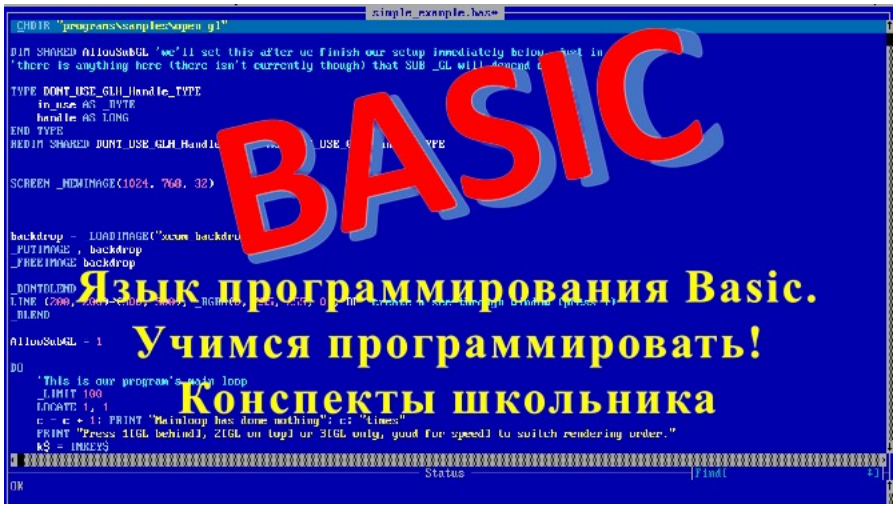
Главная
Новости
Документы
Сценарии
Мелодии
Софт
Авторы
Контакт
Копилка
Баннерообмен

Язык программирования Basic. Учимся программировать! Конспекты школьника

Использованы материалы : информационно-образовательного портала КЛЯКС@.net
www.klyaksa.net

Учимся программировать!
Конспекты школьника.
Составил: Башлаков А.С.
-Б@С- ©2003-04г.



Содержание.

12. Создание движущихся изображений.
13. Работа с файлами.
14. Комбинированные типы.
15. Задания для самостоятельного выполнения.
16. Литература.

Данный материал не является полноценным учебником по программированию на языке Basic, а содержит только необходимый минимум для выработки у школьников алгоритмического мышления и начальных навыков программирования.

В качестве системы программирования можно использовать QBasic.

Язык программирования Basic. Учимся программировать! Конспекты для начинающих.

Алгоритмы.

Появление алгоритмов связывают с зарождением математики. Более 1000 лет назад (в 825 году) ученый из города Хорезма Абдулла (или Абу Джафар) Мухаммед бен Муса аль-Хорезми создал книгу по математике, в которой описал способы выполнения арифметических действий над многозначными числами. Само слово алгоритм возникло в Европе после перевода на латынь книги этого математика.

Алгоритм – описание последовательности действий (план), строгое исполнение которых приводит к решению поставленной задачи за конечное число шагов.

Вы постоянно сталкиваетесь с этим понятием в различных сферах деятельности человека (кулинарные книги, инструкции по использованию различных приборов, правила решения математических задач...). Обычно мы выполняем привычные действия не задумываясь, механически. Например, вы хорошо знаете, как открывать ключом дверь. Однако, чтобы научить этому малыша, придется четко разъяснить и сами эти действия и порядок их выполнения:

1. 1. Достать ключ из кармана.
2. 2. Вставить ключ в замочную скважину.
3. 3. Повернуть ключ два раза против часовой стрелки.
4. 4. Вынуть ключ.

Если вы внимательно оглянитесь вокруг, то обнаружите множество алгоритмов которые мы с вами постоянно выполняем. Мир алгоритмов очень разнообразен. Несмотря на это, удается выделить общие свойства, которыми обладает любой алгоритм.

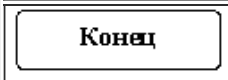
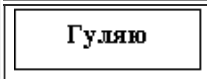
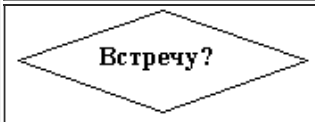
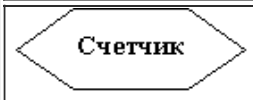
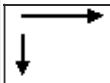
Свойства алгоритмов:

1. 1. Дискретность (алгоритм должен состоять из конкретных действий, следующих в определенном порядке);
2. 2. Детерминированность (любое действие должно быть строго и недвусмысленно определено в каждом случае);
3. 3. Конечность (каждое действие и алгоритм в целом должны иметь возможность завершения);
4. 4. Массовость (один и тот же алгоритм можно использовать с разными исходными данными);
5. 5. Результативность (отсутствие ошибок, алгоритм должен приводить к правильному результату для всех допустимых входных значениях).

Виды алгоритмов:

1. 1. Линейный алгоритм (описание действий, которые выполняются однократно в заданном порядке);
2. 2. Циклический алгоритм (описание действий, которые должны повторятся указанное число раз или пока не выполнено задание);
3. 3. Разветвляющий алгоритм (алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий)
4. 4. Вспомогательный алгоритм (алгоритм, который можно использовать в других алгоритмах, указав только его имя).

5. Для более наглядного представления алгоритма широко используется графическая форма - блок-схема, которая составляется из стандартных графических объектов.

Вид стандартного графического объекта	Назначение
	Начало алгоритма
	Конец алгоритма
	Выполняемое действие записывается внутри прямоугольника
	Условие выполнения действий записывается внутри ромба
	Счетчик кол-во повторов
	Последовательность выполнения действий.

Стадии создания алгоритма:

1. 1. Алгоритм должен быть представлен в форме, понятной человеку, который его разрабатывает.
2. 2. Алгоритм должен быть представлен в форме, понятной тому объекту (в том числе и человеку), который будет выполнять описанные в алгоритме действия.

Объект, который будет выполнять алгоритм, обычно называют исполнителем.

Исполнитель - объект, который выполняет алгоритм.

Идеальными исполнителями являются машины, роботы, компьютеры...

Компьютер – автоматический исполнитель алгоритмов.

Алгоритм, записанный на «понятном» компьютеру языке программирования, называется программой.

Язык программирования Basic. Учимся программировать! Конспекты для начинающих

Введение в язык программирования Basic.

Бейсик это (BASIC, сокращение от англ. Beginner's All-purpose Symbolic Instruction Code — универсальный код символических инструкций для начинающих) — семейство высокоуровневых языков программирования.

Был разработан в 1964 году профессорами Дартмутского колледжа Томасом Курцем и Джоном Кемени.

Язык создавался как инструмент, с помощью которого студенты-непрограммисты могли самостоятельно создавать компьютерные программы для решения своих задач. Получил широкое распространение в виде различных диалектов, прежде всего как язык для домашних компьютеров. К настоящему моменту претерпел существенные изменения, значительно отойдя от характерной для первых версий простоты, граничащей с примитивизмом, и превратившись в достаточно ординарный язык высокого уровня с типичным набором возможностей.
Используется как самостоятельный язык для разработки прикладных программ, главным

образом, работающих под управлением ОС Windows различных версий. Также широко распространён в качестве встроенного языка прикладных программных систем различного назначения и в качестве языка для программируемых калькуляторов. **Википедия**

Для представления алгоритма в виде, понятном компьютеру, служат *языки программирования*. Сначала разрабатывается алгоритм действий, а потом он записывается на одном из таких языков. В итоге получается текст программы - полное, законченное и детальное описание алгоритма на языке программирования. Затем этот текст программы специальными служебными приложениями, которые называются *трансляторами*, либо переводится в машинный код (язык нулей и единиц), либо исполняется.

Языки программирования - это искусственные языки. От естественных они отличаются ограниченным числом "слов", значение которых понятно транслятору, и очень строгими правилами записи команд (*операторов*).

Для написания текста программы можно использовать обычный текстовый редактор (например, Блокнот), а затем с помощью компилятора перевести её в машинный код, т.е. получить исполняемую программу. Но проще и удобнее пользоваться специальными интегрированными средами программирования (например QBasic "КьюБейсик").

QBasic — диалект языка программирования Бейсик (BASIC) разработанный компанией Microsoft, а также среда разработки, позволяющая писать, запускать и отлаживать программы на этом языке.

QBasic удобен для выполнения несложных вычислений и для прямой работы с портами. Наряду с Pascal, язык был довольно популярен для обучения программированию, и долгое время использовался во многих школах.

QBasic широко использовался в школах для обучения основам программирования. В России вплоть до 2010 года QBasic использовали 60-80% учителей информатики. По состоянию на 2020 год, продолжают публиковаться отдельные методические рекомендации по использованию QBasic в обучении информатике. Несмотря на отсутствие поддержки в новых операционных системах, для его запуска используется DOSBox. **Википедия**

Basic (Бейсик) создавался в 60-х годах в качестве учебного языка и очень прост в изучении. По популярности занимает первое место в мире.

Некоторые операторы языка Basic.

REM – оператор комментария.

Все что следует после этого оператора до конца строки игнорируется компилятором и предназначено исключительно для человека. Т.е. здесь можно писать что угодно. Удобно использовать комментарий в начале программы для указания её названия и назначения.

пример:
REM Это комментарий
можно и так:
' Это тоже комментарий

CLS - очистить экран.

Вся информация, которая была на экране стирается.

PRINT (вывод, печать) – оператор вывода.

пример:
PRINT "Привет! Меня зовут Саша."

На экран будет выведено сообщение: Привет! Меня зовут Саша.

INPUT (ввод) – оператор ввода. Используется для передачи в программу каких-либо значений.

пример:
INPUT a

На экране появится приглашение ввести данные (появится знак "?") и компьютер будет ждать их ввода. Для ввода необходимо ввести данные с клавиатуры и нажать ввод (enter).

INPUT "Введите число a: ", a

Компьютер выведет на экран: 'Введите число a:' и будет ждать ввода данных.

DIM – оператор описания типа переменной.

Под **переменной** языках программирования понимают программный объект (число, слово, часть слова, несколько слов, символы), имеющий имя и значение, которое может быть получено и изменено программой.
Если "заглянуть" в компьютер, то переменную можно определить так:

Переменная - это имя физического участка в памяти, в котором в каждый момент времени может быть только одно значение.

Переменная - это ячейка в оперативной памяти компьютера для хранения какой-либо информации.

Само название "переменная" подразумевает, что содержимое этого участка может изменяться.
В качестве имен переменных могут быть латинские буквы с индексами. Причем может быть не одна буква, а несколько.

