

## Где скачать BASIC-256:

Для дистрибутивов ALT Linux

- ветка 4.0 [basic256-0.9.6-alto.M40.1.i586.rpm](#)
- ветка 4.1 [basic256-0.9.5-alto.M41.1.i586.rpm](#)
- ветка p5 [basic256-0.9.6-alto.M50P.1.i586.rpm](#)
- ветка 5.1 [basic256-0.9.5-alto.M51.1.i586.rpm](#)

Windows версия

<http://basic256.org> (<http://www.sourceforge.net/projects/kidbasic>)

## Как установить BASIC-256 в Linux

Для Альт Линукс: настроить репозиторий и обновить/установить пакет через synaptic или apt

Для rpm-based дистрибутивов: rpm -Uvh <имя\_пакета>.rpm

## [BASIC-256 Глава 1](#)

Posted: 11 Nov 2010 04:00 PM PST

Предлагаем вашему вниманию перевод книги по BASIC-256. Недавно в репозиториях появилась новая версия с новыми возможностями. Поддержку пакета в репозитории ALT Linux ведет Сергей Ирюпин. Он же переводчик книги. Перевод планируется печатать по главам. Это первая.

### От переводчика

BASIC-256 – это версия классического языка BASIC, созданная для обучения детей основам программирования. Использует традиционные управляющие структуры, такие как gosub, for/next и if и т.п., что помогает детям легче понять, как происходит управление выполнением программы. Имеет встроенный редактор текста программ, пошаговый отладчик, специальные окна для текстового и графического вывода.

Начал разрабатываться в 2006 году как свободная кроссплатформенная альтернатива коммерческим реализациям BASIC. Долгое время развивался слабо. В ПСПО была включена версия 0.9.2 с весьма ограниченными возможностями. С 2008 года новый разработчик, Джеймс Рено (James M. Repeau), преподаватель и программист, стал активно развивать BASIC-256. В 2010 году к разработке и переводу подключился российский программист Сергей Ирюпин (lamp@), одновременно став майнтейнером пакета BASIC-256 в ALT Linux. В мае 2010 года среди пакетов ALT Linux стала доступна версия 0.9.5. Весной 2010 года Джеймс начал писать книгу — руководство по BASIC-256, которую фактически закончил к концу июля. Затем он начал активно добавлять новые возможности — работу с базами данных, портами ввода-вывода, сетевые функции, одновременно создавая соответствующие новые главы в книге.

Вашему вниманию представляется перевод [книги Джеймса Рено](#), описывающий работу в BASIC-256 версии 0.9.6w. Именно такая версия доступна сейчас в пакетной базе ALT Linux. Она имеет некоторые отличия от текущей версии, 0.9.6.48 (после буквы z, автор перешел на цифровое обозначение релизов), бинарную версию которой можно загрузить только для Windows. В частности, версия 0.9.6w для ALT Linux снабжена оффлайновой справкой на русском языке (начиная с 0.9.6g автор сделал справку онлайнной — в окне открывается вики раздел сайта [basicbook.org](http://basicbook.org)). В книге много примеров, которые у автора собраны [на одной странице](#)

# Глава 1: Знакомство с BASIC-256 – скажи

## «Привет»<sup>1</sup>.

В этой главе вы познакомитесь со средой BASIC-256, на примере операторов print и say. Вы увидите разницу между командами, которые вы отдаете 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 ее можно исправить.

### 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).

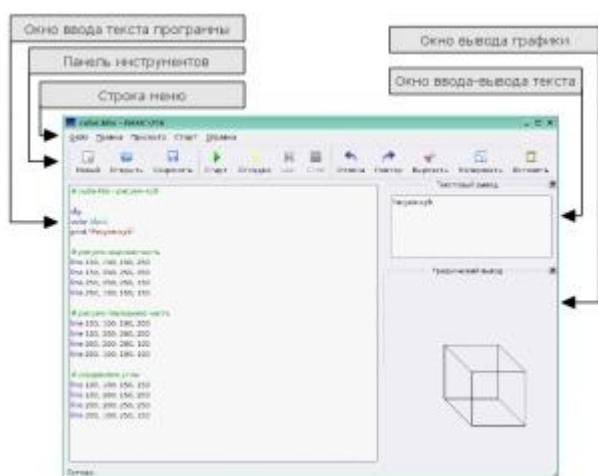








Рисунок 1. Экран BASIC-256







### 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 сохраненные 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.

### Панель инструментов

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

-  Новый – начать новую программу.
-  Открыть – загрузить сохраненную программу.
-  Сохранить – сохраняет программу на диск или USB устройство.
-  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 отмененное 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.

### 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 сохраненные ранее программы.

### Окно ввода-вывода текста

Это 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.

### 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.

### 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 наберете эту команду, щелкните мышью по кнопке «Старт»  на панели инструментов. BASIC-256 поздоровался с вами через динамики компьютера?<sup>2</sup>



**Новое  
понятие**

*say* выражение

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



**Новое  
понятие**


“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

**Новое понятие** 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.

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:

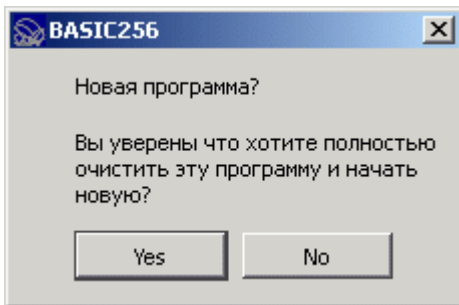


Рисунок 2. BASIC-256 – окно начала новой программы

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



**Новое понятие**



«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.




**Эксперимент**

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 наберете эту команду, щелкните мышью по кнопке  «Старт» на панели инструментов. Сказал ли BASIC-256 то, что вы хотели?<sup>3</sup>



**Новое понятие**

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.

## BASIC-256 действительно хорошо работает с числами – простая арифметика

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

Четыре основные действия: сложение, вычитание, умножение и деление приводятся в исполнение, используя операторы, показанные в Табл. 1.

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

Таблица 1. Основные математические операции

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»?



**Новое  
понятие**

+ - \* / ( )

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 круглые скобки " (" и ")" для группировки операций.



**Эксперимент**

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

## 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»).



**Новое  
понятие**

+ (конкатенация)

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



**Эксперимент**

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

## 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 наберете текст этой программы, щелкните мышкой по кнопке «Старт» на панели инструментов. В окне для ввода-вывода текста появятся слова: «привет» на первой строке и «всем» — на второй.



**Новое  
понятие**

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



**Новое  
понятие**

cls  
Оператор cls стирает всю информацию в окне ввода-вывода текста.<sup>4</sup>



**Эксперимент**

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

**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.

<sup>1</sup>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)

<sup>2</sup>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)

<sup>3</sup>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)

<sup>4</sup>Каждый раз, при запуске программы для исполнения, окно ввода-вывода текста очищается автоматически. (прим. разработчика)

## [BASIC-256 Глава 2](#)

Posted: 18 Nov 2010 04:00 PM PST

Продолжаем публиковать перевод [книги Джеймса Рено](#). Эту главу перевел Сергей Ирюпин. Начало публикации [здесь](#). Мы также позволили себе сменить «скучные» значки автора (см. предыдущую [статью](#)) на другие, ведь книга предназначена не только учителям, но и самим школьникам!

## Глава 2: Рисуем основные фигуры

В этой главе мы начнем 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.

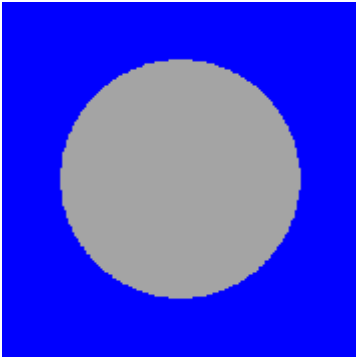
### Рисуем прямоугольники и окружности

Давайте начнем с написания графической программы для нашей любимой спортивной команды «Grey Spots»<sup>1</sup>. Ее цвета – голубой и серый.

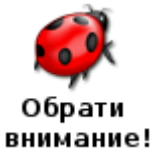
```
1 # c2_greyspots.kbs
2 # программа для нашей команды - the grey spots
3 clg
4 color blue
5 rect 0,0,300,300
6 color grey
7 circle 149,149,100
8 say "Grey Spots, Grey Spots, Grey spots rule!"
```

Программа 9: Grey Spots





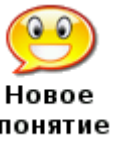
Вывод программы 9: Grey Spots



### Предупреждение:

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

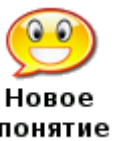
Давайте изучим каждую строку приведенной выше программы. Первая строка называется ремарка или оператор комментария. Комментарий – это место, где программист оставляет свои пометки в компьютерном коде, игнорируемые системой. Такие пометки хороши для того, чтобы описать что делает тот или иной фрагмент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



#  
`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.

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



`cls`

Оператор `cls` 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. рис. 3), или определив один из 16 миллионов вариантов, смешивая основные цвета (красный (Red), зеленый (Green) и голубой (Blue)) разной интенсивности.

Если вы используете цифровой способ определения цвета, учтите, что числа должны быть в диапазоне от 0 до 255. Ноль (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), черный — как 0,0,0 (нулевая яркость всех цветов). Такое числовое представление известно как «RGB триплет». Рисунок 3 показывает имена некоторых цветов и их

числовые значения.

`color имя_цвета`  
`color красный,зеленый,голубой`  
`color RGB_число`



Новое  
понятие

вместо **color** можно также использовать **colour**<sup>2</sup>

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, зеленого и голубого цветов (R,G,B) или одним значением, полученным в результате вычисления выражения: `красный*256^2+зеленый*256+голубой`.

Образец цвета	код	название цвета
	<code>black (0,0,0)</code>	чёрный
	<code>white (248,248,248)</code>	белый
	<code>red (255,0,0)</code>	красный
	<code>darkred (128,0,0)</code>	темно-красный
	<code>green (0,255,0)</code>	зеленый
	<code>darkgreen (0,128,0)</code>	темно-зеленый
	<code>blue (0,0,255)</code>	синий
	<code>darkblue (0,0,128)</code>	темносиний
	<code>cyan (0,255,255)</code>	голубой
	<code>darkcyan (0,128,128)</code>	темно-голубой
	<code>purple (255,0,255)</code>	пурпурный
	<code>darkpurple (128,0,128)</code>	темно-пурпурный
	<code>yellow (255,255,0)</code>	желтый
	<code>darkyellow (128,128,0)</code>	темно-желтый
	<code>orange (255,102,0)</code>	оранжевый
	<code>darkorange (170,51,0)</code>	темно-оранжевый
	<code>gray или grey (164,164,164)</code>	серый
	<code>darkgray или darkgrey (128,128,128)</code>	темно-серый
	<code>clear (-1)</code>	прозрачный

Рисунок 3. Названия цветов

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.

