Оглавление.

Типы данных.

Целый	**INTEGER**	-32768 ÷ 32767
Длинный целый	**LONG**	-2147483648 ÷ 2147483647
Вещественный	**SINGLE**	9..12 зн.ц. $10^{-39} \div 10^{40}$
Вещественный двойной точности	**DOUBLE**	19..20 зн.ц. $10^{-4000} \div 10^{4000}$
Строковый	**STRING**	Строка (max длина 255 символов).

По умолчанию все переменные имеют тип **SINGLE**. Переопределить тип переменной можно несколькими способами:

1. оператором **DEF…**

DEFINT	Целый.
DEFLNG	Длинный целый.
DEFSNG	Вещественный.
DEFDBL	Вещественный, двойной точности.
DEFSTR	Строковый.

Например:

```
DEFINT A
DEFSTR B, C
```

2. спецсимволами

%	Целый.
&	Длинный целый.
!	Вещественный.
#	Вещественный, двойной точности.
$	Строковый.

Например:

A$ - переменная символьного типа

Операторы ввода/вывода.

Оператор ввода INPUT.

Останавливает выполнение программы для запроса значений с клавиатуры. Пример:

```
INPUT A
INPUT " Name : ", Name$
```

Оператор вывода PRINT.

Производит вывод числовых данных, символов и строк на экран. Пример:

```
PRINT A
PRINT " Name : ", Name$
```

Оператор присвоения.

Например:

```
A=A+B
```

сначала вычисляется правая часть выражения, и полученный результат присваивается левой части.

Условный оператор IF.

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие.

IF <**условие**> **THEN** <**оператор1**> **ELSE** <**оператор2**> **END IF**

<**условие**> - произвольное выражение логического типа;
<**оператор1**>, <**оператор2**> - любые исполняемые операторы.

Пример написания условного оператора:

```
IF a>5 THEN
      x:=y*a
ELSE
      x:=x+a
END IF (полная форма записи);

IF a>5 THEN
      x:=y*a
END IF (краткая форма записи).
```

Логические операции

AND	- Логическое «**И**».
OR	- Логическое «**ИЛИ**».
NOT	- Логическое отрицание «**НЕ**».

Пример использования логических операций:

```
INPUT A
IF (A>10) AND (A<=20) THEN
      PRINT "10"
ELSE
      PRINT A
END IF
```

Оператор выбора CASE.

Позволяет выбрать действие из списка в зависимости от значения параметра.

```
SELECT CASE <ключ_выбора>
      CASE <список_выбора>
            <оператор>
END SELECT
```

<**ключ_выбора**>	- переменная;
<**список_выбора**>	- список значений (условие, диапазон значений);
<**оператор**>	- исполнительный оператор.

Пример программы:

```
INPUT a%
SELECT CASE a%
        CASE IS >= 5
                PRINT "Введенное значение больше или равно 5"
        CASE 2 TO 4
                PRINT "Значение в диапазоне от 2 до 4"
        CASE 1

                PRINT "Значение равно 1"
END SELECT
```

Метки и оператор перехода GOTO.

Метки предназначены для принудительного изменения процесса выполнения программы. В программе после метки ставится двоеточие.
Пример программы:

```
1 :
 INPUT "Введите положительное число: ", a
 IF a <= 0 THEN
  GOTO 1
 END IF
 PRINT "Число :", a
```

Операторы циклов.

Безусловный оператор цикла FOR.

Применяется когда количество повторений известно.

FOR <**п.ц.**>=<**н.з.**> **TO** <**к.з.**> **STEP** <**шаг**>
 <**оператор**>
NEXT <**п.ц.**>

п.ц. - параметр цикла;
н.з. - начальное значение;
к.з. - конечное значение;
шаг - шаг изменения параметра цикла;
оператор - выполняемый оператор.

Пример программы:

```
FOR i = 1 TO 10 STEP .5
 PRINT i
NEXT i
```

Оператор цикла с предусловием WHILE.

WHILE <**условие**>
 <**оператор**>
WEND

условие - выражение логического типа;
оператор - выполняемый оператор.

Если <**условие**> имеет значение **TRUE**, то выполняется <**оператор**>, после чего повторяется проверка условия. Если <**условие**> имеет значение **FALSE**, оператор **WHILE** прекращает свою работу.
Пример программы:

```
PRINT "складываю до 100."
sum = 0
WHILE sum < 100
 INPUT a
 sum = sum + a
WEND
PRINT "Сумма превысила 100.", sum
```

Условный оператор цикла DO … LOOP.

Условный оператор цикла, который может использоваться как с предусловием, так и с постусловием. Может иметь две записи:

1. с предусловием:
   DO [**WHILE**, **UNTIL**] <**условие**>
    <**оператор**>
   LOOP

2. с постусловием:
   DO
    <**оператор**>
   LOOP [**WHILE**, **UNTIL**] <**условие**>

WHILE – если условие выполняется, то выполнить <**оператор**>, если нет – завершить работу.
UNTIL – если условие не выполняется, то выполнить <**оператор**>, иначе завершить работу.