Пример:
DIM a, b, chislo1 AS INTEGER

Integer – *целые числа от -32768 до 32768*

Если в программе используются переменные не описанные с помощью оператора DIM, то компьютер будет рассматривать их как универсальные переменные. Это может привести к неэффективному использованию оперативной памяти. К тому же, такие программы не всегда легки для восприятия - плохо читаемы.

Для задания значения переменной служит оператор присваивания. Он записывается так:

LET *переменная = значение* (или просто: *переменная = значение*)

Пример:
LET a = 3
chislo1 = 15

END – оператор конца программы.

Арифметические операции на языке Basic.

Операция	Обозначение	Пример	Результат
Сложение	+	2+5	7
Вычитание	-	10-8	2
Умножение	*	3*4	12
Деление	/	15/3 15/4	5 3.75
Целочисленное деление	\	15\4	3
Возведение в степень	^	2^3	8

Остаток от деления	MOD	13 MOD 5	3

Математические функции на языке Basic.

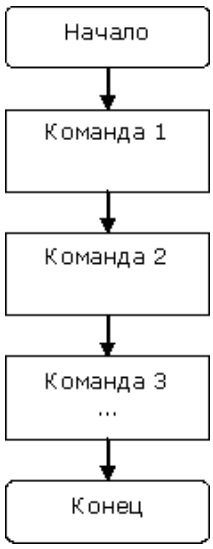
Корень	SQR(X)
Модуль числа	ABS(X)
Синус	SIN(X)
Косинус	COS(X)
Тангенс	TAN(X)
Целая часть числа	INT(X)
Натуральный логарифм	LOG(X)

Теперь уже без осложнений можно переходить непосредственно к составлению программ.

Язык программирования Basic. Учимся программировать! Конспекты школьника

Линейная структура программы.

Программа имеет линейную структуру, если все операторы (команды) выполняются последовательно друг за другом.



Пример: программа, выводящая на экран сообщение: Привет! Меня зовут Саша!

```
REM Первая программа
PRINT “Привет! Меня зовут Саша!”
END
```

Пример: программа, складывающая два числа

```
REM Сумма двух чисел
a = 5
b = 6
c = a + b
PRINT "Результат: ", c
END
```

или так:

```
REM Сумма двух чисел
DIM a, b, c AS INTEGER
a = 5
b = 6
c = a + b
PRINT "Результат: ", c
END
```

Пример: Вычислите площадь прямоугольника по его сторонам.

```
REM Площадь прямоугольника
INPUT "Введите сторону a", a
INPUT "Введите сторону b", b
```

```
s = a * b
PRINT "Площадь равна: ", s
END
```

Пример: Вычислить выражение $c = \frac{\sqrt{2ab}}{a+b}$

```
REM Вычисление выражения
INPUT "Введите a", a
INPUT "Введите b", b
c = SQR(2*a*b)/(a+b)
PRINT "Площадь равна: ", c
END
```

Пример: Вычислите длину окружности и площадь круга по данному радиусу.

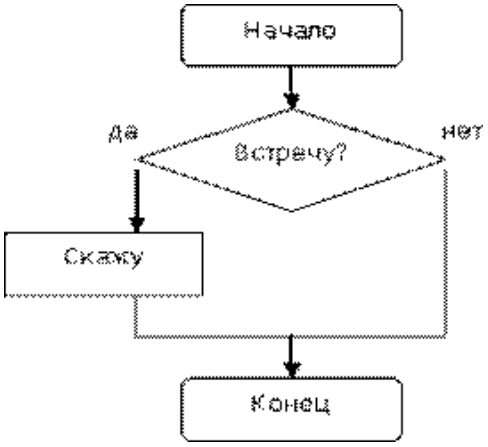
```
REM Вычисление длины окружности и площади круга
INPUT "Введите радиус ", r
PI = 3.14
l = 2 * PI * r
s= PI * r * r
PRINT "Длина окружности равна: ", l
PRINT "Площадь равна: ", s
END
```

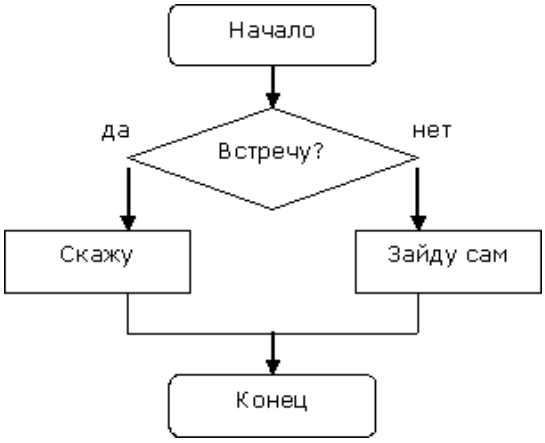
Язык программирования Basic. Учимся программировать! Конспекты школьника

Ветвление в алгоритмах и программах.

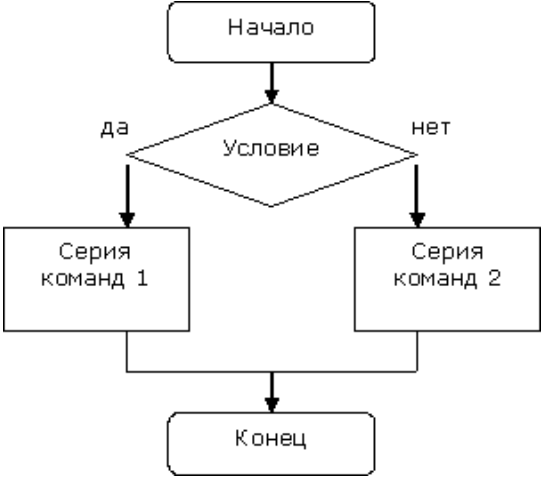
Разветвляющий алгоритм – это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.

Во многих случаях требуется, чтобы при одних условиях выполнялась одна последовательность действий, а при других - другая.





Вся программа состоит из команд (операторов). Команды бывают простые и составные (команды, внутри которых встречаются другие команды). Составные команды часто называют управляющими конструкциями. Этим подчеркивается то, что эти операторы управляют дальнейшим ходом программы.



Рассмотрим запись условного оператора на языке Basic.

Простая форма оператора выглядит следующим образом:

IF <УСЛОВИЕ> THEN <ОПЕРАТОР>

или

IF <УСЛОВИЕ>
<ОПЕРАТОР 1>
<ОПЕРАТОР 2>
...
<ОПЕРАТОР N>
END IF

Если *условие справедливо*, то программа *выполняет* тот оператор, который стоит после ключевого слова **THEN** (или серию операторов от ключевого слова **THEN** до **END IF**), и дальше руководствуется обычным порядком действий. Если *уловие не справедливо*, то оператор, стоящий после **THEN** (или серия операторов от **THEN** до **END IF**) *не выполняется*, и программа сразу переходит к обычному порядку действий.
Конструкция **IF...THEN** позволяет в зависимости от справедливости условия либо выполнить оператор, либо пропустить этот оператор.
Конструкция **IF...THEN...END IF** позволяет в зависимости от справедливости условия либо выполнить группу операторов, либо пропустить эту группу операторов.

Условия - еще один тип логических выражений. В них используются следующие *операторы сравнения*:

=	равно
<>	не равно
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

Справа и слева от знака сравнения должны стоять величины, относящиеся к одному типу. В результате сравнения получается логическая величина, имеющее значение ИСТИНА (TRUE) или ЛОЖЬ (FALSE).

Пример:
5<7 - ИСТИНА;
8=12 -ЛОЖЬ (проверяем равно ли 8 12, *именно проверяем, а не утверждаем, что 8=12*);

Предыдущие конструкции позволяли обойти или выполнить серию оператор в зависимости от справедливости условия. Это еще не было ветвлением. Чтобы вычисления могли разветвляться по нескольким направлениям, служит конструкция **IF...THEN...ELSE...END IF**.

IF <УСЛОВИЕ> THEN
<ОПЕРАТОРЫ 1>

ELSE
<ОПЕРАТОРЫ 2>
END IF

Если условие справедливо (ИСТИНА), то выполняются <операторы 1> (стоящие между **THEN** и **ELSE**), а <операторы 2> (стоящие между **ELSE** и **END IF**) будут пропущены.
Если условие не справедливо (ЛОЖЬ), то <операторы 1> игнорируются и выполняются <операторы 2>.

IF - если, THEN - тогда, ELSE - иначе.

Если в комнате темно, **тогда** надо включить свет.

Если пойдет дождь, **тогда** надо взять зонтик,
иначе, зонтик не брать.

Пример: Проверить, равно ли введенное число некоторому значению, и в случае равенства выдать на экран сообщение о равенстве чисел.

```
REM сравнить число со каким-то значением
INPUT "Введите a", a
IF a=7 THEN PRINT "Числа равны"
END
```

После запуска программы проверяется равно ли введенное значение семи или нет. Если равно, то на экран выводится сообщение 'Числа равны'.

Пример: Определить большее из двух чисел, вывести его на экран, затем - увеличить его в двое и вывести результат на экран.

```
REM определить большее из двух чисел...
INPUT "Введите a", a
INPUT "Введите b", b
IF a>b THEN
PRINT "Большее число: ", a
c=2*a
ELSE
PRINT "Большее число: ", b
c=2*b
END IF
PRINT "результат: ", c
END
```

Сначала программа запрашивает оба числа, затем проверяет условие a>b. Если условие верно, то на экран выводится число a, затем это число удваивается. Иначе на на экран выводится число b, затем число b удваивается. В завершении на экран выводится удвоенное значение большего числа.

Обратите внимание: программа имеет один недостаток - не учитывается тот случай, когда введенные числа равны. Исправим это, использовав вложение одного условия в другое.

```
REM определить большее из двух чисел...
INPUT "Введите a", a
INPUT "Введите b", b
IF a=b TNEN
PRINT "Числа равны"
c=2*a
ELSE
   IF a>b THEN
   PRINT "Большее число: ", a
   c=2*a
   ELSE
   PRINT "Большее число: ", b
   c=2*b
   END IF
END IF
PRINT "результат: ", c
END
```

В этой программе два условных оператора, первым проверяется условие равенства чисел и, в случае его выполнения, будет выдано сообщение о равенстве чисел, если числа не равны, то проверяется второе условие...

Пример: Решение квадратного уравнения.
Решение квадратного уравнения зависит от значения дискриминанта.

```
REM Решение квадратного уравнения
INPUT "Введите коэффициент a: ", a
INPUT "Введите коэффициент b: ", b
INPUT "Введите коэффициент c: ", c
d=b*b-4*a*c
IF d<0 THEN
PRINT "Корней нет"
ELSE
   IF d=0 THEN
   x=-b/(2*a)
   PRINT "корень уравнения: ", x
   ELSE
   x1=(-b-SQR(d))/(2*a)
   x2=(-b+SQR(d))/(2*a)
   PRINT "корни уравнения: ", x1, x2
   END IF
END IF
END
```

Структура "Выбор".

