

Глава 1

Язык описания постпроцессоров



Язык описания постпроцессоров

Язык GML (GeMMa Macro Language) предназначен для создания программных процедур (макропрограмм пользователя), использующих геометрические и графические функции интерфейса прикладного программиста системы ГеММа-3D.

Файлы с макропрограммами находятся в каталогах /GML2 и /GML3 и имеют расширение GML. Редактировать имеющиеся файлы и создавать новые можно с помощью любого текстового редактора, который не добавляет в текст форматирующие символы. Например, для этой цели подойдет NOTEPAD (Блокнот) из системы Windows.

В состав системы ГеММа-3D входит специализированный редактор GMLedit, имеющий систему контекстной подсказки и другие функции.

Программы, написанные на языке GML, состоят из последовательности строк, содержащих команды. В одной строке может быть не более одной команды. Пустые строки и строки с комментариями командами не являются и при выполнении программы игнорируются. Команда состоит из следующих лексических единиц:

1. идентификаторы;
2. операторы;
3. литералы;
4. метки;
5. разделители.

Программа может иметь, кроме основной программы, ряд вспомогательных подпрограмм и функций. Таким образом, вся макропрограмма структурирована в соответствии с логикой работы. Основная программа располагается в начале файла. Дополнительные подпрограммы и функции располагаются после основной программы. Выполнение макропрограммы начинается с основной программы.

Комментарии

В программе могут содержаться поясняющие комментарии. Строка, содержащая комментарий, начинается оператором REM. Далее следует произвольный набор символов. Комментарий не является командой, т.е. при выполнении программы строка с комментарием игнорируется. Пример:

```
REM ======= Это комментарий =========
```

Другим способом ввода комментария служит символ одинарной кавычки(’). Все символы от кавычки до конца строки будут считаться комментарием и игнорироваться при выполнении программы.

Пример:

```
PRINT "A= ", A     'Печать значения переменной A
```

Идентификаторы

Идентификаторы используются в языке GML для идентификации переменных, констант, функций и процедур. Идентификатором является ряд букв и цифр, начинающийся с буквы. Знак подчеркивания является буквой. Также могут использоваться в качестве суффикса (последнего символа идентификатора) символы # и @. В качестве букв допускается использование только символов латинского алфавита. Символы верхнего и нижнего регистра являются различными для переменных и констант. Имена встроенных функций и процедур регистронезависимы. Примеры правильных идентификаторов:

```
Angle N33   _125  A#    B@
```

Неправильные идентификаторы:

```
Угол  33N   @Length     #N
```

Операторы

Операторами являются лексические единицы, имеющие вид идентификаторов. Эти слова зарезервированы и не могут использоваться в качестве идентификаторов. К операторам относятся:

APPEND	AS	CALL	CLOSE	CONST
DIM	DO	ELSE	ELSEIF	END
ENDFUNCTION	ENDSUB	EXIT	FUNCTION	
FOR	GOTO		IF	INPUT
LOOP	NEXT	ON	OPEN	OUTPUT
PRINT	RANDOMIZE	READ	REM	
RND	SUB	THEN	TO	UNTIL
WEND	STEP	SUB	WHILE	WRITE

Операторы могут состоять как из строчных, так и из прописных букв:

```
For i#=0 TO 10 sTeP 'Операторы регистронезависимы.
    a = b [i#]
NexT
```

Литералы

Литералы подразделяются на числовые целые, с плавающей точкой и строки (строчные литералы). Числовые целые литералы имеют вид десятичных чисел. Они могут быть как положительные, так и отрицательные. Примеры числовых целых литералов:

```
128         -255        1234567890
```

Литералы с плавающей точкой представляют данные с двойной точностью. Они состоят из целой части, точки, дробной части, а также из показателя степени десяти, имеющего вид целого числа со знаком, перед которым находится прописная буква E. Допускается опускать целую или дробную часть, а также знак показателя, если он положительный. Кроме этого, можно не указывать точку или показатель вместе с буквой E. Примеры литералов с плавающей точкой:

```
3.14  -10.5 3.E-6 0.0   0.    .0    0E0
```

Строка – это последовательность символов, заключенная в кавычки. Строка может содержать любые символы. Пример строкового литерала:

```
"This is a string"
```

Метки

Метка – это целое положительное число, при необходимости стоящее в начале строки команды. Метка служит для передачи управления к определенной команде. Передача управления может быть как безусловной (оператор **GOTO**), так и условной (операторы **IF, ELSEIF, ELSE, ON** и т.д.). Любая строка программы может иметь свою метку. Каждая метка должна быть единственной в программе. Метки в программе не обязательно появляются в возрастающем порядке.