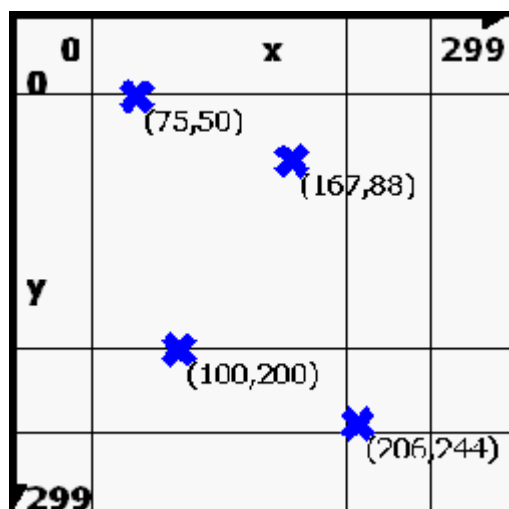


Рисунок 4. Декартова система координат окна вывода графики

Следующий оператор (строка 5) — **rect**. Он позволяет рисовать прямоугольники. **Rect** использует четыре цифры, разделенные 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).

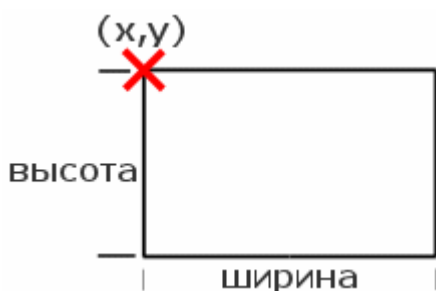


Рисунок 5. Прямоугольник

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



**Новое понятие**

**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).

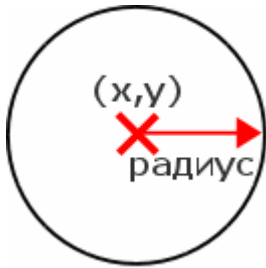


Рисунок 6. Окружность



Новое  
понятие

**circle**  $x, y, \text{радиус}$

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



Пробуй,  
исследуй!

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 "Hello."
```

## Программа 10: Лицо, составленное из прямоугольников



### Вывод программы 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
21 circle 200,100,30
```

## Программа 11: Улыбающееся лицо, составленное из окружностей




### Вывод программы 11: Улыбающееся лицо, составленное из окружностей




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

Сохранение вашей программы и её повторная загрузка

Теперь, когда программы становятся все 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 идет 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.

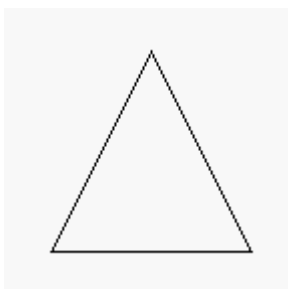
Чтобы открыть ранее сохраненную программу, используйте кнопку «Открыть»  на панели инструментов, либо пункт «Открыть» в выпадающем меню «Файл».

## 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



Вывод программы 12: Рисуем треугольник



**Новое  
понятие**

**line** *старт\_x, старт\_y, финиш\_x, финиш\_y*

Рисует линию шириной один пиксель от стартовой точки до конечной, используя текущий цвет.



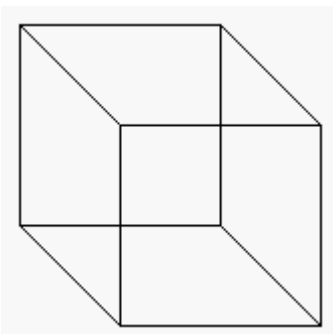
**Пробуй,  
исследуй!**

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
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**



Вывод программы 13: Рисуем куб

## 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
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



Вывод программы 14: Используем **plot** для рисования точек (обведено для привлечения внимания)



**Новое  
понятие**

**plot**  $x, y$

Закрашивает один пиксель (точку на экране) текущим цветом.



**Большая  
программа**

В конце каждой главы вы найдете 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.

Приведенная 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 создает 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
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



Вывод программы 15: Большая программа — разговаривающее лицо

-----  
<sup>1</sup>Серые пятнышки (прим. редактора)

<sup>2</sup>Это сделано из-за того, что в разных англоязычных странах разное написание слова «цвет» (прим. переводчика)

## [BASIC-256. Глава 3](#)

### Глава 3: Звуки и музыка

Теперь, когда у нас есть цвет и графика, давайте добавим звук и немного музыки. Вашему вниманию будут представлены основные концепции физики звука, числовых переменных и нотных записей. Вы научитесь переводить мелодию в частоты и длительность для того, чтобы компьютер синтезировал музыку.

#### Основы звука – все, 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.

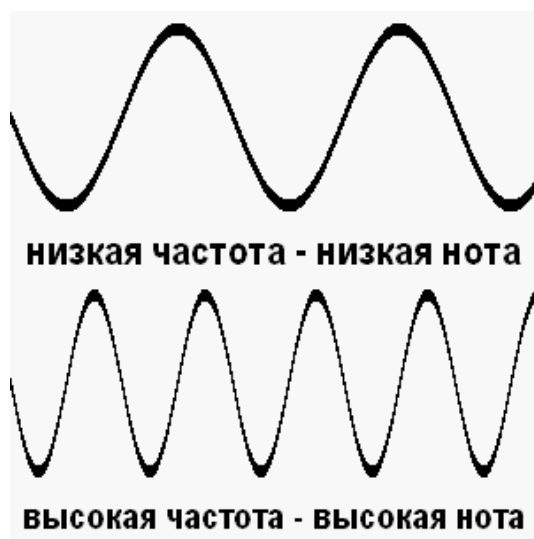


Рисунок 7. Звуковые волны

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 щелкающий звук в колонках между звуками, сыгранными в этом примере. Это вызвано тем, что компьютер создает 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 создает и проигрывает все звуки сразу, делая их более гладкими.

**sound** *частота, длительность*

**sound** {*частота1, длительность1, частота2, длительность2, ...*}

**sound** *массив\_чисел*



**Новое  
понятие**

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, заключенном в фигурные скобки `{}`. Третья форма **sound** использует массив, который содержит частоты и длительности. Речь о массивах пойдет в Главе 11.

Как же BASIC-256 воспроизводит мелодию? Первое, что мы должны сделать — преобразовать ноты на нотном стане в частоты. На рисунке 8 показаны две октавы нот, их названия и приблизительная частота, которая их создает. В музыке есть ещё особое понятие — пауза. Пауза означает — «не воспроизводить музыку в определенный интервал времени». Если вы используете список звуков, то можете вставить паузу, указав частоту нуль (0) и необходимое время паузы.

B	C	C Sharp	D	D Sharp	E	F	F Sharp	
494	523	554	587	622	659	698	740	
D	D Sharp	E	F	F Sharp	G	G Sharp	A	A Sharp
294	311	330	349	370	392	415	440	466
F	F Sharp	G	G Sharp	A	A Sharp	B	C	C Sharp
175	185	196	208	220	233	247	262	277

Рисунок 8. Ноты<sup>1</sup>

Возьмите небольшой музыкальный фрагмент, а затем посмотрите значения частоты для каждой ноты. Почему бы нам не попросить компьютер сыграть «Атака!» (смотрите рисунок 9). Вы наверное заметили, что нота «соль» (G) второй октавы находится выше нотного стана. Если нота располагается не на нотном стане, ее частоту можно удвоить, чтобы сделать выше, или уменьшить наполовину, чтобы сделать ниже. Получается та же самая нота, только на октаву выше или ниже.<sup>2</sup>



Рисунок 9. Атака!

Теперь, когда у нас есть частоты, нам нужны ещё длительности звучания для каждой из нот. Таблица 2 показывает наиболее распространенные 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 битах в минуту (BPM – beats per minute), используя формулу 1.

$$\text{Длительность ноты} = 1000 * 60 / \text{BPM} * \text{Относительная длина}$$

Формула 1. Вычисление длительности ноты

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
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

Таблица 2. Ноты и обычная длительность

Теперь с формулой и таблицей для расчета 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! [Скачать](#)



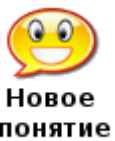
Зайдите в интернет и найдите музыку для «Row-row-row Your Boat» или другую мелодию, напишите программу для её воспроизведения.

## 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.

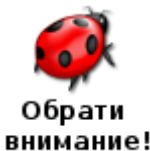
### Числовая переменная



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<sup>3</sup> и числа и чувствительно к регистру. Запрещается использовать слова, принадлежащие языку BASIC-256 при наименовании переменных (см. Приложение I).

Примеры правильных имен 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»<sup>4</sup>.