Структура **IF...** позволяет выбрать между двумя вариантами. Если требуется осуществить выбор между большим числом вариантов, то это можно организовать используя лишь структуру **IF**... Но можно (что чаще проще) и с помощью структуры "Выбор". Эта структура имеет вид:

```
SELECT CASE <Выражение>
CASE <условие 1>
<серия 1>
CASE<условие 2>
<серия 2>
...
CASE ELSE
<серия иначе>
END SELECT
```

Выражение, заданное после ключевых слов **SELECT CASE**, сравнивается с определенными значениями - условиями и если они истинны, то выполняется соответствующая серия команд. Если не одно условие не истинно, то выполняется серия команд между **CASE ELSE** и **END SELECT.**

Пример: Выдать словесное значение числа

```
REM Преобразование чисел в слова
INPUT "Введите число", a
SELECT CASE a
CASE 1
PRINT "один"
CASE 2
PRINT "два"
CASE 3
PRINT "три"
...
CASE 10
PRINT "десять"
CASE ELSE
PRINT "это число не могу перевести"
END SELECT
END
```

В данном примере введенное число сравнивается с числами от 1 до 10 и если наше число равно одному из этих чисел, то на экран выводится словесное

значение числа. Если это не так на экран выводится сообщение: "это число не могу перевести".

Язык программирования Basic. Учимся программировать! Конспекты школьника

Циклы в алгоритмах и программах.

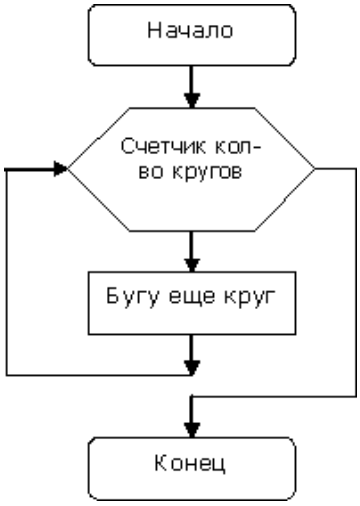
Лучшее качества компьютеров проявляются не тогда, когда они рассчитывают значения сложных выражений, а когда многократно, с незначительными изменениями, повторяют сравнительно простые операции. Даже очень простые расчеты могут поставить человека в тупик, если их надо повторить тысячи раз, а повторять операции миллионы раз человек совершенно не способен.

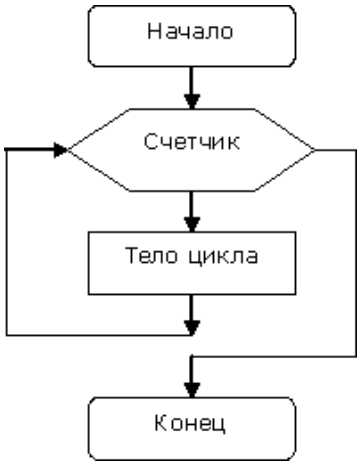
С необходимостью повторяющихся вычислений программисты сталкиваются постоянно. Например, если надо подсчитать, сколько раз буква "о" встречается в тексте необходимо перебрать все буквы. При всей простоте этой программы исполнить ее человеку очень трудно, а для компьютера это задача на несколько секунд.

Циклический алгоритм - описание действий, которые должны повторяться указанное число раз или пока не выполнено заданное условие.

Перечень повторяющихся действий называют телом цикла.

Например, на уроке физкультуры вы должны пробежать некоторое количество кругов вокруг стадиона.





Такие циклы называются - **циклы со счетчиком**.

На языке Basic они записываются следующим образом:

FOR Счетчик=НачЗнач TO КонЗнач [STEP шаг]
тело цикла
NEXT [Счетчик]

Параметры указанные в квадратных скобках являются не обязательными (их можно не записывать). По умолчанию шаг цикла равен одному, т.е. каждый раз после прохождения тела цикла счетчик увеличивается на единицу.

Пример: Вывести на экран все числа от 1 до 100. Для этого можно было бы написать следующую программу:

```
REM Вывод чисел от 1 до 100
PRINT 1
PRINT 2
PRINT 3
PRINT 4
PRINT 5
PRINT 6
PRINT 7
```

```
...
PRINT 98
PRINT 99
PRINT 100
END
```

Всего каких-то 102 строчки ;-). Хотя эту же программу можно написать намного короче:

```
REM Вывод чисел от 1 до 100
FOR I=1 TO 100
PRINT I
NEXT
END
```

Немного исправив программу можно сделать, чтобы она выводила все числа от a до b.

```
REM Вывод чисел от a до b
a=55
b=107
FOR I=a TO b
PRINT I
NEXT
END
```

В этом случае счетчик при первом прохождении цикла принимает значение переменной a, после чего выполняются операторы до ключевого слова NEXT. После этого счетчик увеличивается на единицу и сравнивается со значение переменной b, если счетчик меньше, то цикл выполняется еще.

Легко сделать чтобы программа выводила числа в обратном порядке. Для этого шаг цикла должен быть равен -1 (минус один). В этом случае значение счетчика каждый раз после прохождения цикла будет уменьшено на единицу.

```
REM Вывод чисел от b до a
a=55
b=107
FOR I=b TO a STEP -1
PRINT I
NEXT
END
```

Пример: Вычислить сумму двухзначных натуральных чисел.

```
REM Вычислить сумму двухзначных натуральных чисел
FOR I=10 TO 99
s=s+I
NEXT
PRINT "Результат = ",s
END
```

Программа перебирает числа от 10 до 99 каждый раз выполняя действия **s=s+I**. С точки зрения математики это совершенно бессмысленная запись, но рассмотрим её внимательней.
Процесс решения вычислительной задачи - это процесс последовательного изменения значений переменных. В итоге - в определенных переменных получается результат. Переменная получает определенное значение в результате *присваивания*. Вы помните, что присваивание - это занесение в ячейку, отведенную под переменную, определенного значения в результате выполнения команды.
В результате операции **a=5** переменная **a** получает значение 5.
В результате операции **c=a+b** переменная **c** получает значение равное сумме значений переменной **a** и **b**.
В результате операции **s=s+I** переменная **s** получает значение равное сумме предыдущего значения переменной **s** и значения переменной **I**. Т.е., если до операции присваивания значение **s** было равно 5, а переменной **I** равно 3, то после операции значение переменной **s** будет равно 8 (5+3, старое значение **s** + значение **I**).

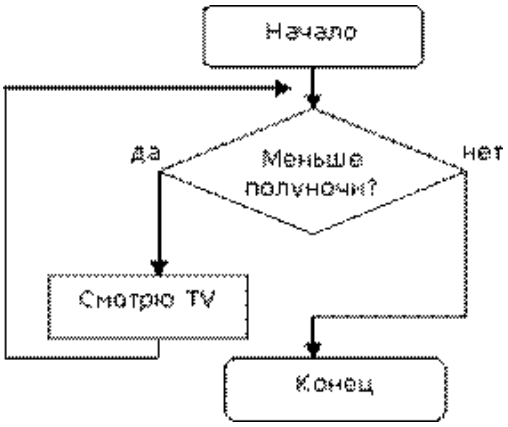
Значит после выполнения нашей программы в переменной s будет хранится сумма всех двузначных чисел от 10 до 99.

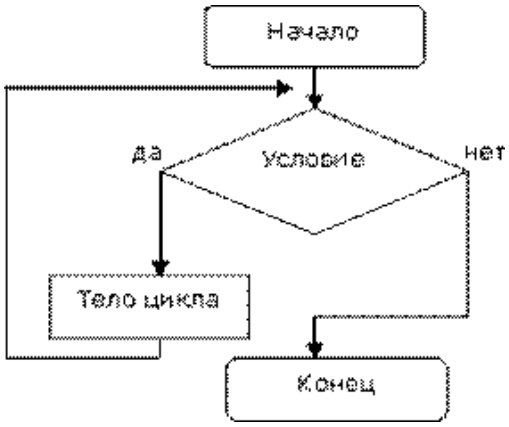
Пример: вычислить факториал числа a (записывается так: a!). Факториал - это произведение чисел от 1 до a. Например, 5! (факториал пяти) - это 5!=1*2*3*4*5

```
REM Вычислить факториал числа
a=5
f=1
FOR I=1 TO a
f=f*I
NEXT
PRINT f
END
```

Вы, конечно, заметили, что до начала цикла мы присвоили переменной f значение равное единице. Иначе бы мы получили в результате ноль.

В субботу вечером вы смотрите телевизор. Время от времени поглядываете на часы и если время меньше полуночи, то продолжаете смотреть телевизор, если это не так, то вы прекращаете просмотр телепередач.





Циклы такого вида называют - **циклы с предусловием**.

На языке Basic они записываются следующим образом:

DO WHILE условие
Тело цикла
LOOP

В этом цикле проверяется условие и если оно выполняется (ИСТИНА), то выполняется тело цикла до ключевого слова LOOP, затем условие проверяется снова ... и так до тех пор пока условие истино.

DO UNTIL условие
Тело цикла
LOOP

Этот цикл отличается от предыдущего только тем, что он выполняется до тех пор пока условие не истинно (т.е. совсем наоборот).

Пример: Вывести все натуральные числа меньше данного.

```
REM Вывод всех чисел меньше данного
a=0
chislo=10
DO WHILE a<chislo
PRINT a
a=a+1
LOOP
END
```

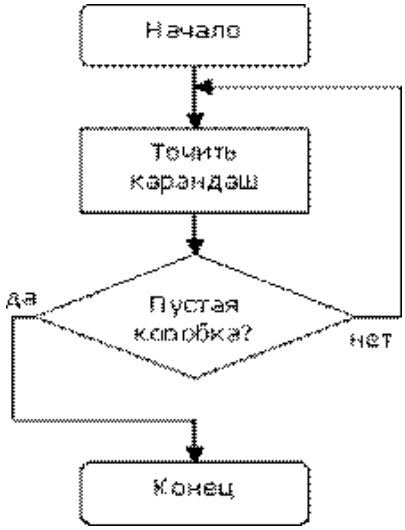
Стоит обратить внимание на то, что цикл *может быть не выполнен ни разу* (если условие первоначально не истинно, например, **a**=5, а **chislo**=4). И наоборот, если

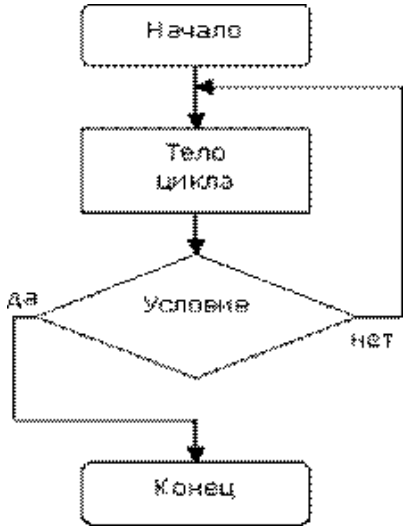
условие будет истинно при любых значениях переменный, то цикл будет *выполнятся бесконечное число раз* (произойдет *зацикливание*).

