Дубовиченко С.Б.

TURBO И QUICK БЕЙСИК

Алматы
2002

Дубовиченко С.Б.

TURBO И QUICK БЕЙСИК

Алматы
2002

ББК 73я7
УДК 681.14
Д 79

Рецензенты:
Декан факультета подготовки иностранных граждан КазГУ, к.ф.-м.н., доцент ***Аккушкарова К.А..***

Дубовиченко С.Б.

Turbo и Quick Бейсик. Учебное пособие. Издание третье, дополненное и исправленное. Алматы. 2002. 160с.

Настоящее учебное пособие предназначено для студентов вузов и других начинающих пользователей персональных ЭВМ типа IBM PC AT.

Книга содержит сравнительно полное описание алгоритмических языков программирования Quick Basic и Turbo Basic (в дальнейшем просто Basic), работающих на основе операционной системы MS - DOS. Приведены все основные сведения о многих элементах и конструкциях языка Basic. Описаны директивы, операторы и функции Basic, а также некоторые сообщения об ошибках в программах, выдаваемых компиляторами этих языков. Даны некоторые примеры программ для выполнения различных математических расчетов.

Книга может использоваться, как справочное пособие по языку Basic, а приведенные программы читатели могут использовать для решения своих задач.

4310020000
Д ----------
00(05) - 01

SBN 9965-511-11-X

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая книга содержит сравнительно полное описание алгоритмических языков программирования Quick Basic и Turbo Basic (в дальнейшем просто Basic), работающих на основе стандартной операционной системы MS - DOS. Описываемые здесь версии языка Basic представляет собой наиболее мощную реализацию этого языка для персональных ЭВМ типа IBM PC. Они имеют намного более широкие возможности по сравнению со старыми версиями языка Basic A или Basic Q. Рассматриваемые версии по своим структурным возможностям очень похожи на хорошо известный алгоритмический язык Fortran версии Дубна или 77.

Основными особенностями данных версий языка Basic являются:

1. Пять типов переменных: целые (от –32767 до +32767), строковые (до 32767 символов), с плавающей точкой обычной точности (7 цифр), с плавающей точкой двойной точности (16 цифр) и длинные целые (от –65000 до +65000).
2. Возможность трассировки (TRON/TROFF), т.е. пошагового выполнения программы с выдачей на экран промежуточных результатов для облегчения процесса отладки.
3. Организация ловушек, используемая в операторе ON ERROR GO TO, используемая для определения мест ошибочных действий в программе.
4. Операторы PEEK и POKE для чтения и записи в любую ячейку оперативной памяти.
5. Булевские (логические) операции OR, AND, NOT, XOR, EQV, IMP.
6. Форматный вывод с помощью оператора PRINT USING, позволяющий выдавать на экран нужное число символов любой цифры.
7. Прямой доступ к 64K портов ввода/вывода с помощью функций INP и OUT.
8. Возможность вызовов подпрограмм на ассемблере и исполняемых .exe файлов, которые могут быть созданы самим компилятором на основе ваших программных блоков.

9. Операторы условного перехода IF/THEN/ELSE и вложенные IF/THEN/ELSE конструкции DO...LOOP, SELECT CASE.

10. Работа с файлами прямого и последовательного доступа с помощью операторов OPEN, CLOSE, GET, PUT, KILL, NAME.

11. Операторы работы с графикой PSET, PRESET, POINT, LINE, CIRCLE, GET, PUT, PAINT, DRAW, VIEW, WINDOW.

12. Операторы работы со звуковым устройством BEEP, SOUND, PLAY.

В описании синтаксиса директив, операторов и функций языка Basic будем далее использовать следующие обозначения:

1. Знак [ ] - квадратные скобки указывает, что элементы конструкции, заключенные между ними, являются необязательными и их можно не задавать.

2. Знак < > - угловые скобки, обозначают данные, вводимые пользователем. Если в угловые скобки заключен текст из строчных букв, пользователь должен вводить данные, определяемые этим текстом. Например, <имя файла> - в данном месте надо вводить именно "имя файла", а не какие-то другие данные или имена. Если же в угловые скобки включен текст из прописных букв, необходимо нажать клавишу, обозначенную этим текстом.

3. Знак { } - фигурные скобки, указывает на возможность выбора между двумя или более возможностями.

4. Знак ... - последовательность точек, обозначает повторение действий необходимое число раз.

5. Знак | - вертикальная черта, обозначает выбор одного из двух возможных вариантов, которые описаны до и после нее.

