Ильдар Ахметов

Программирование на Small Basic

Учебник для тех, кто не понимает никакие другие учебники

Художник: Евгений Шишков

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, то есть уничтожаем. А тут наоборот – мы программируем, то есть 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 как-то вот так, 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 и 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, она 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»[1].

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, например, 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

[1] К счастью, можно предусмотреть такой случай и заранее проинструктировать собаку: “Если на хозяина напали, надо кусать”. В программировании это называется условным оператором.

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 компьютер. Языков программирования много. Люди говорят на русском, английском языке, иврите или африкаансе, а для машин есть языки C, Pascal, Basic, Java, PHP, perl и так далее. 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 хорошо 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[2], конечно, но не будем нарушать традиций.

Итак, вот она – ваша первая программа:

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

Попробуйте написать эту программу и запустить ее (запускается программа большим синим треугольником с надписью «Запуск»). Получилось? Видите черное окно, в котором написано «Привет, мир!»? Тогда поздравляю – первую программу вы написали.

[2] В одной книжке, например, автор предлагал в первой программе выводить фразу «Наше вам с кисточкой».

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 писать 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 значений `"Red"` – «красный». Можете попробовать теперь раскрасить строку в другие цвета.

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 ящике 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 позже).

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

```
a = 17
```

Это значит «положить число 17 в переменную `a`», то есть «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.

```
a = a + 5
```

1. Начинаем обрабатывать правую часть. Открываем ящик `a`. Там лежит 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[3].

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 (то есть 17*2+10).

[3] Поэтому и название такое – переменная, потому что значения меняются.

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.

	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

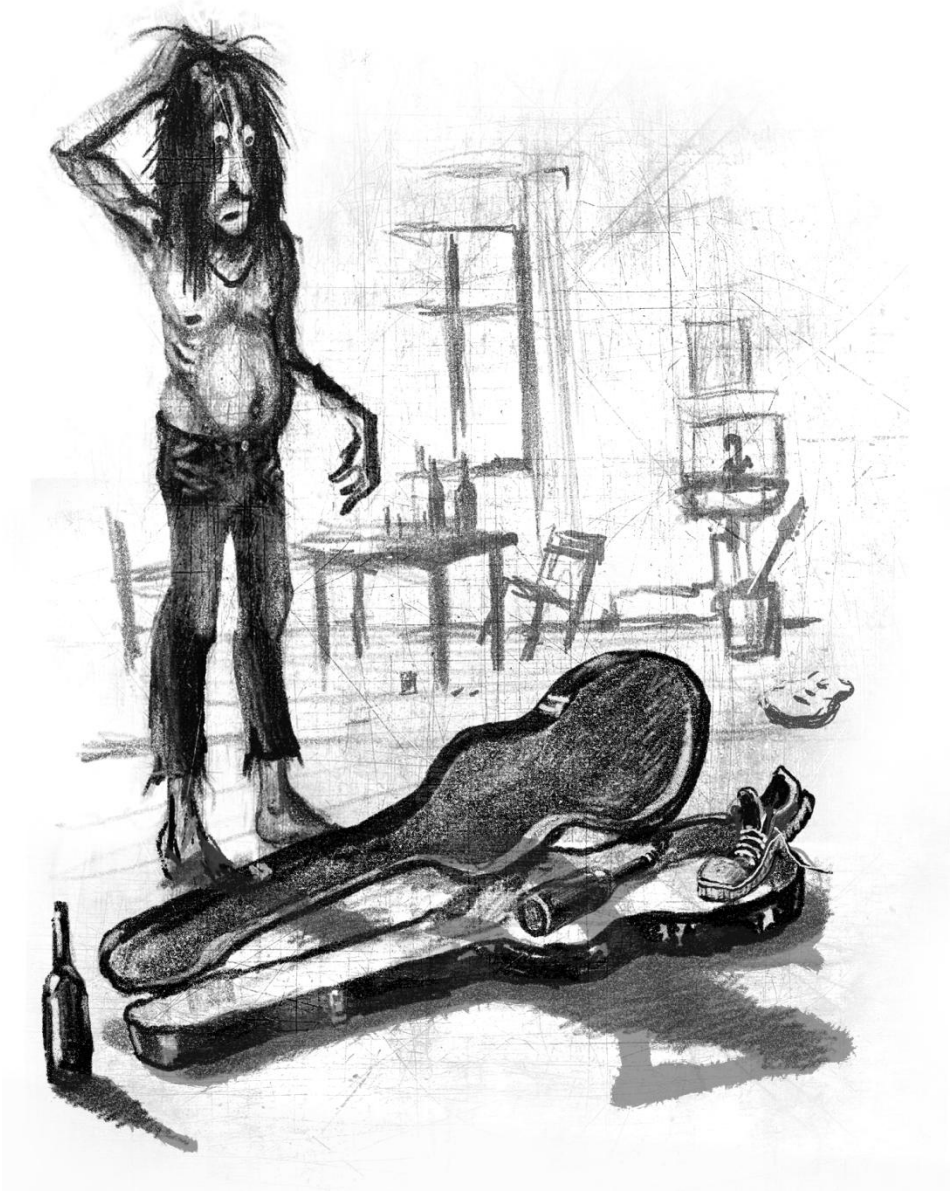
Посмотрите еще раз – переменные, которые находятся в левой части оператора присваивания, не меняются. Меняются только те, что стоят справа.