Пример цикла, который будет выполнятся бесконечное число раз:

```
REM зацикливание
DO WHILE a=a
PRINT "Это сообщение будет выводится на экран постоянно"
LOOP
PRINT "А это сообщение вы не увидите никогда"
END
```

Вам надо поточить все карандаши в коробке. Вы точите один карандаш и откладываете его в сторону. Затем проверяете, остались ли карандаши в коробке. Если условие ложно, то снова выполняется действие 'заточить карандаш'. Как только условие становится истинным, то цикл прекращается.





Циклы такого вида называют - **циклы с постусловием**.

На языке Basic они записываются следующим образом:

DO
Тело цикла
LOOP WHILE условие

DO
Тело цикла
LOOP UNTIL условие

Циклы такого рода отличаются тем, что хоть один раз, но тело цикла будет выполнено вне зависимости от условия. Условие проверяется после первого выполнения тела цикла.

Пример: Вычислите сумму цифр в числе.

```
REM Сумма цифр числа
DIM a, chislo, s AS INTEGER
INPUT "Введите число: ", chislo
a=chislo
DO
s=s+a MOD 10
a=a/10
a=INT(a)
LOOP UNTIL a=0
PRINT "Сумма цифр числа ",chislo ," равна: ", s
END
```

Переменную **s** используем для хранения суммы цифр. До начала цикла в переменную **a** заносим значение переменной **chislo**. Все дальнейшие преобразования осуществляем с переменной **a**. В цикле берем остаток от деления на 10 (последняя цифра числа) и прибавляем к тому, что уже есть в переменной **s**; делим значение переменной **a** на 10, берем целую часть (т.е. отбрасываем

последнюю цифру числа) и заносим в **a**. Цикл продолжается до тех пор пока значение переменной **a** не окажется равным нулю (перебрали все цифры числа). Результат выводим на экран.

Массивы. Одномерные массивы.

При работе с большим числом данных одного типа очень удобно использовать массивы.

Итак, что же такое массивы...
Массив, это разновидность переменной. Он дает возможность хранить сколько угодно значений под одним и тем же именем. К каждому конкретному значению массива, необходимо обращаться через числовой индекс.

Массив - это набор переменных, имеющих одинаковое имя (идентификатор), но различающихся порядковыми номерами (индексами).

Обычно массивы применяют для группировки переменных, имеющих много общих свойств. Например, если в классе 30 учеников, то имя каждого ученика можно было бы сохранить в отдельной строковой переменной: name1, name2, ... Но вводить 30 новых переменных крайне неудобно. Можно сделать проще: объявить один массив name(), имеющий 30 элементов. В скобках проставляется индекс когда надо обратиться к какому-то конкретному элементу.

Отсчет элементов массива во многих языках начинается с нуля. Поэтому имя первого (по классному журналу) ученика будет храниться в переменной name(0), второго - в переменной name(1), а последнего (тридцатого) - в переменной name(29).

Для того чтобы использовать массив его надо сначала объявить в программе. Для этого используют оператор **DIM**. По умолчанию (если нет оператора **DIM** в программе) считается заданным массив из 10 элементов.

Пример:

DIM a(100) AS INTEGER
Это массив из ста элементов, каждый из которых может быть целым числом.
DIM name(30) AS STRING
DIM mas(20)
Это массив из 20 элементов, тип переменных явно не указан.

DIM mas1(10) AS INTEGER

mas1

5	2	23	111	65	87	65	333	7	21
0	1	2	3	4	5	6	7	8	9

Обращение к элементам массива:

a(24)
name(5)
mas(2)
mas(3)

Основное преимущество массивов перед обычным набором разноименных переменных состоит в том. что индекс нужного элемента можно записывать не числом, а переменной или даже вычислять по выражению. Это дает возможность использовать массивы внутри циклов - собственно для этого они и были придуманы. Если в программе есть массив, то, скорее всего, в ней же вы найдете и цикл.

Можно также объявить массив и таким образом:

```
DIM mas2(1 TO 10) AS INTEGER
```

mas2

3	66	34	76	2	99	345	2	90	4
1	2	3	4	5	6	7	8	9	10

или даже так:

```
DIM a2(5 TO 10) AS INTEGER
```

В чем отличие? В том что данном случае индексация элементов массива начинается не с нуля, а с нужного вам индекса (в примере массив **mas2** имеет индексы от 1 до 10, массив **a2** - от 5 до 10).

Допустим в классе 30 учеников. Предположим, что для хранения их оценок по предмету создан массив **DIM mark(30) AS INTEGER**. Следующая программа, поставит каждому учащемуся случайную оценку от 3 до 5. Конечно, так расставлять оценки нельзя, но этот пример показывает, что программа не становиться сложнее, если в классе не 30 учеников, а сто пятьдесят миллионов. Сочетание массивов и циклов позволяет достичь удивительной простоты.

```
REM Выставление оценок :)
DIM mark(30) AS INTEGER
FOR I=0 TO 29
mark(I)=3+INT(RND*3)
NEXT
END
```

mark

3	3	5	3	4	5	5	3	...	4
0	1	2	3	4	5	6	7	...	29

Для создания случайных чисел в языке Basic служит стандартная функция **RND**. Она создает случайное число в диапазоне от 0 до 1. Умножив его на 3, мы получаем случайное число от 0 до 3. А взяв от него целую часть (с помощью функции INT), получим целое случайное число в диапазоне от 0 до 2. Прибавив к нему число 3, мы поучаем случайную оценку, которая не меньше 3 и не больше 5.

Пример: Составить программу заполнения массива из 15 элементов случайными числами в диапазоне от 1 до 10. Предусмотреть вывод массива на экран.

```
REM Заполнение и вывод массива
DIM mas(15) AS INTEGER
FOR I=0 TO 14
mas(I)=1+INT(RND*10)
NEXT
CLS
PRINT "Вывод массива"
FOR I=0 TO 14
PRINT mas(I);
NEXT
END
```

CLS - очистка экрана. Точка с запятой (;) в операторе **PRINT** позволяет выводить элементы массива в строку.

Тоже самое задание, но отличающиеся объявлением массива:

```
REM Заполнение и вывод массива
DIM mas(1 TO 15) AS INTEGER
FOR I=1 TO 15
mas(I)=1+INT(RND*10)
NEXT
CLS
PRINT "Вывод массива"
FOR I=1 TO 15
PRINT mas(I);
```

```
NEXT
END
```

Всё очень просто. Какой из вариантов использовать решать вам.

Пример: Вывести количество отрицательных элементов массива.

```
REM Вывести количество отрицательных элементов
INPUT "Введите число элементов массива", n
DIM mas(n) AS INTEGER
FOR I=0 TO n-1
INPUT "Введите элемент массива", mas(I)
NEXT
CLS
PRINT "Вывод массива"
FOR I=0 TO n-1
PRINT mas(I);
NEXT
FOR I=0 TO n-1
IF mas(I)<0 THEN k=k+1
NEXT
PRINT
PRINT "Число отрицательных элементов: ",k
END
```

Подсчет количества отрицательных элементов массива происходит в цикле:
FOR I=0 TO n-1
IF mas(I)<0 THEN k=k+1
NEXT

Пример: Составить программу для вычисления наибольшего элемента массива и его номера.

```
REM вычисления наибольшего элемента массива и его номера
INPUT "Введите число элементов массива", n
DIM mas(n) AS INTEGER
FOR I=0 TO n-1
INPUT "Введите элемент массива", mas(I)
NEXT
CLS
PRINT "Вывод массива"
FOR I=0 TO n-1
PRINT mas(I);
NEXT
max=mas(0)
nomer=1
FOR I=0 TO n-1
IF mas(I)>max THEN
max=mas(I)
nomer=I+1
END IF
NEXT
PRINT
PRINT "Максимальный элемент: ", max, " с номером ", nomer
END
```

Задание выполняется в строчках:
max=mas(0)
nomer=1
FOR I=0 TO n-1
IF mas(I)>max THEN
max=mas(I)
nomer=I+1
END IF
NEXT
Вначале примем за наибольший элемент - первый элемент массива **mas(0)**. Затем перебирая все элементы по очереди сравниваем их со значение **max** и если **mas(I)>max**, то принимаем этот элемент за наибольший.

Пример: составить программу сортировки массива по возрастанию.

```
REM сортировка массива
INPUT "Введите число элементов массива", n
DIM mas(n) AS INTEGER
FOR I=0 TO n-1
mas(I)=1+INT(RND*10)
NEXT
CLS
PRINT "Вывод массива"
FOR I=0 TO n-1
PRINT mas(I);
NEXT
REM сортировка массива
FOR I=0 TO n-2
FOR J=I+1 TO n-1
IF mas(I)>mas(J) THEN
REM если нашли меньший элемент, то обменяем их местами
a=mas(I)
mas(I)=mas(J)
mas(J)=a
END IF
NEXT J
NEXT I
REM конец сортировки массива
PRINT
PRINT "Вывод отсортированного массива"
FOR I=0 TO n-1
PRINT mas(I);
NEXT
END
```

Иногда для ввода данных удобно использовать операторы **DATA** и **READ**. **DATA** указывает значения для чтения последующими операторами **READ**. **READ** считывает эти значения и присваивает их переменным. **RESTORE** позволяет **READ** заново считать значения в указанном операторе **DATA**.

```
DATA константы
READ переменные
```

Пример: ввод массива с использование оператора DATA.

```
REM Ввод данных из DATA
DIM mas(5) AS INTEGER
DATA 2, -4, 1, 5, 9
REM ввод массива
FOR I=0 TO 4
READ mas(I);
NEXT
REM вывод массива
FOR I=0 TO 4
PRINT mas(I);
NEXT
END
```

Язык программирования Basic. Учимся программировать! Конспекты школьника

Массивы. Двумерные массивы.

