И. Г. Ахметов

Программирование для студентов и школьников на примере Small Basic

БАЗОВЫЕ ПОНЯТИЯ ПРОГРАММИРОВАНИЯ

ПРОСТОЙ ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ЯЗЫК

ВОЗМОЖНОСТЬ СОЗДАНИЯ ИНТЕРАКТИВНЫХ ПРОГРАММ

СВОБОДНО РАСПРОСТРАНЯЕМАЯ РУСИФИЦИРОВАННАЯ СРЕДА ПРОГРАММИРОВАНИЯ

ИНФОРМАТИКА И ИНФОРМАЦИОННО-КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ

Ильдар Ахметов

Программирование для студентов и школьников на примере Small Basic

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.068(07)
ББК 32.973.26-018.1я7
А95

Ахметов И. Г.

А95 Программирование для студентов и школьников на примере Small Basic. — СПб.: БХВ-Петербург, 2012. — 160 с.: ил. — (ИиИКТ)

ISBN 978-5-9775-0785-1

Книга предназначена для начинающих программировать школьников и студентов. Материал излагается доступным языком на примерах из повседневной жизни. Раскрыты основные определения: алгоритм, программа, программирование. Рассмотрены базовые понятия языков программирования: объекты, переменные, присваивание, типы данных, ввод/вывод. Разобрана работа условных операторов, циклов, обработка одномерных и двухмерных массивов, математические функции и функции работы со строками. Описывается работа с графикой, анимация, обработка событий. Материал излагается на примере объектно-ориентированного языка свободно распространяемой русифицированной среды Small Basic. В каждом разделе имеются задания для самостоятельного решения.

Для образовательных учреждений

УДК 681.3.068(07)
ББК 32.973.26-018.1я7

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 30.12.11.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 10.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

ISBN 978-5-9775-0785-1

Оглавление

Введение

Итак, вы решили заняться программированием. Но с чего начать? Языков программирования много, и растеряться в море их разнообразия не мудрено. Да и просто изучение синтаксиса какого-либо языка еще не сделает вас программистами. С одной стороны, надо научиться строить алгоритмы, которые, как известно, представляют собой наборы инструкций, описывающих порядок действий для достижения результата за конечное время. А с другой стороны, хочется увидеть результат, причем не на бумаге, а на экране монитора. Поэтому в качестве очень простого в освоении языка с несложной и дружественной средой программирования, на базе которого вы постигнете принципы алгоритмизации и, разумеется, научитесь программировать, и был выбран Small Basic.

А дальше что? Начнем, пожалуй, с установки.

В Интернете существует "Русскоязычное сообщество для начинающих программистов". Это сайт, посвященный Small Basic. Проект поддерживается компанией Microsoft. Найти сообщество очень просто: в любой поисковой системе наберите запрос "Small Basic", и вы увидите результат с веб-адресом **www.smallbasic.ru**. Вам туда!

Отыскать ссылки, позволяющие скачать Small Basic с сайта этого проекта, совершенно легко: слева в окне обозревателя есть ссылка **Скачать**, а также справа среди прочих вкладок присутствует вкладка **СКАЧАТЬ SMALL BASIC**.

На открывшейся странице перечислены системные требования для установки и находится ссылка **Скачать Small Basic 1.0**. Щелчок по этой ссылке открывает окно загрузки дистрибутива

Small Basic. Скачайте его на свой компьютер, а затем запустите установку двойным щелчком по файлу SmallBasic.msi. Скорее всего, вы не первый раз устанавливаете программы, поэтому справитесь с этой задачей и без наших комментариев.

После установки в главном меню Windows появится группа **Small Basic**, из которой и запускается среда разработки.

Хочется добавить, что на сайте сообщества есть много полезной информации, в том числе документация, существует форум и публикуются интересные коды программ. Когда вы более-менее освоитесь с языком Small Basic, посмотрите материалы на сайте.

Ну, а мы начинаем...

Глава 1

Знакомимся с языком

Что такое программирование

Давайте для начала определимся, чем будем заниматься. А будем мы писать программы. Так что с определением слова *"программирование"*, думаю, все должно быть ясно. Как, например, есть еда — и мы ее едим, т. е. уничтожаем. А тут наоборот — мы программируем, т. е. создаем (пишем, придумываем) программы.

Что же такое *программа*? Слово вроде вполне знакомое. Есть программа передач на следующую неделю, программа развития сельского хозяйства России, школьная программа пятого класса. Что общего у этих программ? То, что они задают какой-то план действий, определяют, как и что будет происходить в будущем. Здесь есть, заметим, важное отличие программы от плана. В плане мы предусматриваем, что нечто будет происходить определенным образом, потом делаем и смотрим, что в итоге получилось — насколько близко к плану. В случае с программой такого нет. Программа — понятие очень четкое. Пусть только попробуют изменить внезапно программу передач или не показать передачу "Спокойной ночи, малыши" в нужное время!

Так же и компьютерная программа — штука очень четкая. Программист говорит железной машине, что нужно делать, и железная машина делает. Именно так, как сказал программист. Машина, хоть и умная, но совершенно лишена инициативы. Она как слишком дрессированная собака. Выполняет все команды — "Сидеть", "Лежать", "Дай лапу" — но если на хозяина кто-то нападает, будет сидеть и смотреть, пока не услышит команду "Фас".

> ***Кстати***
>
> К счастью, можно предусмотреть такой случай и заранее проинструктировать собаку: "Если на хозяина напали, надо кусать". Это можно сделать с помощью условного оператора или обработки событий.

Для того чтобы написать программу, нужно сначала придумать *алгоритм*. Алгоритм — это последовательность действий, которые нужно сделать. На самом деле, мы постоянно пользуемся алгоритмами — просто не задумываемся об этом. Вот, напри-

мер, решили мы сварить картошку. Для этого есть четкий порядок действий. Что-то вроде такого:

1. Почистить картошку.
2. Налить воду в кастрюлю.
3. Поставить кастрюлю на плиту.
4. Добавить соль.
5. Включить плиту.
6. Положить почищенную картошку в воду.
7. Варить картошку, пока она не станет мягкой.
8. Выключить плиту.
9. Снять кастрюлю.
10. Слить воду.

Вот это и есть алгоритм. Видите, как все просто — вроде бы всего-навсего сварили картошку, а на самом деле использовали при этом алгоритм!

Программа пишется на основе алгоритма с использованием какого-нибудь *языка программирования*, который понимает ком-

пьютер. Языков программирования много. Люди говорят на русском, английском языке, иврите или африкаансе, а для машин есть языки C, Pascal, Basic, Java, PHP, Perl и т. д. Некоторые похожи друг на друга, некоторые — совсем ни на что не похожи. И точно так же, как человеку совсем не обязательно (да и невозможно!) говорить на языке австралийских аборигенов и при этом в совершенстве владеть эсперанто, так и программист вовсе не обязан знать десятки языков программирования. Достаточно овладеть хотя бы одним (а профессионалу — тремя-пятью).

Мы с вами будем разбираться с языком программирования Small Basic. Это вариант известного языка Basic. Small Basic — очень простой, но при этом современный язык.

Привет, мир!

Есть такая традиция. Когда кто-то начинает изучать программирование, то самая первая программа — это всегда программа "Hello, world", или "Привет, мир". Проще ничего не бывает — программа просто выводит на экран этот текст. Можно вывести и что-нибудь другое, конечно, но не будем нарушать традиций.

> ***Кстати***
>
> В одной книге автор предлагал в первой программе выводить фразу "Наше вам с кисточкой".

Итак, вот она — ваша первая программа:

```
TextWindow.WriteLine("Привет, мир!")
```

Попробуйте ввести эту программу в открытом окне Small Basic и запустить ее (запускается программа большим синим треугольником с подписью "Запуск"). Получилось? Видите черное окно, в котором написано "Привет, мир!"? Тогда поздравляю — первую программу вы написали.

> ***Кстати***
>
> Кроме приветствия, в черном окне вы увидите еще одну строку — "Press any key to continue...". Разумеется, вы знаете, как она переводится. Ну а если забыли, то напомним: "Нажмите любую клавишу для продолжения...".

Основы

Объекты

Что же значит эта строка с непонятными словами? Здесь все довольно просто. `TextWindow` — это объект "окно с текстом", то самое черное окошко, в котором можно отображать текст.

Объект — это нечто, чем вы можете пользоваться. У каждого объекта есть свойства и методы. *Свойство* объекта — это какая-то его характеристика, а *метод* объекта — это то, что объект может делать.

Например, у вас дома есть микроволновка. Это — объект. Свойства микроволновки — цвет (белый, черный, красный, синий в крапинку), объем в литрах (20, 30, 130), название (Samsung, Electrolux, "Лысьва"). Методы микроволновки — разогреть, разморозить, поджарить на гриле.

Так же и здесь. `TextWindow` — объект, а `WriteLine` — его метод, который означает "вывести строку". Точка используется как разделитель. Метод `WriteLine` принимает *параметр* — он же должен знать, что именно надо вывести в черное окно! Параметры всегда указываются в скобках.

Давайте теперь усложним программу. Например, вот так:

```
TextWindow.ForegroundColor = "Red"
TextWindow.WriteLine("Привет, мир!")
```

Теперь "Привет, мир!" написано в черном окне красным цветом — и это все благодаря первой строке. `ForegroundColor` — свойство объекта `TextWindow`, которое обозначает "цвет текста". Мы хотим, чтобы цвет был красным, поэтому и присваиваем этому свойству значение `"Red"` — "красный". Можете попробовать теперь раскрасить строку в другие цвета — синий (`"Blue"`), желтый (`"Yellow"`), зеленый (`"Green"`) и т. д.

Теперь, когда первая программа (из целых двух строк кода!) готова, давайте немного разберемся с теорией.

Переменные

Переменная — самое важное понятие. Представьте себе деревянный ящик. Помните, в каком ящике нашли Чебурашку? Да, вот такой деревянный ящик. Это и есть *переменная*. На ящик приклеена бумажка с названием — это *имя переменной*. В ящик можно что-то класть, а потом вытаскивать — обычно кладут числа и буквы (ну и еще всякие штуки, о которых вы узнаете позже).

Например, давайте создадим переменную с именем `a` и положим туда число 17:

```
a = 17
```

Это значит "положить число 17 в переменную `a`", т. е. "положить число 17 в ящик, на котором написано `a`". Знак равенства в этом случае называется *оператором присваивания*, потому что с его помощью *присваивают* значения переменным. Теперь в ящике с надписью `a` лежит число 17. Давайте теперь напишем вот такую конструкцию:

```
a = a + 5
```

Это не уравнение! Это — присваивание. Вот что значит эта строка:

1. Взять значение переменной `a`.
2. Прибавить к нему 5.
3. Положить новое значение в переменную `a`, стерев из нее предыдущее.

У оператора присваивания есть две части — левая и правая. Левая часть находится слева, а правая — да, вы правильно догадались. В левой части обычно пишется переменная, которая будет меняться. А в правой — то, как она вычисляется. То есть оператор присваивания работает *справа налево*, и только так!

Давайте посмотрим еще раз, что происходит. Итак:

```
a = a + 5
```

1. Начинаем обрабатывать правую часть. Открываем ящик `a`. Там находится число 17, которое мы положили туда раньше. Берем это число из ящика. При этом из ящика вытаскивается только копия значения — другая копия остается лежать в ящике!
2. Теперь в правой части вместо имени `a` подставится число 17, которое мы взяли из ящика `a`. Считаем: 17 плюс 5 — будет 22. Правая часть вычислена, но число 22 еще никуда не записано — в ящике `a` все еще лежит 17.
3. А вот теперь работает присваивание. То есть в ящик `a` (который написан слева) кладется число 22. Старое число 17 из ящика исчезает.

> ***Кстати***
>
> Поэтому и название такое — переменная, потому что значения меняются.

Давайте приведем пример посложнее:

```
b = a * 2 + 10
```

Это значит "взять число 17 из ящика `a`, умножить его на 2, прибавить к нему 10, а затем положить в ящик `b`". Обратите внимание — ящика `b` вообще не существовало, а теперь он появился, и в нем лежит число 44 (т. е. 17 × 2 + 10).

А вот еще один пример:

```
a = 5
b = 8
a = a + b
b = b - 1
a = b
```

В этой программе пять строк. Посмотрим, что происходит (табл. 1.1).

Таблица 1.1

Действие	**a**	**b**
В самом начале	Ничто	Ничто
После 1-й строки (`a = 5`)	5	Ничто
После 2-й строки (`b = 8`)	5	8
После 3-й строки (`a = a + b`)	13	8
После 4-й строки (`b = b - 1`)	13	7
После 5-й строки (`a = b`)	7	7

Посмотрите еще раз: переменные, которые находятся в левой части оператора присваивания, не меняются. Меняются только те, что стоят справа.

И еще один важный момент. Обратите внимание на последнюю строку (`a = b`). Она не делает переменные `a` и `b` синонимами, т. е. не перевешивает таблички с названиями с одного ящика на другой. Просто два разных ящика содержат одинаковое значение.

Типы данных

Переменные могут быть разных типов. Тут снова хорошо работает сравнение с ящиками. Представьте себе ящик для апельсинов, футляр для гитары, коробочку для обручального кольца. Все они нужны для того, чтобы что-то в них класть. Но гитара не влезет в коробочку для кольца, а кольцо затеряется в ящике из-под апельсинов.

Кстати

Представьте себе глаза той девушки, которой преподнесут кольцо в деревянном ящике.

Так же и с переменными — не все они одинаковые, отличает их *тип данных*.

Часто типы данных вызывают сложности при изучении программирования. Зачастую языки имеют очень много разных типов, и разобраться в них довольно трудно.

В языке Small Basic типов данных всего два:

- число;
- строка.

Что такое число, всем понятно. Примеры чисел: 2, –17, 0, 3,14. Числа можно складывать, умножать, вычитать, делить. Над ними можно совершать все математические действия — вычислять логарифмы, синусы, косинусы и т. д.

Строка — это последовательность символов. Примеры строк: "собака", "Мама мыла раму", "Съешь еще этих мягких французских булок, да выпей же чаю", "A8bfhGGT71bHtd71vfa". Строки можно склеивать и делить на части, в них можно искать символы и заменять их другими.

Типы в Small Basic задаются косвенно. То есть вам не нужно описывать типы, как во многих других языках программирования. Вы просто пишете:

```
k = 5
s = "It is raining cats and dogs"
```

И Small Basic понимает, что тип переменной `k` — число, а `s` — строка.

Интересный момент есть с оператором `+`. Для чисел он означает сложение, а для строк — склеивание. Но если "сложить" число со строкой — они тоже будут склеены (табл. 1.2).

Таблица 1.2

Выражение	Результат
`2 + 2`	`4`
`"око" + "рок"`	`"окорок"`
`"абв" + 10`	`"абв10"`

Ввод и вывод

Каждая программа должна делать что-то полезное. Обычно программы берут какие-нибудь данные *(входные данные)*, обрабатывают их и предоставляют результат *(выходные данные)*.

Вот, например, приходит человек на вокзал, подходит к справочному окну и спрашивает: "Сколько стоит билет на поезд до Таганрога в плацкартный вагон?" Ему отвечают: "2312 рублей". Здесь входные данные — название города (Таганрог) и тип вагона (плацкартный). Выходные данные — цена. Заметьте, что значение на выходе напрямую зависит от значений входных параметров.

Кстати

Мы-то с вами, конечно, знаем, что вместо справочной службы есть **www.rzd.ru**.

Соответственно, программа должна как-то получать входные данные и выдавать выходные. Для этого есть специальные *операторы ввода* и *вывода*. Эти операторы — методы объекта `TextWindow` (ведь работают они внутри "черного окна").

Есть два оператора ввода, в зависимости от типа данных (табл. 1.3).

Таблица 1.3

Метод	Описание
`TextWindow.Read()`	Ввод строки
`TextWindow.ReadNumber()`	Ввод числа

В конце оператора ввода всегда ставятся пустые скобки.

Операторы вывода не зависят от типа данных, но их тоже два (табл. 1.4).

Таблица 1.4

Метод	Описание
`TextWindow.Write(data)`	Обычный вывод
`TextWindow.WriteLine(data)`	Вывод с переходом на следующую строку экрана

Разница между двумя вариантами вывода представлена в табл. 1.5.

Таблица 1.5

Пример использования метода	Результат
`TextWindow.Write("око")` `TextWindow.Write("рок")`	`окорок`
`TextWindow.WriteLine("око")` `TextWindow.WriteLine("рок")`	`око` `рок`

Вот пример простой программы с вводом и выводом:

```
a = TextWindow.ReadNumber()
b = TextWindow.ReadNumber()
x = a + b
TextWindow.WriteLine(x)
```

Первая строка — ввод переменной `a`. Посмотрите: это такой же оператор присваивания, который мы уже знаем. В левой части — переменная `a`, в которую будем записывать значение. А в правой части — оператор ввода `ReadNumber()`, который будет ждать, какое число мы введем с клавиатуры. То есть в черном окне начинает мигать курсор, мы вводим какое-нибудь число, нажимаем клавишу <Enter> — и это число кладется в переменную `a`. Точно так же работает вторая строка — следующее введенное число окажется в переменной `b`. Третья строка складывает значения переменных `a` и `b`, а результат записывает в переменную `x`. Ну и, последняя строка выводит значение `x` на экран.

Давайте теперь снабдим эту программу подсказками, чтобы было понятно, что за волшебную махинацию она производит:

```
TextWindow.Write("Введите первое число: ")
a = TextWindow.ReadNumber()
TextWindow.Write("Введите второе число: ")
b = TextWindow.ReadNumber()
x = a + b
TextWindow.WriteLine("Сумма чисел равна: " + x)
```

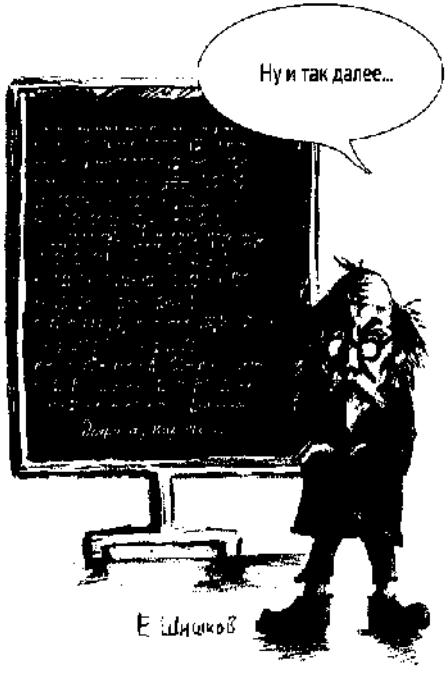
Теперь программа ведет себя гораздо вежливее. Она просит ввести первое число и призывно мигает курсором. Получив число, так же просит ввести второе. Затем считает сумму и пишет, чему она равна. Обратите внимание на знак "+" в последней строке — он склеивает фразу "Сумма чисел равна: " со значением переменной `x`, поэтому получается что-то вроде "Сумма чисел равна: 4".

Задания

1. Вычислите сумму не двух, а трех чисел.
2. А теперь произведение двух (или трех) чисел.
3. Напишите программу для перевода дней в часы (один день — 24 часа).
4. Аналогично напишите программу для конвертации цены из долларов в рубли (курс посмотрите в Интернете).
5. Даны стороны прямоугольника. Нужно найти его площадь.
6. Возведите число в восьмую степень за три операции умножения.

Математические функции

Куда ж без математики! Часто нужно что-то вычислять — синусы, косинусы, логарифмы... Все эти вычисления реализуются с помощью математических функций. В языке Small Basic эти функции находятся в объекте `Math`. Строго говоря, это методы объекта `Math`.



Вот такие математические функции есть в Small Basic (табл. 1.6).

Таблица 1.6

Метод	**Описание**
`Math.Abs(number)`	Модуль
`Math.ArcCos(number)`	Арккосинус
`Math.ArcSin(number)`	Арксинус
`Math.ArcTan(number)`	Арктангенс
`Math.Ceiling(number)`	Округление вверх
`Math.Cos(number)`	Косинус
`Math.Floor(number)`	Округление до ближайшего наименьшего целого
`Math.GetDegrees(angle)`	Перевод числа из радиан в градусы
`Math.GetRadians(angle)`	Перевод числа из градусов в радианы
`Math.GetRandomNumber(maxNumber)`	Случайное число в интервале от 1 до `maxNumber`
`Math.Log(number)`	Десятичный логарифм
`Math.Max(number1,number2)`	Максимум двух чисел
`Math.Min(number1,number2)`	Минимум двух чисел
`Math.NaturalLog(number)`	Натуральный логарифм
`Math.Pi()`	Число π
`Math.Power(baseNumber,exponent)`	Возведение числа `baseNumber` в степень `exponent`
`Math.Remainder(dividend,divisor)`	Остаток от деления
`Math.Round(number)`	Обычное округление
`Math.Sin(number)`	Синус
`Math.SquareRoot(number)`	Квадратный корень
`Math.Tan(number)`	Тангенс

Вот несколько примеров записи математических формул на Small Basic:

$a = x^3$
```
a = Math.Power(x,3)
```

$b = \sin x$
```
b = Math.Sin(x)
```

$c = \cos x^2$
```
c = Math.Cos(Math.Power(x,2))
```
или
```
c = Math.Cos(x*x)
```

$d = \mathrm{tg}^2 x$
```
d = Math.Power(Math.Tan(x),2)
```

$a = \ln\ x + y$
```
a = Math.NaturalLog(x+y)
```

$b = \sqrt{x}$
```
b = Math.SquareRoot(x)
```
или
```
b = Math.Power(x,1/2)
```

$c = \sqrt[3]{x}$
```
c = Math.Power(x,1/3)
```

$d = \dfrac{\sin x^2}{\ln x} + 2\pi$
```
d = Math.Sin(Math.Power(x,2)) /
Math.NaturalLog(x) + 2 * Math.Pi
```

Задания

1. Даны два катета прямоугольного треугольника. Найдите его гипотенузу (это теорема Пифагора, разумеется).
2. Даны длины трех сторон треугольника (уже произвольного). Найдите его площадь (по формуле Герона).
3. Найдите корни квадратного уравнения, если известны его коэффициенты *a*, *b*, *c*. Формулы — стандартные, через дискриминант.
4. Дано трехзначное число. Нужно найти его первую цифру (вам помогут деление и округление).
5. Дано любое целое число. Найдите его последнюю цифру (с помощью остатка от деления).
6. Вычислите вот такое сложное выражение: $x = \dfrac{b^5 - a + \sqrt[3]{bc^2}}{\sqrt{ab}}$.
7. И вот такое: $y = \sin\ 3\pi a + bc\ - \cos a^2$.

Глава 2

Усложняем код

Условный оператор

В жизни постоянно возникают ситуации, которые требуют выбора. Например, собираетесь вы утром выходить на улицу и думаете: "Что же надеть?" Прежде всего, это зависит от погоды. Если льет дождь, то желательно надеть резиновые сапоги и взять зонтик. Если валит снег, то нужны валенки и шапка-ушанка. Ну а если нет ни снега, ни дождя — тогда просто надеваем какую-нибудь обычную одежду.

Смысл условного оператора именно в этом слове — *"если"*. В языке Small Basic (да и в большинстве других языков) это слово пишется просто — *If*.

Если условие истинно (т. е. правда), то оно примет значение "Истина" или `"True"`. Если условие ложно (т. е. совсем даже не правда), то это будет "Ложь" — `"False"` (обратите внимание — кавычки важны!). То есть условие "Париж — столица Франции" истинно (`"True"`), а "Рим — столица Камбоджи" — ложно (`"False"`).

В самом простом варианте условный оператор записывается так:

```
If <условие> Then
  <действия>
EndIf
<продолжение программы>
```

Работает он следующим образом:

1. Вычисляется условие.
2. Если оно истинно, то выполняются действия, а затем — продолжение программы.
3. Если оно ложно, то действия игнорируются, а сразу выполняется продолжение.

Давайте рассмотрим пример. Тот же самый, про погоду, но пока сделаем его попроще, с единственным условием — есть дождь или нет. Запишем это условие в виде псевдопрограммы.