И еще один важный момент. Обратите внимание на последнюю строку (`a = b`). Она не делает переменные `a` и `b` синонимами, то есть 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[4].

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



[4] Да и представьте себе глаза той девушки, которой преподнесут кольцо в деревянном ящике.

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

В языке Small Basic типов данных всего два:

1. Число
2. 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 так далее.

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’s 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:

`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 справочной[5] и говорит – «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.

[5] Хотя мы-то с вами, конечно, знаем, что вместо справочной 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:

`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:

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

Разница между двумя вариантами вывода в следующем:

`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`. А 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 более вежливо. 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».

Условный оператор

В жизни постоянно возникают ситуации, которые требуют выбора. Вот, 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 – то просто надеваем какую-нибудь обычную одежду.

Смысл условного оператора именно в этом слове – **«если»**. В языке Small Basic[6] это слово пишется просто – **`If`**.

Если условие истинно (то есть правда), то оно примет значение «истина» или `"True"`. Если условие ложно (то есть совсем даже не правда), то это будет «ложь» – `"False"`[7]. То есть условие «Париж – столица Франции» – истина (`"True"`), а «Париж – столица Камбоджи» – ложь (`"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 не выполняются, а сразу выполняется продолжение.

Давайте приведем пример. Тот же самый, про погоду, но пока сделаем его попроще, с единственным условием – есть дождь или нет.

Запишем это условие в виде псевдо-программы[8]

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

Видите – все понятно, «эту программу очень легко перевести с компьютерного языка на человеческий[9]. «Если идет дождь, то взять зонт».

[6] Да и в большинстве других языков

[7] Обратите внимание – кавычки важны

[8] Мы будем пользоваться таким приемом еще не раз. Псевдо-код, конечно, нельзя запускать – он нужен для того, чтобы понимать суть.

[9] С бейсика на русский, вообще говоря.

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 уже если ни одно из условие не выполнено (все оказалось ложью[10]), то срабатывает вариант “иначе” – Else.

Вернемся к нашему исходному погодному примеру

```
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 машина 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).

[10] «Все врут» (Доктор Хаус)

5. Если ни дождя, ни снега нет (все условия ложны), то надеть обычную одежду. Выйти из дома.

Приведем теперь несколько работающих примеров

Это уже не псевдо-код, 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` – истина (то есть 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 оказались ложными – то есть 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»[11]. Для этого воспользуемся объектом `Clock` («Часы»), у которого есть свойство `Hour` («Час») – `Clock.Hour` скажем нам, сколько сейчас часов.

```
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 (то есть 18), то «Добрый день», а иначе – «Добрый вечер».

[11] Можно, конечно, здороваться, как в фильме «Шоу Трумана» – «Доброе утро, и, если не увидимся, добрый день и 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 полный список операторов сравнения:

<	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. Знак равенства «=» 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:

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

Примеры условий 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 истинно – 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

Например, 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, например, 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