Двумерные массивы можно представить себе как таблицы, в ячейках которых хранятся значения элементов массива, а индексы элементов массива являются номерами строк и столбцов.

Объявляются двумерные массивы так же, как переменные и одномерные массивы. Например, целочисленный числовой массив, содержащий 3 строк и 4 столбца объявляется следующим образом:

DIM tabl(3 ,4)

DIM tabl(3 ,4) AS INTEGER

tabl

	0	1	2	3
0	2	7	8	3
1	22	1	3	34
2	5	56	9	777

DIM tabl1(1 TO 3 ,1 TO 4) AS INTEGER

tabl1

	1	2	3	4
1	2	7	8	3
2	22	1	3	34
3	5	56	9	777

С помощью двумерного массива 9x9 и двух вложенных циклов можно легко составить программу, реализующую таблицу умножения. Сомножителями будут значения индексов строк и столбцов, а их произведения будут значениями элементов массива.

DIM tablum(1 TO 9 ,1 TO 9) AS INTEGER

tablum

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

```
REM Таблица умножения
DIM tabum(1 TO 9, 1 TO 9) AS INTEGER
REM Заполнение массива - создание таблицы умножения
FOR I=1 TO 9
FOR J=1 TO 9
tabum(I, J)=I*J
NEXT J
NEXT I
REM Вывод массива на экран в виде таблицы
FOR I=1 TO 9
FOR J=1 TO 9
PRINT tabum(I,J);
NEXT J
```

```
PRINT
NEXT I
END
```

Пример: В таблице 3x4 вычислить количество отрицательных элементов, сумму четных элементов, произведение элементов второй строки.

```
REM вычислить количество...
DIM tabl(1 TO 3, 1 TO 4) AS INTEGER
REM Заполнение массива
FOR I=1 TO 3
FOR J=1 TO 4
INPUT "Введите элемент массива:", tabl(I, J)
NEXT J
NEXT I
REM Вывод массива на экран в виде таблицы
CLS
FOR I=1 TO 3
FOR J=1 TO 4
PRINT tabl(I,J);
NEXT J
PRINT
NEXT I
REM требуемые вычисления
k=0
s=0
p=1
FOR I=1 TO 3
FOR J=1 TO 4
IF tabl(I, J)<0 THEN k=k+1
IF tabl(I, J) MOD 2 = 0 THEN s=s+tabl(I, J)
IF I=2 THEN p=p*tabl(I, J)
NEXT J
NEXT I
PRINT
PRINT "результ:"
PRINT "отрицательных элементов: ", k
PRINT "сумма четных элементов: ", s
PRINT "произведение элементов второй строки: ",p
END
```

Язык программирования Basic. Учимся программировать! Конспекты школьника

Символьные и строчные переменные.

Очень часто в программах требуется использовать символьные или строчные переменные. Что же это такое? Это переменные, значениями которых являются либо алфавитно-цифровые символы, либо несколько таких символов.

Строки - последовательность алфавитно-цифровых символов.

Для того, чтобы использовать такие переменные в программе необходимо их соответствующим образом объявить. Для этого используется уже известный оператор **DIM**.

```
DIM s AS STRING
s="Строка123"
```

Или добавлять справа от переменной символ **$**.

```
s$="Тоже строка 987"
```

Пример: Эта программа выводит на экран две строки. Обратите внимание на два способа использования и объявления строковых переменных.

```
DIM stroka AS STRING
stroka="Один"
stroka2$="Два"
PRINT stroka
```

```
PRINT stroka2$
END
```

Еще пример: Та же программа, но с ОШИБКАМИ. Не указано, то что наши переменные строчные.

```
REM ЗДЕСЬ ОШИБКИ
stroka="Один"
stroka2="Два"
PRINT stroka
PRINT stroka2
END
```

Строчные переменные можно склеивать и сравнивать друг с другом. Для склеивания строк (конкатенации) используют знак плюс (+).

Пример.

```
REM конкатенация строк
s1$="Привет! "
s2$="Меня зовут Саша."
s$=s1$+s2$
PRINT s$
END
```

На экране появится надпись: Привет! Меня зовут Саша.

Для сравнения срок используют операции: >, <, =, >=, <=, <>.

Пример:

```
REM Сравнение строк
s1$="abc"
s2$="abc"
s3$="klmn"
IF s1$=s2$ THEN PRINT "Строки равны"
IF s1$=s3$ THEN PRINT "Строки равны"
END
```

Программа выведет "Строки равны только один раз".

Функции для работы со строками:

LEN(s$)	Вычисляет длину строки (количество символов).
MID$(s$,n,k)	Выделяет из строки s$ k символов начиная с n-го символа.
VAL(s$)	Преобразует числовую часть начала строки в число.
STR$(x)	Преобразует число в символьную форму.
ASC(s$)	Вычисляет десятичный код символа.
CHR$(x)	Преобразует код в символ.
INKEY$	Функция опроса клавиш, нажатых на клавиатуре.

Пример: составить программу подсчитывающую, количество букв "а" в предложении.

```
REM кол-во букв "а"
INPUT "Введите предложение", s$
FOR I=1 TO LEN(s$)
IF MID$(s$,I,1)="а" THEN k=k+1
NEXT
PRINT "Кол-во букв а =", k
END
```

Пример: Заменить все буквы "а" в предложении на буквы "о".

```
REM замена букв
ss$=""
INPUT "Введите предложение", s$
FOR I=1 TO LEN(s$)
IF MID$(s$,I,1)="а" THEN
ss$=ss$+"о"
```

```
ELSE
ss$=ss$+MID$(s$,I,1)
END IF
NEXT
PRINT "Исправленная строка: ", ss$
END
```

Пример: Получить предложение в обратном порядке следования символов.

```
REM обратный порядок букв
ss$=""
INPUT "Введите предложение", s$
FOR I=LEN(s$) TO 1 STEP -1
ss$=ss$+MID$(s$,I,1)
NEXT
PRINT "Исправленная строка: ", ss$
END
```

Язык программирования Basic. Учимся программировать! Конспекты школьника

Подпрограммы. Процедуры.

При создании средних по размеру программ используется *структурное программирование*, идея которого заключается в том, что структура программы должна отражать структуру решаемой задачи, чтобы алгоритм решения был ясно виден из исходного текста.

С этой целью в программирование введено понятие *подпрограммы - набора операторов (команд), выполняющих нужное действие и не зависящих от других частей исходного кода*. Программа разбивается на множество подпрограмм, каждая из которых выполняет какое-то действие, предусмотренное исходным заданием.

Подпрограммой называется группа операторов, к которой обращаются из основной программы несколько раз.

Комбинируя подпрограммы, удается сформировать итоговый алгоритм используя блоки кода (подпрограммы), имеющих определенную смысловую нагрузку. Обращаться к этим подпрограммам можно по их имени.

Принято различать два вида подпрограмм - процедуры и функции. Впрочем, это деление весьма условно, потому что они очень близки. Отличаются они тем, что процедура просто выполняет группу операторов, а функция вдобавок вычисляет некоторое значение и передает его в программу.

Когда в программе необходимо выполнить какое-то стандартное действие происходит вызов процедуры. Процедура выполняет действие и возвращает управление обратно программе, которая ее вызвала. В ходе работы процедуры могут вызвать другие процедуры. Прием когда подпрограмма вызывает саму себя называют рекурсией.

Очень важная характеристика подпрограмм - это возможность их повторного использования.

Чтобы работа подпрограммы имела смысл, ей надо получить данные из внешней программы, которая эту подпрограмму вызывает. Данные передаются подпрограмме в виде параметров или аргументов, которые обычно описываются в ее заголовке так же, как и переменные.

Вы уже использовали стандартные процедуры и функции при составлении программ. Теперь пришло время научиться создавать свои процедуры и функции.

Процедуры состоят из трех частей: заголовка, тела процедуры, завершения процедуры.

SUB имя (список параметров)
тело процедуры - список операторов
END SUB

Пример:

```
SUB hello (s$)
PRINT "Привет, ", s$,"! Как твои дела?"
END SUB

REM приветствие
name1$="Саша"
name2$="Вася"
REM процедуру можно вызвать так
CALL hello(name1$)
REM а можно вызвать так
hello(name2$)
REM или даже так
hello("Марина")
END
```

В результате выполнения программы на экране будет выведено:
Привет, Саша! Как твои дела?
Привет, Вася! Как твои дела?
Привет, Марина! Как твои дела?

Параметры, которые указываются в заголовке подпрограммы, называются формальными. Они нужны только для описания тела подпрограммы. А параметры (конкретные значения), которые указываются в момент вызова подпрограммы, называются фактическими параметрами. При выполнении операторов подпрограммы формальные параметры как бы временно заменятся на фактические.

Подпрограммы. Функции.

Функции отличаются от процедур тем, что не только выполняют определенные действия, но еще и возвращают вызывающей программе какое-то значение.

Процедуры и функции бывают стандартными и нестандартными. Стандартные подпрограммы входят в библиотеку, которая поставляется вместе с системой программирования. Нестандартные процедуры и функции программисты пишут сами.

Вы уже использовали стандартные функции, теперь давайте напишем свою функцию.

FUNCTION имя (список параметров)
тело функции - список операторов
END FUNCTION

Пример: функция возвращающая куб числа

```
FUNCTION kub (x)
kub=x*x*x
END FUNCTION

REM Вывод кубов натуральных чисел от 1 до 10
CLS
FOR I=1 TO 10
PRINT kub(I)
NEXT
END
```

В этой программе в цикле происходит обращение к функции **kub**, которая вычисляет куб числа.

Процесс, когда в процедуре происходит обращение к самой себе, называется рекурсией (рекурсия - возврат). (Происходит от латинского recurreus - возвращающийся).

Рекурсия - это такой способ организации подпрограммы, при котором в ходе выполнения она обращается сама к себе.

Ниже приведена программа вычисления факториала числа, в которой используется рекурсивная процедура **fak**:

```
FUNCTION fak (f)
IF f = 0 OR f = 1 THEN
fak = 1
ELSE
fak = fak(f - 1) * f
END IF
END FUNCTION

REM "Вычисление факториала"
INPUT "Введите число: ", a
PRINT "Факториал = ", fak(a)
END
```

Для вычисления факториала числа n, т.е. n! надо умножить последовательно n натуральных чисел от 1 до n: n!=1*2*3*4. Так, 4! будет равно: 4!=1*2*3*4. Это прямой путь вычисления или итеративный.
Возможен и другой путь вычисления: n!=n*(n-1)*...*1. Т.е. 4!=4*3*2*1. Этот путь можно назвать возвратным или рекурсивным.
Именно на этом принципе основана работа приведенной функции.