Теперь, когда мы увидели переменные в действии, мы можем переписать программу «Атака!», используя переменные и формулу для расчета 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,
7 659, eighth, 784, eighth}
7 say "Атака!"

```

### Программа 20: Атака! с переменными. [Скачать](#)



#### Обрати внимание!

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



#### Большая программа

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. 10. 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 задается 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. [Скачать](#)

-----

<sup>1</sup>Латинские названия нот можно посмотреть [в википедии](#). За эталон частоты ноты берётся нота ля (A) первой октавы, частота которой должна быть равной 440 Гц, что и видно на рисунке (*прим. редактора*).

<sup>2</sup>Отношение частот одинаковых нот из соседних октав равно двум или 1/2. (*прим. редактора*).

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

<sup>4</sup>Имена переменным 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*).



## 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 — заключенная 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 «трех 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.

### 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 фактически операторы программы. Псевдокод создается не для компьютера, а для того, чтобы помочь вам осознать сложность задачи и разбить ее на смысловые части.

Нет «самого лучшего» способа написания псевдокода. Существуют десятки стандартов, и каждый из них подходит для определенного 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 11).



Рисунок 11. Школьный автобус

Разобьем эту задачу на две части:

- нарисовать колеса
- нарисовать корпус

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

Выбрать черный цвет.  
 Нарисовать колеса.  
 Выбрать желтый цвет.  
 Нарисовать кузов автобуса.  
 Нарисовать переднюю часть автобуса.

Таблица 3. Школьный автобус — псевдокод

Чтобы наша программа заработала, все, что осталось сделать – это записать ее:

Выбрать черный цвет.	color black
Нарисовать колеса.	circle 50,120,20 circle 200,120,20
Выбрать желтый цвет.	color yellow
Нарисовать кузов автобуса.	rect 50,0,200,100
Нарисовать переднюю часть автобуса.	rect 0,50,50,50

Таблица 4. Школьный автобус — псевдокод и команды BASIC-256

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 колеса
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 колеса, мы также могли нарисовать сначала переднюю часть, а затем заднюю... Мы можем назвать десятки различных способов решения этой простой задачи.

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.







**Пробуй,  
исследуй!**

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

## 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 столкнетесь. Вот несколько основных элементов:

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.


	<p>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.</p>
---	--

Таблица 5. Основные элементы блок-схем

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

### 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 пойдете в школу голодным.

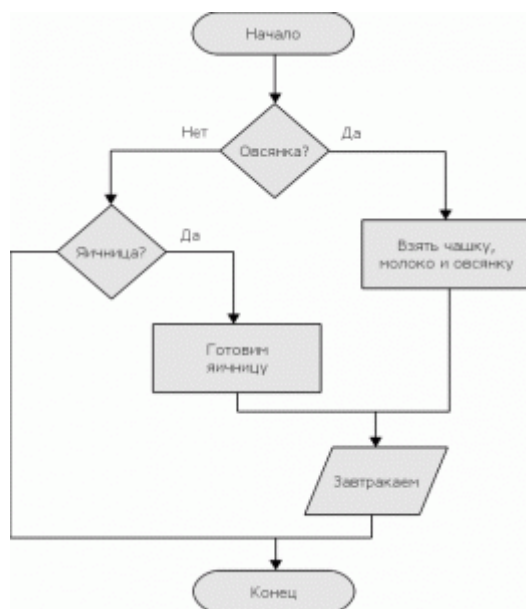


Рисунок 12. Завтрак — блок-схема

Взгляните на рисунок 12 (выше) и проследите за всеми стрелками. Вы видите, как эта схема представляет собой сценарий?

### Блок-схема, пример второй

Еще один пример, связанный с едой. Вы очень хотите пить и намереваетесь купить газировку в торговом автомате. Посмотрите на рисунок 13 (ниже).

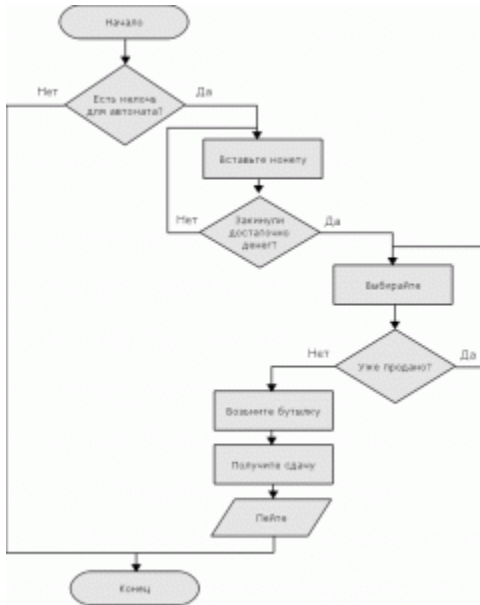


Рисунок 13. Автомат с газировкой — блок-схема

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.



**Пробуй,  
исследуй!**

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

## Глава 5: Программа задает вам вопросы

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

### 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, заключенный 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 имен, чувствительность к регистру и правила зарезервированных слов одинаковы как для строковых, так и для числовых переменных.

```

1 # ilikejim.kbs
2 name$ = "Сереза"
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 Сереза

Сереза мой друг.  
Мне нравится Сереза.

### Вывод программы 23: Мне нравится Сереза



**Новое  
понятие**

#### 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 имен строковых переменных: **d\$, c7\$, book\$, X\$** и **barnYard\$**.



**Обрати  
внимание!**

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

## Input – получение текста или чисел от пользователя

До сих пор в коде самой программы содержалась вся необходимая информация для ее 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$
```



#### Новое понятие

```
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 равно 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



### Большая программа

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

**Пример работы программы:**

*Глупая история.*

*Существительное? кот*

*Глагол? идти*

*Название комнаты в доме? кухня*

*Глагол? рисовать*

*Существительное? колбаса*

*Прилагательное? зеленый*

*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 зеленый, из-за того, что я начал играть в карандаш.*

*Конец.*

# 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.

## Истина и ложь

Язык BASIC-256 имеет еще 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).



Новое  
понятие

*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).

## 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.

Оператор	Операция
<	Меньше <i>выражение1 &lt; выражение2</i> Возвращает истину, если <i>выражение1</i> меньше <i>выражения2</i> и ложь в противном случае
<=	Меньше или равно <i>выражение1 &lt;= выражение2</i> Возвращает истину, если <i>выражение1</i> меньше или равно <i>выражению2</i> и ложь в противном случае
>	Больше <i>выражение1 &gt; выражение2</i> Возвращает истину, если <i>выражение1</i> больше <i>выражения2</i> и ложь в противном случае
>=	Больше или равно <i>выражение1 &gt;= выражение2</i> Возвращает истину, если <i>выражение1</i> больше или равно <i>выражению2</i> и ложь в

	противном случае
=	Равно <i>выражение1 = выражение2</i> Возвращает истину, если <i>выражение1</i> равно <i>выражению2</i> и ложь в противном случае
<>	Не равно <i>выражение1 &lt;&gt; выражение2</i> Возвращает истину, если <i>выражение1</i> не равно <i>выражению2</i> и ложь в противном случае

Таблица 6. Операторы сравнения



**Новое  
понятие**

< <= > >= = <>

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.

## 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.

```

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

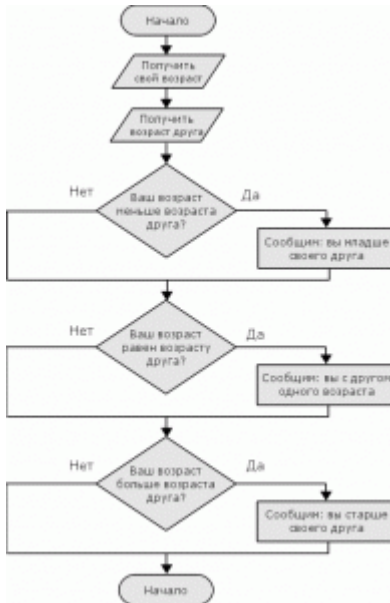


Рисунок 14. Сравним два возраста – блок-схема



**Новое  
понятие**

**if условие then оператор**

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

## 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.



**Новое  
понятие**

**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  $n = \text{int}(\text{rand} * r) + 1$

```
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**.

**Обрати внимание!**

Это создало бы то, что называют «логическая ошибка». Запомните, что каждый раз, когда выполняется **rand**, он возвращает разные случайные числа.

## Логические операторы:

Иногда необходимо соединить вместе несколько простых условий. Это может быть сделано с помощью четырех 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».

### Оператор Операция

Логическое И

выражение1 **AND** выражение2

Если оба выражения истинны, тогда и результат будет истинным, в противном случае он будет ложным

**AND**

<b>AND</b>		<b>выражение2</b>	
		<b>ИСТИНА</b>	<b>ЛОЖЬ</b>
<b>выражение2</b>	<b>ИСТИНА</b>	ИСТИНА	ЛОЖЬ
	<b>ЛОЖЬ</b>	ЛОЖЬ	ЛОЖЬ

Логическое Или

выражение1 **OR** выражение2

Если выражение1 или выражение2 истинны, то результат будет истинным, если оба ложны — ложным.

**OR**

<b>OR</b>		<b>выражение2</b>	
		<b>ИСТИНА</b>	<b>ЛОЖЬ</b>
<b>выражение2</b>	<b>ИСТИНА</b>	ИСТИНА	ИСТИНА
	<b>ЛОЖЬ</b>	ИСТИНА	ЛОЖЬ

**XOR**

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».

XOR	выражение2		
	ИСТИНА	ЛОЖЬ	
выражение2	ИСТИНА	ЛОЖЬ	ИСТИНА
	ЛОЖЬ	ИСТИНА	ЛОЖЬ

Логическое отрицание

**NOT** выражение1

Возвращает противоположное значение. Если выражение1 истинно, результат будет ложь, если ложно — истина.

**NOT**

NOT		
выражение1	ИСТИНА	ЛОЖЬ
	ЛОЖЬ	ИСТИНА

Таблица 7. Логические операторы



**Новое  
понятие**

#### **and or xor not**

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

## **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**.



**Новое  
понятие**

**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 кости



**Новое  
понятие**

«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**).

## 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.



**Новое  
понятие**

```
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**

## 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**

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

```
1 # dieroll.kbs - «бросаем кубик» и рисуем его  
2 # hw - высота и ширина точки  
3 hw = 70  
4 # расстояние между точками
```

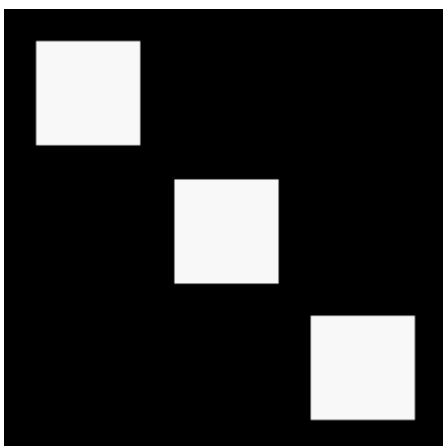


```

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



Пример вывода программы 32: Большая программа – «Бросаем» кубик и рисуем его

# Глава 7: Циклы и счетчики — 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 счетчиком, или и то и другое вместе. В этой главе мы рассмотрим три вида оператора цикла, освоим режим «быстрой графики», и научимся замедлять выполнение программы.

## Цикл For

Наиболее распространенный вид оператора цикла — это **for**. Цикл **for** повторяет блок операторов указанное число раз, отслеживая значение счетчика. Счетчик цикла может начинаться с любого значения и завершаться любым значением, шаг приращения счетчика 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 43: Оператор for вместе с step

```
for счетчик = выражение1 to выражение2 [step выражение3]  
    оператор(ы)  
next счетчик
```



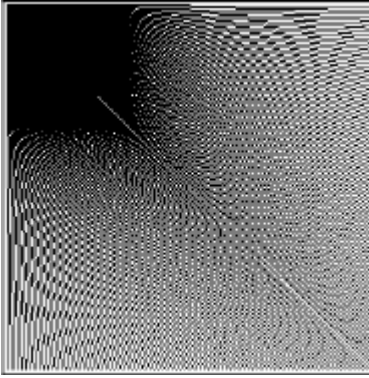
#### Новое понятие

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

Используя цикл, можно легко рисовать очень интересные картинки. Программа 35 рисует муаровый узор<sup>1</sup>. Это действительно интересное изображение возникает потому, что компьютер не в состоянии нарисовать абсолютно прямую линию<sup>2</sup>. 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



Вывод программы 35: Муаровый узор



**Пробуй,  
исследуй!**

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 счета в обратном направлении. Для этого укажите после **step** отрицательное число.

```
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 – счет в обратном направлении

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

Вывод программы 36: Оператор for — счет в обратном направлении



**pause** секунды

**Новое  
понятие**

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.

## 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 введет 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)*

**Новое  
понятие**

**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 счета 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

## 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.

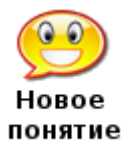
Иногда нам нужна программа с бесконечным циклом, пока сам пользователь не остановит её. Это может быть легко сделано с помощью логической константы *true* (*истина*) (см. программу 39).