Ппроверяем, отдыхать сегодня или учиться. Отдыхаем мы в выходные и на каникулах, поэтому если сейчас июль или август 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
```

Циклы

Циклы нужны, когда нужно 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 и мы задумаемся – как же правильно подниматься по лестнице?

Здесь есть два варианта

1. Мы знаем количество ступеней – например, 10. Тогда нам нужно шагнуть **ровно** 10 раз – и мы придем к вершине (цикл `For`).
2. Мы не знаем количество ступеней. Тогда нам нужно шагать, **пока** впереди есть ступени. И как только ступени закончились – 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 нужно 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 псевдо-код для задачи «поднимаемся по лестнице, в которой 10 ступеней»:

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

Здесь в качестве счетчика выступает переменная `i`[12]. 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:

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

[12] Обычно счетчик обозначают именно этой буквой.

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, потом 2, потом 3... и так далее до 10.

Чтобы шаг цикла был равен не единице, а чему-то другому, используется параметр `Step`.

Например, вот такая программа выведет числа 1, 1.5, 2, ..., 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, потом – 2, потом – 3 и так далее. В итоге за 10 шагов цикла мы прибавим к сумме все числа по очереди. Вот как это будет происходить:

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[13]:

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

Заметили, что здесь мы начинаем не с нуля, а с единицы? Вычисление произведения так и нужно начинать. Если бы мы начали с нуля, то и все произведение осталось бы нулем. Ведь что на ноль не умножай…

[13] Оно, кстати, называется факториал.

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 дошли 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 псевдо-коде 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, поэтому 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 так далее… И вот, пусть мы шагнули на 28-ю ступень и проверяем, есть ли впереди 29-я? А ее нет. Поэтому условие становится ложью и происходит выход из цикла. Все, подъем завершен!

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.

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[14]. 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:

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

[14] Вы же помните, что означает строка k=k/2? Это значит «взять значение переменной k, поделить его на 2, результат положить в переменную k»

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!	-	-

И еще один интересный пример – Что-то вроде игры в лото

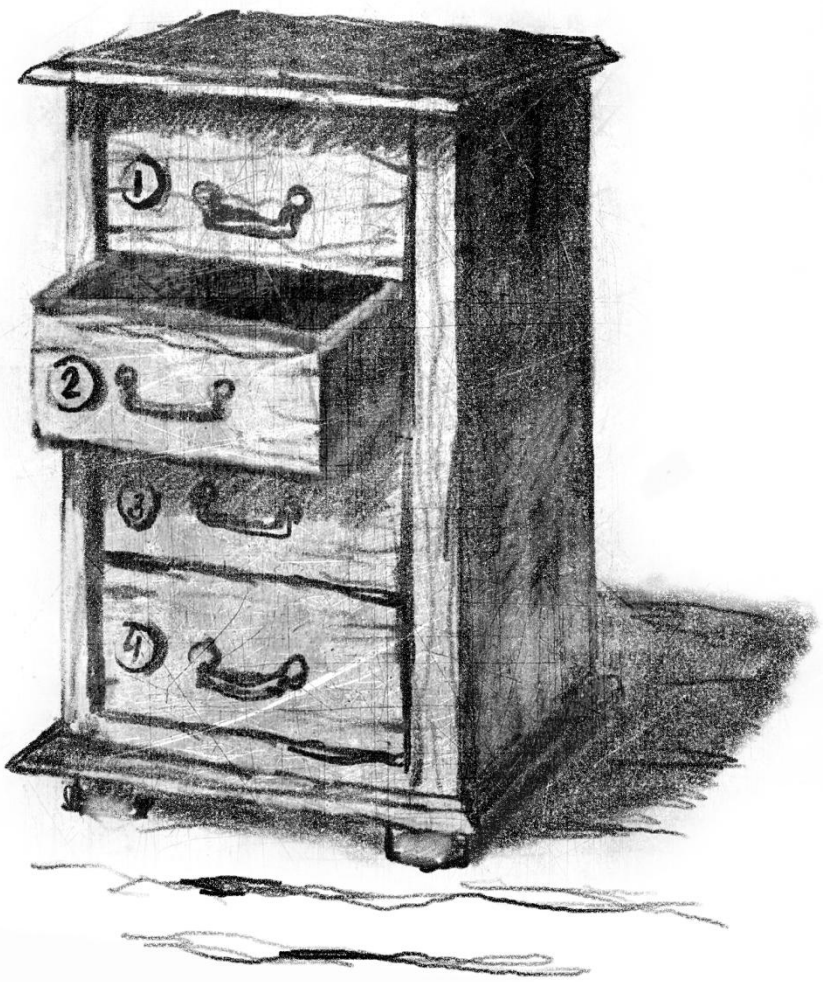
Предположим, что мы загадали число – например, 69. А машина пытается его угадать – она называет случайные числа от 1 до 100 до тех пор, пока не назовет то, что мы загадали. Для выдачи случайных чисел воспользуемся методом `Math.GetRandomNumber(100)`. В скобках стоит число 100 – это значит как раз то, 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 какая-то последовательность вроде такой: 32, 93, 5, 61, 77, 28, 40, 1, 84, 95, 14, 69. Рано или поздно машина обязательно угадает наше число 69 – и на этом остановится – условие «n не равно 69» (`n<>69`) станет ложным.

Массивы

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



Пусть полка называется `M`[15] и в ней 4 ящика. Тогда каждый ящик полки – **элемент массива** – имеет имя:

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

[15]Договоримся для удобства обозначать имена массивов большими буквами (хотя принципиальной разницы нет)

То есть для обращения к элементу массива используется конструкция `M[n]`, где число `n` – это номер ящика на полке, 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])
```