Графический режим работы.

Ну и теперь, наверное, самое интересное. Будем рисовать. Кто же не любит это занятие?!

Программы могут выводит данные на экран в текстовом и графическом режиме работы. Для перехода в графический режим работы служит оператор:

SCREEN <mode>

<mode> - целочисленная константа, указывающая режим работы для данного экрана и адаптера.

Пример:

```
SCREEN 1
SCREEN 2
...
SCREEN 11
...
```

Для рисования можно использовать следующие операторы:

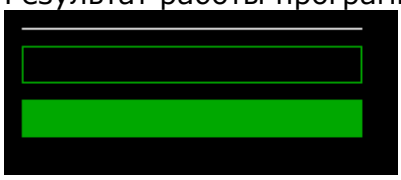
CLS	Очистка экрана
PSET(X,Y),C	Изобразить точку. X,Y - координаты точки, C -цвет.
PSET STEP(X,Y),C	Изобразить точку. X,Y - смешение от данной точки, C - цвет.
LINE(X1,Y1)-(X2,Y2),C	Прямая линия.X1,Y2 и X2,Y2- координаты концов линии, C - цвет.
LINE -(X2,Y2),C	Прямая линия. От текущего положения курсора до X2,Y2- координаты конца линии, C - цвет.
LINE(X1,Y1)-(X2,Y2),C,B	Прямоугольник. X1,Y2 и X2,Y2- координаты концов диагонали, C - цвет.
LINE(X1,Y1)-(X2,Y2),C,BR	Закрашенный прямоугольник. X1,Y2 и X2,Y2- координаты концов диагонали, C - цвет.
CIRCLE(X,Y),R,C	Окружность. X,Y - координаты центра, C -цвет.
CIRCLE STEP(X,Y),R,C	Окружность. X,Y - смешение от данной точки, C -цвет.
CIRCLE(X,Y),R,C,A1,A2	Дуга окружности. X,Y - координаты центра, C -цвет, A1,A2 - угловые меры начальной и конечной точки дуги.
CIRCLE(X,Y),R,C,,, K CIRCLE s(X,Y),R,C, A 1, A 2,K	Элипс. K - коэффициент сжатия.

PAINT(X,Y),C1,C2	Закрасить область. C1 - цвет закраски, C2 - цвет границы.
LOCATE T1,T2	Установка курсора в данную позицию. T1, T2 - номер строки и столбца.
PRINT	Оператор вывода текста

Пример: использования **LINE**

```
REM использование LINE
SCREEN 12
LINE (10, 10)-(200, 10)
LINE (10, 20)-(200, 40), 2, B
LINE (10, 50)-(200, 70), 2, BF
END
```

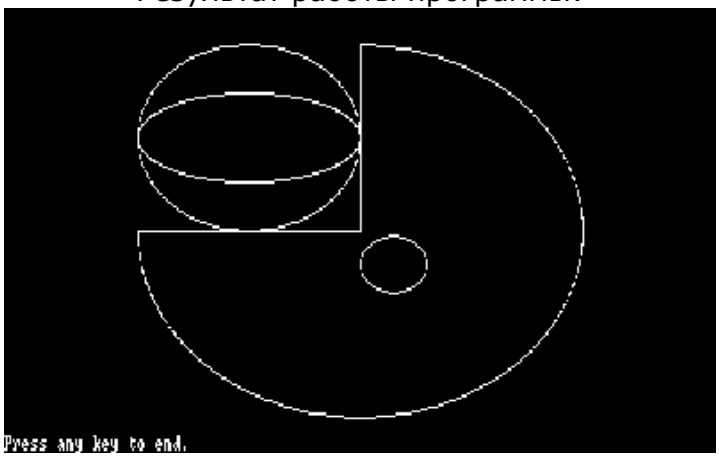
Результат работы программы:



Пример: использование **CIRCLE**

```
REM ОКРУЖНОСТЬ, ДУГА, ЭЛЛИПС
CONST PI = 3.141593
SCREEN 2
REM ОКРУЖНОСТЬ
CIRCLE (350, 115), 30
REM ДУГА ОКРУЖНОСТИ
CIRCLE (320, 100), 200, , -PI, -PI / 2
REM ОКРУЖНОСТЬ
CIRCLE STEP(-100, -42), 100
REM ЭЛЛИПС
CIRCLE STEP(0, 0), 100, , , , 5 / 25
REM ВЫВЕСТИ НАДПИСЬ В СТРОКЕ 25 И СТОЛБЦЕ 1
LOCATE 25, 1: PRINT "Press any key to end.";
REM ЖДЕМ НАЖАТИЯ ЛЮБОЙ КЛАВИШИ
DO
LOOP WHILE INKEY$ = ""
```

Результат работы программы:



Пример: построение окружности

```
REM окружность
CLS
INPUT "Введите координаты центра x,y: ", x,y
INPUT "Введите радиус окружности R: ", r
SCREEN 1
CIRCLE (x, y), r
END
```

Сейчас на улице зима, а значит и ...
Пример: программа "Снеговик"

```
REM Снеговик
SCREEN 12
```

```
x = 320
y = 240
r = 50
c = 3
c1 = 8
c2 = 5
c3 = 6

REM Снег
FOR i = 1 TO 300
PSET (RND * 640, RND * 480), 1
NEXT

REM Сугроб
FOR i = 1 TO 20
LINE (0 + 2 * i, y + 3 * r + 61 - i)-(640 - 2 * i, y + 3 * r + 61 - i), 1
NEXT

REM Тело снеговика
CIRCLE (x, y - 80), r - 20, c
CIRCLE (x, y), r, c
CIRCLE (x, y + 120), r + 20, c
CIRCLE (x - 52, y - 30), 10, c
CIRCLE (x + 52, y - 30), 10, c

PAINT (x, y - 80), c, c
PAINT (x, y), c, c
PAINT (x, y + 120), c, c
PAINT (x - 52, y - 30), c, c
PAINT (x + 52, y - 30), c, c

REM Оформление лица
CIRCLE (x - 15, y - 90), 2, 1
PAINT (x - 15, y - 90), 1, 1
CIRCLE (x + 15, y - 90), 2, 1
PAINT (x + 15, y - 90), 1, 1
CIRCLE (x, y - 80), 10, 4, , , .5
PAINT (x, y - 80), 4, 4
CIRCLE (x, y - 75), 20, 4, 4, 6, 0.5

REM Метла
LINE (x - 50, y - 100)-(x - 54, y + 100), c1, BF
FOR i = 1 TO 20
LINE (x - 53, y - 100)-(x - 53 - 40 * RND + 20, y - 100 - 40 * RND), c1
NEXT

REM Ведро
LINE (x - 30, y - r - 2 * (r - 20) + 10)-(x + 30, y - r - 2 * (r - 20) + 10), c2
LINE (x - 15, y - r - 2 * (r - 20) - 30)-(x + 15, y - r - 2 * (r - 20) - 30), c2
LINE (x - 30, y - r - 2 * (r - 20) + 10)-(x - 15, y - r - 2 * (r - 20) - 30), c2
LINE (x + 15, y - r - 2 * (r - 20) - 30)-(x + 30, y - r - 2 * (r - 20) + 10), c2
PAINT (x, y - r - 2 * (r - 20)), c2, c2

REM Пуговицы
FOR i = 1 TO 5
CIRCLE (x, y - r + 30 * i), 3, c3
PAINT (x, y - r + 30 * i), c4, c3
NEXT

END
```

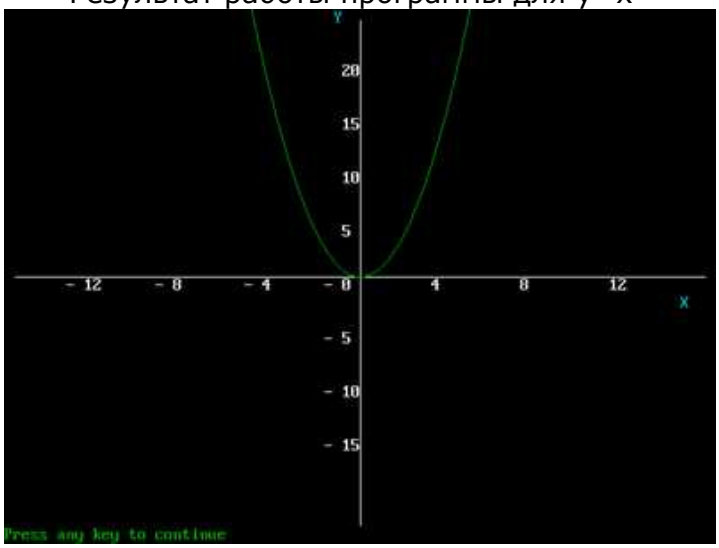
Результат работы программы:

Пример: построение графика функции.

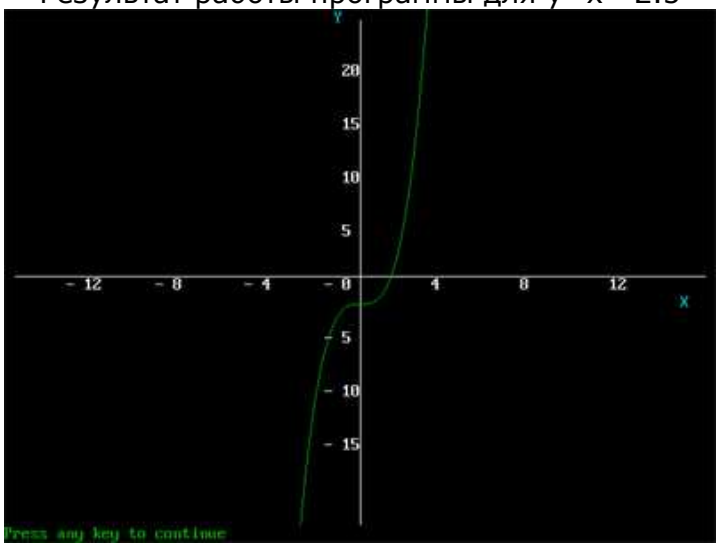
```
FUNCTION F (x)
F = x * x
END FUNCTION

REM ГРАФИК ФУНКЦИИ
SCREEN 12
REM Строим оси
FOR I = 0 TO 3
LOCATE 16, 38 + 10 * I: PRINT 4 * I
NEXT
FOR I = 0 TO 3
s$ = "-" + STR$(4 * I)
LOCATE 16, 38 - 10 * I: PRINT s$
NEXT
FOR I = 0 TO 4
LOCATE 16 - 3 * I, 38: PRINT 5 * I
NEXT
FOR I = 0 TO 3
s$ = "-" + STR$(5 * I)
LOCATE 16 + 3 * I, 37: PRINT s$
NEXT
LINE (10, 240)-(630, 240)
LINE (320, 10)-(320, 470)
COLOR 3
LOCATE 1, 38: PRINT "Y"
LOCATE 17, 77: PRINT "X"
REM Строим график функции
COLOR 2
PSET (20*(-10) + 320, 240 - 10 * F(-10))
FOR x = -10 TO 10 STEP .1
XX = 20 * x + 320
YY = 240 - 10 * F(x)
LINE -(XX, YY)
NEXT
END
```

Результат работы программы для $y=x^2$

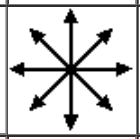


Результат работы программы для $y=x^3-2.5$



Оператор **DRAW** позволяет выполнять разнообразные графические операции.