> ***Кстати***
>
> Мы будем пользоваться таким приемом еще не раз. Псевдокод, конечно, нельзя запускать — он нужен для того, чтобы понимать суть.

```
If идет_дождь Then
  взять_зонт
EndIf
выйти_из_дома
```

Видите: все понятно, эту программу очень легко перевести с компьютерного языка на человеческий (т. е. с Бейсика на русский). "Если идет дождь, то взять зонт".

А теперь давайте усложнять. Если условий несколько, то условный оператор записывается так:

```
If <условие 1> Then
  <действия 1>
ElseIf <условие 2> Then
  <действия 2>
...
Else
  <действия, если все условия ложны>
EndIf
```

Слово `ElseIf` (пишется слитно) означает "иначе если", а слово `Else` — просто "иначе". Только самое первое условие записывается в виде `If` (если). Все остальные условия — это уже `ElseIf`. А уж если ни одно из условий не выполнено (все оказалось ложью — "Все врут!"), то срабатывает вариант "иначе" — `Else`.

Вернемся к нашему исходному погодному примеру и запишем его в виде псевдопрограммы.

```
If идет_дождь Then
  взять_зонт
  надеть_сапоги
ElseIf идет_снег Then
  надеть_валенки
  надеть_шапку
Else
  надеть_обычную_одежду
EndIf
выйти_из_дома
```

Посмотрите внимательно на пример. На русский она переводится совершенно очевидно, достаточно перевести ключевые слова. `If` — "если", `Then` — "то", `ElseIf` — "иначе если", `Else` — "иначе".

А вот как компьютер будет разбирать эту программу:

1. Проверить, идет ли дождь.
2. Если дождь идет (условие истинно), то взять зонт и надеть сапоги. Выйти из дома (выход из условного оператора — другие условия уже не проверяются).

3. Если дождя нет (предыдущее условие ложно), то проверить, идет ли снег.
4. Если снег идет (условие истинно), то надеть валенки и шапку. Выйти из дома (выход из условного оператора).
5. Если ни дождя, ни снега нет (все условия ложны), то надеть обычную одежду. Выйти из дома.

Приведем теперь несколько работающих примеров — уже не псевдокод, а настоящие работающие программы. Попробуйте их написать и запустить в Small Basic.

```
TextWindow.Write("Введите число: ")
a = TextWindow.ReadNumber()
If a > 0 Then
  TextWindow.WriteLine("Число положительное")
EndIf
```

Эта программа просит нас ввести число и проверяет, положительное ли оно. Конструкция простейшая. Если условие `a > 0` истинно (т. е. если значение переменной `a` больше нуля), то на экран выводится строка "Число положительное".

А как же узнать, что число отрицательное или равно нулю? Давайте немного усложним программу.

```
TextWindow.Write("Введите число: ")
a = TextWindow.ReadNumber()
If a > 0 Then
  TextWindow.WriteLine("Число положительное")
ElseIf a < 0 Then
  TextWindow.WriteLine("Число отрицательное")
Else
  TextWindow.WriteLine("Ноль")
EndIf
```

Теперь сначала проверяется условие положительности — `a > 0`. Если оно истинно, то выводится фраза "Число положительное", и на этом проверки заканчиваются. Если же оно ложно, то проверяется следующее условие — `a < 0`. Если оно истинно, то выводится фраза "Число отрицательное". Если же оба условия оказа-

лись ложными, т. е. число и не больше нуля, и не меньше нуля, то остается только один вариант: это самый настоящий ноль! Соответственно, и выводится "Ноль".

А вот еще один любопытный пример. Помните нашу первую программу — "Привет, мир!"? Давайте сделаем ее более доброжелательной. Пусть она здоровается с миром по-разному в зависимости от времени суток — "Доброе утро", "Добрый день" или "Добрый вечер". Для этого воспользуемся объектом `Clock` (Часы), у которого есть свойство `Hour` (Час). `Clock.Hour` скажет нам, сколько сейчас часов.

> ***Кстати***
>
> Можно, конечно, здороваться, как в фильме "Шоу Трумана" — "Доброе утро, и, если не увидимся, добрый день и добрый вечер!"

```
If Clock.Hour < 12 Then
  TextWindow.WriteLine("Доброе утро, мир!")
ElseIf Clock.Hour < 18 Then
  TextWindow.WriteLine("Добрый день, мир!")
Else
  TextWindow.WriteLine("Добрый вечер, мир!")
EndIf
```

Посмотрите, если на часах меньше 12, то мы скажем миру "Доброе утро", если меньше шести вечера (т. е. 18), то "Добрый день", а иначе — "Добрый вечер".

Операторы сравнения

В условиях используются *операторы сравнения*, с помощью которых можно сравнивать значения. Мы уже пользовались оператором < (меньше), когда сравнивали `a` и 0.

А вот в табл. 2.1 представлен полный список операторов сравнения.

Таблица 2.1

Оператор	Описание	Пример
<	Меньше	`2 < 5`
>	Больше	`8 > 3`
<=	Меньше или равно	`3 <= 7, 7 <= 7`
>=	Больше или равно	`5 >= 1, 5 >= 5`
=	Равно	`6 = 6`
<>	Не равно	`2 <> 3`

Обратите внимание на важный момент. Символ = может означать как присваивание, так и сравнение. Все зависит от того, в каком месте программы он находится.

Например:

- `a = 2` — присваивание (кладем число 2 в переменную `a`);
- `If a = 2` — сравнение (проверяем, равно ли значение переменной `a` числу 2).

Логические операторы

С помощью *логических операторов* можно строить более сложные условия. Логических операторов в Small Basic два (табл. 2.2).

Таблица 2.2

Название	Оператор	Описание
И	`And`	Истина, если оба условия истинны
ИЛИ	`Or`	Истина, если хотя бы одно из условий истинно

Приведем пару примеров с логическими операторами.

"Если медведь серый **И** летает, то он тучка" (оба условия должны быть истинны: если не серый, но летает — не тучка; если серый, но по земле ходит — тоже).

```
If серый And летает Then
  тучка
EndIf
```

"Если повысилась шерстистость **ИЛИ** отваливается хвост, то надо сходить к ветеринару" (хотя бы одно условие должно быть истинным: если просто отваливается хвост, а с шерстистостью все в порядке, все равно надо идти к доктору, а если и то, и другое — тем более).

```
If шерстистость_повысилась Or хвост_отваливается Then
  идти_к_ветеринару
EndIf
```

А вот пример с числами. Двойное неравенство `1 < x < 100` означает, что "`x` больше нуля **И** `x` меньше десяти":

```
If x > 1 And x < 100 Then
  TextWindow.WriteLine("Число от 1 до 100")
EndIf
```

Или, скажем, условие "сегодня пятница тринадцатое" (снова пользуемся объектом `Clock`):

```
If Clock.WeekDay = "пятница" And Clock.Day = 13 Then
  TextWindow.WriteLine("Сегодня пятница тринадцатое!")
EndIf
```

Более сложный пример с календарем — проверяем, отдыхать сегодня или учиться. Отдыхаем мы в выходные и на каникулах, поэтому если сейчас июль или август[1] или же любое воскресенье, то отдыхаем, иначе — учимся:

```
If Clock.Month = 7 Or Clock.Month = 8 Or Clock.WeekDay =
"воскресенье" Then
  TextWindow.WriteLine("Отдыхаем")
Else
  TextWindow.WriteLine("Учимся")
EndIf
```

Задания

1. Даны два числа. Определите, сколько из них положительных.
2. То же самое, только для трех чисел.
3. Даны три числа. Найдите самое большое среди них.
4. Даны три числа. Найдите среднее из них (т. е. то, которое расположено между наименьшим и наибольшим).
5. Даны три числа. Нужно выяснить, существует ли треугольник, длины сторон которого равны этим числам.
6. Дано число от 1 до 999. В зависимости от количества знаков надо вывести либо "Однозначное число", либо "Двухзначное число", либо "Трехзначное число".
7. Дано трехзначное число. Проверьте, является ли оно палиндромом (т. е. читается ли одинаково слева направо и справа налево).

[1] Школьники могут добавить в условие еще и июнь.

Циклы

Циклы нужны, когда требуется выполнить какие-то действия несколько раз подряд.

Например, рассмотрим задачу "подняться по лестнице" — попытаемся сформулировать алгоритм ее решения. Конечно, мы не задумываемся, как подниматься по лестнице, а просто шагаем по ступенькам. Но и сороконожка не задумывалась, как ходит, пока ее не спросили об этом. Вот мы и поразмыслим — как же правильно подниматься по лестнице?

Здесь есть два варианта.

- Мы знаем количество ступеней, например, 10. Тогда нам нужно шагнуть *ровно* 10 раз — и мы придем к вершине (цикл `For`).
- Мы не знаем количество ступеней. Тогда нам нужно шагать, *пока* впереди есть ступени. И как только ступени закончатся — больше не шагать (цикл `While`).

Цикл *For* ("для каждого")

Цикл `For` служит для того, чтобы выполнить какое-то действие ровно *N* раз. Он нужен тогда, когда мы точно знаем, сколько раз следует выполнить действие. Цикл `For` записывается вот так:

```
For <счетчик цикла> = <от> To <до>
  <действия>
EndFor
```

Счетчик цикла — это переменная, которая "пробегает" значения от начального до конечного. На каждом проходе цикла выполняются определенные действия.

Давайте напишем псевдокод для задачи "подняться по лестнице, у которой 10 ступеней":

```
For i = 1 To 10
  шагнуть на i-ю ступень
EndFor
```

Здесь в качестве счетчика выступает переменная `i` (обычно счетчик обозначают именно этой буквой). Этот цикл выполнится ровно 10 раз, причем на каждом проходе цикла переменная `i` будет увеличиваться на единицу. Таким образом, сначала мы шагнем на 1-ю ступень, потом на 2-ю и т. д. После каждого шага обязательно проверяется, не дошли ли мы до конца. Если дошли, то дальше не идем и из цикла выходим. Это будет выглядеть вот так, как представлено в табл. 2.3.

Таблица 2.3

Чему равно `i`	**Что происходит**	**Проверка**
1	Шагнуть на 1-ю ступень	До 10 не дошли, идем дальше
2	Шагнуть на 2-ю ступень	До 10 не дошли, идем дальше
...	...	...
10	Шагнуть на 10-ю ступень	Дошли до 10, дальше не идем, выход из цикла

Давайте теперь напишем работающую программу. Пусть она выводит на экран числа от 1 до 10.

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

Все просто — на каждом шаге цикла будет выводиться текущее значение переменной `i`. Сначала 1, затем 2, потом 3 и т. д. до 10.

Чтобы шаг цикла был равен не единице, а чему-то другому, используется ключевое слово `Step`. Например, вот такая программа выведет числа 1, 1,5, 2, 2,5, ..., 9,5, 10:

```
For i = 1 To 10 Step 0.5
  TextWindow.WriteLine(i)
EndFor
```

А для того чтобы идти в обратном порядке, используется отрицательный шаг. Следующая программа выведет числа 10, 9, 8, ..., 2, 1:

```
For i = 1 To 10 Step -1
  TextWindow.WriteLine(i)
EndFor
```

Теперь давайте не просто выведем числа от 1 до 10, а посчитаем их сумму (это, кстати, будет 55).

```
s = 0
For i = 1 To 10
  s = s + i
EndFor
TextWindow.WriteLine(s)
```

Посмотрим внимательно, как работает эта программа. Подсчитать сумму сразу мы не можем — мы ее накапливаем в переменной `s`. Считаем последовательно. Начинаем с нуля (`s = 0`), потом прибавляем 1, затем — 2, потом — 3 и т. д. В итоге за 10 шагов цикла мы прибавим к сумме все числа по очереди. Как это будет происходить, представлено в табл. 2.4.

Таблица 2.4

Шаг цикла	**i**	**s**
До цикла	Ничто	0
1-й	1	1 (0 + 1) Предыдущая сумма — 0 плюс текущее число — 1
2-й	2	3 (1 + 2) Предыдущая сумма — 1 плюс текущее число — 2
3-й	3	6 (3 + 3) Предыдущая сумма — 3 плюс текущее число — 3
4-й	4	10 (6 + 4) Предыдущая сумма — 6 плюс текущее число — 4
5-й	5	15 (10 + 5) Предыдущая сумма — 10 плюс текущее число — 5
...	...	...
10-й	10	55 (45 + 10) Предыдущая сумма — 45 плюс текущее число — 10

Таким образом, после 10 шагов цикла в переменной `s` будет лежать как раз сумма всех 10 чисел. Ее и выводим на экран.

Аналогичным образом можно найти и произведение чисел:

```
p = 1
For i = 1 To 10
  p = p * i
EndFor
TextWindow.WriteLine(p)
```

Кстати

Такое произведение называется факториалом.

Заметили, что здесь мы начинаем не с нуля, а с единицы? Вычисление произведения так и нужно начинать. Если бы мы нача-

ли с нуля, то и все произведение осталось бы нулем. Ведь что на ноль не умножай...

Задания

1. Известна цена одного килограмма конфет. Нужно вывести стоимость 1, 2, ..., 10 кг конфет в виде "*n* кг конфет стоят *k* рублей".
2. Теперь выведите стоимость 100, 200, ..., 900 грамм конфет.
3. Дано целое число *n*. Нужно вывести все степени двойки: 1, 2, 4, ..., 2^n.
4. Даны два целых числа — *a* и *b*. Найдите сумму квадратов всех чисел от *a* до *b* включительно.
5. Дано целое число *n*. Найдите сумму только четных чисел от 0 до *n*.
6. Вычислите выражение $\sqrt{2+\sqrt{2+...+\sqrt{2}}}$ (*n* корней).

Цикл *While* ("до тех пор, пока истинно")

А это второй вид цикла. И если в цикле `For` число повторений известно заранее, то в цикле `While` — нет.

Помните два варианта восхождения по лестнице? Так вот, цикл `While` приходит на помощь тогда, когда мы не знаем, сколько ступеней нам нужно преодолеть — мы просто шагаем до тех пор, пока не дойдем до конца.

Вот так выглядит цикл `While`:

```
While <условие>
  <действия>
EndWhile
```

То есть действия будут выполняться до тех пор, пока условие истинно.

Здесь нужно понять самый важный момент цикла `While`. Те действия, которые происходят в теле цикла, должны менять какую-то переменную, от которой зависит условие. Иначе, если каждый раз условие будет одним и тем же, мы получим бесконечный цикл, который будет выполняться бесконечное число раз. А это вряд ли входит в наши планы.

По традиции запишем в псевдокоде решение задачи о лестнице с неизвестным числом ступеней:

```
i = 1
While есть_i-я_ступень
  шагнуть_на_i-ю_ступень
  i = i + 1
EndWhile
```

Давайте разберем эту программу. Смотрите. Вначале мы стоим перед лестницей. Подниматься еще не начинали, перед нами — первая ступенька. Поэтому присваиваем переменной `i` (счетчик ступеней) значений 1 (`i = 1`). После этого приступаем к восхождению. Проверяем условие — "есть ли 1-я ступень"? Если есть, то выполняем действие — "шагнуть на 1-ю ступень". После этого увеличиваем счетчик ступеней на единицу и повторно возвращаемся к условию цикла — "есть ли 2-я ступень?" Если есть, то шагаем на нее, увеличиваем счетчик и т. д. И вот, пусть мы шагнули на 10-ю ступень и проверяем, есть ли впереди 11-я? А ее нет. Поэтому условие становится ложным и происходит выход из цикла. Все, подъем завершен!

Вывод чисел от 1 до 10

Вот так с помощью цикла `While` можно вывести на экран числа от 1 до 10:

```
i = 1
While i <= 10
  TextWindow.WriteLine(i)
  i = i + 1
EndWhile
```

Несложно понять, что происходит в этом цикле. Перед началом цикла мы задаем `i` равным единице. Затем перед каждым шагом цикла проверяем, не дошли ли до 10. И если еще не дошли, то выводим текущее число на экран, а затем увеличиваем `i`. На самом деле, здесь мы с помощью цикла `While` реализовали цикл `For` из предыдущего раздела.

Деление числа пополам

А вот задача, для которой `For` уже не поможет — только `While`. Давайте возьмем число 100 и будем делить его пополам до тех пор, пока не дойдем до 1.

```
k = 100
While k > 1
  TextWindow.WriteLine(k)
  k = k / 2
EndWhile
```

Разберемся, как программа работает. Начинаем с числа 100 (`k = 100`). Условие цикла — `k > 1`. Пока оно истинно, выводим `k` на экран и делим его на 2. Ведь вы же помните, что означает строка `k = k / 2`? Это значит "взять значение переменной `k`, разделить его на 2, результат положить в переменную `k`". Последнее число, которое будет выведено, — 1,5625. Ведь потом оно поделится на 2, и результат уже будет меньше единицы! Условие станет ложным, цикл завершится. Вот так будут выглядеть шаги цикла (табл. 2.5).

Таблица 2.5

Шаг цикла	Условие `k > 1`	Вывод `k`	Новое `k`
До цикла	—	—	100
1	100 > 1: истина	100	50
2	50 > 1: истина	50	25
3	25 > 1: истина	25	12,5
4	12,5 > 1: истина	12,5	6,25
5	6,25 > 1: истина	6,25	3,125
6	3,125 > 1: истина	3,125	1,5625
7	1,5625 > 1: истина	1,5625	0,78125
8	0,78125 > 1: ложь!	—	—

Задача про лыжника

Следующая интересная задачка для цикла `While` — про лыжника, который поставил себе цель пробежать 200 км. Не за один день, конечно, а за несколько. Причем лыжник поставил себе задачу не совсем обычным образом. В первый день он решил пробежать 10 км, а затем каждый день увеличивать дистанцию на 20%. То есть во второй день лыжник пробежит 12 км (общий пробег будет 22 км), в третий — 14 км и 400 м (всего — 36,4 км), ну и т. д. Рано или поздно общий пробег достигнет 200 км. Надо понять главное — через сколько дней?

Такой простой, удобный и, казалось бы, универсальный цикл `For` здесь оказывается совершенно беспомощным. А вот `While` решает эту задачку очень просто.

```
k = 10
s = 10
i = 1
While s < 200
  k = k * 1.2
  s = s + k
```

```
    i = i + 1
    TextWindow.Write("День " + i + " ")
    TextWindow.Write("Пробег " + k + " ")
    TextWindow.WriteLine("Всего "+ s)
EndWhile
```

В переменной `k` у нас хранится пробег лыжника за текущий день, в переменной `s` — общий пробег, `i` — это счетчик дней. Начинаем мы с первого дня, когда лыжник пробежал 10 км. Присваиваем всем трем переменным нужные значения и идем в цикл, который должен повторяться до тех пор, пока общий пробег лыжника не превысит 200 км (`s < 200`). На очередном шаге цикла пересчитываем каждую переменную. Пробег текущего дня увеличивается на 20% (`k = k * 1.2`), общий пробег `s` увеличивается на значение `k` (`s = s + k`), ну и счетчик дней прирастает на единицу (`i = i + 1`). После пересчета, конечно, выводим новые значения всех переменных на экран, красиво разделяя их пробелами.

Попробуем запустить программу и узнаем, что своей великой цели лыжник добьется совсем скоро — в 9-й день он пробежит почти 43 километра, а общий его пробег составит без малого 208 км, т. е. больше, чем 200.

Для интереса можете проверить, сколько дней пришлось бы бегать лыжнику, если бы он увеличивал дистанцию не на 20%, а, скажем, на 10%. Или вообще на 5%.

Игра в лото

Еще один интересный пример — что-то вроде игры в лото. Предположим, что мы загадали число, например 69. А компьютер пытается его угадать — он выводит случайные числа от 1 до 100 до тех пор, пока не укажем то, что мы загадали. Для выдачи случайных чисел воспользуемся методом `Math.GetRandomNumber(100)`. В скобках стоит число 100. Это значит как раз то, что случайные числа будут генерироваться в интервале 1...100. Вот как будет выглядеть программа:

```
While n <> 69
  n = Math.GetRandomNumber(100)
  TextWindow.WriteLine(n)
EndWhile
```

Запустим программу — она будет пытаться угадать число. Получится некоторая последовательность вроде такой: 32, 93, 5, 61, 77, 28, 40, 1, 84, 95, 14, 69. Рано или поздно компьютер обязательно угадает наше число 69 и на этом остановится — условие "`n` не равно 69" (`n <> 69`) станет ложным.

Сумма цифр числа

Дано число. Нужно вычислить сумму его цифр. Например, сумма цифр числа 652 равна 13 (6 + 5 + 2). Как обычно, приводим программу и разбираемся, как она работает.

```
a = TextWindow.ReadNumber()
s = 0

While a > 0
  s = s + Math.Remainder(a,10)
  a = Math.Floor(a/10)
EndWhile

TextWindow.WriteLine(s)
```

После того как поместили в переменную a число и обнулили переменную s, заходим в цикл While. (Да, без него тут совершенно никак!) Мы последовательно прибавляем к s все цифры числа a, двигаясь по нему справа налево. Как получить крайнюю правую цифру? Очень просто — взять остаток от деления a на 10! Берем его и прибавляем к s (s = s + Math.Remainder(a,10)). Теперь нужно избавиться от этой цифры в числе a. Для этого делим a на 10 и берем целую часть частного (a = Math.Floor(a/10)).

К примеру, для числа 5291 этот процесс приведен в табл. 2.6.

Таблица 2.6

s	**a**
0	5291 (до цикла)
1 (0 + 1)	529 (Math.Floor(5291/10))
10 (1 + 9)	52
12 (10 + 2)	5
17 (12 + 5)	0 (выход из цикла)

Простые делители числа

Задача, чем-то похожая на предыдущую. По крайней мере, тем, что используются практически те же самые функции. Но цикла здесь уже два. Обратите внимание — цикл While вложен внутрь цикла For.

> ***Кстати***
>
> Если вложенные циклы выглядят немного непонятно, то вернитесь к этому примеру после *разд. "Двумерные массивы" далее в этой главе* — там вложенные циклы описаны очень подробно.

```
a = TextWindow.ReadNumber()

For i = 2 To a
  While Math.Remainder(a,i) = 0
```

```
      a = a / i
    EndWhile
    TextWindow.WriteLine(i)
  EndFor
```

Здесь цикл `For` проходит от 2 до `a`, проверяя все возможные числа, которые могут быть делителями `a`. Почему от 2, а не от 1? Потому что единица делит любое число и проверять ее нам совершенно ни к чему. Внутри цикла `For` находится еще один цикл, вложенный — `While`. Здесь он, по сути дела, заменяет условный оператор `If`, проверяя, делится ли `a` на `i`. Только в отличие от `If`, он делает это не один раз, а до тех пор, пока условие не перестанет выполняться.

Давайте проиллюстрируем работу программы на примере числа 260 (табл. 2.7).

Таблица 2.7

i	**Условие** `Math.Remainder(a,i) = 0`	**a**
До цикла		260
2	Истина (260 делится на 2)	130
	Истина (130 делится на 2)	65
3	Ложь (65 не делится на 3)	65
4	Ложь (65 не делится на 4)	65
5	Истина (65 делится на 5)	13
6, 7, ..., 12	Ложь	13
13	Истина (13 делится на 13)	1
14, 15, ..., 260	Ложь	1

Обратите внимание на жуткую неэффективность нашего алгоритма. Уже найден последний делитель (13), совершенно ясно, что больше делителей не будет (т. к. `a` равно единице), но программа все равно продолжает проверять все остальные числа,

вплоть до 260. Давайте исправим эту ошибку, заменив цикл `For` циклом `While`:

```
a = TextWindow.ReadNumber()
i = 2

While a > 1
  While Math.Remainder(a,i) = 0
    a = a / i
  EndWhile
  TextWindow.WriteLine(i)
  i = i + 1
EndWhile
```

Задания

1. Дано целое число *n*. Определите, является ли оно степенью числа 3. Корни не извлекать!
2. Дано все то же целое число *n*. Определите, есть ли среди его цифр двойка?
3. Вы положили в банк 10 000 рублей. В конце каждого года размер вклада увеличивается на *p* процентов. Через сколько лет у вас будет 20 000 рублей?
4. Определите, является ли число простым.
5. Найдите наибольший общий делитель двух чисел (для этого нужно разобраться с алгоритмом Евклида).

