69		[	91		q	113
ET	4		SUB	26		0	48		F	70		\	92		r	114
ENQ	5		ESC	27		1	49		G	71		]	93		s	115
ACK	6		FS	28		2	50		H	72		^	94		t	116
BEL	7		GS	29		3	51		I	73		_	95		u	117
BS	8		RS	30		4	52		J	74		‘	96		v	118
HT	9		US	31		5	53		K	75		a	97		w	119
LF	10		Space	32		6	54		L	76		b	98		x	120
VT	11		!	33		7	55		M	77		c	99		y	121
FF	12		"	34		8	56		N	78		d	100		z	122
CR	13		#	35		9	57		O	79		e	101		{	123
SO	14		$	36		:	58		P	80		f	102		|	124
SI	15		%	37		;	59		Q	81		g	103		}	125
DC1	17		’	39		=	61		S	83		i	105		DEL	127
DC2	18		(	40		>	62		T	84		g	106			
DC3	19		)	41		?	63		U	85		k	107			
DC4	20		*	42		@	64		V	86		l	108			
NAK	21		+	43		A	65		W	87		m	109			

Коды с номерами 0-31 и 127 непечатаемые.

Эта таблица взята из стандарта Unicode 6.0 и доступна по адресу http://www.unicode.org/charts/PDF/U0000.pdf.

Кириллическую таблицу можно взять по адресу http://www.unicode.org/charts/PDF/U0400.pdf.

Приложение I

Зарезервированные слова

Ниже находится список зарезервированных слов, которые BASIC-256 использует для служебных нужд. Ни одно из этих слов нельзя использовать в качестве имени переменной или метки в операторах **goto** или **gosub**.

#
abs
acos
and
asc
asin
atan
black
blue
ceil
changedir
chr
circle
clear
clg
clickb
clickclear
clickx
clicky
close
cls
color
colour
cos
currentdir
cyan
darkblue
darkcyan
darkgray
darkgreen
darkgrey
darkorange
darkpurple
darkred
darkyellow
day
dbclose
dbcloseset
dbexecute
dbfloat
dbint
dbopen
dbopenset
dbrow
dbstring
decimal
degrees
dim
do
else
end
endif
endwhile
eof
exists
false
fastgraphics
float
floor
font
for
getcolor
getsetting
getslice
gosub
goto
graphheight
graphsize
graphwidth
gray
green
grey
hour
if
imgload
imgsave
input
instr
int
key
kill
lasterror
lasterrorextra
lasterrorline
lasterrormessage
left
length
line
log
log10
lower
md5
mid

minute
month
mouseb
mousex
mousey
netaddress
netclose
netconnect
netdata
netlisten
netread
netwrite
next
not
offerror
onerror
open
or
orange
pause
pi
pixel
plot
poly
portin
portout
print
purple
putslice
radians
rand
read
readline
rect
red
redim
refresh
rem
reset
return
rgb
right
say
second
seek
setsetting
sin
size
sound
spritecollide
spritedim
spriteh
spritehide
spriteload
spritemove
spriteplace
spriteshow
spriteslice
spritev
spritew
spritex
spritey
stamp
step
string
system
tan
text
then
to
true
until
upper
volume
wavplay
wavstop
wavwait
while
white
write
writeline
xor
year
yellow

Приложение J

Коды ошибок

Ниже приведён список ошибок времени исполнения (runtime) возвращаемый функцией **lasterror** и текстовых сообщений об ошибке, возвращаемых функцией **lasterrormessage**.