Команды черчения по восьми направлениям: **U, D, L, R, E, F, G, H**.

H	U	E
L		R
G	D	F

- Команда **M** чертит от текущей точки до точки с координатами x, y. Если перед координатами стоит знак плюс или минус, то координаты относительные, а не абсолютные.
- Команда **S** изменяет масштаб черчения.
- Команда **A** поворачивает изображение (**A0**-0, **A1**-90, **A2**-180, **A3**-270 градусов).

Пример: Рисование флагов.

```
REM Флаги
SCREEN 1
PSET (50, 10)
DRAW "R20 G5 F5 L20 U10"
DRAW "B D20"
DRAW "S5 R20 G5 F5 L20 U10"
DRAW "BD20"
DRAW "S4 R20 G5 F5 L20 U10"
DRAW "BD20"
DRAW "R20 G5 F5 L20 U10"
DRAW "BD40"
```

```
DRAW "A3 R20 G5 F5 L20 U10"
END
```

Результат работы программы:

Язык программирования Basic. Учимся программировать! Конспекты школьника

Создание движущихся графических изображений в Basic.

Как нарисовать графический объект в Basic вам уже понятно. Но, как заставить это изображение двигаться с помощью языка программирования Basic?

Очень просто!
1. Рисуем объект цветом отличным от цвета фона.
2. Рисуем объект цветом фона.
3. Изменяем координаты.
4. Повторяем 1-3 столько раз сколько потребуется.

Пример 1: Движущийся графический круг в Basic.

```
REM Движущийся круг
SCREEN 1
x = 1
y = 1
REM цвет фона - 0(черный), цвет рисунка - 1

FOR i = 1 TO 150

REM Рисуем объект цветом отличным от цвета фона.
c = 1
CIRCLE (x, y), 2, c

REM задержка
FOR j = 1 TO 250000
NEXT j

REM Рисуем объект цветом цветом фона.
c = 0
CIRCLE (x, y), 2, c

REM Изменяем координаты
x = x + 2
y = y + 1

NEXT i
END
```

Для того чтобы глаз мог зафиксировать нарисованное изображение используем пустой цикл:

```
REM задержка
FOR j = 1 TO 250000
NEXT j
```

Пример 2: Усложним траекторию движения. Пусть шарик прыгает по поверхности, а когда поверхность закончится - упадет вниз.

```
REM Прыгающий шарик
SCREEN 1
x = 1
y = 100

REM поверхность
LINE (0, y + 20)-(220, y + 20)
```

```
FOR i = 1 TO 140

c = 1
CIRCLE (x, y), 2, c
PAINT (x, y), c, c

FOR j = 1 TO 250000
NEXT j

c = 0
CIRCLE (x, y), 2, c
PAINT (x, y), c, c

x = x + 2
IF i < 115 THEN
y = y + 10 * COS(.5 * i)
ELSE
y = y + 4
SOUND 200, 1
END IF

NEXT i

LOCATE 10, 15: PRINT "GAME OVER :)"
END
```

Пример 3: Шарик, заключенный в прямоугольную область. При касании границ отскакивает обратно.

```
REM Шарик, заключенный в прямоугольную область
SCREEN 1

REM Границы области
xx1 = 1
xx2 = 200
yy1 = 1
yy2 = 150

LINE (xx1, yy1)-(xx1, yy2)
LINE (xx2, yy1)-(xx2, yy2)
LINE (xx1, yy1)-(xx2, yy1)
LINE (xx1, yy2)-(xx2, yy2)

REM Начальные координаты и скорость шарика
x = RND * (xx1 + (xx2 - xx1) / 2)
y = RND * (yy1 + (yy2 - yy1) / 2)
vx = RND * 20 - 10
vy = RND * 20 - 10

CIRCLE (x, y), 2, c
PAINT (x, y), c, c

REM Движение шарика, до тех пор пока не нажмем любую клавишу

DO

c = 1
CIRCLE (x, y), 2, c
PAINT (x, y), c, c

FOR j = 1 TO 150000
NEXT j

c = 0
CIRCLE (x, y), 2, c
PAINT (x, y), c, c

IF x < (xx1 + 5) OR x > (xx2 - 5) THEN vx = -vx
IF y < (yy1 + 6) OR y > (yy2 - 7) THEN vy = -vy
x = x + vx
```

```
y = y + vy

LOOP WHILE INKEY$ = ""

LOCATE 10, 15: PRINT "GAME OVER :)"
END
```

LOCATE - перемещает курсор на экране в указанную позицию.

Пример 4: Идущие часы (входит в состав примеров QBasic 4.5).

```
' *** DRAW_EX.BAS ***
'
' Объявление процедуры.
DECLARE SUB Face (Min$)
'
' Установка графического режима 640 x 200
SCREEN 2
DO
CLS
' Получаем строковое значение количества минут
Min$ = MID$(TIME$,4,2)
' Рисуем изображение часов
Face Min$
' Ждем пока не изменится минута или пока не будет нажата клавиша
DO
' Печатаем время вверху экрана
LOCATE 2,37
PRINT TIME$
' Проверяем нажатие клавиши
Test$ = INKEY$
LOOP WHILE Min$ = MID$(TIME$,4,2) AND Test$ = ""
' Конец программы если нажата клавиша
LOOP WHILE Test$ = ""
END
'
' Процедура рисования часов
SUB Face (Min$) STATIC
LOCATE 23,30
PRINT "Press any key to end"
CIRCLE (320,100),175
' Преобразовываем строку в число
Hr = VAL(TIME$)
Min = VAL(Min$)
' Преобразовываем число в угол
Little = 360 - (30 * Hr + Min/2)
Big = 360 - (6*Min)
' Рисуем стрелки
DRAW "TA=" + VARPTR$(Little) + "NU40"
DRAW "TA=" + VARPTR$(Big) + "NU70"
END SUB
```

INKEY$ считывает символ с клавиатуры.

Пример:

```
PRINT "Для выхода нажмите Esc..."
DO
LOOP UNTIL INKEY$ = CHR$(27)    '27 - это ASCII код для клавиши Esc.
```

- **INKEY$** возвращает нулевую строку символов, если нет символа для возврата.
- Для стандартных клавиш **INKEY$** возвращает 1-байтовую строку символов, содержащую считанный символ.
- Для расширенных клавиш **INKEY$** возвращает 2-байтовую строку символов, состоящую из символа нуля (ASCII 0) и скан-кода клавиатуры.

Несложно осуществить не просто движение объекта, а управляемое движение.
Пример 5: Художник (входит в состав примеров QBasic). Управление художников клавишами со стрелками.

```
' Значение для клавиш управления и пробела:
CONST UP = 72, DOWN = 80, LFT = 75, RGHT = 77
CONST UPLFT = 71, UPRGHT = 73, DOWNLFT = 79, DOWNRGHT = 81
CONST SPACEBAR = " "

' Null$ это первый байт(символ) 2-байтовой строки символов
' для расширенных клавиш (таких, например, как ВВЕРХ и ВНИЗ)
' значение которой возвращает INKEY$

Null$ = CHR$(0)

' Plot$ = "" рисование линий; Plot$ = "B" только перемещение
' Перемещаемся, но не рисуем линии:
Plot$ = ""

PRINT "Use the cursor movement keys to draw lines."
PRINT "Press the spacebar to toggle line drawing on and off."
PRINT "Press <ENTER> to begin. Press q to end the program."

' ждем нажатие клавиши для начала рисования
DO: LOOP WHILE INKEY$ = ""

SCREEN 1
CLS

DO
SELECT CASE KeyVal$
CASE Null$ + CHR$(UP)
DRAW Plot$ + "C1 U2"
CASE Null$ + CHR$(DOWN)
DRAW Plot$ + "C1 D2"
CASE Null$ + CHR$(LFT)
DRAW Plot$ + "C2 L2"
CASE Null$ + CHR$(RGHT)
DRAW Plot$ + "C2 R2"
CASE Null$ + CHR$(UPLFT)
DRAW Plot$ + "C3 H2"
CASE Null$ + CHR$(UPRGHT)
DRAW Plot$ + "C3 E2"
CASE Null$ + CHR$(DOWNLFT)
DRAW Plot$ + "C3 G2"
CASE Null$ + CHR$(DOWNRGHT)
DRAW Plot$ + "C3 F2"
CASE SPACEBAR
IF Plot$ = "" THEN Plot$ = "B " ELSE Plot$ = ""
CASE ELSE
' Пользователь нажал какую-то из клавиш,
' кроме клавиш управления (вверх, вниз, вправо, влево, пробел, выход(q))
' так что ничего не делаем
END SELECT

KeyVal$ = INKEY$

LOOP UNTIL KeyVal$ = "q"

END
```

Язык программирования Basic. Учимся программировать! Конспекты школьника

Работа с файлами.

Файлы широко применяются для решения различных задач. В них размещаются данные, предназначенные для длительного хранения. Каждому файлу присваивается уникальное имя, которое используется для обращения к нему. Использование файлов освобождает разработчика от хранения требуемых данных в тексте программы или многократном вводе их с клавиатуры, что само по себе весьма утомительно и приводит к появлению различных ошибок в программах. Гораздо удобнее ввести эту информацию один раз и сохранить ее в файле на диске.

Имена файлов состоят из двух частей, разделяемых точкой: filename.ext (имя_файла.расширение)
Имя файла может включать от 1 до 8 знаков, а соответствующее расширение - до трех знаков.

Имена файлов и расширения могут содержать следующие символы:

A-Z 0-9 () {} @ # $ % ^ ! - _ ' / ~

Файлы можно создавать, переименовывать, стирать; производить операции считывания и записи.

Basic предлагает три различных способа сохранения и востребования информации с диска: последовательный, прямой и двоичный ввод/вывод файла. У каждого есть свои преимущества и недостатки.