Массивы

Помните, мы говорили о том, что переменная — это ящик, в который можно положить какое-нибудь значение? Так вот, *массив* — это тумбочка из нескольких ящиков, в каждый из которых можно положить значение.

Договоримся для удобства обозначать имена массивов прописными буквами (хотя принципиальной разницы нет).

Пусть тумбочка называется `M` и в ней 5 ящиков. Тогда каждый ящик тумбочки — *элемент массива* — имеет имя:

```
M[1]
M[2]
M[3]
M[4]
M[5]
```

То есть для обращения к элементу массива используется конструкция `M[n]`, где число `n` — это номер ящика в тумбочке, который называется *индексом*.

Приведем пример программы, работающей с массивом:

```
M[1] = 2
M[2] = 5 + M[1]
M[3] = M[1] * M[2]
M[4] = (M[1] + 3) * M[2]
TextWindow.WriteLine(M[4])
```

Как меняются значения элементов массива в ходе выполнения этой программы, представлено в табл. 2.8.

Таблица 2.8

Строка	M[1]	M[2]	M[3]	M[4]	M[5]
1	2	Ничто	Ничто	Ничто	Ничто
2	2	7	Ничто	Ничто	Ничто
3	2	7	14	Ничто	Ничто
4	2	7	14	35	Ничто

Посмотрите: каждый оператор изменяет только один элемент массива за один раз. В этом и есть вся суть. Массив — это тумбочка с ящиками, и за одно действие можно поменять содержимое только одного ящика. То есть, например, положить одновременно какое-то значение во все элементы массива сразу нельзя, только последовательно. Вот для этого и нужны циклы!

Приведем простые примеры. Для начала заполним массив числами. Нет, не от 1 до 10 — везде они. Давайте, например, заполним 10 элементов массива числами 3, 6, 9, ..., 30. (Да, это арифметическая прогрессия.) То есть в первом элементе должно находиться число 3, во втором — 6, в третьем — 9 и т. д. То есть значение каждого элемента будет равно его индексу, умноженному на 3. Воспользуемся циклом `For`.

```
For i = 1 To 10
  M[i] = i * 3
EndFor
```

Вот что происходит — табл. 2.9.

Таблица 2.9

i	Действие	M[i]
1	`M[1] = 1 * 3`	3
2	`M[2] = 2 * 3`	6
3	`M[3] = 3 * 3`	9
...	...	...
10	`M[10] = 10 * 3`	30

Теперь осуществим вывод элементов массива на экран:

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

Легко видеть, что элементы выводятся на экран последовательно. На первом шаге цикла будет выведен элемент `M[1]`, на втором — `M[2]` и т. д. вплоть до `M[10]`.

А вот так будет выглядеть ввод элементов массива с клавиатуры:

```
For i = 1 To 10
  M[i] = TextWindow.ReadNumber()
EndFor
```

А так — заполнение массива случайными числами (от 1 до 100):

```
For i = 1 To 10
  M[i] = Math.GetRandomNumber(100)
EndFor
```

Давайте теперь рассмотрим несколько примеров по обработке массивов.

Вычисление суммы элементов массива

Пусть в массиве будет, например, 5 элементов.

```
For i = 1 To 5
  TextWindow.Write("Введите элемент массива номер " + i)
  M[i] = TextWindow.ReadNumber()
EndFor

sum = 0
For i = 1 To 5
  sum = sum + M[i]
EndFor

TextWindow.WriteLine(sum)
```

В программе два цикла.

Первый цикл — ввод элементов массива. На каждом шаге цикла выводится сообщение-подсказка "Введите элемент массива номер `i`", где вместо `i` подставляется 1, 2, 3, 4, 5. Затем с клавиатуры вводится число и записывается в элемент массива.

Второй цикл — обработка массива, вычисление суммы. Мы уже рассматривали подобный пример чуть раньше, когда изучали циклы. Перед циклом переменная `sum` обнуляется, а потом в ней последовательно накапливается сумма.

Давайте посмотрим, как программа работает. Начнем сразу со второго цикла — с обработки. Предположим, что ввели мы, например, числа 8, 2, 13, 7, 5, т. е.:

```
M[1] = 8
M[2] = 2
M[3] = 13
M[4] = 7
M[5] = 5
```

Как происходит накопление, показано в табл. 2.10.

Таблица 2.10

Шаг цикла	**i**	**sum**
До цикла		0
1	1	8 (0 + 8)
2	2	10 (8 + 2)
3	3	23 (10 + 13)
4	4	30 (23 + 7)
5	5	35 (30 + 5)

Таким образом, сумма элементов нашего массива — 35.

Сумма положительных элементов

Давайте теперь найдем сумму не всех элементов массива, а только положительных.

```
For i = 1 To 5
  TextWindow.Write("Введите элемент массива номер " + i)
  M[i] = TextWindow.ReadNumber()
EndFor

sum = 0
For i = 1 To 5
  If M[i] > 0 Then
    sum = sum + M[i]
  EndIf
EndFor

TextWindow.WriteLine("Сумма равна " + sum)
```

Посмотрите: внутри цикла появилось условие — `If`. Теперь мы увеличиваем сумму на значение текущего элемента, только если он больше нуля.

Предположим, что мы ввели вот такие элементы:

```
M[1] = 7
M[2] = -4
M[3] = 12
M[4] = -5
M[5] = 8
```

Вот что получится — табл. 2.11.

Таблица 2.11

Шаг цикла	**i**	**Условие**	**sum**
До цикла			0
1	1	7 > 0 (истина)	7 (0 + 7)
2	2	–4 > 0 (ложь)	7 (не меняется)
3	3	12 > 0 (истина)	19 (7 + 12)
4	4	–5 > 0 (ложь)	19 (не меняется)
5	5	8 > 0 (истина)	27 (19 + 8)

Как видим, цикл проходит по всем элементам массива, каждый проверяется на выполнение условия (`M[i] > 0`), и к сумме прибавляются только те элементы, которые этому условию удовлетворяют.

Поиск максимального элемента массива

Этот пример уже будет посложнее. Найдем среди элементов массива самое большое значение — максимум. Алгоритм здесь такой:

1. Предположим, что максимум — это самый первый элемент.
2. Сравним с ним второй элемент. Если он больше, то он становится новым максимумом. Если нет, то максимумом остается первый.
3. Так будем сравнивать третий, четвертый и все остальные элементы. Получается, как только встречается элемент, который больше всех предыдущих, он будет становиться максимумом.

Программа получится вот такая:

```
For i = 1 To 5
  TextWindow.Write("Введите элемент массива номер " + i)
  M[i] = TextWindow.ReadNumber()
EndFor

max = M[1]
For i = 2 To 5
  If M[i] > max Then
    max = M[i]
  EndIf
EndFor

TextWindow.WriteLine("Максимальный элемент равен " + max)
```

Посмотрите внимательно. Перед циклом обработки мы присваиваем переменной `max` значение первого элемента массива: `max = M[1]`. Затем проходим по оставшимся элементам (начиная со второго) и сравниваем каждый с максимумом, переписывая максимум по мере необходимости.

Пусть, например, в нашем массиве вот такие значения:

```
M[1] = 8
M[2] = 11
M[3] = 4
M[4] = 17
M[5] = 9
```

Как происходит поиск максимального элемента, представлено в табл. 2.12.

Таблица 2.12

Шаг цикла	**i**	**Условие**	**max**
До цикла			8
1	2	`M[2] > max` 11 > 8 (истина)	11
2	3	`M[3] > max` 4 > 11 (ложь)	11
3	4	`M[4] > max` 17 > 11 (истина)	17
4	5	`M[5] > max` 9 > 17 (ложь)	17

Посмотрите. Сначала максимум — это число 8. Потом находится число 11 — оно становится максимумом. А потом обнаруживается и число 17, которое становится новым максимумом. Чисел, которое больше 17, в массиве нет, поэтому это и есть итоговый максимум.

Все ли элементы равны?

Если нужно проверить, все ли элементы массива равны между собой, то понятно — их нужно сравнивать. Первое, что приходит на ум, — сравнивать каждый элемент с каждым. Если в массиве, например, 100 элементов, то нужно сделать 990 сравнений: каждый элемент сравнить со всеми остальными (т. е. с 99 оставшимися).

Можно ли как-то оптимизировать алгоритм, сделать его эффективнее? Безусловно. Более того, можно сделать этот алгоритм эффективнее на целый порядок, сократив число сравнений до 99! Ведь на самом деле достаточно сравнивать лишь соседние элементы массива — первый со вторым, второй с третьим и т. д. Если где-нибудь в массиве встретится элемент, отличающийся от остальных, то он будет отличаться и от своих соседей, поэтому не обязательно сравнивать его со всеми подряд.

Получим вот такой простой код:

```
f = "True"
For i = 1 To n-1
  If A[i] <> A[i+1] Then
    f = "False"
  EndIf
EndFor
```

Здесь мы рассматриваем массив, в котором содержится `n` элементов. Цикл `For` проходит от первого элемента до элемента с номером `n-1`. Почему не до `n`? Все просто. Мы сравниваем текущий элемент со следующим, т. е. каждый `i`-й элемент с элементом `i+1`. Получается, на последнем шаге нам нужно будет сравнить предпоследний элемент с последним. Счетчик `i` будет равен `n-1`, соответственно элемент с номером `n-1` будет сравниваться с **n**-м элементом, как мы того и хотели.

С циклом и сравнением все ясно. Но остается важный вопрос: что это за переменная `f`, которая равна то истине, то лжи? Переменная `f` — это так называемый *флаг*. Смотрите. До проверки ее значение — "Истина". То есть мы как бы предполагаем, что все элементы массива равны между собой. Затем мы сравниваем все соседние элементы, и если хотя бы где-то выполнилось условие `A[i] <> A[i+1]`, т. е. нашлись неравные элементы, то флаг сразу изменит значение на "Ложь"! А если ни на одном шаге цикла не встретились неравные элементы, то условие не сработает ни разу и значение флага останется прежним — "Истина". Это значит, что все хорошо — все элементы массива равны между собой.

Теперь легко осуществить вывод:

```
If f = "True" Then
  TextWindow.WriteLine("Все элементы массива равны между собой")
Else
  TextWindow.WriteLine("Не все элементы массива равны между собой")
EndIf
```

Все ли элементы различны?

Проверим теперь элементы на различность. Это значит, что в массиве вообще не должно быть одинаковых элементов. Задача звучит очень похоже, но решается немного по-другому. Только сравнением соседей здесь уже не обойтись. Но все же и тут есть место для оптимизации. Алгоритм здесь будет как бы "треугольным". Первый элемент мы будем сравнивать со всеми остальными (от 2 до `n`), второй элемент — с элементами от 3 до `n`, и т. д. Это позволит нам сократить количество сравнений в два раза, что вполне неплохо:

```
f = "True"
For i = 1 To n
  For j = i+1 To n
    If A[i] = A[j] Then
      f = "False"
    EndIf
  EndFor
EndFor
```

Как видно, здесь используются вложенные циклы. Внешний цикл проходит элементы от 1 до `n`, а внутренний для каждого `i`-го элемента пробегает по элементам от `i+1` до `n`.

> ***Кстати***
>
> За разъяснениями по вложенным циклам снова отсылаю вас к *разд. "Двумерные массивы" далее в этой главе*.

Оптимальный способ выплаты

Продавцы в магазинах всегда просят деньги без сдачи. Давайте напишем программу, которая будет рассчитывать самый-самый оптимальный способ выплаты каждой суммы купюрами и монетами. Оптимизировать будем по количеству, т. е. будем считать оптимальным наименьшее количество купюр и монет, которые потребуются для выплаты заданной суммы. Например, 25 рублей можно заплатить пятью пятирублевыми монетами, а можно дать и двадцать пять рублевых монеток. Но оптимальным способом будут две десятирублевки и одна монета в пять рублей, т. е. всего 3 монеты.

Кстати

Яндекс.Деньги, QIWI-кошелек или WebMoney все равно удобнее!

```
A[1] = 5000
A[2] = 1000
A[3] = 500
A[4] = 100
A[5] = 50
A[6] = 10
A[7] = 5
A[8] = 2
A[9] = 1

n = TextWindow.ReadNumber()

i = 1
For i = 1 To 9
  While n > A[i]
    C[i] = C[i] + 1
    n = n - A[i]
  EndWhile
  i = i + 1
EndFor
```

```
For i = 1 To 9
  If C[i] > 0 Then
    TextWindow.WriteLine(A[i] + " руб. x " + C[i])
  EndIf
EndFor
```

Первым делом вручную прописываем массив `A`, содержащий все возможные купюры или монеты. В России есть 9 номиналов — от красивой красной купюры в 5000 руб. до маленькой монетки в 1 руб. Копейки уж рассматривать не будем, с ними все совершенно аналогично.

И снова используем вложенные циклы. Внешний `For` проходит путь от 1 до 9 (по количеству разных номиналов). Для каждого номинала запускается вложенный цикл `While`, который отнимает из числа `n` этот номинал до тех пор, пока это можно сделать. Одновременно увеличивается соответствующий элемент массива `C` (т. е. количество купюр или монет текущего номинала).

Самый последний цикл `For` нужен исключительно для красивого вывода результата.

Самовлюбленные числа

И еще одна задача про целые числа. Самовлюбленным числом (или числом Армстронга) называется такое число, которое равно сумме своих цифр, возведенных в степень, равную количеству его цифр. Например, $153 = 1^3 + 5^3 + 3^3$. Давайте найдем все трехзначные числа Армстронга. Здесь нам массивы пока не нужны.

```
For i = 100 To 999
  a = i
  s = 0
  While a > 0
    s = s + Math.Power(Math.Remainder(a,10),3)
    a = Math.Floor(a/10)
  EndWhile
  If i = s Then
    TextWindow.WriteLine(i)
  EndIf
EndFor
```

Так как мы пока ограничились лишь трехзначными числами, то двигаемся циклом `For` от 100 до 999. На каждом проходе цикла используем уже известный нам алгоритм разделения числа на цифры. Только суммируем не сами цифры, а их кубы (`Math.Power(Math.Remainder(a,10),3)`). После этого проверяем, равна ли полученная сумма `s` числу `i`. Если равна, значит, мы нашли очередное самовлюбленное число.

А как проверить не только трехзначные числа? Если мы просто изменим границы цикла `For`, это не решит проблему — нам ведь нужно знать, в какую степень возводить цифры! Поэтому изменения будут чуть более серьезными:

```
For i=10 To 99999
  a = i
  k = 0
  s = 0
  While a > 0
    k = k + 1
```

```
      C[k] = Math.Remainder(a,10)
      a = Math.Floor(a/10)
    EndWhile
    For j = 1 To k
      s = s + Math.Power(C[j],k)
    EndFor
    If i = s Then
      TextWindow.WriteLine(i)
    EndIf
  EndFor
```

Смотрите, что здесь происходит. В большой цикл `For` у нас теперь вложены два цикла — `While` и `For`. Во внутреннем цикле `While` мы считаем количество цифр в числе и складываем сами цифры в массиве `C`. А во внутреннем цикле `For` мы уже считаем `s`, суммируя `k`-е степени всех цифр, которые берем из того самого массива `C`. Теперь мы знаем еще больше самовлюбленных чисел, например, $92727 = 9^5 + 2^5 + 7^5 + 2^5 + 7^5$.

Не только числа!

Да, в массивах могут храниться не только числа, но и строки. Давайте, например, введем в массив имена нескольких человек, а потом со всеми ними поздороваемся.

```
For i = 1 To 5
  TextWindow.Write("Человек номер " + i + ", введи свое
имя! ")
  name[i] = TextWindow.Read()
EndFor


TextWindow.Write("Привет, ")
For i = 1 To 5
  TextWindow.Write(name[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

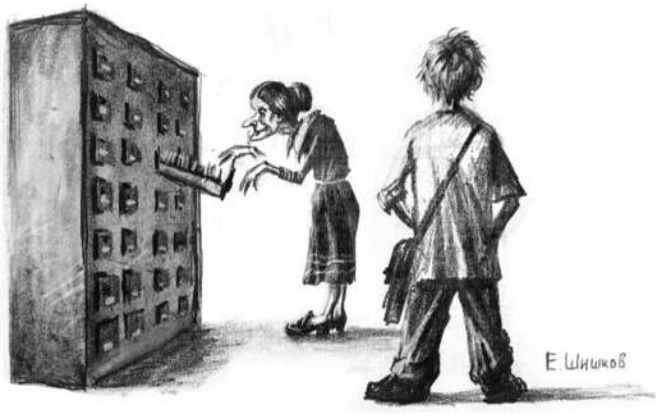
Попробуйте сами разобраться, как работает эта программа!

Задания

1. Перепишите из массива `A` в массив `B` все элементы в обратном порядке.
2. Перепишите из массива `A` в массив `B` все отрицательные элементы (в массиве `A` останутся только положительные элементы и нули).
3. Определите, является ли массив возрастающим (нужно проверять условие "каждый последующий элемент больше предыдущего").
4. В массиве поменяйте местами максимальный и минимальный элементы.
5. В массиве поменяйте местами попарно соседние элементы (первый со вторым, третий с четвертым и т. д.).
6. Найдите количество различных элементов в массиве.
7. Запишите в массив `n` первых чисел Фибоначчи, выведите их.
8. Отсортируйте массив размера `n` методом простого обмена ("пузырьковая" сортировка). Суть метода в том, что массив последовательно просматривается, и если `i`-й элемент больше `i+1`-го, то они меняются местами. Таким образом, по массиву нужно пройти `n-1` раз.
9. Отсортируйте массив методом простого выбора. В этом методе нужно искать максимальный элемент и менять его местами с последним. Это тоже делается `n-1` раз, причем на каждом проходе количество просматриваемых элементов уменьшается на единицу.

Двумерные массивы

И снова возвращаемся к ящикам. Если обычные, одномерные массивы, о которых мы говорили только что, это тумбочки с ящиками, то *двумерный массив* можно представить себе как библиотечный каталог или камеру хранения в супермаркете. То есть это такой шкаф, в котором ящики располагаются и по горизонтали, и по вертикали.



Соответственно, нумеруются элементы двумерного массива не одним индексом, а двумя. Вот так:

N[1][1]	N[1][2]	N[1][3]
N[2][1]	N[2][2]	N[2][3]
N[3][1]	N[3][2]	N[3][3]

Например, чтобы поместить число 17 в элемент, расположенный во второй строке третьего столбца массива, мы напишем:

```
M[2][3] = 17
```

Обрабатываются двумерные массивы с помощью *вложенных циклов*. С ними мы уже встречались в нескольких примерах, но все-таки давайте посмотрим на них более пристально.

Например, заполним двумерный массив размером 3×3 числами от 1 до 9:

```
k = 1
For i = 1 To 3
  For j = 1 To 3
    N[i][j] = k
    k = k + 1
  EndFor
EndFor
```

Здесь один цикл вложен в другой. Первый цикл называется *внешним*, второй — *внутренним*. Счетчики у них, разумеется, разные: у внешнего цикла — `i`, у внутреннего — `j`.

Работает это вот как:

1. Запускается внешний цикл. На первом шаге его счетчик `i` становится равным 1.
2. Запускается внутренний цикл. Его счетчик `j` тоже становится равным 1. Счетчик внутреннего цикла `j` проходит от 1 до 3, в то время как `i` остается равным 1.
3. Только после того как `j` дошел до 3, происходит выход из внутреннего цикла, `i` увеличивается до 2.
4. Снова запускается внутренний цикл. Его счетчик `j` опять проходит от 1 до 3. Счетчик `i` при этом равен 2.
5. И т. д.

Вот что получается — табл. 2.13.

Таблица 2.13

i	j	Действие	k
			1
1	1	`N[1][1] = 1`	2
1	2	`N[1][2] = 2`	3
1	3	`N[1][3] = 3`	4
2	1	`N[2][1] = 4`	5
2	2	`N[2][2] = 5`	6
2	3	`N[2][3] = 6`	7
3	1	`N[3][1] = 7`	8
3	2	`N[3][2] = 8`	9
3	3	`N[3][3] = 9`	10

Ввод двумерного массива 3×3:

```
For i = 1 To 3
  For j = 1 To 3
```

```
    TextWindow.Write("Введите элемент массива " + i + ", "
+ j)
    N[i][j] = TextWindow.Read()
  EndFor
EndFor
```

Аналогично можно запрограммировать вывод двумерного массива. Здесь элементы каждой строки выводятся друг за другом через пробел (с помощью метода `Write()` во внутреннем цикле), а после внутреннего цикла происходит переход на новую строку с помощью метода `WriteLine()`. Поэтому элементы выводятся так, как надо — по строкам, практически в виде таблицы:

```
For i = 1 To 3
  For j = 1 To 3
    TextWindow.Write(N[i][j] + " ")
  EndFor
  TextWindow.WriteLine()
EndFor
```

Рассмотрим пару примеров обработки двумерных массивов.

Сумма всех элементов двухмерного массива

Здесь все просто. Алгоритм тот же, что и для одномерных массивов — сумма последовательно накапливается. Единственное отличие в том, что используются вложенные циклы.

```
For i = 1 To 3
  For j = 1 To 3
    TextWindow.Write("Введите элемент массива " + i + ", "
+ j)
    N[i][j] = TextWindow.Read()
  EndFor
EndFor

sum = 0
For i = 1 To 3
  For j = 1 To 3
    sum = sum + N[i][j]
```

```
  EndFor
EndFor


TextWindow.WriteLine("Сумма всех элементов двумерного мас-
сива: " + sum)
```

Комментарии, пожалуй, излишни.

Суммы по строкам

Посчитаем теперь суммы элементов отдельно по каждой строке. Сколько сумм нам нужно найти? Столько, сколько строк в массиве. В нашем случае — три. Поэтому будем записывать эти суммы в отдельный одномерный массив, который назовем, например, S.

```
For i = 1 To 3
  For j = 1 To 3
    TextWindow.Write("Введите элемент массива " + i + ", "
+ j)
    N[i][j] = TextWindow.Read()
  EndFor
EndFor


For i = 1 To 3
  S[i] = 0
  For j = 1 To 3
    S[i] = S[i] + N[i][j]
  EndFor
EndFor


For i = 1 To 3
  TextWindow.WriteLine("Сумма элементов " + i + "-й строки:
" + S[i])
EndFor
```

Обратите внимание на циклы обработки. Если в предыдущем примере мы обнуляли сумму перед внешним циклом, то здесь обнуляем соответствующий элемент массива S перед внутренним

циклом, на каждом шаге внешнего. Это легко объяснимо — ведь сумм у нас три штуки, и каждая считается отдельно!

Давайте внимательно проанализируем, что же происходит.

Пусть массив будет вот таким:

4	10	8
16	9	12
7	5	3

Начинаем искать суммы (табл. 2.14).

Таблица 2.14

`i`	`j`	`S[1]`	`S[2]`	`S[3]`
1	До внутреннего	0	Ничто	Ничто
1	1	4 (0 + 4)	Ничто	Ничто
1	2	14 (4 + 10)	Ничто	Ничто
1	3	22 (14 + 8)	Ничто	Ничто
2	До внутреннего	22	0	Ничто
2	1	22	16 (0 + 16)	Ничто
2	2	22	25 (16 + 9)	Ничто
2	3	22	37 (25 + 12)	Ничто
3	До внутреннего	22	37	0
3	1	22	37	7 (0 + 7)
3	2	22	37	12 (7 + 5)
3	3	**22**	**37**	**15** (12 +3)

В итоге в одномерном массиве `S` оказались суммы по каждой строке двухмерного массива — 22, 37, 15.

Треугольник Паскаля

Это очень известный и важый арифметический треугольник, который позволяет легко рассчитать так называемые коэффици-

енты бинома Ньютона — основу для всяческих комбинаторных вещей вроде перестановок или сочетаний.

Выглядит треугольник Паскаля вот так:

1						
1	1					
1	2	1				
1	3	3	1			
1	4	6	4	1		
1	5	10	10	5	1	
1	6	15	20	15	6	1

и т. д.

Именно через биномиальные коэффициенты получаются всем известные формулы сокращенного умножения. Например:

- $a+b\ ^2=a^2+2ab+b^2$ (коэффициенты 1, 2, 1 — третья строка треугольника Паскаля);
- $a+b\ ^3=a^3+3a^2b+3ab^2+b^3$ (следующая строка треугольника).

Легко получить и все остальные формулы. Например, вот такую:

$$a+b\ ^6=a^6+6a^5b+15a^4b^2+20a^3b^3+15a^2b^4+6ab^5+b^6.$$

Но вернемся к сути. Алгоритм построения треугольника Паскаля легко виден — каждый его элемент равен сумме двух соседних элементов предыдущей строки. Пользуясь этим простым правилом, напишем программу:

```
k = TextWindow.ReadNumber()

For i = 1 To k
  P[i][1] = 1
  For j = 2 To i
    P[i][j] = P[i-1][j-1] + P[i-1][j]
  EndFor
```

```
EndFor

For i = 1 To k
  For j = 1 To i
    TextWindow.Write(P[i][j] + " ")
  EndFor
  TextWindow.WriteLine("")
EndFor
```

Переменная `k` — количество строк треугольника Паскаля, который мы хотим получить. Основной алгоритм размещается в двух вложенных циклах `For`. Первый (`i` от 1 до `k`) отвечает за строки, второй (`j` от 2 до `i`) — за столбцы. Первый элемент каждой строки — всегда единица (`P[i][1] = 1`), а начиная со второго значения элементов, рассчитываются по известному нам правилу (`P[i][j] = P[i-1][j-1] + P[i-1][j]`).

Вывод элементов массива `P` снова вынесем в отдельный блок, чтобы не запутывать основную часть алгоритма.

Латинские квадраты

Последний пример — самый сложный. Давайте напишем программу, которая проверяет, является ли двумерный массив латинским квадратом. Что это такое? Латинский квадрат — это массив размера $n \times n$, заполненный числами от 1 до n таким образом, чтобы в каждой строке и в каждом столбце встречались все n символов, причем каждый — ровно по одному разу.

Вот, например, латинский квадрат третьего порядка:

1	2	3
2	3	1
3	1	2

Кстати

"Латинским" квадрат назвал великий математик Леонард Эйлер. Он использовал вместо цифр латинские буквы — A, B, C и т. д. А японцы придумали латинский квадрат размером 9×9 еще раньше и назвали его "судоку" (в переводе — "одинокое число").

В общем, довольно вводных слов, давайте программировать.