```
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<sup>3</sup>



**Новое  
понятие**

**while** *условие*  
*оператор(ы)*  
**end while**

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

Программа 40 использует **while/end while** для счета 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
```

Вывод программы 40: While считает до 10

## 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**.



**Новое  
понятие**

**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<sup>4</sup>.

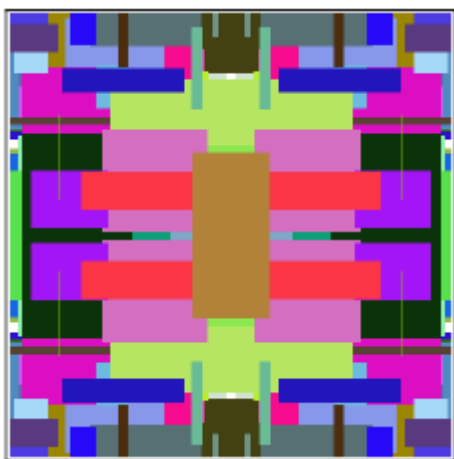
```
1 # kalidescope.kbs
2 clg
3 fastgraphics
4 for t = 1 to 100
5     r = int(rand * 256)
```

```

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: Kaleidoscope



Пример вывода программы 41: Kaleidoscope



**Пробуй,  
исследуй!**

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



**Большая  
программа**

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
5 # начальная позиция мяча
6 x = rand * 300

```

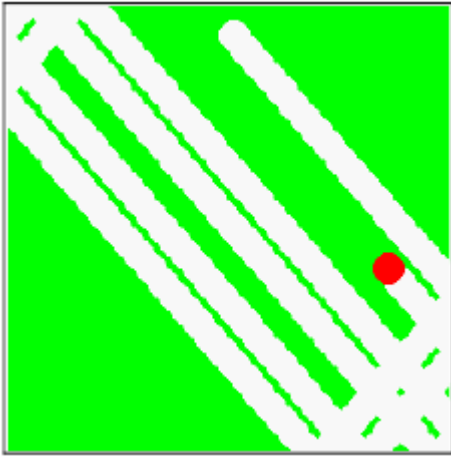


```

7  y = rand * 300
8  # size of ball
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 и 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 и 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**



Пример вывода программы 42: Большая программа — прыгающий мяч

-----  
<sup>1</sup> 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*).

<sup>2</sup> Разумеется, если только она не строго вертикальная или горизонтальная (*прим. разработчика*).

<sup>3</sup> У автора используется слово «nevermore» — намек на стихотворение Эдгара Алана По «Ворон», в котором есть строчка: Quoth the Raven, “Nevermore.” — [Каркнул: «Больше никогда!»](#) (*прим. редактора*).

<sup>4</sup> После завершения выполнения программы режим «быстрой графики» выключается автоматически (*прим. разработчика*).

## Глава 9: Подпрограммы — повторное использование кода

В этой главе 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.

### 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 определенному месту в программе для того, чтобы вы смогли туда перейти во время выполнения программы. Меток в программе может быть много.



**Новое  
понятие**

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 учетом регистра букв. Кроме того, нельзя использовать зарезервированные слова BASIC-256 или используемые имена переменных (см. приложение I).

Примеры правильных имен меток: *top:*, *far999:*, *About:*.



**Новое  
понятие**

#### **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.

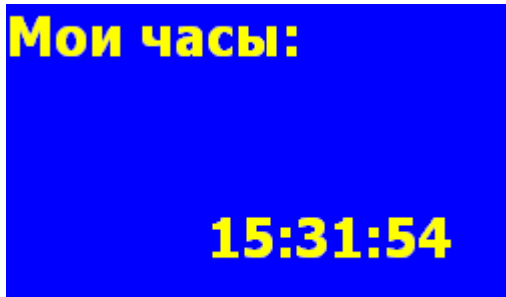
```
1 # textclock.kbs
2 fastgraphics
3 font "Tahoma", 20, 100
4 color blue
5 rect 0, 0, 300, 300
6 color yellow
7 text 0, 0, "Мои часы:"
8 showtime:
9 color blue
10 rect 100, 100, 200, 100
```

```

11 color yellow
12 text 100, 100, hour + ":" + minute + ":" + second
13 refresh
14 pause 1.0
15 goto showtime

```

Программа 50: Цифровые часы



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.



**Новое  
понятие**

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

## Повторное использование кода — оператор Gosub

Во многих программах мы найдем 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 определенную 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



**Новое  
понятие**

**gosub** *метка*

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



**Новое  
понятие**

**return**

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



**Новое  
понятие**

**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 два цифры в строку line$
36 if n < 10 then line$ = line$ + "0"
37 line$ = line$ + n
38 return

```

Программа 52: Цифровые часы — усовершенствованные



Пример вывода программы 52: Цифровые часы — усовершенствованные (фрагмент)

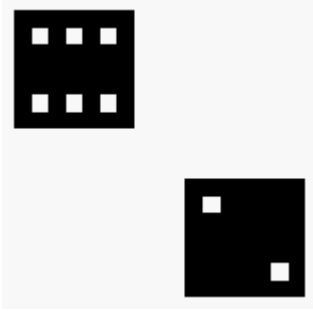


## Большая программа

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.

### Программа 53: Большая программа — бросаем два кубика

```
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
```



Примера 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 мышку: двигать ее и нажимть на кнопки. Обе эти возможности обсуждаются на простых примерах.

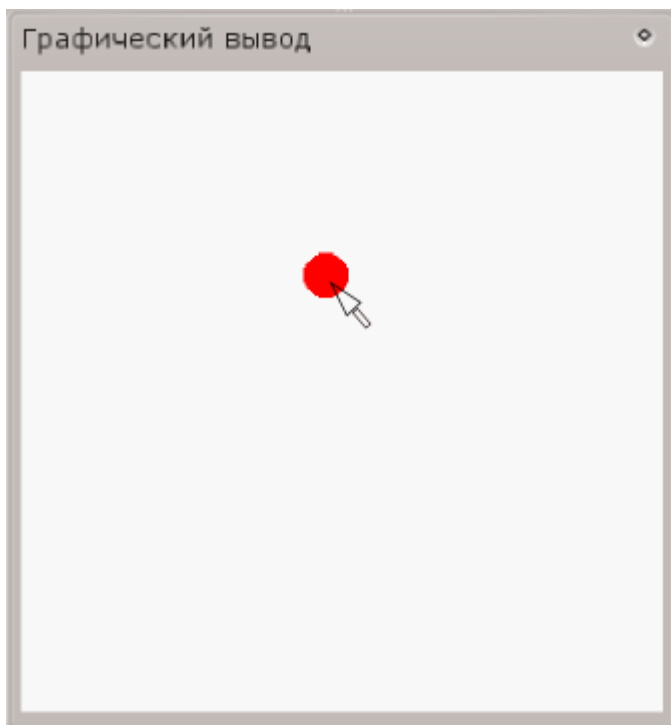
### Режим перемещения

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, ее перемещение не регистрируется, а функции возвращают последнее сохраненное значение.

```
1 # mousetrack.kbs
2 print "Подвигай мышкой по окну графического вывода."
3 print "Щелкни левой кнопкой, чтобы завершить работу."
4
5 fastgraphics
6
7 # Повторять, пока пользователь не нажмет 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.



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.



**Новое  
понятие**

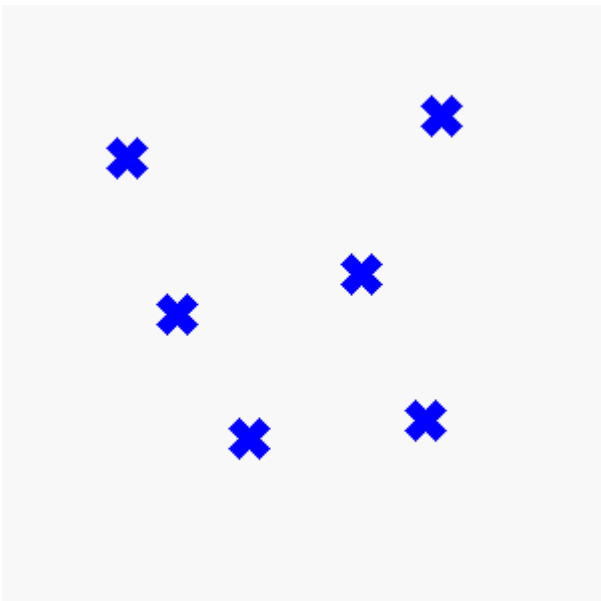
<b>mousex</b>	Возвращает x-координату указателя мышки. Диапазон: от 0 до graphwidth – 1
<b>mousey</b>	Возвращает y-координату указателя мышки. Диапазон: от 0 до graphheight – 1
<b>mouseb</b>	0 Возвращает это значение, когда ни одна кнопка не нажата
	1 Возвращает это значение, когда левая кнопка нажата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

## Режим щелчков

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

```
1 # mouseclick.kbs
2 # X отмечает место где вы щелкнули мышкой
3 print "Подвигайте мышкой в окне графического вывода"
4 print "Щелкните левой кнопкой, чтобы отметить точку"
5 print "Щелкните правой кнопкой, чтобы закончить."
6 clg
7 clickclear
8 while clickb <> 2
9     # стираем значения предыдущего нажатия на кнопку мыши
10    # и ждем 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,
18    2, 0, 1, -1, 2, -2, 1, -1, 0, -2, -1}
19 end while
19 print "all done."
20 end
```

Программа 55 Щелчки мышкой



Пример вывода программы 55 Щелчки мышкой



### Новое понятие

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

Значения этих трех 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 трех 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, зеленый и синий ползунок"
6
7 # Переменные для хранения красной (r), зеленой (g) и синей (b) составляющих
цвета
8 r = 128
9 g = 128
10 b = 128
11
12 gosub display
13
14 while true
15 # ждем 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
```

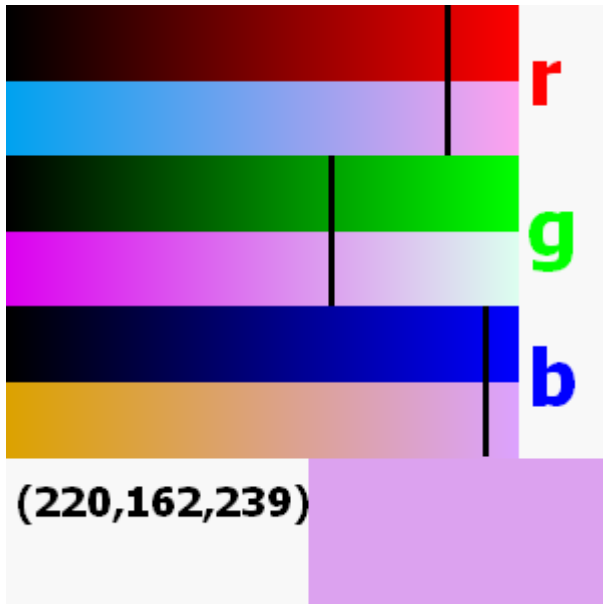
```
28     if mousey >= 150 and mousey < 225 then
29         b = mousex
30         if b > 255 then b = 255
31     end if
32     gosub display
33 end while
34 end
35
36 display:
37 clg
38 # рисуем красным
39 color 255, 0, 0
40 font "Tahoma", 30, 100
41 text 260, 10, "r"
42 for t = 0 to 255
43     color 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 зеленым
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



Образец 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.