Все остальные символы, такие как запятая, двоеточие, косая черта, знак равенства, должны вводиться при наборе программ, как приведено в тексте. Малейшая неточность при вводе текста программы приведет к ее неверной работе и даже к “зависанию” компьютера.

ЭЛЕМЕНТЫ ЯЗЫКА

Строки программы на языке Basic в общем случае имеют следующий формат:

[метка] [LET] BASIC ОПЕРАТОР: [:BASIC ОПЕРАТОР] : ….

В одной строке программы может располагаться более чем один оператор, но при этом каждый оператор в строке отделяется от других операторов с помощью символа ":" - двоеточие. Строка с операторами языка может содержать не более 255 символов. Оператор присваивания LET вводить в начале строки не обязательно, т.е. для того, чтобы присвоить переменной значение выражения, достаточно знака равенства. Метка строки или ее номер, также не обязательные элементы программы и могут использоваться только для задания номера метки строки в операторах типа GOTO, THEN, GOSUB и т.д.

Константы

Существует два типа констант - строковые и числовые. Строковая константа представляет собой последовательность до 255 символов, заключенных в кавычки.

Пример:

"HELLO"
"-25,00.00"
"A+B*200"

Числовые константы представляют собой положительные и отрицательные числа. Числовые константы в Basic не могут содержать запятой, а десятичная часть числа отделяется с помощью знака “.” - точка.

Существует 5 типов числовых констант:

1. Целые константы - представляют собой целые числа в диапазоне от -32768 до +32767 и не могут содержать десятичную точку.

2. Константы с фиксированной точкой - положительные или отрицательные вещественные числа, т.е. числа с десятичной точкой.

3. Константы с плавающей точкой - числа, имеющие экспоненциальную форму. Эти константы состоят из целого или вещественного числа (мантиссы) и буквы E (или D - двойная точность), за которой следует целое число (экспонента). Если необходимо, мантисса и экспонента могут иметь знак. Константы с плавающей точкой имеют диапазон представления от 10E-38 до 10E+38.

4. Шестнадцатеричные константы - представляются шестнадцатеричным числом с префиксом &H.

5. Восьмеричные константы - представляются восьмеричным числом с префиксом & или &O.

Числовые константы могут быть обычной или двойной точности. Константы обычной точности представляются 7 цифрами, а константы двойной точности 16 цифрами.

Константой обычной точности является любая числовая константа, которая имеет один из следующих признаков:

1. 7 или меньше цифр.
2. Экспоненциальная форма с использованием E и мантиссой не более 7 знаков.
1. Заканчивается восклицательным знаком.

Константой двойной точности является любая числовая константа, у которой есть один из следующих признаков:

1. Больше 8 цифр, но не более 16.
2. Экспоненциальная форма с использованием D и мантиссой не больше 16 цифр.
3. Заканчивается символом "#".

Примеры числовых констант:

1. 645 - целая константа обычной точности.
2. -1.09E-06 - константа обычной точности с плавающей точкой.

3. 22.5! - константа обычной точности с фиксированной точкой.
4. 1.09432D-05 - константа двойной точности с плавающей точкой.
5. 345692811 - целая константа двойной точности (больше 8 цифр).
6. &H32F - шестнадцатеричная константа.
7. &0347 - восьмеричная константа.

Переменные

Переменные представляют собой символы (буквы или комбинации букв и цифр), использующиеся для обозначения величин, которые присутствуют в программе. Значение переменной может быть непосредственно присвоено программистом, либо оно может определяться в результате вычислений в программе. До того, как переменной будет присвоено значение, она принимается равной нулю.

Имена переменных в Basic могут быть любой длины, однако распознавание переменной осуществляется по первым 40 символам. Для обозначения имени переменной можно использовать буквы, цифры и десятичную точку, причем первым символом имени должна быть буква. Кроме того, допустимо использовать специальные символы объявления типа, о которых будет сказано ниже. Имя переменной не может совпадать с зарезервированным именем. К зарезервированным именам в Basic относятся имена стандартных директив, операторов, функций и операций.

Переменные могут обозначать либо числовые величины, либо строки. Имя строковой или текстовой переменной должно заканчиваться символом $. Символ $ в данном случае является символом объявления, указывающим на то, что переменная является строковой - текстовой и состоит из символов текста. Имена числовых переменных могут обозначать целые числа, числа обычной и двойной точности.