Вот как меняются значения элементов массива в ходе выполнения этой программы:

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 полка 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[16]. То есть в первом элементе должно находиться число 3, во втором – 6, в третьем – 9 и так далее. 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
```

[16] Да, это называется арифметической прогрессией.

Вот что происходит:

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 так далее 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 приведем несколько примеров по обработке массивов.

Первый пример. 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, то есть:

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

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.

Второй пример. 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:

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.

Третий пример. 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.Read()
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
```

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

Четвертый пример. Не только числа!

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

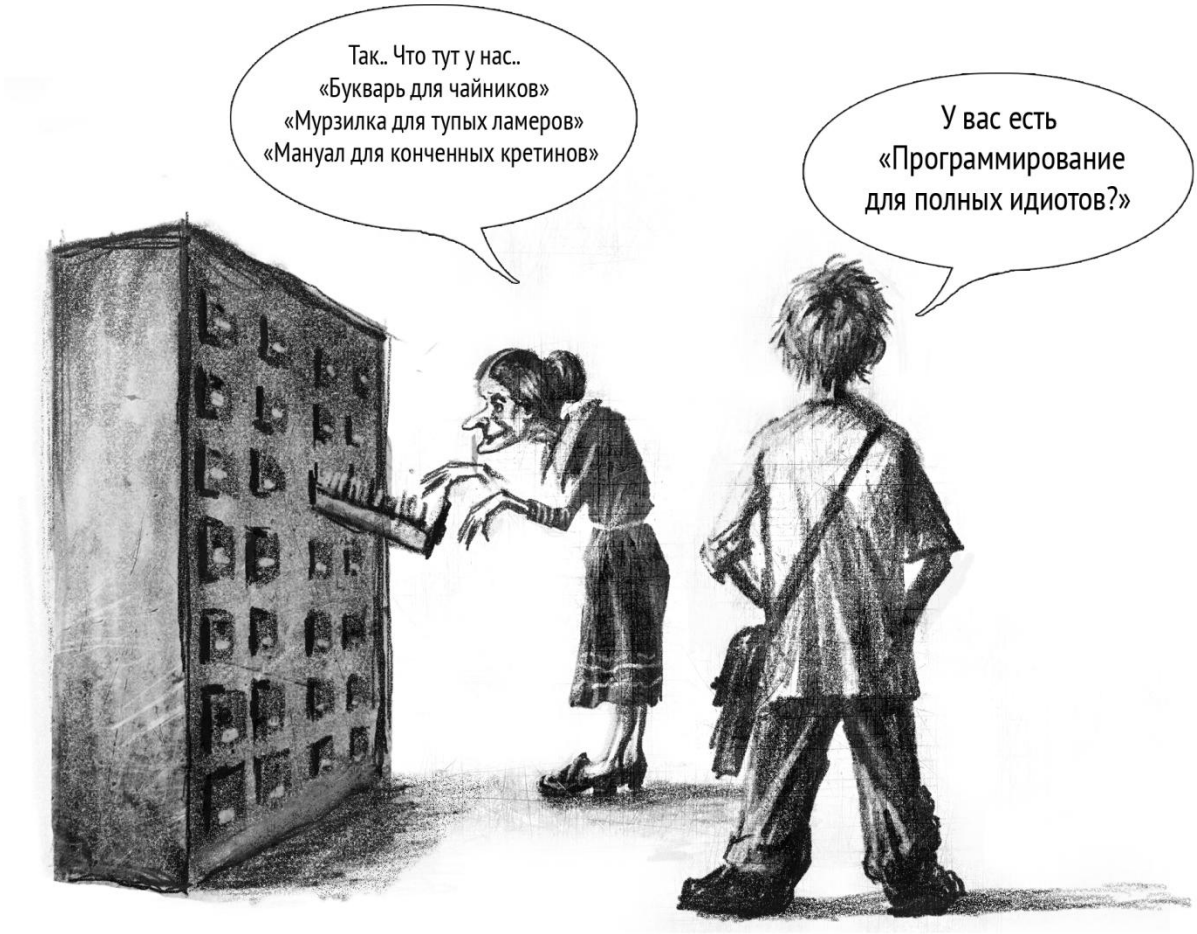
```
For i = 1 To 5
    TextWindow.Write("Человек номер " + i + ", введи свое имя!")
    name[i] = TextWindow.Read()
EndFor

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

Попробуйте сами разобраться, как работает эта программа!

Двухмерные массивы

И снова возвращаемся к ящикам. Если обычные, одномерные массивы, о которых мы говорили только что – это полки 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**.

Давайте, например, заполним двухмерный массив размером 3x3 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` снова проходит от 1 до 3. Счетчик `i` при этом равен 2
5. И так далее.

Вот что получается:

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 двухмерного массива 3 на 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, вывод двухмерного 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
```