### 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 ее значение обнуляется (становится равным нулю), пока не будет нажата еще какая-нибудь клавиша.

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



**Новое  
понятие**

**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)**



**Новое  
понятие**

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)

**Новое  
понятие**

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



**chr**(выражение)

**Новое  
понятие**

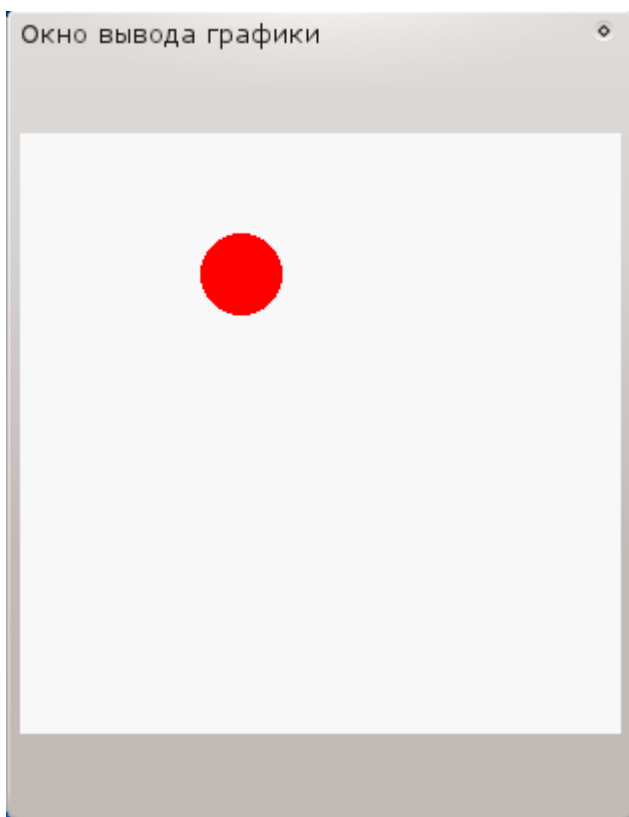
Функция **chr** возвращает строку, содержащую единственный символ, код которого в Юникод-таблице равен целому числу, которому равно «выражение»

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<sup>1</sup>



Примерный вывод программы 58: Двигай мяч



#### **Большая программа**

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.



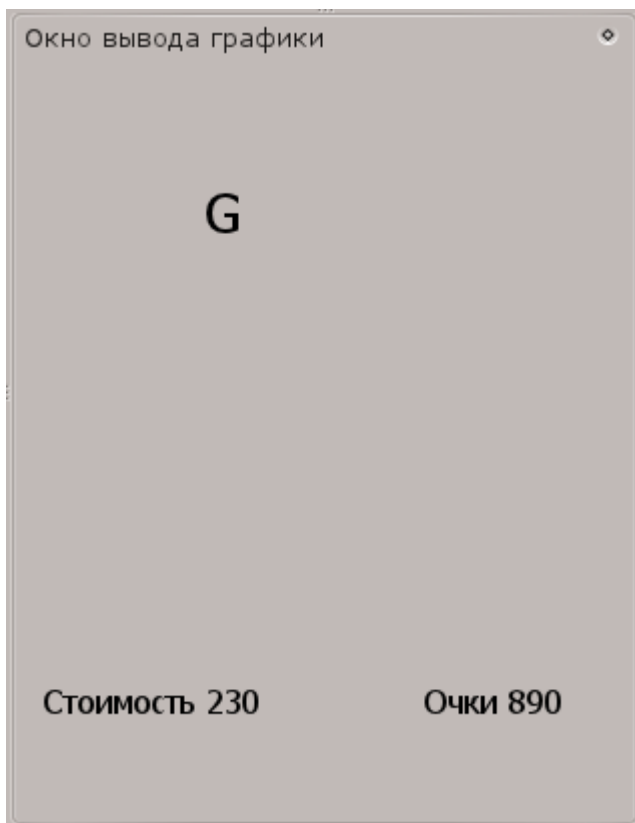
```
1 # fallinglettergame.kbs
2
3 speed = .25 # скорость падения - чем число меньше, тем быстрее
4 nletters = 10 # количество букв в игре
5
6 score = 0
7 misses = 0
8 color black
9
10 fastgraphics
11
12 clg
13 font "Tahoma", 20, 50
14 text 20, 60, "Падающие буквы"
15 font "Tahoma", 18, 50
16 text 20, 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 ждем 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 счет 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<sup>2</sup>



Пример экрана программы 59 Большая программа — Падающие буквы

-----

<sup>1</sup>В данной программе поле представляет собой сферу: когда вы подводите мячик к верхнему краю и нажимаете на "Г" он появляется снизу, аналогично при движении мячика вправо и влево или вниз. Хорошее упражнение — исправить программу так, чтобы поле было прямоугольником со стенками. (прим. переводчика).

<sup>2</sup>В данную программу внесены изменения:

- строка 3: было speed = .15 У меня, на 2 гГц, с таким значением буквы мелькают как сумасшедшие 😊
- строки 18-20 добавлены мной, поскольку программа выдает 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 нее любое нажатие на клавишу (даже неверную) считается попаданием.

(прим. переводчика)

## Глава 12 Картинки, музыка и спрайты

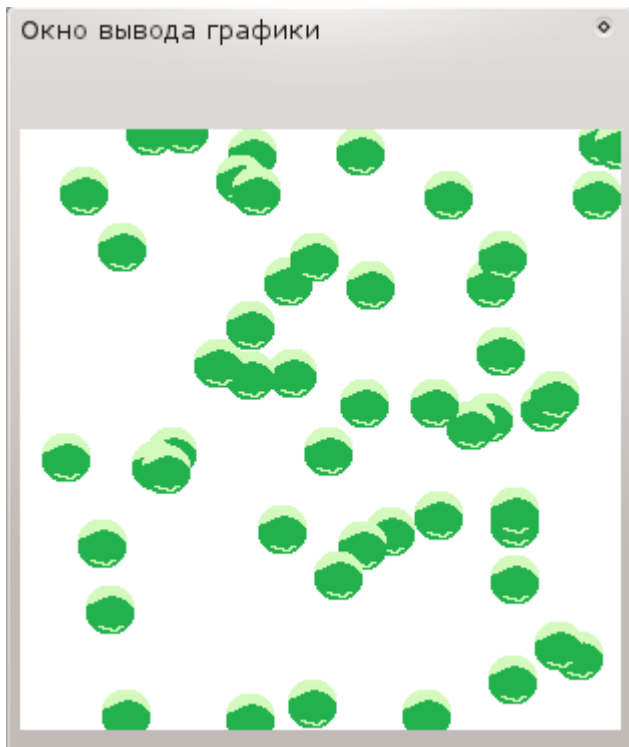
В этой главе мы начинаем изучать управление мультимедийным и графическим содержанием программ. Научимся загружать графические образы (картинки) из файлов, проигрывать асинхронно звуковые файлы типа WAV, а также создавать анимацию (мультфильмы) с использованием спрайтов.

### 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.

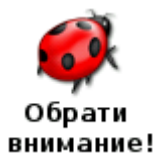
```
1 # imgload_ball.kbs – Демонстрация функции Imgload
2 clg
3 for i = 1 to 50
4   imgload rand * graphwidth, rand * graphheight, "greenball.png"
5 next i
```

Программа 60 Загрузка графического файла.



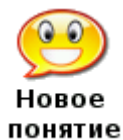
Пример вывода программы 60 Загрузка графического файла

Программа 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 картинке, 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 ее запустите. Это задает текущий рабочий каталог программы так, что BASIC-256 сможет эти файлы найти и загрузить.

**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 ее 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 картинке 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  $2\pi$ )

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

```
1 # imgload_picasso.kbs — Загрузка картинке с вращением и масштабированием
```

```

2 graphsize 500,500
3 clg
4 for i = 1 to 50
5     imgload graphwidth/2, graphheight/2, i/50, 2*pi*i/50,
"picasso_selfport1907.jpg"
6 next i
7 say "Здравствуй, Пикассо."

```

Программа 61 Загрузка картинки с масштабированием и вращением.



Примерный вывод программы 61 Загрузка картинки с масштабированием и вращением.

## 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 # задаем 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

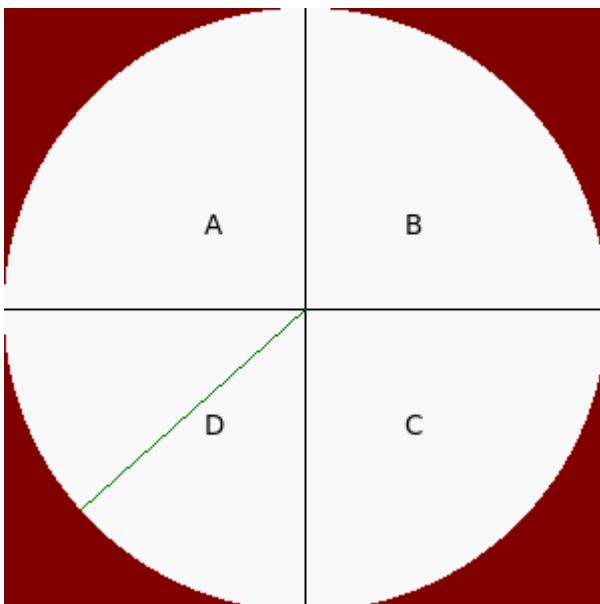
```

```

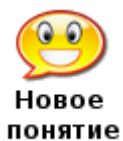
13   color white
14   circle 150,150,150
15   color black
16   line 150,300,150,0
17   line 300,150,0,150
18   text 100,100,"A"
19   text 200,100,"B"
20   text 200,200,"C"
21   text 100,200,"D"
22   color darkgreen
23   line 150,150,150 + cos(angle)*150, 150 + sin(angle)*150
24   refresh
25 # update angle for next redraw
26   angle = angle + speed
27   speed = speed * .9
28   pause .05
29 next t
30
31 # ждем, пока перестанет звучать воспроизводимый звуковой файл.
32 wavwait

```

Программа 62 Вращающийся луч (радар) со звуковым сопровождением.