Пример:

```
100 CALL BEEP ()
I# = I# + 1
IF I# = 10 THEN 17
GOTO 100
17 END
```

Разделители

Разделители служат для разделения отдельных лексических единиц. Разделителями являются пробел, табуляция и следующие символы:

```
      ,     ;     (     )     +     -     *     /     =     |
&     ^     <     >     %
```

Переменные

Объекты, над которыми выполняются действия в макропрограмме, представляются посредством имен. Простейшими именами объектов являются литералы и идентификаторы. Литералы представляют в программах постоянные, а идентификаторы – переменные величины. В языке GML поддерживаются четыре типа объектов: целые числа (тип INT), числа с плавающей точкой (тип REAL), идентификаторы геометрических объектов (тип ENT) и строковые переменные (тип STR). Чтобы отличать их в программе один от другого, используется суффикс (последний символ в идентификаторе). Таким образом, тип имени переменной определяет тип переменной.

Целые числа имеют суффикс #. Примеры:

```
I#    COUNT#      NUMBER#
```

Идентификаторы геометрических объектов имеют суффикс @. Примеры:

```
LINE@       SURFACE@
```

Строковые переменные имеют суффикс $. Примеры:

```
FILENAME$   STRING$
```

Идентификаторы чисел с плавающей точкой не имеют суффикса. Примеры:

```
X     Y     LENGTH
```

Когда переменная встречается в программе первый раз, ее значение равно 0.

Предопределенные константы

Ряд идентификаторов зарезервирован под предопределенные константы. Значение константы определяется конкретной величиной и остается неизменным на все время выполнения программы. Константы можно использовать в любых выражениях. Попытка изменить значение константы без оператора CONST приведет к сообщению об ошибке и завершению выполнения программы, однако предопределенные константы постпроцессора могут модифицироваться, при этом необходимо использовать модификатор CONST, если модификатор не был использован, то будет выдано предупреждение и выполнение программы продолжится.

Пример:

```
If a@ = NULL@ Then 100  ' OK.
x = R*SIN(ang*PI/360.)  ' OK.
PI = 6.26               ' Ошибка - присвоение константе
```

В таблице приводится список констант, их значение и описание:

Для универсального постпроцессора определен целый ряд дополнительных констант, таких как TRUE#, FALSE#, MOVE# и т.д. Их значение имеет

Имя константы	Значение	Описание
PI	3.1415...	Число π.
NULL@	0	Нулевой указатель на объект.
NULL$	0	Нулевой указатель на строку.
NL$	LF	Символ перевода строки.

смысл только в контексте использования конкретных внешних переменных – параметров постпроцессора и исходной программы. Параметры, принимающие ряд определенных значений, описаны как имеющие тип перечислимый или логический (см. главу «**Параметры постпроцессора и исходной программы**» в руководстве пользователя «**Универсальный постпроцессор управляющих программ системы ГеММа-3D**»). Возможные значения соответствующего перечислимого типа (набор предопределенных констант) указаны для каждого конкретного параметра. Тип логический принимает значения TRUE# и FALSE#.

В макроязыке есть возможность ввести новые константы с помощью оператора **CONST** (см. Оператор **CONST**).

Массивы

Язык GML позволяет использовать массивы объектов. Массив – это совокупность объектов (компонентов) одинакового типа. Компоненты массива называются его элементами. Перед первым использованием элементов массива необходимо его описать. Для этого используется оператор **DIM**. При описании массива необходимо указать в квадратных скобках число элементов. Число элементов массива является динамически расширяемым. Т.е. если происходит обращение к элементу массива с номером, большим, чем указано при описании, размер массива автоматически увеличивается. Таким образом, при описании массива не обязательно знать его максимальную длину. Один оператор **DIM** может описать сразу несколько массивов. Нумерация элементов массива начинается с 0.

Пример:

```
DIM A[1], B[10]
B[0] = 10
A[5] = B[0]
```

Все элементы массива после описания инициализированы нулем.