```
TextWindow.Write("Введите размер квадрата: ")
n = TextWindow.ReadNumber()

For i = 1 To n
  For j = 1 To n
    TextWindow.Write("Введите элемент (" + i + "," + j +
")")
    A[i][j] = TextWindow.ReadNumber()
  EndFor
EndFor

f = "True"

For i = 1 To n
  For j = 1 To n
    If A[i][j] < 1 Or A[i][j] > n Then
      f = "False"
    EndIf
  EndFor
EndFor

For i = 1 To n
  For j = 1 To n
    For k = j + 1 To n
      If A[i][j] = A[i][k] Then
        f = "False"
      EndIf
    EndFor
  EndFor
EndFor

For j = 1 To n
  For i = 1 To n
    For l = i + 1 To n
      If A[i][j] = A[l][j] Then
```

```
          f = "False"
        EndIf
      EndFor
    EndFor
  EndFor

  If f = "True" Then
    TextWindow.WriteLine("Латинский квадрат!")
  Else
    TextWindow.WriteLine("Обычный квадрат")
  EndIf
```

В самом начале вводим размер квадрата (`n`), а затем все его элементы. А дальше стартует проверка. Она состоит из трех этапов:

1. Проверяем, все ли элементы квадрата — это числа от 1 до `n`, иными словами, нет ли среди элементов каких-то неподходящих значений (например, в латинском квадрате третьего порядка никак не может быть числа 5 — только 1, 2 и 3).
2. Проверяем на различность элементы по строкам.
3. Проверяем на различность элементы по столбцам.

Помните, как мы проверяли, являются ли различными все элементы одномерного массива? Так вот, здесь ровно то же самое. Пусть тройные вложенные циклы вас не пугают — всмотритесь в них повнимательнее, и все станет ясно.

Кстати, двумерными массивами дело не ограничивается. Массивы могут быть и трехмерными (представьте себе кубик Рубика — ящики в трех измерениях!), и четырех-, и пятимерными и т. д. Используются они, конечно, довольно редко. Но они есть.

Задания

1. Заполните двумерный массив размера $n \times n$ последовательностью чисел 1, 2, ..., n^2.
2. Найдите сумму элементов главной диагонали массива (на главной диагонали находятся элементы, у которых номер строки равен номеру столбца).
3. Найдите сумму элементов побочной диагонали массива (несложно догадаться, какая диагональ будет побочной).

4. Заполните массив так, чтобы на главной диагонали были расположены нули, выше нее — единицы, а ниже — двойки.
5. Найдите максимум и минимум в каждой строке двумерного массива.
6. В каждом столбце массива найдите количество элементов, больших среднего арифметического элементов этого столбца.
7. Найдите количество строк массива, в которых все элементы равны между собой.
8. Напишите программу, которая проверяет, является ли двумерный массив магическим квадратом. В магическом квадрате суммы по всем строкам, столбцам, а также по двум диагоналям должны быть одинаковыми.

Работа со строками

До этого мы работали в основном с числами. Но вы же помните, что есть и второй тип данных — строки.

Строка — это последовательность символов (слово, предложение или вообще несвязная ерунда из букв, цифр и других значков). Каждый символ в строке имеет свой номер. Этим строка чем-то напоминает одномерный массив.

Например, строка "окорок":

1	2	3	4	5	6
о	к	о	р	о	к

Для обработки строк в языке Small Basic есть специальный объект `Text`.

Основные функции объекта `Text` приведены в табл. 2.16.

Таблица 2.16

Функции	**Описание**
`Text.Append`	Склеивает две строки (можно использовать вместо оператора +) `Text.Append("око","рок") = "окорок"`

Таблица 2.16 *(продолжение)*

Функции	Описание
`Text.ConvertToLowerCase`	Конвертирует символы в нижний регистр `Text.ConvertToLowerCase ("БейСИк") = "бейсик"`
`Text.ConvertToUpperCase`	Конвертирует символы в верхний регистр `Text.ConvertToLowerCase ("БейСИк") = "БЕЙСИК"`
`Text.EndsWith`	Проверяет, заканчивается ли строка заданной подстрокой `Text.EndsWith("окорок", "рок") = Истина`
`Text.GetCharacterCode`	Получает код символа (в стандарте Юникод) `Text.GetCharacterCode("A") = 65` (английская буква "A") `Text.GetCharacterCode("А") = 1040` (русская буква "А")

`Text.GetCharacter`	Получает символа по его коду (в стандарте Юникод)
	`Text.GetCharacter(100) = "d"`
	`Text.GetCharacter(1100) = "ь"`

`Text.GetIndexOf`	Находит позицию в строке, с которой начинается подстрока `Text.GetIndexOf("собака", "оба") = 2`
`Text.GetLength`	Определяет длину строки `Text.GetLength("собака") = 6`
`Text.GetSubText`	Получает часть строки `Text.GetSubText("котелок",3, 4) = "тело"`
`Text.GetSubTextToEnd`	Получает часть строки, начиная с заданной позиции до конца `Text.GetSubText("котелок",3) = "телок"`

Таблица 2.16 *(окончание)*

Функции	Описание
`Text.IsSubText`	Проверяет, входит ли одна строка в другую `Text.IsSubText("собака", "оба") = Истина`
`Text.StartsWith`	Проверяет, начинается ли строка заданной подстрокой `Text.EndsWith("окорок", "око") = Истина`

Приведем несколько примеров обработки строк.

Вывод символов по отдельности

Дана строка. Вывести все символы этой строки по отдельности.

```
s = "Мама мыла раму"
For i = 1 To Text.GetLength(s)
  TextWindow.WriteLine(Text.GetSubText(s,i,1))
EndFor
```

Программа выведет символы по одному на каждой строке экрана:

```
М
а
м
а

м
ы
л
а

р
а
м
у
```

Вот как эта программа работает. Основа алгоритма — цикл `For`, который последовательно проходит по строке `s` с первого символа до последнего. Первый символ, очевидно, имеет номер 1. Как узнать номер последнего символа? Элементарно — `Text.GetLength(s)`!

Теперь на каждом шаге цикла у нас есть номер текущего символа. Чтобы вытащить из строки символ с этим номером, пользуемся функцией `Text.GetSubText(s,i,1)` — "взять 1 символ из строки `s`, начиная с символа с номером `i`". И, наконец, выводим полученный символ на экран с помощью `TextWindow.WriteLine()`.

Таким образом, на первом шаге цикла на экран выведется символ "М", на втором — "а", на третьем — "м" и т. д.

Количество букв "а"

Дана строка. Определить, сколько раз в ней встречается буква "а".

```
s = "собака пошла в магазин"
c = 0
For i=1 To Text.GetLength(s)
  If Text.GetSubText(s,i,1) = "а" Then
    c = c + 1
  EndIf
EndFor
TextWindow.WriteLine("Количество букв а в строке: " + c)
```

Алгоритм здесь практически такой же. Цикл, который проходит строку от начала до конца. На каждом шаге цикла проверка — не "а" ли равен текущий символ? Если это правда "а", то увеличиваем на единицу переменную `c` (заботливо обнуленную заранее).

Программа подсчитает, что в предложении "собака пошла в магазин" 5 букв "а". И это правильно.

Но для предложения "А роза упала на лапу Азора" программа скажет, что букв "а" — 6. Хотя невооруженным глазом видно, что их там 8! Просто две из них — прописные. Как сделать так, чтобы считались и прописные (большие), и строчные (маленькие) буквы? Есть два способа.

Можно изменить условие, добавив еще один вариант с помощью логического оператора `Or` ("или"):

```
If Text.GetSubText(s,i,1) = "а" Or Text.GetSubText(s,i,1) =
"А" Then
```

А можно перед проверкой просто перевести все символы в нижний регистр и спокойно искать маленькую "а". Для этого перед циклом просто добавляется вот такая конструкция:

```
s = Text.ConvertToLowerCase(s)
```

Замена буквы "a" буквой "o"

Дана строка. Заменить в ней все буквы "a" на "o".

```
s = "магазин"
t = ""
For i = 1 To Text.GetLength(s)
  If Text.GetSubText(s,i,1) = "a" Then
    t = t + "o"
  Else
    t = t + Text.GetSubText(s,i,1)
  EndIf
EndFor
TextWindow.WriteLine(t)
```

Здесь суть в том, что мы из исходной строки (переменная `s`) последовательно формируем новую строку (в переменной `t`), делая нужную нам замену. Как и раньше, проходим по строке `s` циклом `For`. Проверяем каждую букву, равна ли она "a". Если равна, то в строку `t` добавляем букву "o", а если не равна — то ту же букву, которая и была.

Обратите внимание, что строку `t` мы перед циклом тоже "обнуляем", но т. к. это не число, то присваиваем переменной `t` значение "пустая строка", т. е. открываем кавычки и сразу закрываем.

Получается вот так — табл. 2.17.

Таблица 2.17

i	**Текущий символ `Text.GetSubText(s,i,1)`**	**Строка `t`**
1	м	**м**
2	а	м**о**
3	г	мо**г**
4	а	мог**о**
5	з	могo**з**
6	и	могоз**и**
7	н	могози**н**

Определение длины всех слов в строке

Дана строка из нескольких слов. Определить длины всех слов (знаки препинания не учитываем).

```
s = "Жил старик со своею старухой у самого синего моря"
t = ""
n = 0

If Text.EndsWith(s," ") = "False" Then
  s = s + " "
EndIf

For i = 1 To Text.GetLength(s)
  If Text.GetSubText(s,i,1) <> " " Then
    t = t + Text.GetSubText(s,i,1)
    n = n + 1
  Else
    TextWindow.WriteLine(t + " " + n)
    t = ""
    n = 0
  EndIf
EndFor
```

Эта программа уже посложнее, но и в ней нет ничего сверхъестественного.

Итак, в переменной `s` лежит строка, которую мы будем разбивать на слова. В нашем случае — строка про старика и старуху, которые жили возле моря. Переменная `t` нужна для того, чтобы по ходу программы собирать в ней каждое отдельное слово. И `n` — для подсчета количества букв.

Алгоритм следующий. Понятно, что слова в строке разделены между собой пробелами. Поэтому именно по пробелам и будем определять, где заканчивается одно слово и начинается другое. Проходим по строке циклом. Каждый символ сравниваем с пробелом (`" "`). Если текущий символ не равен пробелу, то добавляем его к строке `t` (к текущему слову), а переменную `n` (количест-

во букв в текущем слове) увеличиваем на единицу. Если же нам встретился пробел, значит, слово закончилось. Выводим его на экран, обнуляем переменные `t` и `n`.

Ну и небольшой финт ушами. Мы же определяем конец слова по пробелу, поэтому последнее слово вылетит из алгоритма — пробела-то после него нет! Поэтому пробел в конец строки `s` надо добавить. Перед циклом вставляем небольшую проверку: если строка НЕ заканчивается пробелом, то добавить его в конец.

Шифр Цезаря

Как сделать так, чтобы наше сообщение не прочитали посторонние? Конечно, зашифровать его! Вообще, изучением шифров занимается целая наука — криптография. И ее приверженцы добились серьезных успехов — алгоритмы RSA, DES, RC5, AES и многие другие дают очень мощную защиту.

Мы же с вами пока реализуем один из самых первых методов шифрования, которым, по легенде, пользовался диктатор Римской республики Гай Юлий Цезарь.

Алгоритм этого шифра основан на сдвиге. Это значит, что каждая буква заменяется другой буквой, которая отстоит на некоторое число символов правее в алфавите. Это самое "некоторое число" называется *ключом шифра*. Например, если ключ шифра равен 4, то буква "А" заменится буквой "Д", буква "Б" — буквой "Е" и т. д. Если после прибавления ключа буквы в алфавите закончатся, то происходит возврат в начало алфавита. То есть, например, буква "Я" станет буквой "Г", буква "Ю" — буквой "В" и т. д.

Обратите внимание — наша программа будет шифровать только русские буквы. Букву "Ё" программа тоже обрабатывать не будет — для нее не нашлось места между буквами "Е" и "Ж", и она оказалась в другой части кодовых таблиц. Не будем усложнять программу дополнительными проверками. Просто договоримся, что буквы "Ё" в шифровках не будет.

```
TextWindow.Write("Введите строку: ")
s = Text.ConvertToUpperCase(TextWindow.Read())

TextWindow.Write("Введите ключ: ")
k = TextWindow.ReadNumber()

For i = 1 To Text.GetLength(s)
  p = Text.GetSubText(s,i,1)
  x = Text.GetCharacterCode(p) - 1039
  y = Math.Remainder(x+k,32) + 1040
  p1 = Text.GetCharacter(y)
  s1 = s1 + p1
EndFor

TextWindow.WriteLine(s1)
```

Вводим строку в переменную `s` и для простоты сразу же переводим ее в верхний регистр (маленькие и большие буквы нам будут только мешать — пусть все будут большими). Ключ будет лежать в переменной `k`.

Приступаем к шифрованию:

1. Получаем текущий символ (`p = Text.GetSubText(s,i,1)`).
2. Определяем номер текущего символа в алфавите (`x = Text.GetCharacterCode(p) - 1039`). Функция `Text.GetCharacterCode` даст нам код символа в стандарте Юникод. Код символа "А", т. е. первой буквы алфавита — 1040. Поэтому для того чтобы узнать номер буквы в алфавите, нужно отнять из ее кода число 1039. Тогда буква "А" будет иметь номер 1, "Б" — 2 и т. д. Полученный номер попадает в переменную `x`.
3. Производим сдвиг (`y = Math.Remainder(x+k,32) + 1040`). Для этого нам нужно прибавить к `x` ключ `k`, а затем взять остаток от деления этой суммы на 32 (количество букв в алфавите, не считая злополучной буквы "Ё"). И вновь возвращаемся от номера буквы в алфавите к коду символа Юникод, прибавляя 1040.
4. Получаем символ по коду (`p1 = Text.GetCharacter(y)`).
5. Дописываем этот символ в строку `s1` (`s1 = s1 + p1`).

Попробуем что-нибудь зашифровать — например, слово "ПРОГРАММИРОВАНИЕ" ключом 13. Получится "ЭЮЬСЮОЪЪЦЮЬРОЫЦУ". Вполне неплохо! Теперь можно шифровать все важные сообщения где-нибудь "ВКонтакте". Главное — заранее сообщить друзьям ключ, чтобы они могли расшифровывать сообщения.

А как, зная ключ, расшифровать сообщения? Для этого достаточно сдвигать буквы в обратную сторону, т. е. вычислять не `x + k`, а `x - k`. И, конечно, вычитать 1040, а прибавлять 1039, т. е. тоже наоборот.

Задания

1. Замените в строке все восклицательные знаки вопросительными, а все точки — многоточиями.
2. В строке найдите количество всех знаков препинания.
3. Найдите в строке все слова, у которых первая и последняя буквы совпадают.
4. Каждое слово в строке запишите наоборот (справа налево).
5. Даны два слова. Определите, какое из них меньше (т. е. какое будет раньше расположено в словаре). Например: "кошка" < "собака", "коробка" < "кошка".

6. Замените все последовательности из нескольких одинаковых букв одной такой же буквой.
7. Напишите программу, которая преобразует слова, оканчивающиеся на "онок" и "енок", в множественное число (мышонок — мышата, цыпленок — цыплята, слоненок — слонята и т. д.). Предусмотрите исключения из правила (ребенок — дети, звонок — звонки, жаворонок — жаворонки).
8. Напишите программу, которая записывает трехзначное число прописью. Например, 123 — "сто двадцать три", 511 — "пятьсот одиннадцать" и т. д.
9. Усложним задачу — пусть программа теперь записывает словами все числа от 0 до 1000 (5 — "пять", 18 — "восемнадцать", 1000 — "тысяча" и т. д.).

Стек

Конечно, про стек надо было рассказать чуть раньше — сразу же после массивов. Ведь стек в каком-то роде похож на массив — он тоже позволяет хранить множество элементов и по одному к ним обращаться. Но так уж вышло, что самые простые и интересные примеры использования стека так или иначе связаны с обработкой строк. Поэтому и говорим про стек вот тут, уже изучив работу со строками.

Что же такое *стек*? Проще всего представить его как гору тарелок. Вообразите: мы моем тарелки и складываем их одну на другую. Причем мы не можем положить новую чистую тарелку в любое место стопки — только на самый верх. А когда мы собираемся есть и начинаем брать тарелки из кучи, все происходит точно так же — мы можем взять только верхнюю тарелку. И если кому-нибудь захочется поесть не из первой, а из второй тарелки, то мы сначала снимем первую, а уже затем вторую.

Еще одна классическая аналогия стека — магазин в огнестрельном оружии. Первым пойдет по врагу патрон, заряженный последним.

Называется этот принцип *LIFO (last in — first out)*. То есть, в переводе на русский, "последним пришел — первым вышел". Вполне логично — последняя поставленная тарелка будет взята самой первой. Обратный принцип — *FIFO (first in — first out)*.

Добавление нового элемента в стек (т. е. новой тарелки в стопку) называется *проталкиванием элемента (push)*. Удаление элемента из стека (изъятие тарелки из стопки) называется *выталкиванием (pop)*. При этом "вытолкнут" будет самый верхний элемент стека, после чего верхним окажется второй с конца, т. е. предпоследний пришедший.

В Small Basic за работу со стеком отвечает простой объект `Stack`. У него всего три метода (табл. 2.18).

Таблица 2.18

Методы	**Описание**
`Stack.GetCount(stackName)`	Получает число элементов стека
`Stack.PopValue(stackName)`	Выталкивает верхний элемент стека
`Text.PushValue(stackName,value)`	Проталкивает новый элемент в стек

Давайте рассмотрим пару примеров использования стека.

Баланс скобок

Пожалуй, самый простой пример задачи, которую проще решить с помощью стека, чем каким-то другим способом. Будем рассматривать последовательности скобок — открывающих и закрывающих. Нужно определить, сбалансированы ли эти скобки, т. е. соответствует ли каждой открывающей скобке закрывающая. Если соответствует — то баланс есть, если же какие-то скобки останутся лишними — то баланса нет.

Например, строка `()((())))` сбалансирована, а строка `(()(()()` — нет, потому что есть две открывающие скобки, для которых нет закрывающих.

Вот такой будет программа:

```
s = TextWindow.Read()
myStack = 0

For i = 1 To Text.GetLength(s)
  If Text.GetSubText(s,i,1) = "(" Then
    Stack.PushValue(myStack,1)
  ElseIf Text.GetSubText(s,i,1) = ")" Then
    Stack.PopValue(myStack)
  EndIf
EndFor

If Stack.GetCount(myStack) = 0 Then
  TextWindow.WriteLine("Есть баланс")
Else
  TextWindow.WriteLine("Нет баланса")
EndIf
```

Понять, что тут происходит, несложно. В переменной `s` лежит строка, которую мы будем проверять. Переменная `myStack` — наш стек. Не пугайтесь второй строчки кода (`myStack = 0`). Это всего лишь инициализация стека. Не забывайте, что язык Small Basic настолько прост, что переменные в нем не объявляются специальным образом, как во многих других языках программирования, а создаются автоматически при первом обращении.

Проталкивание или выталкивание не могут создать переменную стека — такова уж особенность. Поэтому этой строкой мы как раз и обеспечиваем первое обращение к переменной, ее создание. При этом совершенно неважно, какое именно значение присвоить переменной `myStack` — хоть ноль, хоть шестьсот шестьдесят шесть.

Затем мы идем в цикл, который проходит по каждому символу строки `s`. Нас волнуют два символа — открывающая и закрывающая скобки. Каждая открывающая скобка будет добавлять один элемент в стек (`Stack.PushValue(myStack,1)`), а каждая закрывающая — удалять (`Stack.PopValue(myStack)`). Опять же, то, что мы "проталкиваем" в стек число 1, а не что-то другое — совершенно не важно, нам важен сам факт того, что стек вырос на один элемент.

После того как мы прошли циклом по всем символам строки, по состоянию стека можно точно узнать, сбалансированы ли скобки в строке. Если стек после цикла оказался пустым, значит, баланс есть.

Вот, например, строка `()(())`. Размер стека будет меняться так, как показано в табл. 2.19. Все должно быть понятно без дополнительных комментариев.

Таблица 2.19

i	**Текущий символ `Text.GetSubText(s,i,1)`**	**Число элементов в стеке**
1	(	1
2	)	0
3	(	1
4	(	2
5	)	1
6	)	0

Вроде бы все просто и, главное, правильно. Но нет — в программе есть серьезная ошибка! Проявится она, например, для строки `())`. Первая скобка — открывающая, в стек добавится

элемент, размер стека станет равным 1. Затем идет закрывающая скобка, элемент из стека удалится, размер будет равен 0. Но следом — еще одна закрывающая скобка! Нужно удалить элемент из стека, но стек уже пуст — возникает ошибка!

Что делать? Усложнять условия. Если мы встретили закрывающую скобку, будем проверять еще и текущий размер стека. Если он больше нуля (`Stack.GetCount(myStack) > 0`), то все в порядке — выталкиваем элемент из стека. Если же размер стека равен нулю (`Stack.GetCount(myStack) = 0`), значит, выталкивать нечего, а баланса скобок уже явно нет. Поэтому в этом случае сразу же, без промедления, выводим сообщение "Баланса нет" и завершаем программу (`Program.End()`). А чтобы успеть увидеть сообщение до закрытия черного окна программы, вставляем `TextWindow.Read()`.

```
  s = TextWindow.Read()
  myStack = 0

  For i = 1 To Text.GetLength(s)
    If Text.GetSubText(s,i,1) = "(" Then
      Stack.PushValue(myStack,1)
    ElseIf Text.GetSubText(s,i,1) = ")" And
  Stack.GetCount(myStack) > 0 Then
      Stack.PopValue(myStack)
    ElseIf Text.GetSubText(s,i,1) = ")" And
  Stack.GetCount(myStack) = 0 Then
      TextWindow.WriteLine("Нет баланса")
      TextWindow.Read()
      Program.End()
    EndIf
  EndFor

  If Stack.GetCount(myStack) = 0 Then
    TextWindow.WriteLine("Есть баланс")
  Else
    k = Stack.PopValue(myStack)
    TextWindow.WriteLine("Нет баланса")
  EndIf
```

В каком месте ошибка?

Теперь мы хотим знать, какая именно скобка нарушает баланс. Как это сделать? Очень просто. Для этого достаточно писать в стек не какую-то условную единицу, а номер текущего символа (`i`). Тогда, если после прохождения по циклу в стеке что-то останется, то это будет как раз номер ошибочной скобки.