Примерный вывод программы 62



**Новое  
понятие**

**wavplay** имя\_файла

**wavwait**

**wavstop**

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.

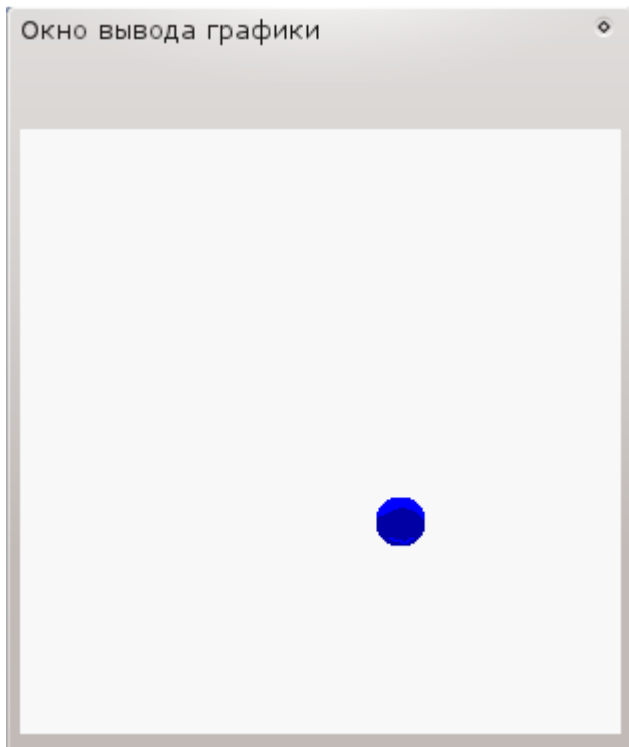
**wavwait** – останавливает исполнение дальнейших операторов программы, пока не закончится воспроизведение текущего звукового файла.

## 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 легкого 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.<sup>1</sup>



Пример экрана программы 63

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 еще функции, позволяющие определить положение спрайта на экране (команды **spritex** и **spritey**), его размеры (команды **spritew** и **spritey**) и его видимость (команда **spritev**).



Новое  
понятие

**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.

**spriteload** номер\_спрайта, имя\_файла



Новое  
понятие

Эта команда загружает графический файл (форматов GIF, BMP, PNG, JPG, или JPEG) с учетом пути (если указан) к файлу, и создает 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**)



Новое  
понятие

**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.



**Новое  
понятие**

**spriteplace** номер\_спрайта, x, y

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

**spitemove** 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.



**Новое  
понятие**

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**.



**Новое  
понятие**

**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)

**spriew**(номер\_спрайта)

**spritex**(номер\_спрайта)

**spritey**(номер\_спрайта)



**Новое  
понятие**

Эти функции возвращают различную информацию о загруженном спрайте  
**spriteh** — возвращает высоту спрайта в пикселях  
**spriew** — 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.

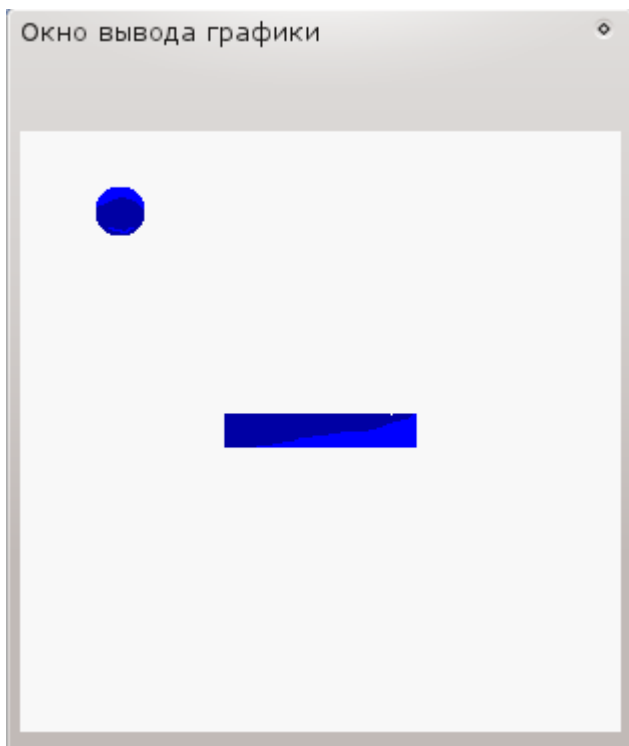
```
1 # sprite_bumper.kbs
2
3 color white
4 rect 0, 0, graphwidth, graphheight
5
6 spritedim 2
7
8 # спрайт-отбойник
9 spriteload 0, "paddle.png"
```

```

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 "Ба-бах!"
30     end if
31     spritemove 1, dx, dy
32     pause .05
33 end while

```

#### Программа 64 Столкновение спрайтов.<sup>2</sup>



Примерный экран программы 64 Столкновение спрайтов



**spritecollide**(номер\_первого, номер\_второго)

**Новое  
понятие**

Эта функция возвращает **true** (истину), если два спрайта столкнулись или один перекрывает другой.<sup>3</sup>



**Большая  
программа**

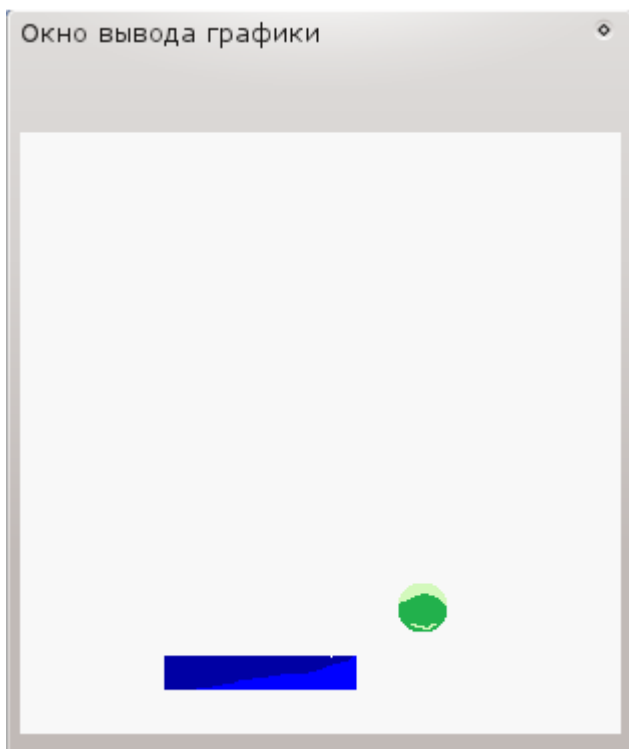
Большая программа данной главы использует спрайты и звуки для создания игры типа пинг-понг.

```
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
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(a)."
```

#### Программа 65. Пинг-понг



Примерный вывод программы 65 Пинг-понг

-----

<sup>1</sup>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 ( $\text{spritew}(0)/2$ ) (прим. переводчика).

<sup>2</sup>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*).

<sup>3</sup>Функция слишком примитивна. На самом деле 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*).

## 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.

### Одномерный числовой массив.

Одномерный массив создает 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)
```



Новое  
понятие

Оператор **dim** создает массив в оперативной памяти компьютера размером равном указанному в скобках и именем `variable` или `variable$` (можно задать свое). 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]
```



Новое  
понятие

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 меньше, чем его размер для последнего, но так принято в программировании. Массивы с отсчетом от нуля – так называют такие объекты программисты.

Мы можем использовать числовые массивы для рисования большого количества одновременно прыгающих на экране мячиков. Программа 66 использует 5 массивов для хранения местоположения каждого из мячей, его направления движения и цвета. С помощью циклов мы задаем начальные значения этим массивам, а также заставляем мячи двигаться. Эта программа также использует функцию **rgb()**, которая определяет и сохраняет цвет каждого мяча.

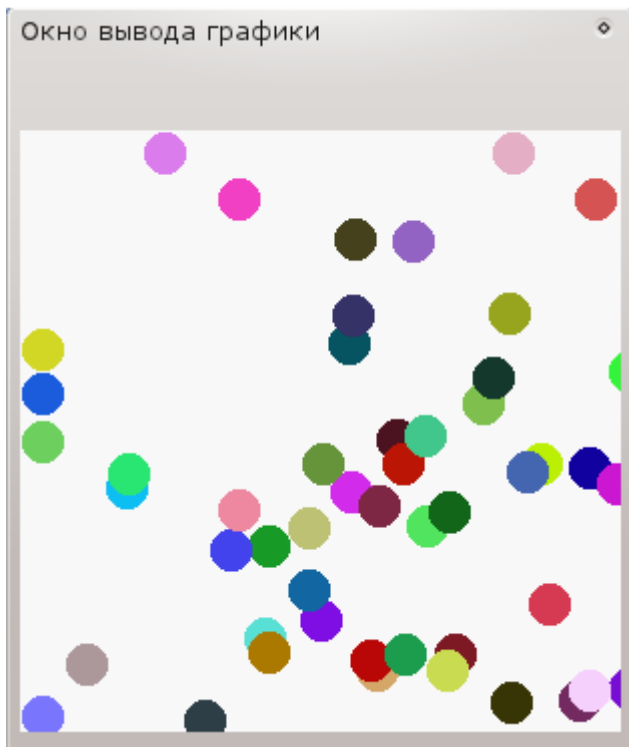
```
1 # manyballbounce.kbs
2 fastgraphics
3
```

```

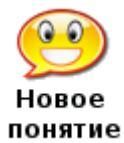
4 r = 10 # размер мяча
5 balls = 50 # количество мячей
6
7 dim x(balls)
8 dim y(balls)
9 dim dx(balls)
10 dim dy(balls)
11 dim colors(balls)
12
13 for b = 0 to balls-1
14     # начальная позиция мяча
15     x[b] = 0
16     y[b] = 0
17     # скорость по осям x и y
18     dx[b] = rand * r + 2
19     dy[b] = 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     clr
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.



Пример экрана программы 67. Много прыгающих мячиков.



**rgb** (красный, зеленый, синий)

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

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

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

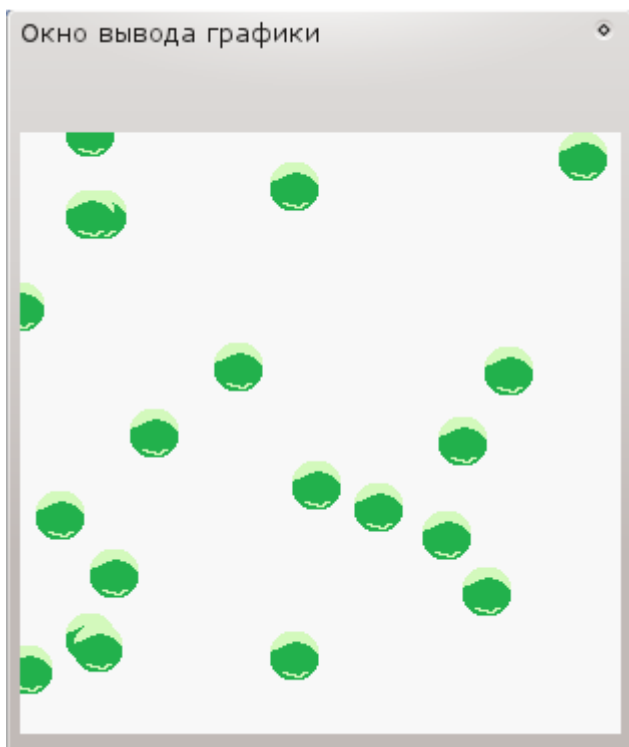


```

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.