Выражения

Выражения используются для операций над переменными. В языке GML используются арифметические, логические и битовые операции. Вычисление значений выражений осуществляется согласно приоритетам содержащихся в них операций. Если в одном выражении находятся операции с одинаковыми приоритетами, то выполнение осуществляется слева направо. В качестве операндов в выражении могут использоваться идентификаторы, элементы массивов и литералы.

Примеры:

```
A = B + 1. / C [I#]
C [I#] = D * E + 10.
```

В таблице приводится список всех операций, а также типы данных операндов, над которыми возможны операции (INT – целые, REAL – с плавающей точкой, ENT – идентификаторы объектов, STR – строки). Операции приведены в порядке уменьшения приоритета. В каждой строке-ячейке таблицы объединены операции с одинаковым приоритетом.

Операция	Описание	Возможные типы данных операндов
- + ~	Унарный минус Унарный плюс Побитовое отрицание	INT, REAL INT, REAL INT
^	Возведение в степень	INT, REAL
* / % | &	Произведение Частное Остаток от деления Побитовое ИЛИ Побитовое И	INT, REAL INT, REAL INT INT INT
+ -	Сложение Вычитание	INT, REAL, STR INT, REAL
  < <= > >= = <>	Логические операции сравнения: Меньше Меньше или равно Больше Больше или равно Равно Не равно	 INT, REAL INT, REAL INT, REAL INT, REAL INT, REAL, ENT, STR INT, REAL, ENT, STR
AND OR	Логическое И Логическое ИЛИ	INT, REAL, ENT, STR INT, REAL, ENT, STR

Если необходимо поменять порядок вычислений, то нужно использовать скобки.

Примеры

```
A = B + C * D
A = (B + C) * D
```

Если в арифметической операции один операнд имеет тип INT, а другой – REAL, то значение выражения будет иметь тип REAL.

В одном выражении могут одновременно присутствовать арифметические, логические и битовые операции. При этом значение логических операций равно 1, если условие верно, и 0 в противном случае. Например, в выражении

```
I# = 10 * (A < B)
```

Переменная I# примет значение 10, если A < B, и 0 в противном случае.

Примечание: операторы **AND** и **OR** регистрозависимые.

Функции

В выражениях могут использоваться функции.

Пример:

```
A = SIN (B) - COS (C)
```

В таблице приводится список имеющихся в языке GML функций, типы данных аргумента и возвращаемого значения:

Функция	Описание	Тип аргумента	Тип значения
ABS	Абсолютное значение	INT, REAL	INT, REAL
ACOS	Арккосинус	INT, REAL	REAL
ASIN	Арксинус	INT, REAL	REAL
ATAN	Арктангенс	INT, REAL	REAL
ATAN2(Y,X)	Арктангенс Y/X	INT, REAL	REAL
CEIL	Округление до ближайшего большего целого	INT, REAL	REAL
COS	Косинус	INT, REAL	REAL
COSH	Косинус гиперболический	INT, REAL	REAL
ETYPE	Тип объекта	ENT	INT
EXP	Экспонента	INT, REAL	REAL
FLOOR	Округление до ближайшего меньшего целого	INT, REAL	REAL
GRADRAD	Градусы в радианы	INT, REAL	REAL
HYPOT(X,Y)	Квадратный корень из суммы квадратов аргументов	INT, REAL	REAL
INT	Приведение к типу INT	INT, REAL	INT
LG	Десятичный логарифм	INT, REAL	REAL
LN	Натуральный логарифм	INT, REAL	REAL
MAX(X,Y)	Максимальное значение из двух аргументов	INT, REAL	INT, REAL
MIN(X,Y)	Минимальное значение из двух аргументов	INT, REAL	INT, REAL
RADGRAD	Радианы в градусы	INT, REAL	REAL
REAL	Приведение к типу REAL	INT, REAL	REAL
ROUND	Округление до ближайшего целого	INT, REAL	REAL
SGN	Знак	INT, REAL	INT
SIN	Синус	INT, REAL	REAL
SINH	Синус гиперболический	INT, REAL	REAL
SQRT	Квадратный корень	INT, REAL	REAL
TAN	Тангенс	INT, REAL	REAL
TANH	Тангенс гиперболический	INT, REAL	REAL

Работа со строками

Строковые переменные могут содержать любую символьную информацию. Для инициализации строковых переменных могут использоваться строковые литералы:

```
STR$ = "file.txt"
```