```
s = TextWindow.Read()
myStack = 0

For i = 1 To Text.GetLength(s)
  If Text.GetSubText(s,i,1) = "(" Then
    Stack.PushValue(myStack,i)
  ElseIf Text.GetSubText(s,i,1) = ")" And
Stack.GetCount(myStack)>0 Then
    Stack.PopValue(myStack)
  ElseIf Text.GetSubText(s,i,1) = ")" And
Stack.GetCount(myStack)=0 Then
    TextWindow.WriteLine("Нет баланса, ошибка в символе
номер " + i)
    TextWindow.Read()
    Program.End()
  EndIf
EndFor

If Stack.GetCount(myStack) = 0 Then
  TextWindow.WriteLine("Есть баланс")
Else
  k = Stack.PopValue(myStack)
  TextWindow.WriteLine("Нет баланса, ошибка в номере "+k)
EndIf
```

Как видно, самое главное изменение в программе — `Stack.PushValue(myStack,i)`. Ну и, соответственно, немного поменялся вывод — в сообщение о том, что баланса нет, тоже добавился номер.

Да, обратите внимание — наша программа будет работать не только для последовательностей из одних скобок, но и для любых

выражений со скобками! Что-то вроде `(2+3)*(15-(4+1))` (сбалансированное) или `((a+b)*(c+d)))/(e+f)` (не сбалансированное, ошибка в 14-м символе). Ведь мы проверяем только скобки, а остальные символы нам совершенно не мешают.

Задания

1. С помощью стека перепишите символы строки в обратном порядке ("привет" — "тевирп").
2. Проверьте правильность расстановки скобок трех видов: круглых ("(" и ")"), квадратных ("[" и "]") и фигурных ("{" и "}").

Файлы

Перед тем как перейти к графике, разберемся еще с одним полезным объектом Small Basic — объектом `File`. Как легко догадаться из названия этого объекта, он позволяет работать с файлами. Что это такое — надеюсь, объяснять не нужно. При этом возможности объекта `File` можно разделить на две части:

- работа с файлом;
- работа с файловой системой.

В чем разница — понятно. В первом случае мы работаем с содержимым файла — читаем из него информацию, что-то в него записываем. А во втором случае мы оперируем файлами целиком — копируем, удаляем, переименовываем. А это, на самом деле, работа с файловой системой.

Работа с файлом

Содержимое файла можно читать и записывать. Вот такие функции существуют для чтения из файла — табл. 2.20.

Таблица 2.20

Метод	Описание
`File.ReadContents(filePath)`	Читает все содержимое файла
`File.ReadLine(filePath,lineNumber)`	Читает указанную строку из файла

В чем между ними разница — понятно. Имейте в виду, что для больших файлов метод `ReadContents()` может работать очень медленно.

Для записи в файл функций больше (табл. 2.21).

Таблица 2.21

Метод	Описание
`File.WriteContents(filePath, contents)`	Записывает новое содержимое в файл (старое полностью удаляется)
`File.AppendContents(filePath, contents)`	Добавляет новое содержимое в конец файла
`File.InsertLine(filePath, lineNumber,contents)`	Вставляет содержимое в указанную строку файла
`File.WriteLine(filePath, lineNumber,contents)`	Записывает содержимое в указанную строку файла (старое удаляется)

В чем разница здесь — наверное, тоже понятно. Но на всякий случай давайте рассмотрим пример:

```
File.WriteContents("file.txt","Первая строка")
File.AppendContents("file.txt","Продолжение первой строки")
File.InsertLine("file.txt",1,"Новая первая строка")
File.WriteLine("file.txt",2,"Новая вторая строка")
```

Вот что будет происходить с файлом file.txt во время выполнения этой простой программы — табл. 2.22.

Таблица 2.22

Действие	Содержимое файла
`File.WriteContents("file.txt", "Первая строка")`	`Первая строка`
`File.AppendContents("file.txt" ,"Продолжение первой строки")`	`Первая строкаПродолжение первой строки`
`File.InsertLine("file.txt", 1, "Новая первая строка")`	`Новая первая строка` `Первая строкаПродолжение первой строки`
`File.WriteLine("file.txt", 2, "Новая вторая строка")`	`Новая первая строка` `Новая вторая строка`

Метод `WriteContents()` просто записывает в файл строку "Первая строка". Если такого файла не существует, он автоматически появится. Где? Скорее всего, в текущей папке, т. е. там, где сохранена текущая программа.

Затем с помощью `AppendContents()` добавляем в файл текст "Продолжение первой строки". Казалось бы, он должен вставиться в новую строку — но нет, этот метод допишет текст в конец строки.

Методом `InsertLine()` вставим "Новая первая строка" — бывшая первая строка сдвинется и станет второй. А вот метод `WriteLine()` перезапишет вторую строку новым текстом.

В общем, вот они — функции для работы с содержимым файла на все случаи жизни. Кстати, все они возвращают результат — либо `"SUCCESS"` (все прошло хорошо), либо `"FAILED"` (произошла какая-то ошибка). Узнать, какая именно произошла ошибка, можно с помощью свойства `LastError`.

Вообще, файлы — очень полезная штука. Во многих примерах нам приходилось вводить с клавиатуры довольно много значений, особенно когда речь шла о массивах. Гораздо удобнее реализовать ввод через файл. Тогда достаточно один раз ввести данные в текстовый файл (например, с помощью программы Блокнот) и не нужно будет при каждом запуске программы вводить все элементы массива один за другим.

Вот так, например, можно ввести из файла содержимое одномерного массива:

```
i = 1
While File.ReadLine("file.txt",i) <> ""
  A[i] = File.ReadLine("file.txt",i)
  i = i + 1
EndWhile
```

Все просто — циклом `While` проходим по каждой строке файла до тех пор, пока строки не закончатся. Каждую строку помещаем в `i`-й элемент массива `A`.

У этого алгоритма, конечно, есть очень серьезный недостаток. Если в файле вдруг встретится пустая строка, то остальные строки будут проигнорированы — программа решит, что файл закон-

чился. Такая проблема возникает из-за того, что в Small Basic никак нельзя узнать, где находится конец файла. Для того чтобы обойти эту ситуацию, приходится выкручиваться — считывать содержимое файла целиком функцией `ReadContents()`, а затем разбирать получившийся большой текст, определяя, где в нем заканчивается каждая строка. Код там получается громоздкий и непонятный, поэтому не будем портить им книгу. Просто имейте в виду — не делайте в файлах пустых строк в середине!

Двумерный массив считывается из файла немного сложнее. Каждую строку файла нужно разбирать на отдельные элементы (по разделителю — пробелу). Помните, мы делили строку на слова? Вот и здесь все примерно так же, попробуйте сами соединить эти два алгоритма.

Работа с файловой системой

Функции Small Basic по работе с файловой системой позволяют делать практически все те вещи, которые умеет стандартный Проводник Windows (табл. 2.23).

Таблица 2.23

Метод	Описание
`File.GetFiles(directoryPath)`	Получает список файлов в папке
`File.GetDirectories(directoryPath)`	Получает список папок в папке
`File.CopyFile(sourceFilePath, destinationFilePath)`	Копирует файл
`File.DeleteFile(filePath)`	Удаляет файл
`File.CreateDirectory(directoryPath)`	Создает папку
`File.DeleteDirectory(directoryPath)`	Удаляет папку

Вот простейший пример, который выводит список папок и файлов в заданной папке:

```
TextWindow.Write("Введите путь к папке: ")
path = TextWindow.Read()
```

```
Directories = File.GetDirectories(path)
Files = File.GetFiles(path)

If Directories <> "FAILED" Then
  TextWindow.WriteLine("Папки:")
  For i = 1 To Array.GetItemCount(Directories)
    TextWindow.WriteLine(Directories[i])
  EndFor
Else
  TextWindow.WriteLine("Ошибка!")
  TextWindow.WriteLine(File.LastError)
EndIf

If Files <> "FAILED" Then
  TextWindow.WriteLine("Файлы:")
  For i = 1 To Array.GetItemCount(Files)
    TextWindow.WriteLine(Files[i])
  EndFor
Else
  TextWindow.WriteLine("Ошибка!")
  TextWindow.WriteLine(File.LastError)
EndIf
```

Сначала вводим путь к папке и записываем его в переменную `path`. Путь обязательно должен быть введен в полном формате — не забудьте про имя диска. Например, `"c:\Users\vasya\Documents"` — это правильный путь, а просто `"vasya\Documents"` — неправильный.

Затем с помощью функций `GetDirectories()` и `GetFiles()` получаем два массива. В массиве `Directories` записаны все папки, а в массиве `Files` — все файлы. Если, конечно, не произошла какая-нибудь ошибка! А если ошибка появилась, то в этих переменных будет только одно слово — `"FAILED"`. Поэтому делаем проверку и, если все в порядке, выводим все элементы массива, а иначе — сообщение об ошибке.

Задания

1. Сделайте в файле ВСЕ БУКВЫ ПРОПИСНЫМИ.
2. Перепишите все строки из одного файла в другой в обратном порядке.
3. В файле записаны фамилии, имена и отчества людей (например, Иванов Иван Иванович). В каждой строке — по одному человеку. Перепишите все строки в другой файл, сокращая инициалы (Иванов И. И.).
4. Придумайте, как можно переименовать файл.

Глава 3

Совершенствуем интерфейс

Графика

До этого мы работали только с объектом `TextWindow`, который называли просто "черным окном". Это, действительно, универсальный объект, который позволяет делать практически все, что нужно программисту, а именно вводить и выводить числа и текстовые строки. Но вот одного черное окно не умеет — рисовать картинки. А иногда и это бывает нужно!

Поэтому знакомьтесь: новый объект — графическое окно `GraphicsWindow`! Оно как раз и умеет рисовать.

Кстати

Оно умеет и кое-что еще, очень интересное и полезное, но об этом немного позже.

Итак, графическое окно. В первую очередь его нужно показать. Для этого есть метод `Show()`, который отобразит окно в стандартном режиме:

```
GraphicsWindow.Show()
```

Настройки окна легко изменить, для этого существует несколько свойств (табл. 3.1).

Таблица 3.1

Свойства	Описание
`GraphicsWindow.BackgroundColor`	Цвет фона
`GraphicsWindow.CanResize`	Можно ли изменять размер
`GraphicsWindow.Height`	Высота
`GraphicsWindow.Width`	Ширина
`GraphicsWindow.Left`	Расстояние от левого края экрана
`GraphicsWindow.Top`	Расстояние от верхнего края экрана
`GraphicsWindow.Title`	Заголовок

Давайте проведем тотальный эксперимент со всеми свойствами сразу:

```
GraphicsWindow.BackgroundColor = "Red"
GraphicsWindow.CanResize = "False"
GraphicsWindow.Height = 300
GraphicsWindow.Width = 500
GraphicsWindow.Left = 10
GraphicsWindow.Top = 10
GraphicsWindow.Title = "Графическое окно"
```

Вот теперь окно у нас персонализированное. Во-первых, оно красное. Во-вторых, изменять его размер нельзя. В-третьих, высота у него 300 пикселов, а ширина — 500 пикселов. В-четвертых, оно находится практически в левом верхнем углу экрана — всего

в 10 пикселах от краев. И, в-пятых, у него крайне креативный заголовок — "Графическое окно".

> ***Кстати***
>
> На всякий случай вспомним, что пиксел — это одна точка экрана. Если очень внимательно присмотреться к монитору, то можно увидеть, что он состоит из мелких-мелких точек. Каждый монитор имеет разное количество пикселов по ширине и высоте (например, 1280×1024). Этот параметр называется *разрешением монитора*.

Внутри графического окна тоже есть точки, те же самые пикселы. И каждая точка окна имеет свою координату. Но для окна не имеет значения разрешение монитора — окно действует само по себе. Таким образом, левая верхняя точка окна имеет координату (0, 0). Первая координата — *X* (по горизонтали), вторая — *Y* (по вертикали). Совсем как в математике.

Давайте теперь, зная про координаты, попробуем нарисовать простейший графический объект — линию (или, если говорить более точно, отрезок):

```
GraphicsWindow.DrawLine(0,0,500,300)
```

Так как наше персонализированное окно имеет ширину в 500 пикселов и высоту в 300, то линия как раз пересечет окно по диагонали. Как работает метод `DrawLine()`, понять таким образом несложно. Два первых числа — координаты начала отрезка, два вторых — координаты конца.

Посмотрите, какого цвета линия? Скорее всего, черного. Как изменить цвет? Очень просто — перед рисованием линии изменить свойство `PenColor`. Изменить толщину? Тоже никаких проблем — свойство `PenWidth`. Вот, например, флаг Швейцарии — толстый белый крест на квадратном красном фоне:

```
GraphicsWindow.BackgroundColor = "Red"
GraphicsWindow.CanResize = "False"
GraphicsWindow.Height = 300
GraphicsWindow.Width = 300
GraphicsWindow.Title = "Флаг Швейцарии"


GraphicsWindow.PenColor = "White"
```

```
GraphicsWindow.PenWidth = 45
GraphicsWindow.DrawLine(150,50,150,250)
GraphicsWindow.DrawLine(50,150,250,150)
```

Здесь надо сделать одно важное замечание — о цветах. Цвет может задаваться двумя способами. Первый — по названию цвета. Именно этим способом мы всегда пользовались до этого, просто называя цвета по-английски: Black, White, Red, Blue, Green и т. д. В Small Basic большое количество таких названий — около 100 штук. Кроме основных там есть такая экзотика как Cornsilk (цвет кукурузного початка), Chartreuse (цвет ликера "Шартрез"), Tan (цвет загара) и др.

Второй способ — более универсальный. Он подразумевает использование так называемой модели RGB. Наверное, вы о ней уже слышали, но все же повторим. RGB означает "Red, Green, Blue", т. е. "Красный, Зеленый, Синий". Это три основных цвета модели RGB, и из них получаются все остальные цвета. Например, если смешать красный цвет с зеленым, получится желтый, а если красный с синим, то будет сиреневый. Более сложные смешивания дают весь возможный спектр цветов. Если, к примеру, взять много красного цвета, чуть-чуть зеленого и среднее количество синего, то цвет будет розовым.

Легко догадаться, как сделать черный и белый цвета. Если не брать ни одного цвета — получится черный. А если взять все по максимуму, то белый.

Записывается цвет в RGB-модели в виде шестнадцатеричного числа из 6 разрядов, перед которым ставится знак решетки — `#`. Каждые два разряда обозначают количество базовых (красный, зеленый, синий) цветов. Минимум цвета — 00, максимум — FF (соответствует десятичному числу 255).

Примеры приведенены в табл. 3.2.

Таблица 3.2

Код RGB	**Цвет**
#000000	Черный
#FFFFFF	Белый

Таблица 3.2 *(окончание)*

Код RGB	Цвет
#FF0000	Красный
#00FF00	Зеленый
#0000FF	Синий
#FFFF00	Желтый
#FF1493	Розовый
#FFF8DC	Цвет кукурузного початка

Кстати

В шестнадцатеричной системе счисления не 10 цифр, а, как не сложно догадаться, 16. Вот они: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Поэтому не пугайтесь букв в числах — здесь буквы это цифры, как бы странно это не звучало.

Не будем вдаваться в суть шестнадцатеричной системы счисления — это потребует слишком много времени. Тем более, Small Basic — настолько дружественный язык, что в нем есть специальная функция для тех, кто не знает шестнадцатеричную систему, но хочет пользоваться всей мощью RGB-модели. Эта функция называется `GetColorFromRGB()`. Она принимает три параметра — количество красного, зеленого и синего цветов в виде обычных десятичных чисел (от 0 до 255) и возвращает значение цвета. Например, эти две строчки сделают фон графического окна бирюзовым (если, конечно, мы правильно понимаем, что такое бирюзовый цвет):

```
color = GraphicsWindow.GetColorFromRGB(72,139,158)
GraphicsWindow.BackgroundColor = color
```

Ну, с цветами немного разобрались — двигаемся дальше.

Кроме линий в графическом окне можно рисовать простые фигуры — прямоугольники, треугольник и эллипсы. Все эти фигуры делятся на два вида: обычные и залитые. Например, если мы рисуем обычный красный прямоугольник, то он будет красным только по контуру. А если он залитый, то будет красным целиком. Вот функции для рисования фигур (табл. 3.3).

Таблица 3.3

Методы	Описание
`GraphicsWindow.DrawRectangle(x,y,width,height)`	Обычный прямоугольник. `x`, `y` — координаты. `width`, `height` — ширина и высота
`GraphicsWindow.FillRectangle(x,y,width,height)`	Прямоугольник, закрашенный текущим цветом
`GraphicsWindow.DrawTriangle(x1,y1,x2,y2,x3,y3)`	Обычный треугольник. `x1`, `y1` и т. д. — координаты вершин
`GraphicsWindow.FillTriangle(x1,y1,x2,y2,x3,y3)`	Треугольник, закрашенный текущим цветом
`GraphicsWindow.DrawEllipse(x,y,width,height)`	Обычный эллипс. `x`, `y` — координаты левого верхнего угла прямоугольника, в который вписан эллипс. `width`, `height` — ширина и высота эллипса
`GraphicsWindow.FillEllipse(x,y,width,height)`	Эллипс, закрашенный текущим цветом

Цвет для обычных и закрашенных (иногда говорят "залитых") фигур задается тоже по-разному. Свойства (табл. 3.4) для обычных фигур начинаются со слова Pen (Перо), для закрашенных — со слова Brush (Кисть).

Таблица 3.4

Свойства	Описание
`GraphicsWindow.PenColor`	Цвет контура для обычной фигуры
`GraphicsWindow.PenWidth`	Толщина контура для обычной фигуры
`GraphicsWindow.BrushColor`	Цвет залитой фигуры

Вот, например, флаг Японии. Круг красного цвета, который нам нужно нарисовать — это эллипс, ширина и высота которого одинаковы (по 100 пикселов). Он вписан в квадрат, левый верхний угол которого находится в точке (100, 50) — 100 пикселов от левого края, 50 пикселов от верхнего:

```
GraphicsWindow.BackgroundColor = "White"
GraphicsWindow.CanResize = "False"
GraphicsWindow.Height = 200
GraphicsWindow.Width = 300
GraphicsWindow.Title = "Флаг Японии"

GraphicsWindow.BrushColor = "Red"
GraphicsWindow.FillEllipse(100,50,100,100)
```

А вот и флаг России. Он состоит, как все знают, из трех прямоугольников. Возьмем для флага нашей Родины не абы какие стандартные цвета, а самые настоящие, подсмотренные в Википедии:

```
GraphicsWindow.CanResize = "False"
GraphicsWindow.Height = 600
GraphicsWindow.Width = 900
GraphicsWindow.Title = "Флаг России"

GraphicsWindow.BrushColor = "#FFFFFF"
GraphicsWindow.FillRectangle(0,0,900,200)
GraphicsWindow.BrushColor = "#0053A0"
GraphicsWindow.FillRectangle(0,200,900,200)
GraphicsWindow.BrushColor = "#E2383F"
GraphicsWindow.FillRectangle(0,400,900,200)
```

Следующая возможность графического окна — вывод текста. Для этого есть два метода: `DrawText()` и `DrawBoundText()`. Второй отличается тем, что подразумевает границу справа: если текст окажется слишком длинным, то будет переноситься на следующую строку. А для оформления текста, разумеется, есть целый ряд свойств (табл. 3.5).

Таблица 3.5

Методы	Описание
`GraphicsWindow.DrawText(x,y,text)`	Вывод текста в графическое окно. `x`, `y` — координаты точки, от которой начнется текст. `text` — сам текст
`GraphicsWindow.DrawBoundText(x,y,width,text)`	Вывод текста с ограничением ширины. `width` — максимальная ширина текста (если текст окажется длиннее, то будет переноситься)
`GraphicsWindow.BrushColor`	Цвет текста
`GraphicsWindow.FontBold`	Полужирный шрифт
`GraphicsWindow.FontItalic`	Курсив
`GraphicsWindow.FontName`	Название шрифта
`GraphicsWindow.FontSize`	Размер шрифта

Например, добавим на флаг России патриотическую надпись оптимистичного зеленого цвета:

```
GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FontSize = "64"
GraphicsWindow.FontName = "Times New Roman"
GraphicsWindow.FontItalic = "True"
GraphicsWindow.DrawText(250,50,"Россия, вперед!")
```

Наконец, в окно можно выводить совершенно произвольные картинки из любых графических файлов. Для этого также есть два метода: `DrawImage()` и `DrawResizedImage()`. Первый выводит картинку в реальном размере, а второй может менять ее размер (табл. 3.6).

Таблица 3.6

Метод	Описание
`GraphicsWindow.DrawImage (imageName,x,y)`	Вывод картинки в графическое окно. `imageName` — картинка (полное имя файла или URL). `x`, `y` — координаты левого верхнего угла изображения
`GraphicsWindow. DrawResizedImage(imageName, x,y,width,height)`	Вывод картинки с изменением размера. `width`, `height` — новые ширина и высота

Кстати

Обратите внимание, что картинка может быть расположена не только на жестком диске вашего компьютера, но и в Интернете! Small Basic — язык современный и продвинутый, поэтому в качестве `imageName` можно указать как c:\photo.jpg, так и что-то вроде **http://cs4733.vkontakte.ru/u780128/110579249/x_7102d579.jpg**.

Давайте теперь попробуем совместить эти простые графические вещи с тем, что мы уже знаем — в частности, с циклами.

Отрезки разной толщины

Это самый простой пример того, как с помощью цикла можно нарисовать несколько параллельных отрезков, толщина которых будет каждый раз все больше и больше.

```
For i = 1 To 20
  GraphicsWindow.PenWidth = i
  GraphicsWindow.DrawLine(0,i*25,500,i*25)
EndFor
```

Цикл `For` проходит 20 шагов. На каждом шаге мы, во-первых, устанавливаем новую толщину пера, равную `i`, т. е. на первом шаге толщина будет равна единице, а на последнем — 20. Затем рисуется отрезок. Все отрезки горизонтальные, поэтому координаты `x1` и `x2` постоянны и равны 0 и 500 соответственно. А вот координаты `y1` и `y2` должны меняться в зависимости от `i` — в нашем примере `i` каждый раз умножается на 25. Вполне можно поэкспериментировать с этим множителем — чем он больше, тем большее расстояние будет между соседними отрезками.

Вложенные квадраты

Еще один интересный и несложный пример. Будем рисовать разноцветные квадраты, вложенные друг в друга.

```
GraphicsWindow.Width = 500
GraphicsWindow.Height = 500


x = 0
y = 0
For i = 25 To 1 Step -1
  GraphicsWindow.BrushColor =
GraphicsWindow.GetRandomColor()
  GraphicsWindow.FillRectangle(x,y,i*20,i*20)
  x = x + 10
  y = y + 10
EndFor
```

У нас есть графическое окно размером 500×500 пикселов. В каком порядке рисовать квадраты — от большего к меньшему или от меньшего к большему? Выбора у нас на самом деле нет. Если мы начнем с самого маленького, то все последующие, бóльшие по размеру, станут закрывать собой предыдущие — и в результате будет виден только последний квадрат.

Поэтому начинаем построение с самого большого квадрата. Левый верхний угол его будет совпадать с углом всего графического окна, т. е. будет иметь координаты (0, 0). Поэтому перед циклом присвоим переменным `x` и `y` значения 0. Входим в цикл с обратным шагом (–1) и выполняем его от 25 до 1. На каждом шаге цикла выбираем для нового квадрата случайный цвет (`GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()`) и рисуем квадрат. Координаты левого верхнего угла — (`x`, `y`), ширина и высота зависят от `i`: на первом шаге это будет квадрат 500×500, затем 480×480 и т. д. вплоть до самого маленького квадрата — 20×20. И, разумеется, на каждом шаге цикла сдвигаем угол следующего квадрата на 10 пикселов вниз и вправо, увеличивая переменные `x` и `y`.

Кляксы

Теперь давайте вообразим себя художниками-авангардистами и создадим нереальный шедевр. На белый лист в случайном порядке накапаем видимо-невидимо разноцветных клякс.