Таблица J.1: Коды ошибок

Номер и код ошибки		**Описание ошибки**	
0	ERROR_NONE		Нет ошибки
1	ERROR_NOSUCHLABEL	No such label	Метка не найдена
2	ERROR_FOR1	Illegal FOR — start number > end number	Ошибка в операторе FOR — начальное значение переменной цикла > конечного
3	ERROR_FOR2	Illegal FOR — start number < end number	Ошибка в операторе FOR — начальное значение переменной цикла < конечного
4	ERROR_NEXTNOFOR	Next without FOR	Next без FOR
5	ERROR_FILENUMBER	Invalid File Number	Неправильный номер файла
6	ERROR_FILEOPEN	Unable to open file	Невозможно открыть файл
7	ERROR_FILENOTOPEN	File not open.	Файл не открыт
8	ERROR_FILEWRITE	Unable to write to file	Ошибка записи в файл
9	ERROR_FILERESET	Unable to reset file	Невозможно удалить данные из открытого файла
10	ERROR_ARRAYSIZELARGE	Array dimension too large	Размер массива слишком велик
11	ERROR_ARRAYSIZESMALL	Array dimension too small	Размер массива слишком мал
12	ERROR_NOSUCHVARIABLE	Unknown variable	Неизвестная переменная
13	ERROR_NOTARRAY	Not an array variable	Переменная не является массивом
14	ERROR_NOTSTRINGARRAY	Not a string array variable	Переменная не является строковой
15	ERROR_ARRAYINDEX	Array index out of bounds	Индекс за границами массива
16	ERROR_STRNEGLEN	Substring length less that zero	Длина подстроки меньше нуля
17	ERROR_STRSTART	Starting position less than zero	Начальная позиция меньше нуля
18	ERROR_STREND	String not long enough for given starting character	Начальная позиция дальше конца строки
19	ERROR_NONNUMERIC	Non-numeric value in numeric expression	Нечисловое значение в числовом выражении

Таблица J.1 — продолжение

Номер и код ошибки		Описание ошибки	
20	ERROR_RGB	RGB Color values must be in the range of 0 to 255.	Значения RGB цветов должны быть в диапазоне от 0 до 255.
21	ERROR_PUTBITFORMAT	String input to putbit incorrect.	Формат строки ввода неверен
22	ERROR_POLYARRAY	Argument not an array for poly()/stamp()	Аргумент функции poly()/stamp() не является массивом
23	ERROR_POLYPOINTS	Not enough points in array for poly()/stamp()	Недостаточно точек в массиве переданном poly()/stamp()
24	ERROR_IMAGEFILE	Unable to load image file.	Невозможно загрузить файл изображения
25	ERROR_SPRITENUMBER	Sprite number out of range.	Номер спрайта вне допустимого диапазона
26	ERROR_SPRITENA	Sprite has not been assigned.	Спрайт не указан
27	ERROR_SPRITESLICE	Unable to slice image.	Невозможно создать спрайт из образа
28	ERROR_FOLDER	Invalid directory name.	Неправильное имя директории
29	ERROR_DECIMALMASK	Decimal mask must be in the range of 0 to 15.	Десятичная маска должна быть в диапазоне от 0 до 15
30	ERROR_DBOPEN	Unable to open SQLITE database.	Невозможно открыть базу данных SQLITE
31	ERROR_DBQUERY	Database query error (message follows).	Ошибка запроса к базе (сообщения следуют)
32	ERROR_DBNOTOPEN	Database must be opened first.	База данных не открыта
33	ERROR_DBCOLNO	Column number out of range.	Номер колонки вне диапазона
34	ERROR_DBNOTSET	Record set must be opened first.	Массив записей не открыт
35	ERROR_EXTOPBAD	Invalid Extended Op-code.	Неизвестный код ошибки.
36	ERROR_NETSOCK	Error opening network socket.	Ошибка открытия сетевого сокета.
37	ERROR_NETHOST	Error finding network host.	Сетевой хост (компьютер) не найден.
38	ERROR_NETCONN	Unable to connect to network host.	Невозможно установить соединение с сетевым хостом (компьютером).
39	ERROR_NETREAD	Unable to read from network connection.	Невозможно прочитать данные из сетевого соединения.
40	ERROR_NETNONE	Network connection has not been opened.	Сетевое соединение не было открыто.
41	ERROR_NETWRITE	Unable to write to network connection.	Невозможно записать в сетевое соединение.
42	ERROR_NETSOCKOPT	Unable to set network socket options.	Невозможно установить параметры сетевого соединения (сокета).
43	ERROR_NETBIND	Unable to bind network socket.	Невозможно присоединиться к сокету.
44	ERROR_NETACCEPT	Unable to accept network connection.	Невозможно принять сетевое соединение.
45	ERROR_NETSOCKNUMBER	Invalid Socket Number	Ошибочный номер сокета.
9999	ERROR_NOTIMPLEMENTED	Feature not implemented in this environment.	Функция не реализована.