К строковым переменным можно применять операции сравнения и слияния (конкатенации):

```
IF STR$ <> "POINT" THEN 10
S1$ = S2$ + S3$
```

В выражениях со строковыми переменными могут использоваться функции. В качестве примера работы со строками приводится фрагмент программы разбора строки из файла управляющей программы (УП) для станка с ЧПУ. УП содержит строки вида

```
. . .
N13X10.1Y32.4Z-7.89
N13Y32.9Z-7.99
N13X10.0Y32.4Z-6.89
N13X9.8
. . .
```

Программа для извлечения значений X,Y,Z из строк УП выглядит следующим образом:

```
Open "file.msh" For Input As id# ' Открываем файл с УП.
Do
  Input id#, s$ ' Считываем очередную строку.
  If LEFT$ (s$,3) = "N13" Then ' Строки с "N13" в начале.
    len# = LEN (s$) ' Определяем длину строки.
    ' Последовательно проверяем каждый символ строки,
    ' находим 'X', 'Y' и 'Z' и переводим следующее за ними
    ' значение из символьного в числовое.
      For i#=3 To len#-1
      c$ = MID$ (s$,i#,1)
      If c$ = "X" Then X = VAL (MID$ (s$,i#+1,10))
      ElseIf c$ = "Y" Then Y = VAL (MID$ (s$,i#+1,10))
      ElseIf c$ = "Z" Then Z = VAL (MID$ (s$,i#+1,10))
    Next
  EndIf
  ' *** Здесь другие команды ***
Loop Until EOF (id#) ' Если дошли до конца файла то закончим.
Close id#
```

Некоторые системы управления станками с ЧПУ имеют особенности в представлении координат. Например, в системе управления FANUC запись X100. соответствует 100 милиметрам, а запись X100 - 100 импульсам (*.001 мм) - .1мм.

Для правильного формирования записи координат для таких станков применяются специальные функции вывода числовых значений - STR_FP$ и STR_NZ$. Эти функции преобразуют числовое значение в строку, которая

затем выводится в файл по соответствующему формату.

Функция	Описание	Тип значения
ASC(S$)	ASCII-код первого символа строки	INT
CHR$(N#)	Символ, соответствующий ASCII-коду	STR
HEX$(N#)	Шестнадцатеричное строковое представление числа	STR
OCT$(N#)	Восьмеричное строковое представление числа	STR
LCASE$(S$)	Переводит строку в нижний регистр	STR
UCASE$(S$)	Переводит строку в верхний регистр	STR
LEFT$(S$,N#)	Часть строки S$ из первых N# символов	STR
RIGHT$(S$,N#)	Часть строки S$ из последних N# символов	STR
MID$(S$,I#,N#)	Часть строки S$, начинающаяся с позиции I#, длиной N# символов	STR
LEN(S$)	Длина строки	INT
SPACE$(N#)	Строка из N# пробелов	STR
STR$(A)	Строковое представление числа A	STR
STR_FP$(A,N#)	Строковое представление числа A, в котором гарантированно присутствует плавающая точка и нет нулей в конце. N# – макс. число знаков после запятой	STR
STR_NZ$(A,N#)	Строковое представление числа A, в котором гарантированно присутствует плавающая точка и нет нуля в начале (если |A| < 1) и нулей в конце. N# – макс. число знаков после запятой	STR
STRING$(N#,{C$|C#})	Строка из N# символов C$ или C#	STR
VAL(S$)	Числовое представление строки	REAL

Пример:

```
'Вывод координаты X с двумя знаками после запятой и

'обязательной десятичной точкой

write id# using "X%s", STR_FP$(x*RDISCR,2)
```

В таблице приводится полный список функций для работы со строками:

Описание операторов

Оператор INPUT

Оператор **INPUT** предназначен для ввода чисел. Можно ввести до шести чисел. Формат:

```
INPUT “Строка приглашения”, A1, A2,..., A6
```

Здесь:

“Строка приглашения” – появляется при вводе чисел; A1, ..., A6 – вводимые числа.

Для ввода чисел из файла с помощью оператора **INPUT** смотри ниже раздел **Работа с файлами**.

Оператор CONST

Оператор служит для определения констант – объектов, значение которых нельзя изменить. Значение константы должно быть обязательно задано. С помощью оператора **CONST** можно изменить значение константы (определенной оператором **CONST** или предопределенной).