Пример вывода программы 68 Много прыгающих мячиков с использованием спрайтов.

## Массивы строк

Массивы могут хранить не только числа, но и строки. Чтобы открыть строковый массив необходимо использовать идентификатор строковой переменной в операторе **dim**. Все, что мы знаем про числовые массивы применимо и к строковым с учетом 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 ?Артем
Напиши имя друга ?Денис
-- очистка экрана --
Мои друзья
Друг номер 1, это Иван
Друг номер 2, это Артем
Друг номер 3, это Денис
```

Пример вывода программы 69 Список друзей.

## 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, заключенных в фигурные скобки. Это применимо как к числовым, так и строковым массивам.

## Звуки и массивы

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

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

```

1 # spacechirp.kbs
2
3 # четные значения 0,2,4... - частота звука
4 # нечетные значения 1,3,5... - длительность
5
6 # чириканье начинается со 100гц и увеличивается на 40 для каждого из 50
звуков в списке, длительность всегда равна 10
7

```

```

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 чирикание



**Пробуй,  
исследуй!**

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

## Графики и массивы

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

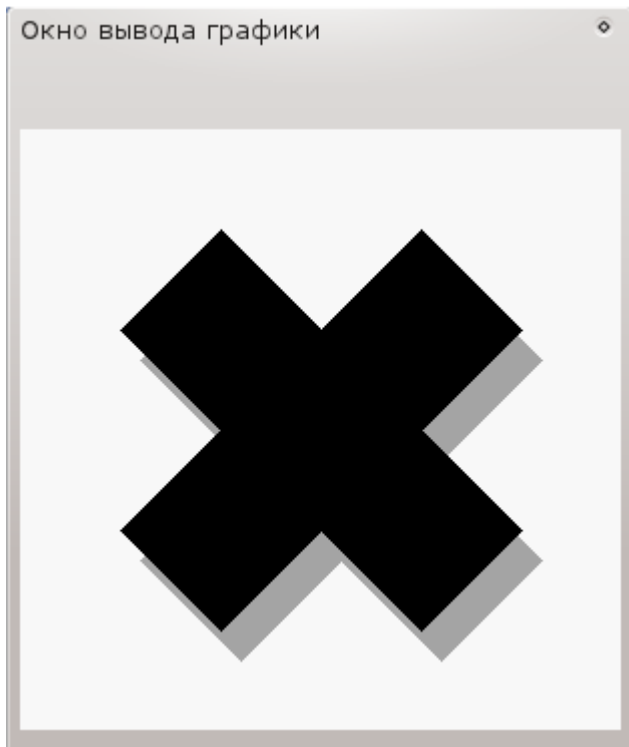
В массиве, используемом для штампов и многоугольников четные элементы (0,2,4,...) содержат абсциссы (x), а нечетные элементы (1,3,5,...) содержат ординату (y) точек, образующих фигуру, по два значения на каждую точку.

```

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

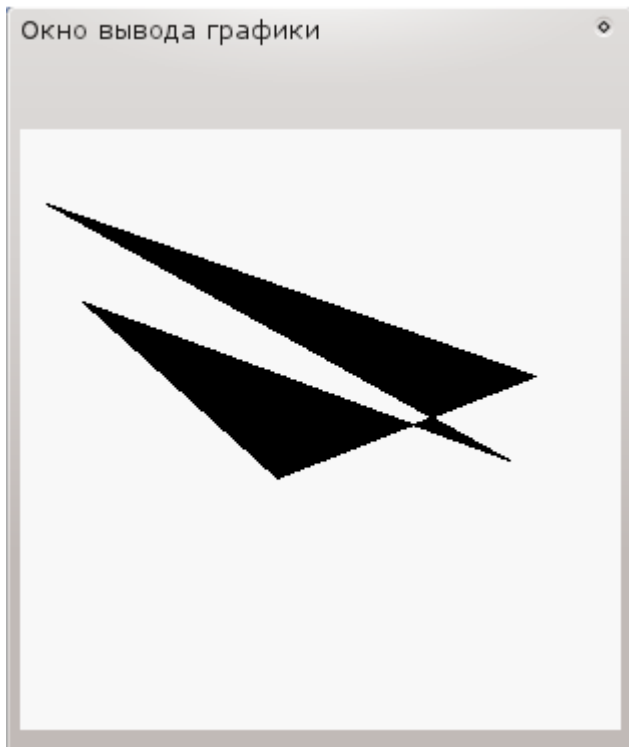


Пример вывода программы 72 Штамп с тенью

Массивы также можно использовать, чтобы создать многоугольник математически (по формулам). В программе 73 мы создаем 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



Примерный вывод программы 73 создание случайного прямоугольника

### 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. 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 к программе 74)

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

Таблица к программе 74<sup>1</sup>

```
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"
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] + " - среднее равно ";
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

## 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 вернет 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
```

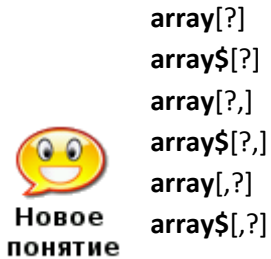
Программа 75 Получение размера массива

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



33,  
3,  
22,

Пример вывода программы 75 Получение размера массива



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.

## 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<sup>2</sup>.

```
1 # redim.kbs
2 dim number(3)
3 number = {77, 55, 33}
4 # создаем 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
12 miny = ballr # минимальное значение координаты y мяча
13 maxy = graphheight - ballr # максимальное значение координаты y мяча
14 score = 0 # в начале счет равен 0
15 playerw = 30 # ширина игрока
16 playerm = 10 # шаг игрока
17 playerh = 10 # высота игрока
18 playerx = (graphwidth - playerw)/2 # начальное значение координаты x
игрока - середина поля
19 keyj = asc("J") # значение для клавиши 'j'
20 keyk = asc("K") # значение для клавиши 'k'
21 keyq = asc("Q") # значение для клавиши 'q'
22 growpercent = .20 # случайное расширение игрока - чем больше, тем быстрее
23 speed = .15 # скорость - чем меньше, тем быстрее
24
25 print "Космический мусор - использовать j и k клавиши чтобы избежать
столкновения с космическим мусором"
26 print "q - закончить"
27
```

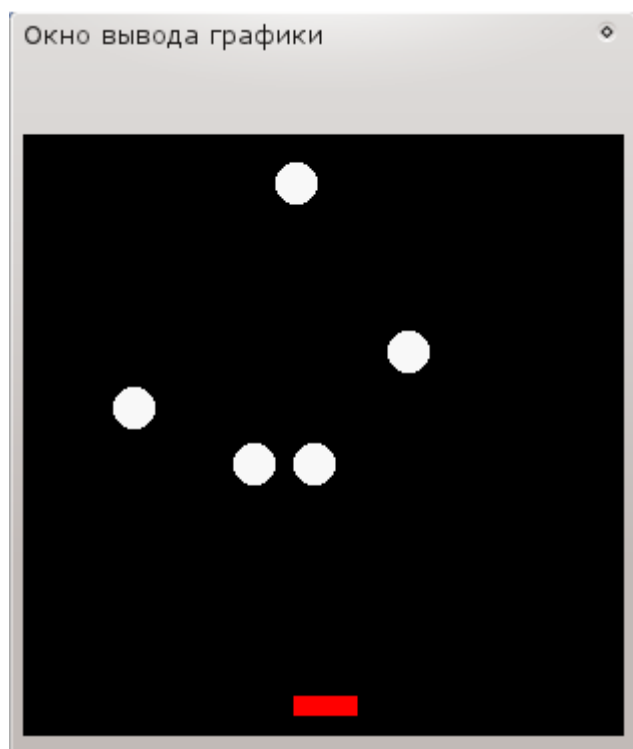
```

28 fastgraphics
29
30 # установка начальных значений скорости и положения мячей
31 for n = 0 to balln-1
32     gosub setupball
33 next n
34
35 more = true
36 while more
37     pause speed
38     score = score + 1
39
40     # очистка экрана
41     color black
42     rect 0, 0, graphwidth, graphheight
43
44     # рисуем мячи и проверяем на наличие столкновений
45     color white
46     for n = 0 to balln-1
47         bally[n] = bally[n] + ballspeed[n]
48         if bally[n] > maxy then gosub setupball
49         circle ballx[n], bally[n], ballr
50         if ((bally[n]) >= (maxy-playerh-ballr)) and ((ballx[n]+ballr) >=
playerx) and ((ballx[n]-ballr) <= (playerx+playerw)) then more = false
51     next n
52
53     # рисуем игрока
54     color red
55     rect playerx, maxy - playerh, playerw, playerh
56     refresh
57
58     # игрок растёт в ширину
59     if (rand<growpercent) then playerw = playerw + 1
60
61     # проверяем нажата ли клавиша и перемещаем игрока, если нажата
62     k = key
63     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 "счет " + 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



Примерный экран программы 77 Большая программа – Игра Космический мусор

-----  
<sup>1</sup>В школах многих зарубежных стран используется 100-бальная система оценок (*прим. переводчика*)

<sup>2</sup>Программа 75 также является демонстрацией динамических массивов. Обратите внимание на оператор **redim** (*прим. переводчика*).

## Глава 14. Математика — развлечемся с числами

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

## Новые операторы

В дополнение к базовым математическим операциям, которые мы использовали с первой главы книги, есть еще 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)

### 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*



**Новое  
понятие**

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) и 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.