Метод последовательных файлов - это способ прямого чтения и записи файлов. Команды последовательных файлов в Basic создают текстовые файлы: файлы ASCII-символов с парами "возврат каретки/перевод строки", разделяющими записи. Вероятно одной из основных причин использования последовательных файлов является степень их "переносимости" в другие программы, языки программирования и компьютеры. Они читаются программами подготовки текстов и редакторами, принимаются другими прикладными программами и могут посылаться через серийные порты на другие компьютеры.

В основе последовательных файлов лежит сама простота: пишите в них так, словно они - экран, и читайте с них так, словно они - клавиатура.

Создание последовательного файла:

1. ОТКРЫТЬ файл в режиме последовательного ВВОДА. Для создания файла необходимо использовать оператор **OPEN**.

В последовательных файлах есть два пути подготовки файла к выводу:

OUTPUT (ВЫВОД): Если файл не существует- создается новый файл. Если файл уже существует, его содержание уничтожается, а сам файл рассматривается как новый.

APPEND (ДОБАВИТЬ В КОНЕЦ): Если файл не существует- создается новый файл. Если файл уже существует, любые данные дописываются в конец этого файла.

2. Ввод данных в файл. Используйте **WRITE# PRINT#** или **PRINT#USING** для записи данных в последовательный файл.

3. ЗАКРЫТИЕ файла. Оператор **CLOSE** закрывает файловую переменную после завершения всех операций ввода/вывода.

Для чтения последовательного файла:

1. ОТКРЫТЬ файл в режиме последовательного ВВОДА. Подготовить файл для считывания.

2. Считывать данные с файла. Использовать операторы **INPUT#**, **INPUT$**, или **LINE INPUT#**.

3. ЗАКРЫТЬ файл. Оператор **CLOSE** закрывает файловую переменную после выполнения всех операций ввода/вывода.

Недостаток последовательных файлов в том, что возможен только последовательный доступ к данным.

Можно создавать последовательные файлы двух типов: 1 - последовательные файлы с разделенными полями, где все поля на каждой строке файла разделяются (ограничиваются) особыми символами, и 2- неразделенные последовательные файлы, когда каждый файл выглядит абсолютно одинаково и на экране, и на распечатке. Эти два типа файлов создаются с помощью операторов **WRITE#** и **PRINT#.** соответственно. Используйте I**NPUT#**, **INPUT$**,

или **LINE INPUT#** для обратного считывания информации с последовательного файла любого типа.

Пример 1: записать строку в файл, считать строку из файла.

```
REM Работа с файлами. Пример 1.
REM Запись в файл
OPEN "file01.dat" FOR OUTPUT AS #1
A$ = "Это наша текстовая строка"
PRINT #1, A$
CLOSE #1
REM Чтение из файла
OPEN "file01.dat" FOR INPUT AS #1
INPUT #1, B$
PRINT b$    'вывод на экран
CLOSE #1
```

В результате выполнения программы будет создан файл "file01.dat" и файл будет содержать строку *Это наша текстовая строка.* Затем файл будет открыт для чтения и из него будет прочитана и выведена на экран данная строка.

Пример 2. Напишем программу для записи в файл отметки ученика на уроке. В файле будет хранится следующая информация:

ФИО	ДАТА	ОТМЕТКА
Иванов Иван	22 февраля	4
Петров Петя	3 марта	5
...	...	...

```
REM Работа с файлами. Пример 2.
REM Запись в файл
OPEN "journal.dat" FOR APPEND AS #1
INPUT "Введите ФИО", FIO$
INPUT "Введите дату", DAY$
INPUT "Введите отметку", MARK
WRITE #1, FIO$, DAY$, MARK
CLOSE #1
REM Чтение из файла
OPEN "journal.dat" FOR INPUT AS #1
INPUT #1, FIO$, DAY$, MARK
PRINT FIO$, DAY$, MARK    'вывод на экран
CLOSE #1
```

Эта программа будет создавать файл journal.dat, записывать введенные пользователем данные, а затем считывать из файла journal.dat данные и выводить их на экран. Но в данной версии программы из файла мы будем получать всегда первую строчку (порцию) данных. Исправим это. Будем использовать функцию **EOF**, проверяющую достигнут ли конец файла.

```
REM Работа с файлами. Пример 2_2.
REM Запись в файл
OPEN "journal.dat" FOR APPEND AS #1
INPUT "Введите ФИО", FIO$
INPUT "Введите дату", DAY$
INPUT "Введите отметку", MARK
WRITE #1, FIO$, DAY$, MARK
CLOSE #1
REM Чтение из файла
OPEN "journal.dat" FOR INPUT AS #1
DO WHILE NOT EOF(1)
INPUT #1, FIO$, DAY$, MARK
PRINT FIO$, DAY$, MARK    'вывод на экран
LOOP
CLOSE #1
```

Теперь программа выводит из файла все данные.

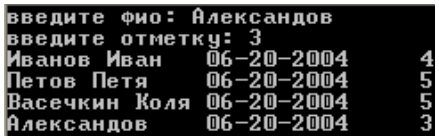
Продолжим работу. Упростим задачу пользователя - дату будем получать с помощью функции **DATE$**, которая возвращает текущую дату в формате mm-dd-

yyyy.

```
REM Работа с файлами. Пример 2_3.
REM Запись в файл
OPEN "journal.dat" FOR APPEND AS #1
INPUT "Введите ФИО", FIO$
INPUT "Введите отметку", MARK
WRITE #1, FIO$, DATE$, MARK
CLOSE #1
REM Чтение из файла
OPEN "journal.dat" FOR INPUT AS #1
DO WHILE NOT EOF(1)
INPUT #1, FIO$, DAY$, MARK
PRINT FIO$, DAY$, MARK    'вывод на экран
LOOP
CLOSE #1
```

Итак, что у нас получилось? Мы написали программу для заполнения и вывода на экран классного журнала (для простоты мы не стали разделять эти две части программы). Данные журнала хранятся в файле на диске.

Результат работы программы:



Примечание: Кроме операторов для создания, считывания и записи файлов, Basic имеет средства для осуществления определенных DOS-подобных сервисных программ внутри программы. Оператор **NAME** переименовывает файлы, **KILL** - стирает файлы, **MKDIR** - создает каталоги, **CHDIR** - меняет текущий каталог, **RMDIR** - уничтожает каталоги.

Примечание: рассмотрим еще два примера (назначение ясно из коментариев).

```
'Пример открыть файл, назначенный принтеру
OPEN "LPT1:" AS #1
'послать строку на принтер
PRINT# 1,"THIS IS A TEST"
CLOSE# 1 'закрыть переменную файла

'открыть два разных файла
OPEN "CLOSEFIL.ONE" FOR AS #1
OPEN "CLOSEFIL.TWO" FOR AS #2
'вписать строку в каждый файл
PRINT# 1,"THIS IS A TEST"
PRINT# 2,"THIS IS A TEST"
'закрыть все файлы
CLOSE
END
```

Язык программирования Basic. Учимся программировать! Конспекты школьника

Комбинированные типы.

Под **переменной** в языках программирования понимают программный **объект** (число, слово, часть слова, несколько слов, символы), имеющий имя и значение, которое может быть получено и изменено программой. При объявлении переменных можно указать тип данных. Это делается с помощью оператора **DIM**. В бейсике имеется несколько встроенных типов: числовые (integer, long, single, double) и стоковые (string, string *).

При работе с большим числом данных одного типа очень удобно использовать массивы. Массив, это разновидность переменной. Он дает возможность хранить

сколько угодно значений одного типа под одним и тем же именем. К каждому конкретному значению массива, необходимо обращаться через числовой индекс.

При написании программы, возникает необходимость описать **характеристики** (свойства) некоторого **объекта**, представляемого и обрабатываемого в программе. Таким объектом может быть человек, некоторый вычислительный комплекс, письмо, посылаемое по почте и т. д. Во всех подобных случаях свойства объекта представляются значениями различных типов и поэтому для их описания не могут быть использованы массивы.

Для описания объекта «ученик» могут понадобиться, например, следующие характеристики:

- фамилия, имя и отчество (строки);
- возраст (integer);
- пол (строка);
- класс (integer);
- буква класса (символ);
  и т.д.

Для представления такой разнородной, но логически связанной информации удобно использовать **комбинированный тип**. Необходимо отметить, что в данном случае определенные компоненты комбинированного типа, ввиду их различной природы, не могут идентифицироваться порядковыми номерами (индексами), как в массивах, поэтому для обозначения компонентов используются идентификаторы (имена). Таким образом, описание комбинированного типа представляет собой список описаний его элементов; каждое описание похоже на описание простой переменной. Для примера, приведенного выше, описание комбинированного типа PUPIL (ученик) может выглядеть следующим образом:

```
TYPE Pupil
   fio AS STRING * 20
   age AS INTEGER
   sex AS STRING * 6
   class AS INTEGER
   classname AS STRING * 1
END TYPE
```

Определив собственный тип данных, вы можете использовать его для объявления переменных этого типа.

```
DIM  Schoolchildrens AS Pupil

DIM Group(1 TO 25) AS Pupil
```

Доступ к компонентам (свойствам) переменной пользовательского типа осуществляется путем указания точки после имени переменной.

```
Schoolchildrens.fio = "Иванов Иван"
Schoolchildrens.age = 15
Schoolchildrens.sex = "male"
Schoolchildrens.class = 10
Schoolchildrens.classname = "A"

PRINT Schoolchildrens.fio, Schoolchildrens.age, Schoolchildrens.sex, Schoolchildrens.class,
Schoolchildrens.classname
```

Пример простой программы:

```
REM использование комбинированных типов

REM описание типа ученик
TYPE Pupil
   fio AS STRING * 20
   age AS INTEGER
   sex AS STRING * 6
   class AS INTEGER
   classname AS STRING * 1
END TYPE
```

```
REM объявление массива из 3 элементов типа ученик
DIM Group(1 TO 3) AS Pupil

Group(1).fio = "Иванов Иван"
Group(1).age = 15
Group(1).sex = "male"
Group(1).class = 10
Group(1).classname = "А"

Group(2).fio = "Петрова Маша"
Group(2).age = 14
Group(2).sex = "female"
Group(2).class = 10
Group(2).classname = "Б"

Group(3).fio = "Сидоров Вася"
Group(3).age = 16
Group(3).sex = "male"
Group(3).class = 11
Group(3).classname = "В"

REM выводим на экран учеников 10 класса
FOR i=1 TO 3
IF Group(i).class = 10 THEN PRINT Group(i).fio
NEXT i

END
```