Пример:

```
CONST PI2=PI/2. ‘Определение константы
PI2=2. ‘Ошибка – нельзя изменить константу
CONST PI2=PI*2. ‘OK - Переопределение константы
CONST PI=3.14 ‘OK - Переопределение предопределенной константы
```

Оператор CALL

Оператор служит для вызова процедур, предоставляемых геометрическим и графическим ядром системы ГеММа-3D (в файлах описания постпроцессоров данные процедуры недоступны), а также процедур пользователя (см. оператор **SUB**).

Пример:

```
CALL CREATE_POINT (X, Y, Z, E@, ERR#)
```

Описание всех процедур системы приводится в отдельном разделе. При вызове процедур необходимо соблюдать соответствие типов фактических параметров требуемым типам формальных параметров конкретной процедуры. Например, следующий вызов процедуры имеет синтаксическую ошибку:

```
CALL CREATE_POINT (X#, 10., 10., E@, ERR#)
```

– первый параметр процедуры CREATE_POINT должен иметь тип REAL. В подобных случаях необходимо явное преобразование типа:

```
CALL CREATE_POINT (REAL (X#), 10., 10., E@, ERR#)
```

При вызове подпрограмм параметры передаются по ссылке. Т.е. если формальный параметр в вызванной подпрограмме изменяется, то фактический параметр в вызывающей процедуре также изменяется. Для того чтобы защитить параметр от изменения, его нужно заключить в скобки.
Пример:

```
A=1.
```

```
Call A10 (A)
Print "A=",A 'Напечатает A=10
A=1.
Call A10 ((A)) 'Защита A от изменения.
Print "A=",A 'Напечатает A=1
End

Sub A10(B) 'B - формальный параметр.
B=10.
EndSub
```

Оператор DIM

Оператор **DIM** предназначен для описания массивов. Формат:

```
DIM V1 [N1], V2 [N2] …
```

Здесь:
V1, V2 – имена описываемых массивов;
N1, N2 – начальные размеры массивов.

Описание массива должно быть сделано до первого использования его элементов.

Оператор END

Обычно ставится в конце программы, хотя его присутствие и не является обязательным. Возможно использование для завершения программы. **Пример:**

```
IF (I# < 0) THEN END
```

Циклы: Операторы FOR, TO, STEP, NEXT, EXIT FOR

Операторы служат для организации циклов путем задания начального и конечного значений управляющей переменной, а также величины шага (приращения), которая прибавляется к текущему значению управляющей переменной при каждом повторении цикла. Формат:

```
FOR V = E1 TO E2 [STEP E3]
   ' Команды, составляющие тело цикла
NEXT
```

Здесь:
V – имя управляющей переменной;
E1 – выражение, значение которого дает начальную величину управляющей переменной;
E2 – выражение, значение которого дает конечную величину управляющей переменной;



E3 – выражение, значение которого дает величину приращения управляющей переменной.
Если величина шага не задана, то значением по умолчанию принимается 1. Если тип переменной – REAL, то шаг должен быть указан явно. Значения выражений E1, E2, E3 вычисляется один раз при начале цикла. Таким образом, если в теле цикла поменять значение переменных, присутствующих в выражениях E1, E2 или E3, то это никак не повлияет на условие завершения цикла. Проверка условия завершения цикла производится в начале каждого цикла. Например, цикл

```
FOR I# = 2 TO 1
  . . .
NEXT
```

не исполнится ни разу.

Для перехода к следующему циклу и изменению текущего значения управляющей переменной на величину шага служит оператор **NEXT**. Циклы могут быть вложенными.

Пример:

```
FOR I# = 0 TO 10
  FOR X = 0. TO 1. STEP 0.1
    . . .
  NEXT
NEXT
```

Для прерывания цикла может быть использован оператор **EXIT FOR**:

```
FOR I#=0 TO 10
  INPUT "Введите K#:", K#
  IF K# = 0 THEN EXIT FOR
    . . .
NEXT
```

Циклы: Операторы DO, LOOP, WHILE, UNTIL, EXIT DO

Операторы служат для организации циклов путем задания условия продолжения исполнения команд, составляющих тело цикла. Формат:

```
DO [{WHILE | UNTIL} E]
  ' Команды, составляющие тело цикла
LOOP
```

или

```
DO
  ' Команды, составляющие тело цикла
LOOP [{WHILE | UNTIL} E]
```

Здесь:
E – выражение, задающее условие продолжения выполнения цикла.
Оператор **WHILE** используется для организации выполнения цикла до тех пор, пока значение выражения E не равно 0. Оператор **UNTIL** используется

для организации выполнения цикла до тех пор, пока значение выражения E равно 0.