Пример

```
# Тестирование перехвата ошибок
onerror nicetrap
print 1
next haha
print 2
open 999, "bogus.file"
print "Всё равно продолжаем"
# Отключаем перехват ошибок, обработка программы
# прерывается на первой же ошибке (основное поведение)
offerror
print 3
next hoho
print "Сюда никогда не попадём"
end
nicetrap:
# Это прекрасный обработчик ошибок
print "Обнаружена ошибка на строке " + lasterrorline +
      " - номер ошибки " + lasterror + " сообщение " +
      lasterrormessage + " (" + lasterrorextra + ")"
return
```

Будет напечатано:

```
1
Обнаружена ошибка на строке 4 - номер ошибки 4 сообщение Next without FOR ()
2
Обнаружена ошибка на строке 6 - номер ошибки 5 сообщение Invalid File Number ()
Всё равно продолжаем
3
ERROR on line 12: Next without FOR
```

Смотри также: Lasterror, Lasterrorextra, Lasterrorline, Lasterrormessage, Offerror, Onerror

Приложение K

Словарь терминов

Словарь терминов, используемых в книге.

ASCII

акроним от American Standard Code for Information Interchange — Американский стандартный код обмена информацией. Определяет таблицу числовых кодов для представления символов (букв, цифр и специальных знаков) используемых в английском языке.
Смотри также: *Unicode*

IP адрес

Сокращение от Internet Protocol (интернет протокол) — числовой адрес, присваеваемый устройству или компьютеру в сети Интернет

false

Булево (логическое) представление не истины (лжи). В BASIC-256 является просто другим представлением целого числа нуль (0).
Смотри также: *Булева Алгебра, true*

Font

Шрифт, способ изображения букв.

RGB

Акроним от Red Green Blue (Красный Зелёный Синий). Все цвета состоят из комбинации этих трёх базовых цветов.

SQL

Акроним от Structured Query Language — Структурированный язык запросов (термин применяется без перевода). SQL — основной язык для манипулирования данными в реляционных базах данных.

true

Булево (логическое) представление истины. В BASIC-256 является просто другим представлением целого числа один (1).
Смотри также: *Булева Алгебра, false*

Unicode

Современный стандарт кодирования целыми числами букв и знаков всех мировых языков.

Алгоритм

Пошаговый процесс решения проблемы

Аргумент

Значение, включённое в вызов функции или оператора для передачи информации. В BASIC-256 значение аргумента не изменяется функцией или оператором.

Асинхронно

Процессы или события происходящие один после другого.
Смотри также: *Синхронно*

База данных

Организованная коллекция данных. Большинство баз храняться в компьютере и состоят из таблиц однотипных данных, разбитых на строки и колонки
Смотри также: *Колонка, Ряд, SQL, Таблица*

Булева Алгебра

Алгебра логики изобретённая Чарльзом Булем более 150 лет назад. Использует только значения истина/ложь (true/false).

Герц (Гц)

Единица измерения частоты в циклах в секунду. Названа в честь немецкого физика Генриха Герца.
Смотри также: *частота*

Градус

Единица измерения углов. На плоскости угол изменяется от 0° до 360° градусов. Прямой угол составляет 90°, развёрнутый — 180°.
Смотри также: *Угол, Радиан*

Декартова система координат

Система координат однозначно определяющая точку на плоскости с помощью двух перпендикулярных осей. Координатами являются расстояния от точки (0,0) до проекций опущенных из точки на ось) точки на оси Ox и Oy. Проекция точки на ось — основание перпендикуляра опущенного из точки на ось.

Двумерный массив

Множество данных в памяти компьютера, представляющих собой строки и столбцы одной таблицы, которые можно адресовать парой целочисленных индексов.
Смотри также: *Массив*

Именованная константа

Постоянная величина имеющее собственное имя. Например **pi** или **true**

Колонка (базы данных)

Определяет единый для всех строк базы тип хранящейся информации.

Константа

Постоянная величина, значение которой не может быть изменено.

Логическая ошибка

Ошибка в программе, при которой она выполняется, но не так, как ожидалось её создателем.

Массив

Организованное множество данных в памяти компьютера. Обращаться к элементам массива можно с помощью одного или нескольких целочисленных индексов.
Смотри также: *Числовой массив, Одномерный массив, Строковый массив, Двумерный массив*

Математический оператор

Действие над одним или двумя объектами

Метка

Имя, с помощью которого можно пометить какое-то место в программе. Для перехода к метке используются операторы **goto** или **gosub**.