Примеры программы:

1

```
PRINT "Складываю до 100"
sum = 0
DO
 INPUT "Введите число: ", a
 sum = sum + a
LOOP WHILE sum <= 100
```

2

```
PRINT "Складываю до 100"
sum = 0
DO
 INPUT "Введите число: ", a
 sum = sum + a
LOOP UNTIL sum >= 100
```

Форматный вывод данных.

Оператор LOCATE.

Установить курсор на экране по заданным параметрам (координатам).

```
LOCATE Y, X
```

Оператор PRINT USING

Может использоваться для форматного вывода числовых данных. Имеет форму записи:

```
PRINT USING "###.##"; F
```

###.## - количество позиций выделенных под число (целая часть - **###**, дробная - **##**).

Массивы

Описывается оператором **DIM**. Описание типа массива может происходить с помощью спецсимволов, а также с помощью оператора **AS**. Пример:

```
DIM SL$(5)
DIM SL(5,4) AS INTEGER
```

Обращение к элементу массива осуществляется по его порядковому номеру, например:

```
A(4), B(3,2)
```

Пример программы заполнения одномерного массива:

```
DIM A(5)
FOR I=1 TO 5
 INPUT A(I)
NEXT I
```

Работа с данными.

Операторы DATA, READ, RESTORE.

В операторе **DATA** задается список значений, которые считываются при помощи оператора **READ**. Для того, что бы повторить чтение этих значений сначала, используется оператор **RESTORE**. Формы записи:

DATA <**список_значений**>
READ <**список_переменных**>
RESTORE

Пример программы:

```
FOR i% = 1 TO 3
 READ a%, b$
 PRINT a%, b$
 RESTORE
NEXT i%
DATA 1, "Repeat"
```

Работа с графикой.

Оператор SCREEN.

Задать графический режим работы экрана. Форма записи:

SCREEN <**режим_работы**>

Режим работы:
7 – 320 X 200 X 16
8 – 640 X 200 X 16
9 – 720 X 348 X 16

Оператор CLS.

Очистка экрана.

Оператор LINE.

Нарисовать линию (прямоугольник) по заданным координатам. Форма записи:

```
LINE (X1, Y1)-(X2, Y2)[,color][,B[F]]
```

X1, Y1	- координаты левого верхнего угла;
X2, Y2	- координаты правого нижнего угла;
Color	- цвет линии;
B	- нарисовать прямоугольник;
BF	- нарисовать закрашенный прямоугольник.

Оператор CIRCLE.

Нарисовать окружность (дугу). Форма записи:

```
CIRCLE (X, Y), R [,C] [[,S][,E][,A]]
```

X, Y	- координаты центра окружности;
R	- радиус окружности;
C	- цвет линии;
S	- начало дуги (радианы);
E	- конец дуги (радианы);
A	- сжатие окружности (дуги).

Оператор PAINT.

Закрасить замкнутую область. Форма записи:

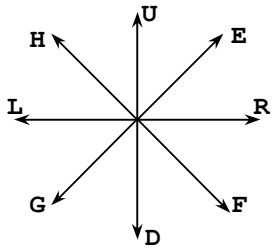
```
PAINT (X, Y), C1, C2
```

X, Y	- координаты внутри замкнутой области;
C1	- каким цветом закрасить;
C2	- до какого цвета закрасить.

Оператор DRAW.

Нарисовать ломаную линию. Форма записи:

```
DRAW "<команды>"
```



B	- не рисовать;
mXY	- переместить на координаты **X**, **Y**;
N	- нарисовать и вернуться в исходную точку.

Оператор COLOR.

Установить цвет выводимых линий, символов. Форма записи:

```
COLOR <номер_цвета>
```

Оператор PSET.

Нарисовать точку в заданных координатах. Форма записи:

```
PSET (X, Y) [,C]
```

X, Y	- координаты точки;
C	- цвет точки.

Пример программы с использованием графики.

```
SCREEN 7
CLS
LINE (100, 100)-(300, 180), , B
LINE (100, 100)-(200, 50)
LINE (300, 100)-(200, 50)
CIRCLE (50, 50), 20
PAINT (50, 50), 14, 15
CIRCLE (200, 75), 20, , -6.28, -3.14, .5
DRAW "bm0,120 e20f20e20f20e20d80l20nu60l20nu80l20nu60l20nu80l20u60"
PSET (10, 10), 14
```

Строки.

Процедуры и функции для работы со строками.

CHR$(<код>)	- возвращает символ по его коду;
ASC(<символ>)	- возвращает код символа;
STR$(<числовая_переменная>)	- преобразовать число в строку;
VAL(<строка>)	- преобразовать строку в число;
LEN(<строковая_переменная>)	- возвращает количество символов в строке;
INKEY$	- остановить программу до нажатия любой клавиши;
INSTR([pos,] <c_п>, <символ>)	- найти позицию первого вхождения символа в строке: **pos** – с какой позиции начать поиск; **c_п** – символьная переменная; **символ** – символ поиска;
LEFT$(<c_п>, <kol>)	- копировать слева **kol** символов из **c_п**;
RIGHT(<c_п>, <kol>)	- копировать справа **kol** символов из **c_п**;
MID$(<c_п>, <pos>, <kol>)	- копировать **kol** символов начиная с позиции **pos** из **c_п**.