Для прерывания цикла может быть использован оператор **EXIT DO**.

Циклы: Операторы WHILE, WEND, EXIT WHILE

Операторы служат для организации циклов путем задания условия продолжения исполнения команд, составляющих тело цикла. Формат:

```
WHILE E
   ' Команды, составляющие тело цикла
WEND
```

Здесь:
E – выражение, задающее условие продолжения выполнения цикла.
Цикл выполняется до тех пор, пока значение выражения E не равно 0.

Для прерывания цикла может быть использован оператор **EXIT WHILE**.

Оператор GOTO

Оператор служит для безусловного перехода на программную строку с указанной меткой.

Пример:

```
10 …PRINT "бесконечный цикл..."
GOTO 10
```

Операторы IF, ELSEIF (ELSE IF), ELSE, ENDIF (END IF)

Операторы служат для организации условного перехода или выполнения указанного действия в зависимости от результата проверки логического условия. Форматы:

```
IF условие THEN N
IF условие THEN действие
IF условие THEN
  действие 1
  . . .
  действие N
ENDIF
```

Здесь:
N – метка программной строки, на которую должен осуществиться переход в случае истинности условия.

В качестве условия может использоваться любое выражение. Условие считается истинным, если значение выражения отлично от нуля, и ложным в противном случае. Примеры:

```
IF A < -1. OR A > 1. THEN 10
IF INT (A) THEN 10
```

Операторы **ELSEIF** и **ELSE** служат для организации более сложных условных выражений:

```
IF I# = 0 THEN
  J# = 1
ELSEIF I# = 1 THEN
  J# = 1
ELSE
  J# = J# * I#
ENDIF
```

Вместо операторов **ELSEIF** и **ENDIF** можно использовать последовательность операторов, соответственно, **ELSE IF** и **END IF**:

```
IF I# = 0 THEN
  J# = 1
ELSE IF I# = 1 THEN
  J# = J# * I#
END IF
```

Оператор ON

Оператор служит для организации перехода на одну из нескольких программных строк, указанных в списке, в зависимости от значения переключающего выражения. Формат:

```
ON E GOTO N1, N2, …, Nk
```

Здесь:
E – переключающее выражение, значение которого определяет номер метки в списке;
Nk – метка программной строки, на которую передается управление в случае, если значение переключающего выражения E равно k.

Значение выражения E должно иметь тип INT.

Оператор PRINT

Оператор служит для отладочной печати значений выражений. Формат:

```
PRINT E1,E2,…,Ek
```

Здесь:
Ek – выражения.

Подпрограммы: операторы SUB, ENDSUB (END SUB)

Оператор **SUB** позволяет определить подпрограмму, которую можно будет

вызывать из основной программы и других подпрограмм с помощью оператора **CALL**. Конец подпрограммы задается оператором **ENDSUB** или последовательностью операторов: **END SUB**. Формат:

```
SUB name(p1,p2,...)
  'Тело подпрограммы.
ENDSUB
```

Здесь:
name – имя подпрограммы;
p1,p2 – формальные параметры подпрограммы.

Имя подпрограммы должно отличаться от имен встроенных процедур и других подпрограмм. Возможное число формальных параметров – от 0 до 20. Параметры передаются по ссылке, т.е. изменение значения формального параметра при вызове процедуры приведет к изменению значения соответствующего фактического параметра.

Для передачи массива в качестве параметра, после соответствующего формального параметра необходимо поставить признак массива – парные скобки:

```
Sub Ex0(t,N#,a[]) 'a - массив
  t=0.
  For i#=0 To N#
    t=t+a[i#]
  Next
End Sub
```

Все переменные и массивы, определенные в теле подпрограммы являются локальными по отношению к подпрограмме. Значения локальных переменных не влияют на значения переменных с такими же именами в основной программе и других подпрограммах. Предопределенные константы и константы, заданные оператором **CONST**, доступны во всех подпрограммах, по существу переменная объявленная с помощью оператора **CONST** становится глобальной.