```
GraphicsWindow.Width = 500
GraphicsWindow.Height = 500

For i = 1 To 500
  x = Math.GetRandomNumber(500)
  y = Math.GetRandomNumber(500)
  w = Math.GetRandomNumber(30)
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
  GraphicsWindow.FillEllipse(x,y,w,w)
EndFor
```

Клякс у нас будет 500 штук — поэтому и цикл будет идти от 1 до 500. Каждая клякса должна быть в случайном месте, поэтому координаты `x` и `y` будут случайными числами в диапазоне от 1 до 500 (`Math.GetRandomNumber(500)`). Диаметр кляксы — тоже случайное число от 1 до 30. Ну и цвет опять же случайный, разумеется, иначе не бывать шедевру. Определившись со всеми случайными параметрами, бросаем кляксу — `GraphicsWindow.FillEllipse(x,y,w,w)`.

Если вы вдруг захотите, чтобы кляксы были не обязательно круглыми, а вытянутыми в разные стороны, добавьте еще одну переменную для высоты эллипсов — например, `h = Math.GetRandomNumber(30)`.

Штриховка

Горизонтальные линии мы уже рисовали, давайте теперь посмотрим, как рисуются диагональные линии. Для этого аккуратно заштрихуем все окно.

```
GraphicsWindow.Width = 500
GraphicsWindow.Height = 500

For i = 1 To 25
  GraphicsWindow.DrawLine(0, (i-1)*20, 500-(i-1)*20, 500)
  GraphicsWindow.DrawLine((i-1)*20, 0, 500, 500-(i-1)*20)
  GraphicsWindow.DrawLine(500-(i-1)*20, 0, 0, 500-(i-1)*20)
  GraphicsWindow.DrawLine(500, (i-1)*20, (i-1)*20, 500)
EndFor
```

Выглядит достаточно сложно? На самом деле это далеко не так. Просто на каждом шаге цикла в зависимости от значения `i` строятся сразу четыре линии.

На первом шаге, если присмотреться внимательно, строятся не 4, а всего лишь две линии: (0, 0)—(500, 500) и (500, 0)—(0, 500). То есть две основные диагонали (которые, вообще говоря, называются главной и вспомогательной).

А вот затем уже от этих двух диагоналей идет движение вверх и вниз. Например, на втором шаге будут построены вот такие четыре отрезка:

- (0, 20)—(480, 500) — вверх от главной диагонали;
- (20, 0)—(500, 480) — вниз от главной диагонали;
- (480, 0)—(0, 480) — вверх от вспомогательной диагонали;
- (500, 20)—(20, 500) — вниз от вспомогательной диагонали.

И т. д., пока не будет заштрихована вся область графического окна.

Задания

1. Нарисуйте елочку из треугольников и прямоугольника.
2. Нарисуйте домик со всеми деталями — дверью, окном, трубой, дымом из трубы и т. д.
3. Нарисуйте еще какой-нибудь флаг, посложнее. Например, флаг Великобритании вполне подойдет.
4. Заполните графическое окно разноцветными квадратами (квадраты должны стоять вплотную друг к другу).
5. Постройте классическую пятиконечную звезду.
6. Постройте правильный шестиугольник (нужно будет придумать, как рассчитывать длины сторон и углы).
7. Дано число `n`. Постройте правильный `n`-угольник (здесь тоже нужно будет серьезно подумать).

Подпрограммы

Иногда нужно какие-то вещи повторить несколько раз. Конечно, часто в этом нам помогают циклы — именно они позволяют произвести похожие действия много раз подряд. Но бывают такие ситуации, когда цикл не может прийти на помощь.

Давайте представим такую ситуацию. Вы изучили рецепт приготовления какого-нибудь очень сложного блюда, например пельменей. Вам подробно объяснили алгоритм: налить воду в кастрюлю, включить плиту, дождаться, пока она закипит, посолить, достать пельмени из морозильника, положить пельмени в воду, дождаться их всплытия, подождать 10 минут, выключить плиту. Вы сделали все четко по алгоритму, получили кучу вкусных пельменей и съели их, все отлично.

А теперь вы ждете гостей. И среди других не менее вкусных блюд решили приготовить пельмени. Что вы запишете в список праздничных блюд? Вряд ли вы будете писать все, как в первый раз — про кастрюлю, воду и всплытие. Вы напишете просто — "сварить пельмени". Таким образом, программа приготовления праздничного обеда будет выглядеть как-то так:

- ❒ сварить пельмени;
- ❒ сделать салат "Оливье";
- ❒ испечь пирожки;
- ❒ сварить кисель.

А где-то в другом месте, в вашей книге рецептов, будут записаны подробные алгоритмы для всех этих сложных действий.

Переведем это в термины программирования. Вы написали программу, которая содержит четыре вызова подпрограмм.

Теперь, наверное, понятно, что такое *подпрограмма*. Это часть программы, содержащая некий набор инструкций. Основное свойство подпрограммы заключается в том, что у нее есть свое *имя*. То есть ее можно один раз описать, а потом много раз вызывать из разных мест основной программы.

Есть две причины использовать подпрограммы.

Во-первых, это улучшает логику программы. Если бы вы в программе приготовления праздничного обеда начали перечис-

лять все элементарные действия подряд, то легко запутались бы, для чего варятся яйца — для киселя или для салата, перемешивать ли пельмени с пирожками и т. д. А так все четко — четыре отдельных пункта.

Во-вторых, подпрограммы можно вызывать с разными *параметрами*. Например, пирожки могут быть с разными начинками — с картошкой, ливером, рисом и яйцом и т. д. Поэтому "Начинка" будет параметром подпрограммы "Испечь пирожки". Алгоритм-то от изменения начинки не изменится, поэтому писать отдельные подпрограммы для каждого вида пирожков совсем ни к чему. Достаточно просто менять значения параметра. К сожалению, в языке Small Basic передача параметров реализована не слишком хорошо. Точнее говоря, в Small Basic передача параметров вообще отсутствует. Поэтому приходится использовать обычные переменные внутри подпрограмм, что в принципе проблему решает.

В Small Basic подпрограмма определяется с помощью ключевого слова `Sub`:

```
Sub <имя_подпрограммы>
  <текст_подпрограммы>
EndSub
```

Рассмотрим пару примеров использования подпрограмм.

Домики

Давайте напишем подпрограмму `DrawHouse()`, которая рисует домик:

```
Sub DrawHouse
  GraphicsWindow.PenColor = "Red"
  GraphicsWindow.DrawRectangle(100,100,50,50)
  GraphicsWindow.DrawTriangle(100,100,150,100,125,75)
  GraphicsWindow.DrawRectangle(120,120,10,10)
EndSub
```

Теперь, чтобы нарисовать домик, достаточно вызвать ее из любого места программы:

```
DrawHouse()
```

Скобки нужны для того, чтобы интерпретатор понял, что `DrawHouse()` — это не переменная, а именно подпрограмма.

Давайте теперь немного изменим код подпрограммы, чтобы можно было вызывать ее с разными параметрами:

```
Sub DrawHouse
  GraphicsWindow.PenColor = color
  GraphicsWindow.DrawRectangle(x, y, 50, 50)
  GraphicsWindow.DrawTriangle(x, y, x+50, y, x+25, y-25)
  GraphicsWindow.DrawRectangle(x+20, y+20, 10, 10)
EndSub
```

Посмотрите — теперь, изменяя значения переменных `x` и `y`, можно сдвигать домик по горизонтали и по вертикали, а через переменную `color` менять его цвет.

Например, вот так:

```
x = 200
y = 200
color = "Blue"
DrawHouse()
Program.Delay(1000)
x = 100
y = 100
color = "Green"
DrawHouse()
```

Сначала мы рисуем синий домик из точки (200, 200), потом делаем небольшую паузу (`Program.Delay(1000)`) и рисуем еще один домик — зеленый из точки (100, 100). При этом повторять все инструкции (рисование двух квадратов и одного треугольника) не требуется — достаточно просто вызвать подпрограмму. Удобно!

> ***Кстати***
>
> В окне системы Small Basic взамное расположение основной программы и подпрограммы не имеет значения. Вы можете сначала написать текст подпрограммы, а затем — текст основной программы. Или наоборот.

График функции

А сейчас научимся строить графики произвольных функций с помощью графических средств Small Basic.

```
Sub F
  y = Math.Sin(x)
EndSub

GraphicsWindow.Width = 800
GraphicsWindow.Height = 600
GraphicsWindow.CanResize = "False"
GraphicsWindow.DrawLine(0,300,800,300)
GraphicsWindow.DrawLine(400,0,400,600)
For x = -4 To 4 Step 0.01
  F()
  GraphicsWindow.SetPixel(x*100+400,y*100+300,"Red")
EndFor
```

График рисуется довольно просто. Как и в тетрадке, он строится по точкам. Значение `x` пробегает диапазон от –4 до 4, а значение `y` рассчитывается подпрограммой `F()`.

Единственная сложность здесь — "нормировать" значения `x` и `y`, т. е. вычислить, какой именно пиксел закрашивать. Ведь масштабы получаются разными: если `x` в функции меняет значение от –4 до 4, то точка *X* на экране изменяется от 0 до 800 (в нашем примере). Поэтому нужно понять, как перейти от реального `x` к экранному. Смотрите, значению –4 соответствует экранная координата 0, значению 0 — координата 400 (середина), значению 4 — 800. Поэтому легко вывести правило перевода — `x*100+400`. Для `y` все аналогично.

Можете теперь попробовать поменять функцию в подпрограмме `F()` — вместо синуса вписать туда какую-нибудь параболу, гиперболу или экспоненту — именно они и будут строиться.

Обработка событий

Мы подошли к крайне важному понятию — *событиям*. Попробуем разобраться, что это такое, на примере собаки. В самом начале книги мы уже обращались к собаке, как к обработчику команд. Помните — "Сидеть", "Лежать", "Фас"? Это команды, которые обученная собака может выполнять.

Но что будет делать собака, если что-то происходит, но хозяина рядом нет, поэтому никто не может дать команду? Собака все равно будет реагировать. Например, если к собаке подойдет незнакомый человек — сработает событие "Подошел чужой". У каждой собаки, конечно, на это событие "прописаны" разные действия, но обычно это что-то вроде "Зарычать", "Залаять" или даже "Укусить". Другое событие — "Подошел хозяин". На него у собаки запрограммированы совершенно другие действия — "Вилять хвостом", "Прыгать от счастья" и т. д. Таким образом, если события запрограммированы, то реакция на них происходит автоматически.

В программировании события тоже применяются повсеместно. Вообще, с тех пор, как появились самые первые операционные системы с графическим интерфейсом, события стали основой программирования. Посмотрите, в Windows абсолютно все завязано на события! Щелчок левой кнопкой мыши, щелчок правой кнопкой мыши, двойной щелчок — это все события. И в каждой программе на эти события есть обработчики. То есть при наступлении события "щелчок левой кнопкой мыши" на кнопке эта кнопка нажимается, и т. д.

И в Small Basic, конечно же, есть поддержка обработки событий. Давайте начнем с самого простого примера.

Щёлк!

```
GraphicsWindow.MouseDown = OnMouseDown


Sub OnMouseDown
  GraphicsWindow.ShowMessage("Щёлк!", "Сообщение")
EndSub
```

Описание обработки события состоит из двух частей. Во-первых, мы пишем подпрограмму, которая делает какие-то действия (в примере — подпрограмма `OnMouseDown()`). Во-вторых, мы присваиваем событию вызов этой подпрограммы (`GraphicsWindow.MouseDown = OnMouseDown`). То есть событию `MouseDown` объекта `GraphicsWindow` присваивается обработчик — подпрограмма `OnMouseDown()`. Она делает одно-единственное действие — показывает окно с сообщением "Щёлк!"

И сколько бы раз мы не щелкали по графическому окну — каждый раз в ответ на это будет показываться сообщение "Щёлк!"

В Small Basic есть три объекта, которые поддерживают обработку событий:

- `Controls`;
- `GraphicsWindow`;
- `Timer`.

Про объекты `Controls` и `Timer` мы будем говорить позже, поэтому пока ограничимся только событиями графического окна (табл. 3.7).

Таблица 3.7

Событие	**Когда происходит**
`GraphicsWindow.KeyDown`	При нажатии клавиши
`GraphicsWindow.KeyUp`	При отпускании клавиши
`GraphicsWindow.MouseDown`	При нажатии кнопки мыши
`GraphicsWindow.MouseMove`	При перемещении мыши
`GraphicsWindow.MouseUp`	При отпускании кнопки мыши
`GraphicsWindow.TextInput`	При вводе текста

Кстати

На клавиатуре — клавиши, а у мыши — кнопки. Не вздумайте перепутать!

Для того чтобы правильно обрабатывать события, нам понадобятся специальные свойства объекта `GraphicsWindow` (табл. 3.8).

Таблица 3.8

Свойство	**Описание**
`GraphicsWindow.LastKey`	Нажатая клавиша
`GraphicsWindow.LastText`	Введенный текст
`GraphicsWindow.MouseX`	Координата *X* курсора мыши
`GraphicsWindow.MouseY`	Координата *Y* курсора мыши

Еще существует объект `Mouse`, у которого есть несколько полезных свойств. Например, только с его помощью можно определить, какая кнопка нажата — левая или правая. Но о нем тоже поговорим немного позже.

Теперь, зная все это, давайте немного усложним наш пример. Нарисуем красный квадрат.

Красный квадрат

```
GraphicsWindow.BrushColor = "Red"
GraphicsWindow.FillRectangle(50,50,50,50)

GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  If GraphicsWindow.MouseX >= 50 And GraphicsWindow.MouseX
<= 100 And GraphicsWindow.MouseX >= 50 And
GraphicsWindow.MouseX <= 100 Then
    GraphicsWindow.ShowMessage("Квадрат!","Сообщение")
  EndIf
EndSub
```

Посмотрите — обработчик выводит сообщение только тогда, когда мы щелкаем по квадрату. Чтобы определить, попали мы в квадрат или нет, проверяются координаты мыши в момент щелчка — `MouseX` и `MouseY`. Если они находятся в диапазоне от 50 до 100 (именно там находится наш квадрат), то сообщение выводится.

Кляксы 2.0

Помните, мы рисовали кляксы случайными цветами? Давайте вернемся к тому примеру и добавим в него обработчик события `MouseDown`, который будет сообщать нам цвет выбранной кляксы.

```
Sub OnMouseDown
  mx = GraphicsWindow.MouseX
  my = GraphicsWindow.MouseY
  color = GraphicsWindow.GetPixel(mx,my)
  If color <> "#000000" Then
    GraphicsWindow.ShowMessage(color,"Цвет кляксы")
  EndIf
EndSub
```

В этом обработчике мы определяем координаты мыши, записываем их в переменные `mx` и `my`. Затем с помощью метода `GetPixel()` определяем цвет пиксела, расположенного в этих са-

мых координатах. Сравниваем получившийся цвет с белым ("#000000"). И, если мы попали не на белый фон, а на цветную кляксу — выводим ее цвет. Конечно, в модели RGB.

Какая клавиша нажата?

Сменим ненадолго мышь на клавиатуру и научимся определять, какая клавиша была нажата.

```
GraphicsWindow.KeyDown = OnKeyDown


Sub OnKeyDown
  key = GraphicsWindow.LastKey
  GraphicsWindow.ShowMessage(key, "Клавиша")
EndSub
```

> ***Кстати***
>
> Хоть мы везде и называем обработчики событий стандартно — `OnKeyDown`, `OnMouseDown` — это совершенно не обязательно. Можно назвать подпрограмму как угодно. BlaBlaBla или xKLghb65DR — вполне допустимые варианты.

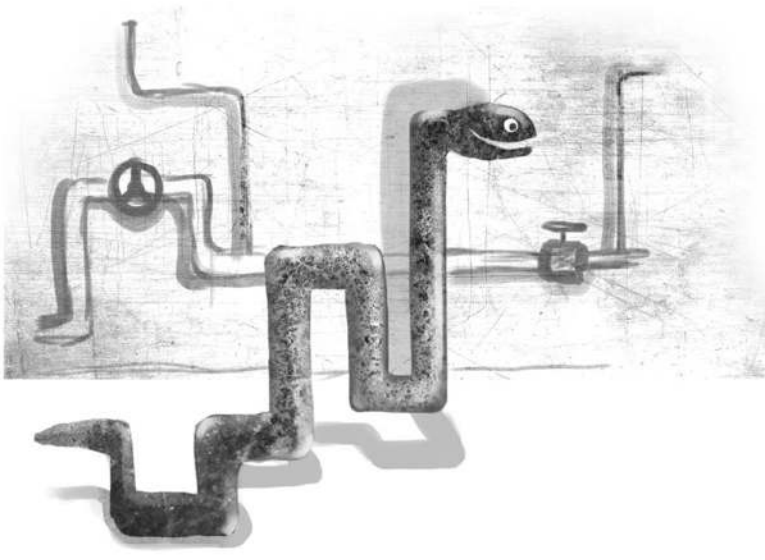
Обязательно попробуйте поэкспериментировать с этой программой и понажимать разные клавиши. Вы увидите, как они называются внутри Small Basic. Некоторые названия вполне очевидны. Так, стрелки называются соответственно Up, Down, Left, Right. Пробел — Space. А вот, к примеру, Caps Lock называется Capital, а клавиша с тильдой (~) и буквой "Ё" — вообще Oem3.

В общем, если в какой-то программе вам нужно будет обрабатывать нажатия разных клавиш — узнайте предварительно, как эти клавиши называет Small Basic.

Snake

Если у вас когда-нибудь был (или даже есть) телефон Nokia, то вы должны знать об игре Snake, в которой длинная змея ползает по экрану и ест разные фрукты. Давайте сделаем сверхпро-

стой вариант этой игры на Small Basic. Наша змея пока не станет заглатывать фрукты и даже не будет погибать, если укусит себя за хвост. Она просто будет расти.



```
x = 250
y = 250

GraphicsWindow.KeyDown = OnKeyDown

Sub OnKeyDown
  x0 = x
  y0 = y
  key = GraphicsWindow.LastKey
  If key = "Up" Then
    y = y - 10
  ElseIf key = "Down" Then
    y = y + 10
  ElseIf key = "Left" Then
```

```
    x = x - 10
  ElseIf key = "Right" Then
    x = x + 10
  EndIf
  GraphicsWindow.DrawLine(x0,y0,x,y)
EndSub
```

Текущие координаты головы змеи будем хранить в переменных `x` и `y`. Пусть змея начинает расти из точки (250, 250).

В обработчике `OnKeyDown()` станем пересчитывать новые координаты в зависимости от того, какая клавиша-стрелка нажата. Если <↑> (вверх), то уменьшим `y` на 10, если <↓> (вниз), то, наоборот, увеличим `y` на 10, и т. д. Но для того чтобы дорисовать змею, нам нужно знать и предыдущие координаты. Поэтому перед тем, как пересчитывать `x` и `y`, сохраним текущие значения в переменные `x0` и `y0`. Теперь, после пересчета, рисуем линию из точки (`x0`, `y0`) в точку (`x`, `y`).

Рисование мышью

Давайте теперь будем рисовать линии не с помощью клавиатуры, а мышью. Это делается практически так же.

```
GraphicsWindow.MouseMove = OnMouseMove


Sub OnMouseMove
    x = GraphicsWindow.MouseX
    y = GraphicsWindow.MouseY
    GraphicsWindow.DrawLine(x0,y0,x,y)
    x0 = x
    y0 = y
EndSub
```

Здесь мы обрабатываем событие `MouseMove`. То есть при каждом перемещении мыши будет вызываться подпрограмма `OnMouseMove()`. В ней мы записываем в переменные `x` и `y` текущие координаты мыши и рисуем линию из предыдущей точки (`x0`, `y0`) в текущую — (`x`, `y`).

Здесь есть две проблемы. Во-первых, изначально в переменных `x0` и `y0` ничего нет — поэтому рисование всегда будет начинаться из левого верхнего угла, т. е. из точки (0, 0). Во-вторых, остановить рисование нельзя — любое перемещение мыши будет рисовать линию, и никак иначе.

Усовершенствуем нашу рисовалку, добавив еще и обработку события `MouseDown`, чтобы рисование начиналось и останавливалось по щелчку.

```
f = "False"

GraphicsWindow.MouseMove = OnMouseMove
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  If f = "False" Then
    x0 = GraphicsWindow.MouseX
    y0 = GraphicsWindow.MouseY
    f = "True"
  Else
    f = "False"
  EndIf
EndSub

Sub OnMouseMove
  If f = "True" Then
    x = GraphicsWindow.MouseX
    y = GraphicsWindow.MouseY
    GraphicsWindow.DrawLine(x0,y0,x,y)
    x0 = x
    y0 = y
  EndIf
EndSub
```

Помогать нам будет флаг, который мы поместим в переменную `f`. От значения флага теперь зависит, можно рисовать (`"True"`) или нет (`"False"`). Для этого в обработчик `OnMouseMove()` добавляем усло-

вие — рисовать, только если флаг равен истине (`If f = "True"`). Изначально флаг имеет значение "Ложь" (`f = "False"`), а обработчик `OnMouseDown()` по щелчку меняет его значение.

Задания

1. Сделайте так, чтобы круг менял цвет по щелчку мыши, как светофор. То есть последовательно: зеленый, желтый, красный, снова желтый, зеленый и т. д.
2. Напишите программу, которая меняет цвет графического окна в соответствии с нажатой клавишей. Например, <R> — красный цвет, <B> — синий, <Y> — желтый и т. д.
3. В примере "Snake" нет проверки того, дошла ли змея до края окна. Попробуйте ее сделать!

Движение фигур

А теперь самое время познакомиться с замечательным объектом `Shapes`. Когда мы только начинали изучать графику, то рисовали прямоугольники, эллипсы, треугольники, выводили текст. Все это было прекрасно, но статично. Добавить динамики позволит объект `Shapes`, который умеет двигать фигуры.

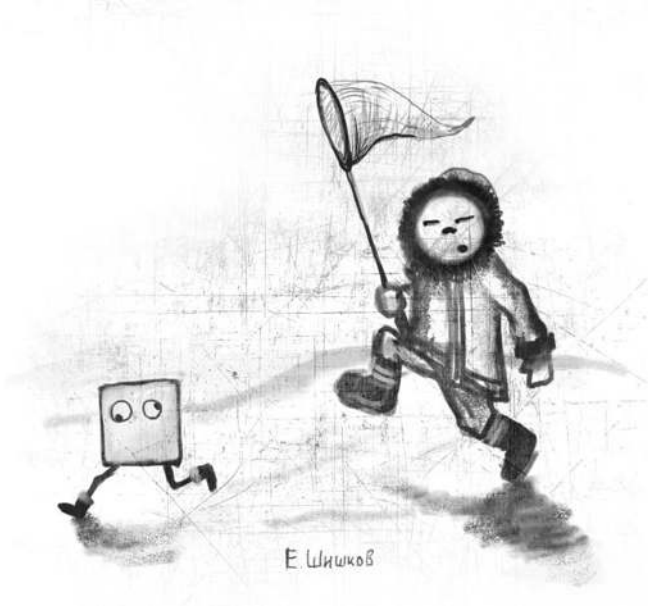
Начнем с простейшего примера. Нарисуем квадрат и заставим его бегать за курсором мыши.

Бегающий квадрат

```
square = Shapes.AddRectangle(100,100)

GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  Shapes.Move(square,x,y)
EndSub
```

Для добавления квадрата (прямоугольника с равными сторонами, конечно) пользуемся методом `AddRectangle()`. У него есть два очень важных отличия от методов `DrawRectangle()` и `FillRectangle()` объекта `GraphicsWindow`.

Во-первых, не указываются координаты квадрата. Он создается в левом верхнем углу графического окна и только там. Если нужно, чтобы он появился в другом месте — нужно сразу же его туда переместить.



Во-вторых, `AddRectangle()` создает новый отдельный объект. Обратите внимание, что мы пользуемся оператором присваивания: созданный объект размещаем в переменной `square`. Теперь `square` — это как бы указатель, ссылка, на наш квадрат. И через этот указатель, переменную `square`, с квадратом можно делать все, что угодно. Например, двигать — с помощью обработчика события `MouseMove`. В нем мы получаем координаты мыши и с помощью метода `Move()` перемещаем в эти координаты квадрат.

Летающий квадрат

Давайте теперь заставим квадрат летать самостоятельно, по случайной траектории.