## 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*



**Новое  
понятие**

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.

## 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



**Новое  
понятие**

*выражение1* ^ *выражение2*

Оператор степень (^) возводит *выражение1* в степень заданную числом *выражение2*.

Математическая запись степени  $a = b^c$  в BASIC-256 записывается как  $a = b^c$

## 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 меньше 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
	<b>int(выражение)</b>	Преобразует <i>выражение</i> (строку, целое или десятичную дробь) в целое число. При этом дробная часть просто отбрасывается. Если строка не содержит числа возвращается ноль.
Новое понятие	<b>ceil(выражение)</b>	Преобразует <i>выражение</i> в ближайшее большее целое
	<b>floor(выражение)</b>	Преобразует <i>выражение</i> 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
```

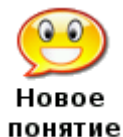
### Программа 82 Различие между *int*, *ceil* и *floor*

```
-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*

## Новые функции для дробных чисел

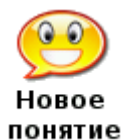
Математические функции, в которые погрузила нас эта глава могут пригодиться при написании некоторых специальных программ. В подавляющем большинстве программ они не нужны.



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

## 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 (0-2π). 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	Описание
<code>cos(выражение)</code>	Возвращает значение косинуса угла
<code>sin(выражение)</code>	Возвращает значение синуса угла
<code>tan(выражение)</code>	Возвращает значение тангенса угла
<code>degrees(выражение)</code>	Преобразует число из радиан (0-2π) в градусы (0° 360°)
<code>radians(выражение)</code>	Преобразует число градусов (0° 360°) в радианы (0-2π)
<code>acos(выражение)</code>	Функция, обратная косинусу (арккосинус)
<code>asin(выражение)</code>	Функция, обратная синусу (арксинус)
<code>atan(выражение)</code>	Функция, обратная тангенсу (арктангенс)

Первые три функции имеют прямое отношение к сторонам прямоугольного треугольника. Рисунок 20 демонстрирует прямоугольный треугольник, его стороны и углы.

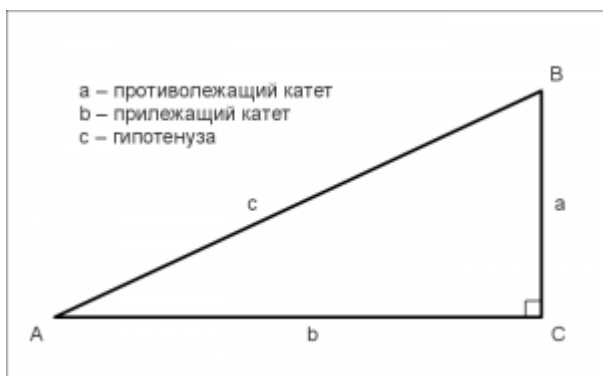


Рисунок 20 Прямоугольный треугольник.

## 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  $2\pi$  радиан.

Рисунок 21 показывает волновой график косинуса в диапазоне от 0 до  $2\pi$ .

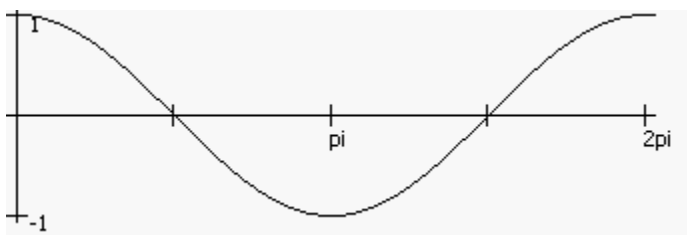


Рисунок 21 Функция cos()

## 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  $2\pi$

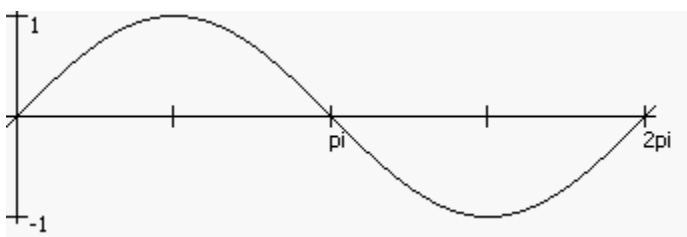


Рисунок 22 Функция sin()

## Тангенс

Тангенс равен отношению противолежащего катета к прилежащему:

$$\operatorname{tg} A = \frac{a}{b}$$

Тангенс повторяет свои значения лежащие в диапазоне от  $-\infty$  до  $\infty$  на каждом отрезке в  $\pi$  радиан. Такой диапазон тангенса объясняется тем, что когда угол треугольника становится очень маленьким, противолежащий катет становится очень маленьким.

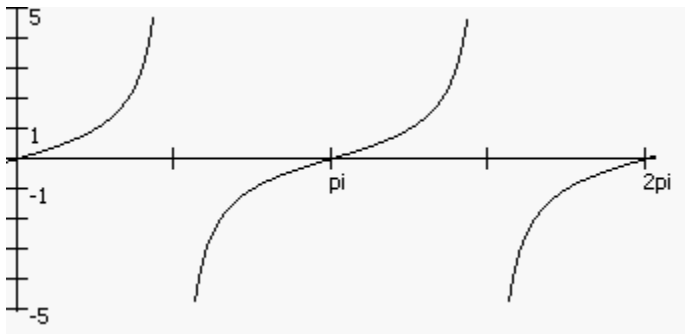


Рис. 23 Функция  $\tan()$

## Функция **degrees**

Функция **degrees()** преобразует значения угла из радиан в градусы по формуле:

$$\text{градусы} = (\text{радианы}/2\pi) * 360.$$

## Функция **radians**

Функция **radians()** преобразует значения угла из градусов в радианы по формуле

$\text{радианы} = (\text{градусы}/360) * 2\pi$  Запомните, все тригонометрические функции BASIC-256 используют в качестве аргумента радианы, а не градусы.

## Аркосинус

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

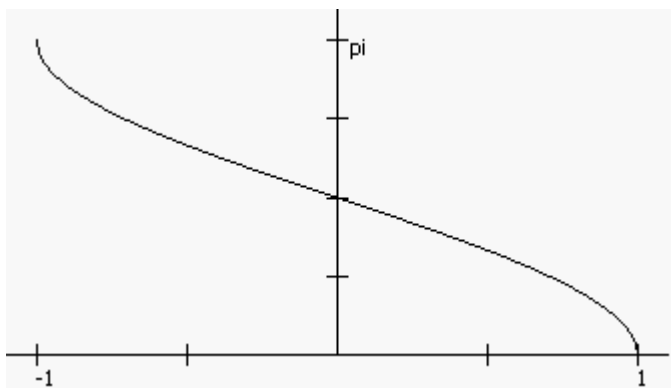


Рисунок 24 Функция  $\operatorname{acos}()$

## Арксинус

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

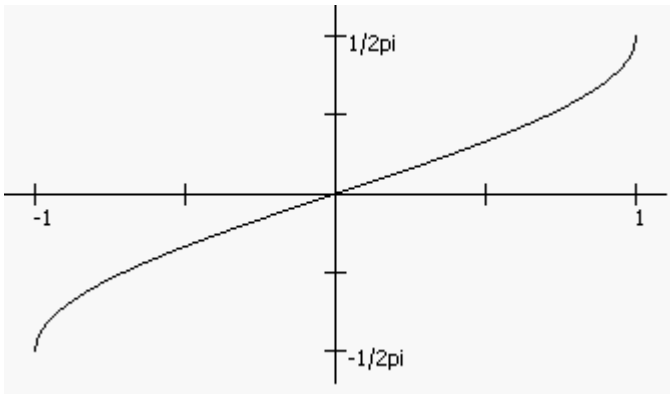


Рисунок 25 Функция asin()

## Арктангенс

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

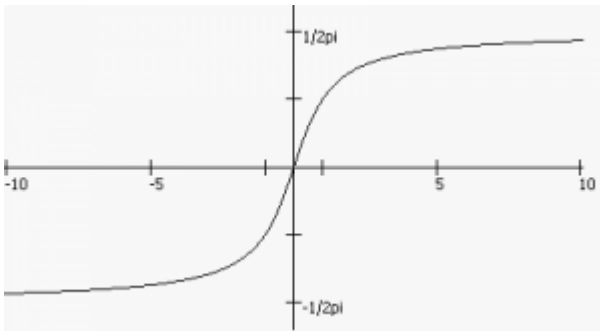


Рисунок 26 Функция atan()



**Большая программа**

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

```
1 # longdivision.kbs
2 # Показывает графически процесс деления уголкоком
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 следующую цифру текущего остатка и убираем ее 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 ее
```

```

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 # передаем 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 drawrighnumber:
100 # передаем 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

```

  001582
78|123456
  0
  12
  0
 123
  78
  454
  390
  645
  624
  216
  156
   60

```

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.

### Строковые функции

BASIC-256 имеет восемь стандартных функций для преобразования строк, которые собраны в таблице 7



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

## Функция `string()`

Функция **`string()`** берет 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
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()`



**Новое  
понятие**

**`string(выражение)`**

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

## 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()



**Новое  
понятие**

**length(выражение)**

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

## Функции left(), right() и mid()

Функции left(), right() и mid() извлекают из строки определенную 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)**

**Новое  
понятие**

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



**right(строка, длина)**

**Новое  
понятие**

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



**mid(строка, начало, длина)**

**Новое  
понятие**

Возвращает подстроку заданной длины переменной *строка* из ее середины. Параметр *начало* указывает, где эта подстрока должна начаться (1 = начало строки)<sup>1</sup>

## 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()`<sup>2</sup>



**lower(строка)**

**upper(строка)**

**Новое  
понятие**

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

## 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()`



**Новое  
понятие**

`instr(строг_сена, иголка)`

Ищет подстроку (*иголка*) в другой строке (*строг\_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 ( $23=2*10^1+3*10^0$ ) станет 43 ( $4*5^1+3*5^0$ ), но будет представлять одно и то же количество предметов. Такое преобразование чисел называется сменой основания системы счисления.



**Большая  
программа**

Компьютер, в своем 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.<sup>3</sup>

```
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

-----

<sup>1</sup>Если сумма параметров *начало+длина* больше, чем длина строки, то вернется

только часть строки от места *начало* до конца строки, что и показано в последнем выводе программы 86. (прим. редактора)

<sup>2</sup>Если у вас эти функции не работают с русскими буквами - обновите BASIC-256 (прим. редактора).

<sup>3</sup>Как известно, в английском алфавите ровно 26 букв. А означает «цифру» 10, В - 12 и т. д. Наиболее известна в среде программистов шестнадцатеричная система счисления, где цифрами от 10 до 15 являются буквы от А до F