Для прерывания выполнения подпрограммы и возврата в вызывающую процедуру может быть использован оператор **EXIT SUB**.

Пример:

```
Sub Ex1(t,k#)
  if k# = 0 Then Exit Sub
  t = PI / k# 'PI - предопределенная константа.
End Sub
```

Все подпрограммы должны быть определены после основной программы.

Функции: операторы FUNCTION, ENDFUNCTION (END FUNCTION)

Оператор **FUNCTION** позволяет определить функцию, возвращающую значение, которую можно будет использовать в выражениях. Конец функции

задается оператором **ENDFUNCTION** или последовательностью операторов: **END FUNCTION**. Формат:

```
FUNCTION name(p1,p2,...)
  'Тело функции.
  name = ret_val 'Возвращаемое значение.
ENDFUNCTION
```

Здесь:
name – имя функции;
p1,p2 – формальные параметры функции;
ret_val – возвращаемое значение.

Имя функции должно отличаться от имен встроенных функций и других функций. Тип имени функции определяет тип возвращаемого значения. Возможное число формальных параметров – от 0 до 20. Параметры (как простые переменные, так и массивы) передаются в функцию по значению. Таким образом, изменения формальных параметров в теле функции не оказывают влияния на значения фактических параметров в вызывающей программе.

Для передачи массива в качестве параметра, после соответствующего формального параметра необходимо поставить признак массива – парные скобки.

Возвращаемое значение функции задается путем присваивания значения переменной с таким же именем, как у функции.

Все переменные и массивы, определенные в теле функции являются локальными по отношению к данной функции. Значения локальных переменных не влияют на значения переменных с такими же именами в основной программе и других функциях. Предопределенные константы и константы, заданные оператором **CONST**, доступны во всех функциях, по существу глобальные переменные.

Для прерывания выполнения функции и возврата в вызывающую процедуру может быть использован оператор **EXIT FUNCTION**.

Пример:

```
For i#=0 To 10
  Print StrNum$("Строка:",i#)
Next
End

Function StrNum$(s$,n#)
  StrNum$ = s$
  if n# < 0 Then Exit Function
  StrNum$ = s$ + STR$(n#)
End Function
```

Все функции должны быть определены после основной программы.

Генератор случайных чисел

Для инициализации генератора случайных чисел используется оператор **RANDOMIZE**. Формат:

```
RANDOMIZE SEED#
```

Здесь:

SEED# – целое число, используемое для инициализации генератора случайных чисел.

Для извлечения очередного случайного числа в диапазоне от 0. до 1. в арифметических выражениях используется оператор **RND.**

Пример:

```
CALL GET_TIME (H#,M#,S#)
RANDOMIZE H#+M#+S#
D1# = INT(RND*5) + 1
D2# = INT(RND*5) + 1
PRINT "Кубик 1 : ", D1#, " Кубик 2 : ", D2#
```

Работа с файлами

Для работы с файлами используются операторы **OPEN, CLOSE, INPUT, WRITE.** Файлы представляются как последовательности записей – текстовых строк переменной длины. Для начала работы с файлами служит оператор **OPEN**. Формат:

```
OPEN "ИМЯ_ФАЙЛА" FOR { INPUT | OUTPUT | APPEND } AS ID#
```

Здесь:

"ИМЯ_ФАЙЛА" – строка, содержащая имя файла (может быть строковым литералом или строковой переменной);
FOR INPUT – открываемый файл будет использоваться только для ввода;
FOR OUTPUT – открываемый файл будет использоваться только для вывода;
FOR APPEND – открываемый файл будет использоваться для записи новых данных в конец к уже существующим;
ID# – программный номер файла, который будет использоваться в дальнейшем в операциях обмена.

Для чтения данных из файла служит оператор **INPUT**. При каждом операторе происходит считывание очередной записи (текстовой строки) файла и чтение из нее данных в переменные, представленные в списке. Формат:

```
INPUT ID#, A1, A2, A3 …
```

Здесь:
A1, A2, A3 – переменные, в которые осуществляется чтение.

Если переменных в списке оператора **INPUT** больше, чем данных в строке файла, операция ввода прервется с сообщением об ошибке. Если переменных в списке меньше, чем данных в строке файла, то из строки будут считаны только данные, соответствующие списку переменных. Оставшиеся данные в строке будут отброшены без сообщения об ошибке.