```
square = Shapes.AddRectangle(100,100)


While 1 = 1
  x = Math.GetRandomNumber(500)
  y = Math.GetRandomNumber(500)
  Shapes.Move(square,x,y,300)
  Program.Delay(300)
EndWhile
```

Для этого сделаем бесконечный цикл `While` с условием `1=1`. Это условие будет истинно всегда, поэтому цикл будет выполняться вечно. Точнее, до тех пор, пока мы не закроем программу.

На каждом шаге цикла определяем новые случайные координаты `x` и `y` и перемещаем туда квадрат. После этого вставляем небольшую задержку — 300 миллисекунд будет вполне достаточно, чтобы увидеть, куда в очередной раз прыгнул квадрат.

Да, он именно прыгает — метод `Move()` просто перемещает графический объект из одного места в другое. А для того чтобы квадрат красиво пролетел, нужно использовать вместо `Move()` метод `Animate()`:

```
Shapes.Animate(square,x,y,300)
```

Заметьте, добавился еще один параметр — время полета, которое мы также определили в 300 миллисекунд.

Летающий квадрат с таймером

Бесконечный цикл — это, конечно, элегантное решение, но есть более корректный способ — объект `Timer`. Это, как не трудно догадаться, *таймер*, который позволяет выполнять какие-либо действия с определенным периодом.

Сделаем то же самое, что в предыдущем примере, но с таймером.

```
square = Shapes.AddRectangle(100,100)

Timer.Tick = OnTick
Timer.Interval = 300

Sub OnTick
  x = Math.GetRandomNumber(500)
  y = Math.GetRandomNumber(500)
  Shapes.Animate(square,x,y,300)
EndSub
```

У объекта `Timer` есть лишь одно событие — `Tick`, т. е. "тиканье", и лишь одно свойство — `Interval`, т. е. интервал этого самого тиканья. И если мы ставим `Interval` равным 300 миллисекунд, то таймер будет тикать каждые 0,3 секунды. И, соответственно, каждый 0,3 секунды будет срабатывать событие `Tick`.

Резиновые стены

А сейчас пускай фигура летает по графическому окну и отталкивается от его краев, как будто они резиновые. Для разнообразия пусть это будет не квадрат, а шарик.

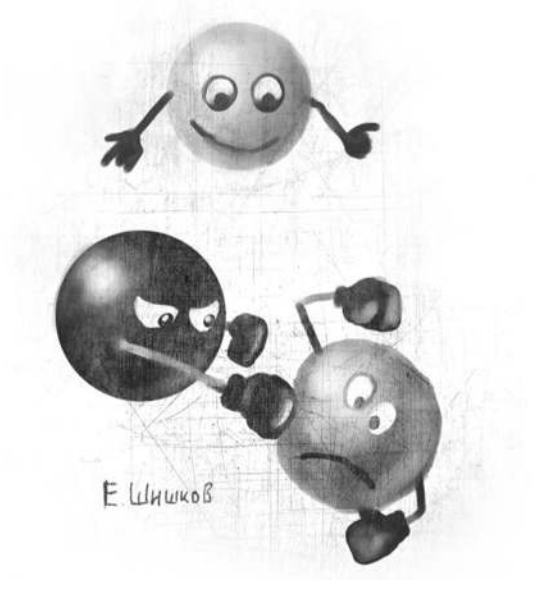
```
circle = Shapes.AddEllipse(100,100)
dx = 5
dy = 5

Timer.Tick = OnTick
Timer.Interval = 50

Sub OnTick
  x = x + dx
  y = y + dy
  If x <= 0 Or x+100 >= GraphicsWindow.Width Then
    dx = - dx
  EndIf
  If y <= 0 Or y+100 >= GraphicsWindow.Height Then
    dy = - dy
  EndIf
  Shapes.Animate(circle,x,y,50)
EndSub
```

Основную роль здесь играют переменные `dx` и `dy` — они определяют направление движения шарика. Если `dx` и `dy` положительны, то шарик летит вниз и вправо (т. к. и `x`, и `y` увеличиваются). Если `dx>0`, а `dy<0`, то шарик полетит вверх и вправо. И т. д. — всего четыре варианта. Мы присвоили им значение 5. Это значит, что за каждый "тик" таймера шарик будет перемещаться на 5 пикселов по горизонтали и на 5 пикселов по вертикали.

Когда шарик ударяется о стенку, действует закон "Угол падения равен углу отражения". Соответственно, если шарик ударился о нижний или верхний край, то изменится направление `dy`, а если удар будет в левый или правый край, то поменяется направление `dx`.

Куча-мала

А теперь пусть летает и отталкивается от стен не один шарик, а сразу десять. Как мы будем ими командовать? Конечно же, с помощью массива!




```
For i = 1 to 10
  color = GraphicsWindow.GetRandomColor()
  GraphicsWindow.BrushColor = color
  Circles[i] = Shapes.AddEllipse(50,50)
  X[i] = Math.GetRandomNumber(GraphicsWindow.Width)
  Y[i] = Math.GetRandomNumber(GraphicsWindow.Height)
  Dx[i] = 5
  Dy[i] = 5
EndFor

Timer.Tick = OnTick
Timer.Interval = 50
```

```
Sub OnTick
  For i = 1 To 10
    X[i] = X[i] + Dx[i]
    Y[i] = Y[i] + Dy[i]
    If X[i] <= 0 Or X[i]+50 >= GraphicsWindow.Width Then
      Dx[i] = - Dx[i]
    EndIf
    If Y[i] <= 0 Or Y[i]+50 >= GraphicsWindow.Height Then
      Dy[i] = - Dy[i]
    EndIf
    Shapes.Animate(Circles[i],X[i],Y[i],50)
  EndFor
EndSub
```

Посмотрите внимательно — и алгоритм тот же, и все переменные те же, только теперь они стали массивами. То есть у нас теперь не одна фигура `circle`, а целый массив фигур `Circles`. Соответственно и массивы `X`, `Y`, `Dx`, `Dy` хранят информацию о координатах и направлениях движения всех десяти шариков.

Создаются шарики в случайных местах. С этим связана небольшая проблема: если шарик появляется слишком близко к краю окна, то он к этому краю "прилипает". Попробуйте сами разобраться, из-за чего так происходит и как это можно побороть!

Задания

1. Переделайте программу с "бегающим квадратом" так, чтобы он реагировал не на мышь, а на клавиатуру.
2. Напишите программу для "перетаскиваемого квадрата", т. е. он должен двигаться, только если взять его мышью и тащить, как значок на рабочем столе.
3. Разработайте теперь полноценный светофор из трех кругов, которые будут по очереди загораться красным, желтым и зеленым цветом. Обязательно учтите то, что желтый горит меньше остальных. Можете еще добавить мигающий зеленый перед желтым. И даже обратный отсчет секунд, если получится!
4. Попробуйте создать модель "Солнце — Земля — Луна". Солнце будем считать неподвижным. Вокруг него вращается Земля, а вокруг Земли — Луна.

Элементы интерфейса

В каждой серьезной программе есть *интерфейс*, т. е. такие элементы, с помощью которых можно программой управлять. Стандартные элементы интерфейса — это кнопки и текстовые поля. В мощных профессиональных средах программирования есть, конечно, еще десятки других элементов — всяческие списки, полосы прокрутки, таблицы, флажки и т. д. Но в Small Basic таких элементов всего два — и для простых задач их, честно говоря, вполне хватает. Вот они:

- `Button` (Кнопка);
- `TextBox` (Текстовое поле).

А отвечает за работу с этими элементами объект `Controls`, у которого есть несколько методов, свойств и событий (табл. 3.9—3.11).

Таблица 3.9

Метод	Описание
`Controls.AddButton(left,top)`	Добавляет новую кнопку
`Controls.AddTextBox(left,top)`	Добавляет новое текстовое поле
`Controls.AddMultiLineTextBox (left,top)`	Добавляет новое текстовое поле из нескольких строк
`Controls.SetSize (control,width,height)`	Устанавливает размеры элемента
`Controls.SetButtonCaption (buttonName,caption)`	Устанавливает надпись на кнопке
`Controls.SetTextBoxText (textBoxName,caption)`	Устанавливает текст поля
`Controls.GetButtonCaption (buttonName)`	Получает текущую надпись на кнопке
`Controls.GetTextBoxText (textBoxName)`	Получает текущий текст поля
`Controls.ShowControl(controlName)`	Показывает элемент
`Controls.HideControl(controlName)`	Скрывает элемент
`Controls.Move(control,x,y)`	Перемещает элемент
`Controls.Remove(controlName)`	Удаляет элемент

Таблица 3.10

Свойство	Описание
`Controls.LastClickedButton`	Последняя нажатая кнопка
`Controls.LastTypedText`	Последний введенный текст

Таблица 3.11

Событие	Описание
Controls.ButtonClicked	Нажата кнопка
Controls.TextTyped	Введен текст

Давайте для начала напишем самую простую программу с одной кнопкой, при нажатии которой выводится сообщение:

```
button = Controls.AddButton("Кнопка",100,100)

Controls.ButtonClicked = OnButtonClicked

Sub OnButtonClicked
  GraphicsWindow.ShowMessage("Нажата кнопка!", "Сообщение")
EndSub
```

Все действительно очень просто. Создается кнопка `button` с банальной подписью "Кнопка" (100, 100 — координаты кнопки, как вы уже догадались). Затем определяется обработчик события `ButtonClicked`, который состоит из одной лишь строки — вывода сообщения о том, что кнопка нажата.

Давайте теперь сделаем вместо одной кнопки две:

```
buttonLeft = Controls.AddButton("Левая кнопка",100,100)
buttonRight = Controls.AddButton("Правая кнопка",300,100)
```

Появилась проблема: при нажатии и на ту, и на другую кнопку появляется одинаковое сообщение — "Нажата кнопка!" Это, конечно, непорядок. Нужно в обработчик добавить условие, которое будет проверять, какая именно кнопка нажата:

```
Sub OnButtonClicked
  If Controls.LastClickedButton = buttonLeft Then
    GraphicsWindow.ShowMessage("Нажата левая кнопка!",
"Сообщение")
  ElseIf Controls.LastClickedButton = buttonRight Then
    GraphicsWindow.ShowMessage("Нажата правая кнопка!",
"Сообщение")
  EndIf
EndSub
```

Давайте теперь придумаем несколько интересных примеров. Заодно, по ходу дела, разберемся и с текстовыми полями.

Инспектор цветов

Помните, мы разбирались в RGB-модели? Давайте напишем простую программку, которая поможет нам проводить с ней любые эксперименты. Для этого добавим текстовое поле (в него будем вводить RGB-код цвета) и кнопку (она будет изменять цвет графического окна):

```
textColor = Controls.AddTextBox(10,10)
buttonChangeColor = Controls.AddButton("Показать цвет",
10,40)

Controls.ButtonClicked = OnButtonClicked

Sub OnButtonClicked
  color = Controls.GetTextBoxText(textColor)
  GraphicsWindow.BackgroundColor = color
EndSub
```

Замена `TextWindow`

С помощью графического окна и элементов интерфейса можно выполнять те же действия, что и с помощью черного окна — `TextWindow`. Реализуем самый-самый простой пример — сложение двух чисел.

```
GraphicsWindow.DrawText(10,10,"Первое число")
GraphicsWindow.DrawText(10,40,"Второе число")
GraphicsWindow.DrawText(10,70,"Сумма")
textNumber1 = Controls.AddTextBox(100,10)
textNumber2 = Controls.AddTextBox(100,40)
textSum = Controls.AddTextBox(100,70)
buttonSum = Controls.AddButton("Сложить!",300,10)

Controls.ButtonClicked = OnButtonClicked

Sub OnButtonClicked
```

```
    a = Controls.GetTextBoxText(textNumber1)
    b = Controls.GetTextBoxText(textNumber2)
    c = a + b
    Controls.SetTextBoxText(textSum,c)
EndSub
```

Видите — вроде бы все просто, но сколько кода! Поэтому мы и начали изучать программирование с черного окна. Представьте себе, если бы в самом начале книги мы начали разбирать такой код. С текстовыми полями, кнопкой, обработчиком события, координатами всех этих элементов... Ведь суть алгоритма здесь в одной-единственной строчке! `c = a + b` — и все! А дополнительных, "интерфейсных" действий — целых 13 строк кода.

Конечно, Small Basic в деле создания интерфейса далеко до старшего брата — среды Visual Studio. Главный недостаток в том, что для каждого элемента надо вручную прописывать координаты и размеры, предварительно высчитав их на листочке. Но для того, чтобы научиться — в самый раз!

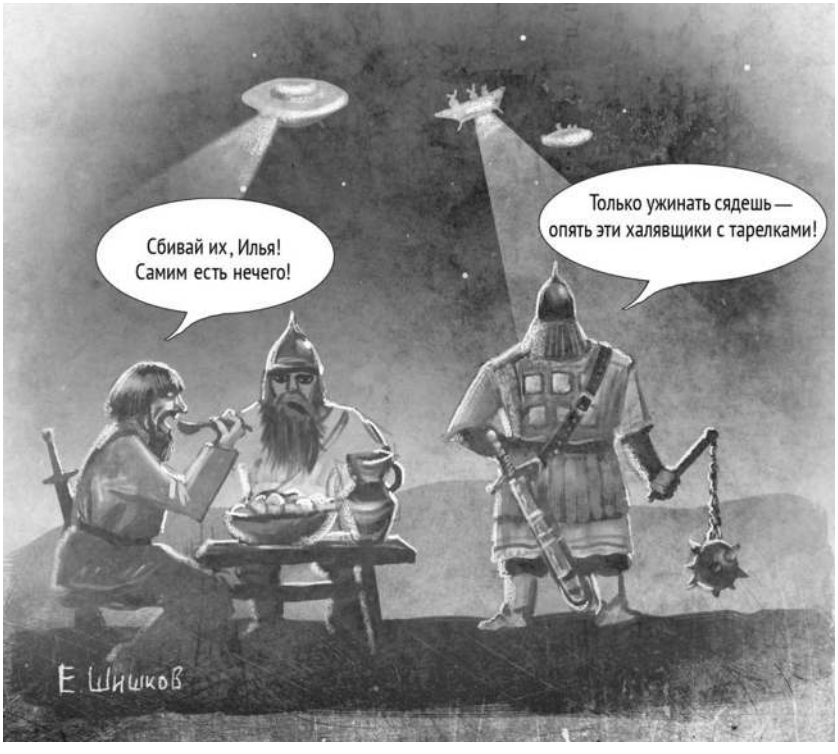
Задания

1. Усовершенствуйте пример "Замена `TextWindow`", создав более навороченный калькулятор: добавьте операции вычитания, умножения и деления и какие-нибудь математические функции (квадратный корень, синус, косинус).
2. Напишите программу для добавления данных в текстовый файл: пользователь вводит текст в поле, нажимает кнопку — текст дописывается в файл.
3. А теперь напишите простейшую программу-блокнот. Вам понадобятся: текстовое поле (для ввода имени файла), кнопка **Открыть файл**, кнопка **Сохранить файл** и многострочное текстовое поле (`MultiLineTextBox`) для редактирования файла.

Игра

А теперь обобщим все, что изучили — графику, события, движения фигур — и напишем настоящую игру! Это будет шутер, т. е. стрелялка, наподобие старой доброй Galaxian. Сюжет

прост. У нас есть космический корабль, он находится где-то в космосе, и его атакуют корабли пришельцев. Задача игрока — расстреливать всех противников.



Будем реализовывать игру последовательно, по шагам.

Для начала настроим графическое окно. Придадим ему размеры 800×600 пикселов и сделаем его черным (космос же!):

```
GraphicsWindow.Height = 600
GraphicsWindow.Width = 800
GraphicsWindow.CanResize = "False"
GraphicsWindow.Title = "Игра"
GraphicsWindow.BackgroundColor = "Black"
```

Затем создадим корабль. Пока что для простоты изобразим его в виде зеленого прямоугольника. Рассчитаем начальные координаты корабля так, чтобы он был в середине нижней части экрана:

```
GraphicsWindow.BrushColor = "Green"
ship = Shapes.AddRectangle(50,50)
shipX = 375
shipY = 550
Shapes.Move(ship,shipX,shipY)
```

Корабль должен перемещаться влево и вправо. Напишем обработчик события `KeyDown`, чтобы он реагировал на стрелки:

```
GraphicsWindow.KeyDown = OnKeyDown

Sub OnKeyDown
  If GraphicsWindow.LastKey = "Left" Then
    If shipX > 10 Then
      shipX = shipX - 10
      Shapes.Move(ship,shipX,shipY)
    EndIf
  ElseIf GraphicsWindow.LastKey = "Right" Then
    If shipX < 750 Then
      shipX = shipX + 10
      Shapes.Move(ship,shipX,shipY)
    EndIf
  EndIf
EndSub
```

Обработчик простой — при нажатии клавиши <←> корабль перемещается на 10 пикселов влево (`shipX = shipX - 10`), при нажатии клавиши <→> — на 10 пикселов вправо (`shipX = shipX + 10`). Чтобы он не улетел за пределы экрана, добавляем простые проверки: `shipX > 10` и `shipX < 750`.

С кораблем разобрались, идем дальше. Кроме корабля нам нужны еще два вида объектов — враги и пули, которыми наш корабль стреляет. Опять же для простоты враги будут синими квадратами, а пули — красными кругами. Нам обязательно придется считать количество и тех, и других объектов, поэтому заве-

дем для этого специальные переменные — `cntBullets` (счетчик пуль) и `cntEnemies` (счетчик противников):

```
cntBullets = 0
cntEnemies = 0
```

Сразу предусмотрим и подсчет очков. Хранить счет будем в переменной `total`, выводить — в углу экрана:

```
total = 0
GraphicsWindow.BrushColor = "White"
GraphicsWindow.FontSize = 24
score = Shapes.AddText(total)
Shapes.Move(score,700,10)
```

Теперь научим корабль стрелять. Напишем подпрограмму `CreateBullet()`, которая будет создавать пулю:

```
Sub CreateBullet
  cntBullets = cntBullets + 1
  GraphicsWindow.BrushColor = "Red"
  Bullets[cntBullets] = Shapes.AddEllipse(25,25)
  Shapes.Move(Bullets[cntBullets],shipX+15,525)
EndSub
```

Информацию о пулях будем хранить в массиве фигур `Bullets`. Увеличиваем счетчик пуль на единицу и создаем новую пулю в массиве. Размещаем ее точно над текущим положением корабля.

За выстрел у нас традиционно будет отвечать пробел. Поэтому в обработчике `OnKeyDown` добавляем еще одно условие:

```
  ElseIf GraphicsWindow.LastKey = "Space" Then
    createBullet()
  EndIf
```

Точно так же пишем подпрограмму для создания врагов:

```
Sub CreateEnemy
  cntEnemies = cntEnemies + 1
  GraphicsWindow.BrushColor = "Blue"
  Enemies[cntEnemies] = Shapes.AddRectangle(50,50)
  enemyX = Math.GetRandomNumber(749)+1
  Shapes.Move(Enemies[cntEnemies],enemyX,10)
EndSub
```

Информацию о врагах будем тоже хранить в массиве, назовем его `Enemies`. Враги должны появляться в разных местах, поэтому координата *X* должна быть случайным числом в интервале от 1 до 750.

Теперь процедура создания врагов есть, но по какому событию ее запускать? Явно не по нажатию клавиши — враги должны появляться сами по себе. Поэтому без таймера не обойтись:

```
Timer.Interval = 50
Timer.Tick = OnTick


Sub OnTick
  rndNum = Math.GetRandomNumber(30)
  If rndNum = 1 Then
    createEnemy()
  EndIf
EndSub
```

Событие таймера срабатывает через равные промежутки времени. Но если враги будут появляться равномерно, будет неинтересно. Поэтому надо и сюда внести элемент случайности! На каждом тике таймера будем генерировать случайное число от 1 до 30. И только если оно окажется равно 1 (т. е. с вероятностью 1/30), будем создавать врага.

Итак, что мы уже имеем? Корабль перемещается, создаются пули, сами собой появляются враги. Теперь сделаем так, чтобы стали двигаться пули и враги. Таймер у нас уже есть. Нужно, чтобы на каждом его тике каждая пуля пролетала несколько пикселов вверх, а каждый враг — вниз. Для движения пуль напишем процедуру `MoveBullets()`:

```
Sub MoveBullets
  For i = 1 To cntBullets
    xBullet = Shapes.GetLeft(Bullets[i])
    yBullet = Shapes.GetTop(Bullets[i])
    If yBullet > 0 Then
      Shapes.Move(Bullets[i],xBullet,yBullet-10)
    Else
```

```
        Shapes.Remove(Bullets[i])
      EndIf
    EndFor
  EndSub
```

Вы же помните — у нас одновременно должно существовать (и обрабатываться!) множество пуль. Ведь нельзя ждать, когда пролетит и исчезнет первая пуля — надо стрелять и стрелять! Поэтому информация обо всех существующих пулях хранится в массиве `Bullets`. Счетчик, соответственно, находится в переменной `cntBullets`. Поэтому для того чтобы переместить все пули, надо сделать это в цикле.

За один тик таймера будем перемещать пулю на 10 пикселов вверх (`Shapes.Move(Bullets[i],xBullet,yBullet-10)`). Конечно, при этом нужна проверка: если пуля ушла за пределы экрана (координата `yBullet` стала отрицательной), ее необходимо удалить (`Shapes.Remove(Bullets[i])`).

Аналогично выглядит процедура перемещения врага `MoveEnemies()`:

```
  Sub MoveEnemies
    For i = 1 To cntEnemies
      xEnemy = Shapes.GetLeft(Enemies[i])
      yEnemy = Shapes.GetTop(Enemies[i])
      If yEnemy < 550 Then
        Shapes.Move(Enemies[i],xEnemy,yEnemy+3)
      Else
        GameOver()
      EndIf
    EndFor
  EndSub
```

Разница в том, что враги двигаются медленнее (всего 3 пиксела за тик). И, конечно, если врагу удалось добраться до нижней границы экрана, то все — игра закончена:

```
  Sub GameOver
    Timer.Pause()
    GraphicsWindow.BrushColor = "Yellow"
```

```
    GraphicsWindow.FontSize = 48
    GraphicsWindow.DrawText(100,100,"Игра окончена!")
  EndSub
```

Теперь вызовы процедур `MoveBullets()` и `MoveEnemies()` нужно добавить в обработчик таймера и — да, они летают!

Остался самый последний этап — непосредственно ликвидация врагов. Если пуля пересекается с врагом, оба объекта должны исчезнуть, а количество подбитых врагов вырасти на единицу. Для проверки соприкосновения объектов напишем процедуру `CheckHit()`:

```
Sub CheckHit
  For i = 1 To cntBullets
    For j = 1 To cntEnemies
      xBullet = Shapes.GetLeft(Bullets[i])
      yBullet = Shapes.GetTop(Bullets[i])
      xEnemy = Shapes.GetLeft(Enemies[j])
      yEnemy = Shapes.GetTop(Enemies[j])
      If xBullet > 0 and yBullet > 0 And xBullet >= xEnemy-
25 And xBullet <= xEnemy+50 And yBullet <= yEnemy+50 Then
        Shapes.Remove(Bullets[i])
        Shapes.Remove(Enemies[j])
        total = total + 1
        Shapes.SetText(score,total)
      EndIf
    EndFor
  EndFor
EndSub
```

Идея проста — мы должны проверить пересечение координат каждой пули и каждого врага. Именно по этой причине не обойтись без вложенного цикла. Во внешнем цикле по `i` идут пули, во внутреннем по `j` — враги. Для каждой пары делаем проверку по координатам (`xBullet >= xEnemy-25 And xBullet <= xEnemy+50 And yBullet <= yEnemy+50`), и если пересечение имеется, то — Бум! — и пуля, и враг исчезают, а счетчик врагов увеличивается.

Обратите внимание на два дополнительных условия (`xBullet > 0 and yBullet > 0`). Они нужны для того, чтобы не обрабатывать удаленные ранее объекты.

Добавим теперь вызов процедуры `CheckHit()` в обработчик таймера — и все, можно играть!

Вот так будет выглядеть наша игра целиком:

```
GraphicsWindow.Height = 600
GraphicsWindow.Width = 800
GraphicsWindow.CanResize = "False"
GraphicsWindow.Title = "Игра"
GraphicsWindow.BackgroundColor = "Black"

cntBullets = 0
cntEnemies = 0

GraphicsWindow.BrushColor = "Green"
ship = Shapes.AddRectangle(50,50)
shipX = 375
shipY = 550
Shapes.Move(ship,shipX,shipY)

total = 0
GraphicsWindow.BrushColor = "White"
GraphicsWindow.FontSize = 24
score = Shapes.AddText(total)
Shapes.Move(score,700,10)

GraphicsWindow.KeyDown = OnKeyDown
Timer.Interval = 50
Timer.Tick = OnTick

Sub OnKeyDown
  If GraphicsWindow.LastKey = "Left" Then
    If shipX > 10 Then
      shipX = shipX - 10
```

```
      Shapes.Move(ship,shipX,shipY)
    EndIf

  ElseIf GraphicsWindow.LastKey = "Right" Then
    If shipX < 750 Then
      shipX = shipX + 10
      Shapes.Move(ship,shipX,shipY)
    EndIf

  ElseIf GraphicsWindow.LastKey = "Space" Then
    CreateBullet()
  EndIf
EndSub

Sub OnTick
  CheckHit()
  MoveBullets()
  MoveEnemies()
  rndNum = Math.GetRandomNumber(30)
  If rndNum = 1 Then
    CreateEnemy()
  EndIf
EndSub


Sub CreateBullet
  cntBullets = cntBullets + 1
  GraphicsWindow.BrushColor = "Red"
  Bullets[cntBullets] = Shapes.AddEllipse(25,25)
  Shapes.Move(Bullets[cntBullets],shipX+15,525)
EndSub

Sub CreateEnemy
  cntEnemies = cntEnemies + 1
  GraphicsWindow.BrushColor = "Blue"
```

```
  Enemies[cntEnemies] = Shapes.AddRectangle(50,50)
  enemyX = Math.GetRandomNumber(749)+1
  Shapes.Move(Enemies[cntEnemies],enemyX,10)
EndSub

Sub CheckHit
  For i = 1 To cntBullets
    For j = 1 To cntEnemies
      xBullet = Shapes.GetLeft(Bullets[i])
      yBullet = Shapes.GetTop(Bullets[i])
      xEnemy = Shapes.GetLeft(Enemies[j])
      yEnemy = Shapes.GetTop(Enemies[j])
      If xBullet > 0 and yBullet > 0 And xBullet >= xEnemy-
25 And xBullet <= xEnemy+50 And yBullet <= yEnemy+50 Then
        Shapes.Remove(Bullets[i])
        Shapes.Remove(Enemies[j])
        total = total + 1
        Shapes.SetText(score,total)
      EndIf
    EndFor
  EndFor
EndSub

Sub MoveBullets
  For i = 1 To cntBullets
    xBullet = Shapes.GetLeft(Bullets[i])
    yBullet = Shapes.GetTop(Bullets[i])
    If yBullet > 0 Then
      Shapes.Move(Bullets[i],xBullet,yBullet-10)
    Else
      Shapes.Remove(Bullets[i])
    EndIf
  EndFor
EndSub

Sub MoveEnemies
```

```
  For i = 1 To cntEnemies
    xEnemy = Shapes.GetLeft(Enemies[i])
    yEnemy = Shapes.GetTop(Enemies[i])
    If yEnemy < 550 Then
      Shapes.Move(Enemies[i],xEnemy,yEnemy+3)
    Else
      GameOver()
    EndIf
  EndFor
EndSub

Sub GameOver
  Timer.Pause()
  GraphicsWindow.BrushColor = "Yellow"
  GraphicsWindow.FontSize = 48
  GraphicsWindow.DrawText(100,100,"Игра окончена!")
EndSub
```

Задания

1. Замените квадраты и круги настоящими изображениями космических кораблей и снарядов.
2. Сделайте исчезновение врагов более красочным, чтобы они, например, плавно уменьшались и только потом исчезали.
3. Добавьте кнопку **Пауза**, чтобы можно было остановить игру, а потом продолжить.
4. Сделайте возможность перемещать корабль влево и вправо не только клавиатурой, но и мышью.
5. Сделайте несколько видов врагов. Пусть они выглядят по-разному и летят с разной скоростью. Соответственно, и очков некоторые будут приносить меньше, а некоторые — больше.
6. Добавьте в игру уровни. Например, если игрок набрал 100 очков, скорости всех врагов стали выше. Набрал 200 — еще выше. И т. д.
7. Реализуйте сохранение рекордов в файл.
8. Напишите другую игру — "Арканоид". Суть игры в следующем. В верхней части окна находятся "кирпичи". В нижней — небольшая платформа, которая ездит влево и вправо (удобнее, чтобы она двигалась с помощью мыши). От платформы под

углом в 45° летит шарик. Шарик может ударяться в стены, кирпичи и платформу — от всех объектов он отскакивает под тем же углом в 45°. Причем, когда шарик попадает в кирпич, кирпич исчезает — за это дается одно очко. Если шарик улетел вниз (игрок не смог поймать его платформой) — игра окончена.

Другие объекты

Осталось еще несколько объектов Small Basic, которые иногда могут быть полезны. Вот они:

- `Clock`;
- `Desktop`;
- `Dictionary`;
- `Flickr`;
- `ImageList`;
- `Network`;
- `Program`;
- `Sound`;
- `Turtle`.

Давайте совсем кратко рассмотрим эти объекты.

Объект *Clock*

`Clock` — это, разумеется, часы. Соответственно, с помощью этого объекта можно узнавать все параметры текущей даты и времени. У объекта `Clock` нет ни одного метода (т. е. изменить дату и время не получится), но достаточно свойств (табл. 3.12).

Таблица 3.12

Свойства	Описание
`Clock.Date`	Текущая дата (например, "24.07.2011")
`Clock.Time`	Текущее время (например, "1:18:06")
`ClockYear`	Год
`Clock.Day`	День месяца (от 1 до 31)

Таблица 3.12 *(окончание)*

Свойства	Описание
`Clock.Hour`	Час (от 0 до 23)
`Clock.Minute`	Минута (от 0 до 59)
`Clock.Second`	Секунда (от 0 до 59)
`Clock.Millisecond`	Миллисекунда (одна тысячная секунды — от 0 до 999)
`Clock.WeekDay`	День недели (от "понедельник" до "воскресенье")
`Clock.ElapsedMilliseconds`	Число миллисекунд, прошедших с 1 января 1900 года

Немного странным здесь может выглядеть только последнее свойство — `ElapsedMilliseconds`, т. е. число миллисекунд с момента 0:00:00 01.01.1900. В UNIX-подобных операционных системах время всегда отсчитывается с 1 января 1970 года — именно эта дата является началом "эры UNIX". Поэтому, если вы захотите узнать текущую дату в Linux или FreeBSD и введете команду `date`, то получите именно число миллисекунд с начала UNIX Epoch.

Почему в Small Basic идет отсчет с 1900 года? Наверное, потому что корпорация Microsoft решила выбрать другой год для начала "эры", а 1900-й — вполне логичный выбор. А вот зачем в Small Basic вообще нужно это свойство — уже другой вопрос.

Объект *Desktop*

Это очень простой объект, отвечающий за рабочий стол Windows. Имеет всего два свойства и один метод (табл. 3.13 и 3.14).

Таблица 3.13

Свойство	Описание
`Desktop.Width`	Ширина рабочего стола
`Desktop.Height`	Высота рабочего стола

Таблица 3.14

Метод	Описание
`Desktop.SetWallPaper(fileOrUrl)`	Устанавливает фоновую картинку (обои) рабочего стола

Ширина и высота рабочего стола — это, на самом деле, ширина и высота всего экрана в пикселах, т. е. разрешение экрана. Эти параметры бывает полезно узнать для того, чтобы, например, рассчитать подходящие размеры графического окна. А менять обои с помощью метода `SetWallPaper()` — это, конечно, просто развлечение.

Объект *Dictionary*

Толковый словарь, в котором есть значения разных слов. К сожалению, пока только английских и французских.

Объект *Flickr*

Flickr — это известный интернет-сервис для хранения фотографий. Соответственно, в архивах Flickr находится огромное количество самых разных картинок, которые можно вытаскивать из Small Basic.

У объекта `Flickr` есть два метода (табл. 3.15).

Таблица 3.15

Метод	Описание
`Flickr.GetPictureOfMoment()`	Получает случайную картинку из Flickr
`Flickr.GetRandomPicture(tag)`	Получает случайную картинку по какой-нибудь теме (`tag`)

Разница между методами небольшая. Первый берет из архивов Flickr абсолютно случайную картинку, а второй — тоже случайную, но по какой-то теме, которую можно задать с помощью параметра `tag`. Тег, кстати, лучше писать по-английски.

Например, вот такая программка выведет случайную картинку с ослом[1]:

```
image = Flickr.GetRandomPicture("donkey")
GraphicsWindow.DrawImage(image,0,0)
```

Объект *ImageList*

Единственное, для чего нужен объект `ImageList`, — это для определения размеров изображений с помощью двух свойств (табл. 3.16).

Таблица 3.16

Свойство	Описание
`ImageList.GetHeightOfImage`	Высота изображения
`ImageList.GetWidthOfImage`	Ширина изображения

Например, ими можно пользоваться, чтобы изменить размеры окна в соответствии с подгруженной картинкой:

```
image = Flickr.GetRandomPicture("donkey")
h = ImageList.GetHeightOfImage(image)
w = ImageList.GetWidthOfImage(image)
GraphicsWindow.Height = h
GraphicsWindow.Width = w
GraphicsWindow.DrawImage(image,0,0)
```

Есть, правда, еще метод `LoadImage()`, который загружает картинку в память (к ней затем можно обращаться через переменную), но его смысл не вполне ясен.

Объект *Network*

Это еще один простой объект — для загрузки файлов из Интернета. Имеет два метода (табл. 3.17).

[1] Иногда могут проявляться курьезы. Например, при тестировании программ вывелась картинка с двумя мужчинами и человеком-роботом, играющими в игровые автоматы. А через 15 секунд — фотография с ослом. А потом опять с ослом, но другая фотография. Но как попал первый робот в этот ряд?.. — *Ред.*

Таблица 3.17

Метод	Описание
`Network.DownLoadFile(url)`	Загружает файл из Интернета и сохраняет его на диске
`Network.GetWebPageContents(url)`	Получает содержимое веб-страницы

Скорее всего, в следующих версиях Small Basic функциональность этого объекта будет расширена и он научится делать еще что-нибудь. А пока — можно загружать файлы и страницы.

Объект *Program*

Как легко понять из названия, этот объект позволяет управлять выполнением программы. Объект более полезен, чем предыдущие. Его методы и свойства представлены в табл. 3.18 и 3.19.

Таблица 3.18

Метод	Описание
`Program.Delay(milliSeconds)`	Приостанавливает программу на заданное число миллисекунд (пауза)
`Program.End()`	Завершает программу
`Program.GetArgument(index)`	Получает параметр, с которым была запущена программа

Таблица 3.19

Свойство	Описание
`Program.ArgumentCount`	Количество параметров
`Program.Directory`	Текущая папка при выполнении программы

Что такое параметры запуска? Когда мы сохраняем программу, она записывается в файл с расширением sb — это исходный код программы. Если мы теперь запустим сохраненную программу, то она будет скомпилирована и в том же каталоге сохранится файл с расширением exe — это исполняемый файл программы. Его можно запускать отдельно от оболочки Small Basic (для этого должен быть установлен .NET Framework). Причем при запуске этого файла можно передавать ему параметры. Проще всего это сделать из командной строки Windows — cmd.exe. Параметры запуска записываются друг за другом и разделяются пробелами.

Для чего нужны параметры? Например, для того, чтобы сразу же сообщить программе, в каком файле брать исходные данные для какого-нибудь расчета и в какой файл выводить результаты.

Запуск программы тогда будет выглядеть следующим образом:

```
myprogram.exe input.txt output.txt
```

А в самой программе мы напишем:

```
inputFile = Program.Directory + "\" +
Program.GetArgument(1)
outputFile = Program.Directory + "\" +
Program.GetArgument(2)
```

Объект *Sound*

Для того чтобы добавить в программу звук, нужен объект `Sound`. Он содержит достаточно много методов. Основные приведены в табл. 3.20.

Таблица 3.20

Метод	Описание
`Sound.Play(filePath)`	Воспроизводит звуковой файл
`Sound.Pause(filePath)`	Делает паузу в воспроизведении файла
`Sound.Stop(filePath)`	Останавливает воспроизведение
`Sound.PlayAndWait(filePath)`	Воспроизводит звуковой файл. Следующая команда выполнится только после того, как файл закончится

Поддерживаются форматы WAV, MP3 и WMA. Файлы других форматов будут прекрасно воспроизводиться, если на компьютере установлены кодеки. И, как и в случае с картинками, файл может быть загружен и с локального диска, и из Интернета.

Кроме этого, у объекта `Sound` есть несколько стандартных звуков — щелчок, звонок и колокольчик.

Объект *Turtle*

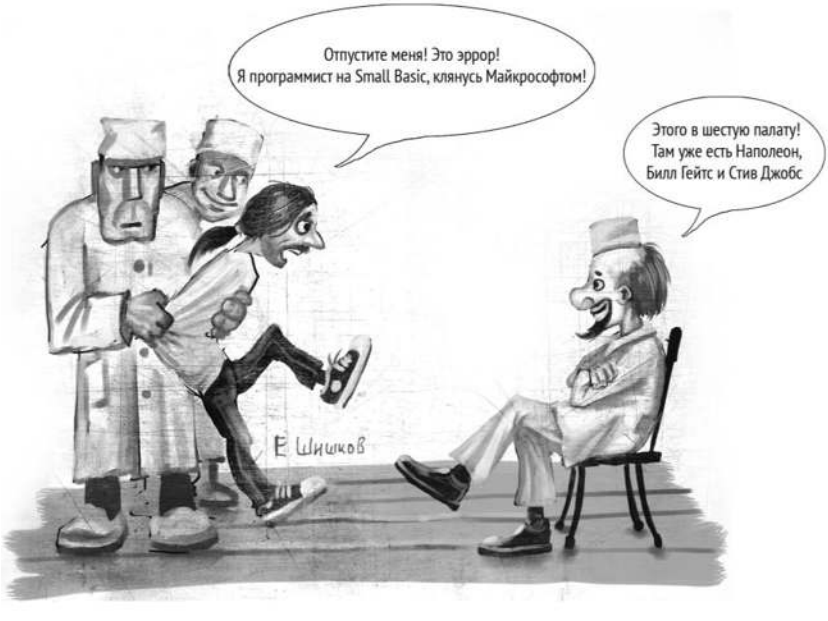
С принципом "черепашьей графики" вы, скорее всего, знакомы по языку Лого. Черепашка понимает несколько команд и, выполняя их, перемещается по экрану и оставляет за собой след заданного цвета и толщины. Стандартные команды Черепашки — повернуть направо/налево или на заданный угол, переместиться вперед/назад, поднять/опустить перо.

Останавливаться на этом объекте мы не будем — все-таки это очень интересный, но совершенно отдельный раздел программирования. Если вы захотите в нем разобраться — смело берите любую книжку по языку Лого!

Напутствие

Книга подошла к концу. Вы ее внимательно прочитали, аккуратно разобрали все примеры, серьезно сделали все задания. И — да! Вы теперь умеете программировать!

Посмотрите с чего мы с вами начинали еще совсем недавно, — учились присваивать значения переменным, вводить и выводить числа в текстовом окне... А теперь — смогли написать настоящую игру!



Самое главное — это то, что теперь вы можете мыслить, как программист, то есть, видя задачу, сразу же прикидывать в голове алгоритм, которым ее можно решить. Уверяю вас — далеко не каждый обладает таким мышлением. Вы теперь — обладаете.

Что же делать дальше? Прежде всего — творить! Для начала перед вами огромное поле для развлечений с языком Small Basic. Придумывайте задачи, ищите их в книжках — и решайте. Ведь мало что может сравниться с тем удовольствием, которое испытывает программист, придумавший красивый, элегантный алгоритм решения сложной задачи.

А когда Small Basic уже станет вам мал, когда его возможностей перестанет хватать — смело переходите к его старшему брату — Visual Basic. Потом — к C#. А потом... впрочем, зачем загадывать так далеко? Главное — вы умеете программировать! А это — просто здорово.

Удачи!

Приложение

Цвета Small Basic

Красные цвета

IndianRed	#CD5C5C	индийский красный, каштановый
LightCoral	#F08080	светло-коралловый
Salmon	#FA8072	лососёвый
DarkSalmon	#E9967A	темно-лососёвый
LightSalmon	#FFA07A	светло-лососёвый
Crimson	#DC143C	темно-красный, кровавый
Red	#FF0000	красный
FireBrick	#B22222	кирпичный
DarkRed	#8B0000	темно-красный

Розовые цвета

Pink	#FFC0CB	розовый
LightPink	#FFB6C1	светло-розовый
HotPink	#FF69B4	ярко-розовый
DeepPink	#FF1493	темно-розовый
MediumVioletRed	#C71585	фиалковый
PaleVioletRed	#DB7093	бледно-фиалковый

Оранжевые цвета

Coral	#FF7F50	коралловый
Tomato	#FF6347	томатный
OrangeRed	#FF4500	оранжево-красный
DarkOrange	#FF8C00	темно-оранжевый
Orange	#FFA500	оранжевый

Желтые цвета

Gold	#FFD700	золотой
Yellow	#FFFF00	желтый
LightYellow	#FFFFE0	светло-желтый
LemonChiffon	#FFFACD	лимонный
LightGoldenrodYellow	#FAFAD2	светло-золотистый
PapayaWhip	#FFEFD5	побег папайи
Moccasin	#FFE4B5	индейские мокасины
PeachPuff	#FFDAB9	персиковый
PaleGoldenrod	#EEE8AA	бледно-золотой
Khaki	#F0E68C	хаки
DarkKhaki	#BDB76B	темный хаки

Фиолетовые цвета

Lavender	#E6E6FA	лаванда
Thistle	#D8BFD8	чертополох
Plum	#DDA0DD	сливовый
Violet	#EE82EE	фиолетовый
Orchid	#DA70D6	орхидея
Fuchsia, Magenta	#FF00FF	фуксия, малиновый, маджента
MediumOrchid	#BA55D3	средняя орхидея
MediumPurple	#9370DB	средне-пурпурный
BlueViolet	#8A2BE2	фиолетово-синий

Фиолетовые цвета (окончание)

DarkViolet	#9400D3	темно-фиолетовый
DarkOrchid	#9932CC	темная орхидея
DarkMagenta, Purple	#8B008B	темно-малиновый, пурпурный
Indigo	#4B0082	индиго
SlateBlue	#6A5ACD	грифельно-серый
DarkSlateBlue	#483D8B	темный грифельно-серый
MediumSlateBlue	#7B68EE	средний грифельно-серый

Зеленые цвета

GreenYellow	#ADFF2F	желто-зеленый
Chartreuse	#7FFF00	салатовый, шартрез
LawnGreen	#7CFC00	зеленая лужайка
Lime	#00FF00	лайм
LimeGreen	#32CD32	лимонно-зеленый
PaleGreen	#98FB98	бледно-зеленый
LightGreen	#90EE90	светло-зеленый
MediumSpringGreen	#00FA9A	средний весенне-зеленый
SpringGreen	#00FF7F	весенне-зеленый
MediumSeaGreen	#3CB371	средний цвет морской волны
SeaGreen	#2E8B57	цвет морской волны
ForestGreen	#228B22	зеленый лесной
Green	#008000	зеленый
DarkGreen	#006400	темно-зеленый
YellowGreen	#9ACD32	желто-зеленый
OliveDrab	#6B8E23	нежно-оливковый
Olive	#808000	оливковый
DarkOliveGreen	#556B2F	темный оливково-зеленый
MediumAquamarine	#66CDAA	средне-аквамариновый
DarkSeaGreen	#8FBC8F	темный цвет морской волны

Зеленые цвета (окончание)

LightSeaGreen	#20B2AA	светлый цвет морской волны
DarkCyan	#008B8B	темно-васильковый
Teal	#008080	сине-зеленый

Синие цвета

Aqua, Cyan	#00FFFF	зеленовато-голубой, васильковый
LightCyan	#E0FFFF	светло-васильковый
PaleTurquoise	#AFEEEE	бледно-бирюзовый
Aquamarine	#7FFFD4	аквамариновый
Turquoise	#40E0D0	бирюзовый
MediumTurquoise	#48D1CC	средне-бирюзовый
DarkTurquoise	#00CED1	темно-бирюзовый
CadetBlue	#5F9EA0	серо-синий
SteelBlue	#4682B4	синий со стальным оттенком
LightSteelBlue	#B0C4DE	светло-синий со стальным оттенком
PowderBlue	#B0E0E6	синий с пороховым оттенком
LightBlue	#ADD8E6	светло-синий
SkyBlue	#87CEEB	небесно-голубой
LightSkyBlue	#87CEFA	светлый небесно-голубой
DeepSkyBlue	#00BFFF	глубокий небесно-голубой
DodgerBlue	#1E90FF	защитно-синий
CornflowerBlue	#6495ED	васильковый
RoyalBlue	#4169E1	королевский синий
Blue	#0000FF	синий
MediumBlue	#0000CD	средне-синий
DarkBlue	#00008B	темно-синий
Navy	#000080	цвет формы морских офицеров США
MidnightBlue	#191970	полуночно-синий

Коричневые цвета

Cornsilk	#FFF8DC	шелковый оттенок
BlanchedAlmond	#FFEBCD	миндаль Крайола
Bisque	#FFE4C4	томатный суп
NavajoWhite	#FFDEAD	белый навахо
Wheat	#F5DEB3	пшеничный
BurlyWood	#DEB887	большое дерево, желтоватый
Tan	#D2B48C	цвет загара
RosyBrown	#BC8F8F	розово-коричневый
SandyBrown	#F4A460	песочный
Goldenrod	#DAA520	золотисто-березовый
DarkGoldenrod	#B8860B	темно-золотой
Peru	#CD853F	цвет Перу
Chocolate	#D2691E	шоколадный
SaddleBrown	#8B4513	кожано-коричневый
Sienna	#A0522D	охра
Brown	#A52A2A	коричневый
Maroon	#800000	темно-бордовый

Белые цвета

White	#FFFFFF	белый
Snow	#FFFAFA	белоснежный
Honeydew	#F0FFF0	медовый
MintCream	#F5FFFA	кремово-мятный
Azure	#F0FFFF	лазурный
AliceBlue	#F0F8FF	синяя Элис
GhostWhite	#F8F8FF	призрачно-белый
WhiteSmoke	#F5F5F5	белый дым
Seashell	#FFF5EE	морская раковина

Белые цвета (окончание)

Beige	#F5F5DC	бежевый
OldLace	#FDF5E6	старое кружево
FloralWhite	#FFFAF0	белый цветок
Ivory	#FFFFF0	слоновая кость
AntiqueWhite	#FAEBD7	антикварный белый
Linen	#FAF0E6	льняной
LavenderBlush	#FFF0F5	розово-лавандовый
MistyRose	#FFE4E1	тускло-розовый

Серые цвета

Gainsboro	#DCDCDC	Гейнсборо
LightGray	#D3D3D3	светло-серый
Silver	#C0C0C0	серебряный
DarkGray	#A9A9A9	темно-серый
Gray	#808080	серый
DimGray	#696969	тускло-серый
LightSlateGray	#778899	светлый грифельно-серый
SlateGray	#708090	грифельно-серый
DarkSlateGray	#2F4F4F	темный грифельно-серый
Black	#000000	черный

Предметный указатель