Приведем пару примеров обработки двухмерных массивов.

Первый пример. 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: " + sum)
```

Комментарии, пожалуй, излишнии.

Второй пример. 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 штуки. 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 эл-тов " + 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:

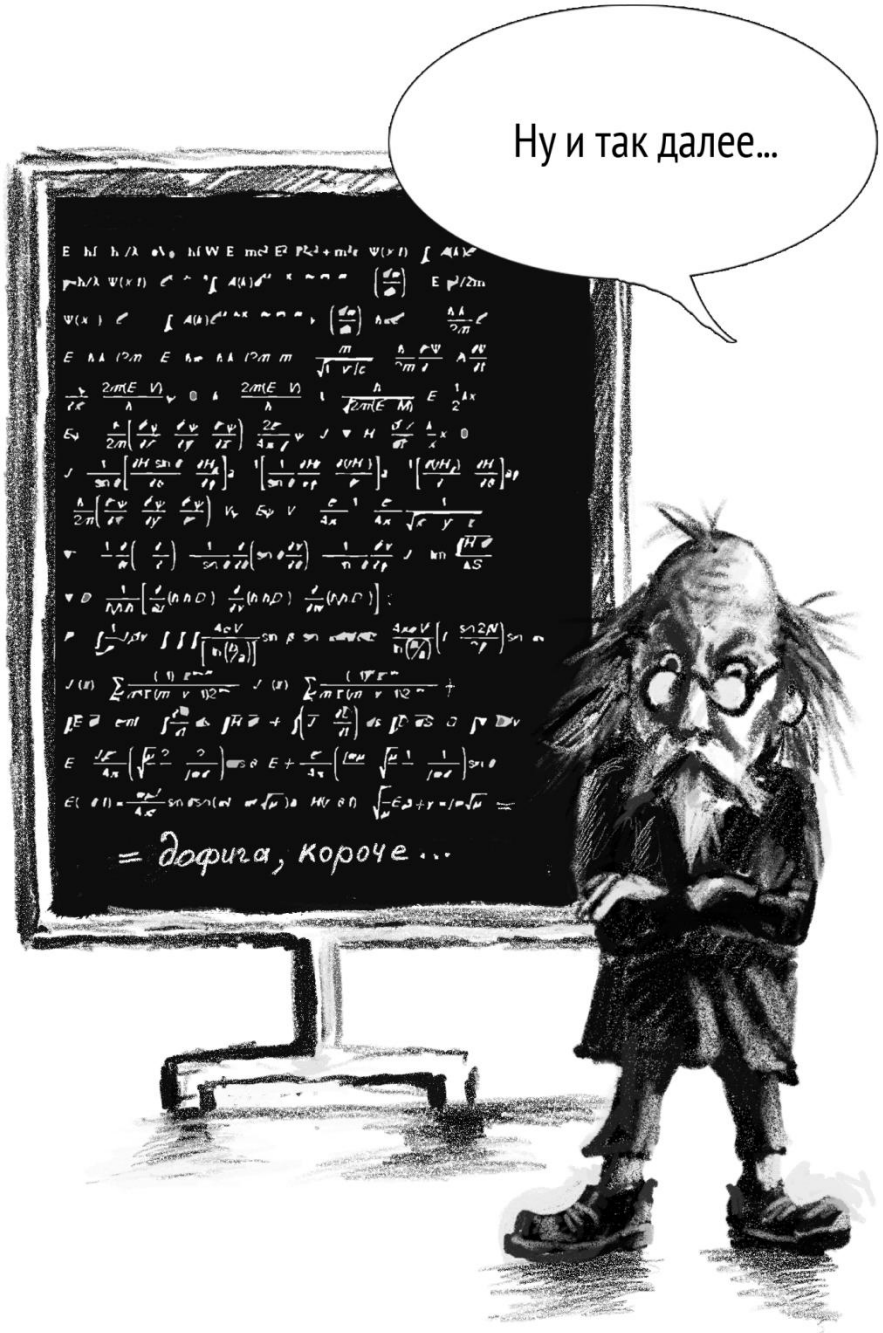
i	**j**	**S[1]**	**S[2]**	**S[3]**
1	До внутреннего	0	ничто	ничто
1	1	4 (0+4)	ничто	ничто
1	2	14 (4+10)	ничто	ничто
1	3	22 (14+8)	ничто	ничто
2	До внутреннего	22	0	ничто
2	1	22	16 (0+16)	ничто
2	2	22	25 (16+9)	ничто
2	3	22	37 (25+12)	ничто
3	До внутреннего	22	37	0
3	1	22	37	7 (0+7)
3	2	22	37	12 (7+5)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

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

Кстати, двухмерными массивами дело не ограничивается. Массивы могут быть и трехмерными (представьте себе кубик Рубика – ящики в трех измерениях!), и четырех-, пяти- и так далее-мерными. Используются они, конечно, довольно редко – но они есть.

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`[17].



[17] Строго говоря, это – методы объекта Math.

Вот такие математические функции есть в Small Basic:

Math.Abs	Модуль
Math.ArcCos	Арккосинус
Math.ArcSin	Арксинус
Math.ArcTan	Арктангенс
Math.Ceiling	Округление вверх
Math.Cos	Косинус
Math.Floor	Округление вниз
Math.GetDegrees	Из радиан в градусы
Math.GetRadians	Из градусов в радианы
Math.GetRandomNumber	Случайное число
Math.Log	Десятичный логарифм
Math.Max	Максимум двух чисел
Math.Min	Минимум двух чисел
Math.NaturalLog	Натуральный логарифм
Math.Pi	Число Пи
Math.Power	Степень
Math.Remainder	Остаток от деления
Math.Round	Обычное округление
Math.Sin	Синус
Math.SquareRoot	Квадратный корень
Math.Tan	Тангенс

Несколько примеров записи математических формул на Small Basic

$$a = x^3$$

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

$$b = sinx$$