Для определения конца файла используется функция EOF(ID#). В качестве параметра ей подается программный номер открытого файла. Функция возвращает 1, если достигнут конец файла, и 0 в противном случае.

Пример:

```
OPEN "POLY.DAT" FOR INPUT AS ID#
10 INPUT ID#, X, Y, Z
PRINT X, Y, Z
IF EOF(ID#) = 0 THEN 10
```

После завершения работы с файлом его следует закрыть. Для этого служит оператор **CLOSE**. Формат:

CLOSE ID1#, ID2#, ...

Здесь:
ID1#, ID2# – программные номера открытых файлов.

После завершения работы программы все открытые файлы закрываются автоматически.

Оператор WRITE: бесформатный вывод

Для записи в файл служит оператор **WRITE**. Формат:

```
WRITE ID#, E1, E2, E3, …
```

Здесь:
ID# – программный номер открытого файла;
E1, E2, E3 – аргументы: выражения, значения которых записываются в файл.

Указанные в качестве параметров оператора **WRITE** значения записываются в файл последовательно в одну строку. Между отдельными числовыми значениями вставляется пробел (добавляется к каждому выводимому числовому значению). Между строковыми значениями пробел не вставляется. Для перевода на новую строку необходимо вывести предопределенную строковую константу NL$.

Пример:

```
WRITE ID#, "Первая строка"    'Запись первой строки.
WRITE ID#, NL$                'Перевод строки.
WRITE ID#, "Вторая строка"    'Запись второй строки.
```

Оператор WRITE: форматированный вывод

Оператор **WRITE** позволяет осуществлять форматированный вывод в файл последовательности аргументов в соответствии с указанной форматной строкой. Формат:

```
WRITE ID# USING Format$, E1, E2, E3 . . .
```

Здесь:
ID# – программный номер открытого файла;
Format$ – форматная строка;
E1, E2, E3 – аргументы: выражения, значения которых записываются в файл.

Форматная строка представляет собой строку (переменную или литерал), содержащую обычные символы и шаблоны преобразования. Обычные символы передаются в файл без изменения. Шаблон преобразования определяет формат, с помощью которого в файл выводится аргумент. Если число аргументов превышает число шаблонов, то избыточные аргументы не выводятся. Не допускается, чтобы число аргументов было меньше числа шаблонов преобразования.

Шаблон в форматной строке определяется символом % и следующими за ним полями и типом аргумента. Для того, чтобы вывести в файл из форматной строки символ %, необходимо указать его дважды

```
WRITE ID# USING "%d%%", 5 ' Вывод в файл '5%'.
```

Шаблон в общем случае имеет следующий формат:

```
% [флаги] [длина] [.точность] тип
```

В квадратных скобках указаны необязательные поля.

Поле **Тип** может принимать следующие значения:

d – десятичное целое число,
f – число с плавающей точкой в неэкспоненциальной форме,
e, E – число с плавающей точкой в экспоненциальной форме,
g, G – самый короткий вариант из f и e,
c – один символ,
s – строка.

Поле **Флаги** может принимать следующие значения:

(нет) – выровнен вправо, остаток заполнен 0 или пробелами,
- – выровнен влево, остаток заполнен пробелами,
+ – всегда начинается с + или -,
пробел – выводить знак только для отрицательных чисел.

Поле **Длина**, задаваемое целым числом, определяет минимальное количество символов, которое будет выведено для данного аргумента. Остаток заполняется пробелами.

Поле **Точность** задает максимальное выводимое количество символов или число символов после десятичной точки для чисел с плавающей точкой.

Пример:

```
WRITE ID# USING "X = %f%s", X, NL$
```

В таблице приводится несколько примеров шаблонов и соответствующих результатов вывода в файл. Символ b в колонке Поле файла означает пробел:

Аргумент	Шаблон	Поле файла (результат вывода)
12.4	%+f	+12.4000
12.4	%.3f	12.400
12.4	%.3e	1.240e+001
-458.7	%.1G	-4.6E+002
5	%d	5
5	%3d	bb5
5	%-3d	5bb
5.0	%f	5.000000
5.0	%5.2f	b5.00
"GeMMa-3D"	%s	GeMMa-3D
"GeMMa-3D"	%.5s	GeMMa
"GeMMa-3D"	%6.5s	BGeMMa
"GeMMa-3D"	%c	G
55	%04d	0055