Пример программы работы со строками.

```
INPUT "Введите строку с цифрой 1: ", a$
PRINT "Длина строка: ", LEN(a$)
PRINT MID$(a$, 1, INSTR(a$, "1"))
INPUT " Введите число: ", b$
a = VAL(b$)
PRINT USING "#####.#####"; a
INPUT "Введите символ: ", b$
PRINT "Код символа: ", ASC(b$)
INPUT "Введите число (0 - 255): ", a%
PRINT "Символ с этим кодом: ", CHR$(a%)
```

Работа с файлами.

Оператор OPEN.

Открыть файл для записи или для чтения. Форма записи:

OPEN <c_n> FOR [INPUT|OUTPUT] AS #n

c_n	- символьная переменная или имя файла;
INPUT	- открыть для чтения;
OUTPUT	- открыть для записи;
#n	- номер файла.

Команды для работы с каталогами и файлами.

EOF(<n>)	- функция тестирующая конец файла. **n** – номер файла;
CLOSE [#n]	- закрыть файл;
KILL <c_n>	- стереть файл;
MKDIR "путь"	- создать каталог;
CHDIR "путь"	- перейти в каталог;
RMDIR "путь"	- удалить каталог;
FILES	- вывести список файлов.

Пример программы работы с файлами.

```
CLS

INPUT "Input file name: ", a$
OPEN a$ FOR OUTPUT AS #1
FOR I% = 1 TO 5
 INPUT b$
 PRINT #1, b$
NEXT I%
CLOSE #1

OPEN a$ FOR INPUT AS #1
WHILE NOT EOF(1)
 INPUT #1, b$
 PRINT b$
WEND
CLOSE #1
```

Подпрограммы.

Оператор GOSUB.

Используется для перехода в подпрограмму, которая записана в основной программе. Форма записи:

```
GOSUB <метка>
...
<метка>
 ...
 RETURN
```

Пример программы:

```
CLS
FOR i% = 1 TO 5
 GOSUB first
NEXT i%
END

first:
 LOCATE i%, 10
 PRINT i%, "строка"
 RETURN
```

Внешние процедуры и функции.

Их исполнительные операторы описываются в отдельном окне редактора, которое открывается после набора служебного слова обозначающего подпрограмму. Переход между основной программой и подпрограммой осуществляется клавишей **F2**. По время сохранения основной программы, в ее начало, добавляются служебные слова описания подпрограмм, которые начинаются со служебного слова **DECLARE**. Вызов подпрограммы осуществляется с помощью ее имени или служебного слова **CALL**.

Процедура SUB.

Форма записи:

```
SUB name [([paramlist])]
 <операторы>
END SUB
```

name	- имя процедуры;
[paramlist]	- список используемых переменных и их тип;
<операторы>	- исполнительные операторы.

Функция FUNCTION.

В отличии от процедуры, которая выполняет какое-то действие, функция, в своем имени, возвращает результат, который получен после или во время исполнения подпрограммы. Форма записи:

```
FUNCTION name [([paramlist])]
 <операторы>
 name = expression
 <операторы>
END FUNCTION
```

expression	- выражение или переменная.

Пример программы с подпрограммами.

```
DECLARE SUB proba ()
DECLARE FUNCTION text$ ()

CLS
 PRINT "procedure or function."
 CALL proba
 b$ = text$
 PRINT b$
END


SUB proba
 CLS
 FOR i% = 1 TO 10
 LOCATE i%, i%
 PRINT i%
 NEXT i%
END SUB

FUNCTION text$
 INPUT a$
 text$ = a$
END FUNCTION
```

Приложение 1.

Математические функции.

ABS(A)	- функция, модуль числа **A**.
SQR(A)	- функция, корень числа **A**.
SIN(A)	- функция, **sin A** (**A** – угол в радианах).
COS(A)	- функция, **cos A** (**A** – угол в радианах).
TAN(A)	- функция, **tg A** (**A** – угол в радианах).
ATN(<число>)	- функция, **arctg числа**.
*	умножение
^	возведение в степень
/	вещественное деление
\	целочисленное деление
INT(A)	получить целую часть от числа
MOD	получить остаток от деления (целый тип)

Пример программы:

```
CONST PI = 3.141592654#
PRINT ATN(TAN(PI / 4!)), PI / 4!  'Output is:  .7853981635  .7853981635
PRINT (COS(180 * (PI / 180)))     'Output is:  -1
PRINT (SIN(90 * (PI / 180)))      'Output is:  1
PRINT (TAN(45 * (PI / 180)))      'Output is:  1.000000000205103
```