```
b = Math.Sin(x)
```

$$c = cosx^2$$

```
c = Math.Cos(Math.Power(x,2))
```

или

```
c = Math.Cos(x * x)
```

$$d = tg^2x$$

```
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)
```

$$d = \frac{sinx^2}{lnx} + 2\pi$$

```
d = Math.Sin(Math.Power(x,2)) / Math.NaturalLog(x) + 2 * Math.Pi
```

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`:

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

Первый пример. 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 один. 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 номер `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 так далее.

Второй пример. 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) = "a" 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 – а 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» тоже 5. Хотя невооруженным глазом видно, что их там 7! Просто две из них – большие. Как сделать так, чтобы считались и большие, и маленький буквы? Есть два способа.

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

```
If Text.GetSubText(s,i,1) = "a" Or Text.GetSubText(s,i,1) = "a" 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 штука:

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

Третий пример. Заменить букву «а» на букву «о»

Дана строка. Заменить в ней все буквы «а» на «о».

```
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 (в переменной `t`), делая нужную нам замену. Как и раньше, проходим по строке `s` циклом `For`. Проверяем каждую букву, равна ли она «а». Если равна, то в строку `t` добавляем букву «о», а если не равна – то ту же букву, которая и была.

Обратите внимание, что строку `t` мы перед циклом тоже «обнуляем» – но, так как это не число, то присваиваем переменной `t` значение «пустая строка» – то есть открываем кавычки и сразу закрываем.

Получается вот так:

`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	мого**з**
6	и	могоз**и**
7	н	могози**н**

Четвертый пример. Найти длины всех слов в строке

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

```
s = "Колокольчики мои цветики степные"
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 степные цветики. Переменная `t` – 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` (количество 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.