Одномерный массив

Набор данных в памяти компьютера, которые можно адресовать с помощью одного индекса.
Смотри также: *Массив*

Окно вывода графики

Специальная область на экране куда выводятся рисунки графическими командами BASIC-256.

Окно вывода текста

Специальная область на экране куда выводятся текстовые сообщения и сообщения об ошибках BASIC-256.

Оператор

Одиночная команда выполняющяя какое-то преобразование данных и не возвращающее никаких значений.

Переменная

Именованная область памяти компьютера, применяемая для хранения изменяемых данных. В BASIC-256 переменные могут быть числовыми и строковыми.
Смотри также: *Числовая переменная* и *Строковая переменная*

Пиксель

Наименьшая адресуемая величина компьютерного экрана.

Пойнт

Типографская единица измерения. 1 пойнт = 1/72" (дюйма). Текст величиной 12 пойнт имеет высоту равную 1/6 дюйма или 4.2 мм

Подстрока

Часть большой строки.

Подпрограмма

Блок кода, исполняющая некое задание независимо от остальной программы. Может использоваться многократно в одной программе.

Порт

Число, используемое в программе для создания сокета и обмена через него информацией.

Прозрачный

Объект, через который виден низлежащий объект.

Псевдокод

Пояснение алгоритма программы на естественном (не компьютерном) языке. Слово «псевдокод» состоит из префикса «псевдо», означающее что-то похожее, но не то (можно перевести «как бы»), и слова «код», означающее текст программы.

Пустая строка

Строка не содержащая ни одного символа и имеющая длину нуль (0). В программе представляется двумя двойными кавычками ("").
Смотри также: *Строка*

Радиан

Единица измерения углов. Углы на плоскости измеряются от 0 до 2π радиан. Развёрнутый угол равен π радиан, а прямой угол равен $\frac{\pi}{2}$ радиан.
Смотри также: *Угол, Градус*

Радиус

Расстояние от центра круга до его границы. Радиус равен половине диаметра круга.

Ряд (базы данных)

Строка или запись в таблице базы. Представляет собой единицу хранения информации в таблице базы данных.

Синтаксическая ошибка

Ошибка в написании оператора или функции при которой программа не может быть исполнена.

Синхронно

События, происходящие одновременно.
Смотри также: *Асинхронно*

Сокет

Программная точка сетевого двунаправленного соединения двух компьютеров или двух процессов на одном компьютере.

Список

Набор данных, присваеваемых массивам или используемых некоторыми операторами.

Спрайт

Графический образ, используемый для отображения как единое целое.

Строка

Последовательность символов (букв, цифр и знаков). Строковые константы должны быть окружены двойными кавычками (").

Строковая переменная

Переменная, предназначенная для хранения строк. Строковая переменная обозначается знаком доллара ($) после имени переменной.

Строковый массив

Массив, состоящий из строк.

Структура данных

Способ организации хранения и использования информации для эффективного использования в компьютерной системе.

Таблица (базы данных)

Набор данных, организованных в виде рядов и колонок. Таблицы имеют определённый набор колонок и могут иметь несколько или ни одной строки.

Угол

Фигура, образованная двумя лучами исходящими из одной точки называется угол. Угол измеряется величиной поворота от одного луча к другому и выражается в *радианах* или *градусах*.

Функция

Специальный тип операторов BASIC-256 который может принимать несколько (или ни одного) входящих данных (*аргументов*), производить вычисления и возвращать информацию в программу.

Целое число

Число являющееся натуральным (применяемым для счёта: 1,2,3 ...) или отрицательным натуральным числом или нулём. В BASIC-256 целые числа лежат в диапазоне от -2 147 483 648 до 2 147 483 647.

Частота

Количество событий (колебаний) в единицу времени.
Смотри также: *Герц*

Число с плавающей точкой

Дробное число в представлении которого нет конкретного места для десятичной точки. Они имеют вид $\pm m \times 10^p$, где m — мантисса, а p — показатель степени. Например $\pm 1.332 \times 10^{12}$. В BASIC-256 числа с плавающей точкой хранятся в диапазоне $\pm 1.7 \times 10^{\pm 308}$ и имеют точность 15 знаков.

Числовая переменная

Переменная, предназначенная для хранения целых чисел или *чисел с плавающей точкой* (дробных чисел).

Числовой массив

Массив состоящий из чисел.

Список иллюстраций

Список программ