Если необходимо, то для задания типа числовой переменной используются следующие символы объявления типа, всегда помещаемые в конце символа переменной:

1. Знак % - целая переменная, которая не содержит десятичную точку (знак % можно не писать.

2. Знак ! - переменная обычной точности содержит точку (знак ! можно не писать, достаточно присутствия десятичной точки).
3. Знак # - переменная двойной точности (знак # или описание переменной оператором DEFDBL обязательны).
4. Знак $ - строковая переменная (знак $ или описание переменной оператором DEFSTR обязательны). Например, ABV$ = “A” - значение строковой переменной A заключается в скобки.

По умолчанию числовые переменные имеют представление с обычной точностью.

Примеры имен переменных:

1. A$ - имя строковой переменной.
2. PI# - имя переменной двойной точности.
3. MINIMUM! - имя переменной обычной точности.
4. LIMIT% - имя целой переменной.
5. ABC - имя переменной обычной точности.

В Basic существует и другой способ объявления типа переменной. Для этого используются операторы DEFINT - целая, DEFSTR - строковая, DEFSNG - обычной точности и DEFDBL - двойной точности, которые будут описаны ниже. Для размещения в памяти целой переменной необходимо - 2 байта, для переменной обычной точности - 4 байта, а для переменной двойной точности - 8 байт. Для размещения строковой переменной в памяти необходимо 3 байта плюс количество байт, равное количеству символов в строке.

Массивы переменных

Массив - это группа чисел, ассоциированных с одним именем. Каждый элемент массива связан с переменной массива, которая имеет индекс, определяющийся целой величиной или целым выражением. Размерность массива определяется числом индексов переменной.

Например, A(10) - одномерный массив, каждый элемент которого A(I) имеет определенную величину, а индекс I может принимать значения от 0 до 10. B(20,30) - двумерный массив,

который, если не использовать нулевые элементы массива можно представить в виде таблицы с 20 строками и 30 столбцами.

Преобразование типов

В Basic существует возможность преобразования типов числовых констант. Если числовая константа одного типа присваивается числовой переменной другого типа, то переменной будет присвоена величина, соответствующая типу переменной. Если строковой переменной присваивается числовое значение или числовой переменной - строковое, то выдается сообщение об ошибке "Type mismatch".

Пример:

```
10 A% =23.42
20 PRINT A%
```

RUN - запуск программы и вывод на экран результата.

```
23
```

Таким образом, величина A будет иметь целое значение. Во время вычисления выражения все команды в арифметических операциях или операциях отношения преобразуются к одному типу представления - типу наибольшей точности. Результат арифметической операции, также будет иметь этот тип представления. При выполнении логических операций значения операндов преобразуются до целых, и результату присваивается целая величина.

Значения операндов должны находиться в пределах от -32768 до +32767, в противном случае выдается сообщение об ошибке "Overflow".

Когда число с дробной частью преобразуется в целую величину, выполняется округление. Если переменной двойной точности присваивается величина обычной точности, то эта величина преобразуется в представление двойной точности. Абсолютная погрешность между прежним и новым значением преобразованной величины не превышает 1.0E-7.

Пример:

```
10  A=2.04
20  B# =A
30 PRINT A;B#
```

RUN - запуск программы и вывод на экран результата.

```
2.04        2.039999961853027
```

Выражения и операции

Выражением может быть просто строковая или числовая константа или переменная. Им может быть комбинация констант, переменных, функций и операций.

Операции в языке Basic можно разделить на четыре категории:

1. Арифметические операции.
2. Операции отношения.
3. Логические операции.
4. Строковые операции.

Рассмотрим последовательно все эти типы операций:

Арифметические операции

Арифметические операции в Basic имеют порядок старшинства, приведенный ниже:

1. Возведение в степень - X^Y .
2. Отрицание - X .
3. Умножение и деление - X*Y; X/Y .
4. Целочисленное деление - X\Y .
5. Модуль числа - X MOD Y .
6. Сложение и вычитание - X+Y; X-Y .

Для изменения порядка выполнения операций используются круглые скобки. Операции внутри скобок выполняются в порядке старшинства первыми. При выполнении операции деления целых чисел и присвоении результату целого числа, он округляется до целых значений (они должны быть в пределах от -32768

до 32767), перед выполнением целочисленного деления частное усекается до целого (не округляется, а берется только целая часть числа). Примеры целочисленного деления:

10 \ 4 = 2
23.68 \ 6.99 = 3

Результатом операции модуля числа, также является остаток целочисленного деления.

Примеры:

10 MOD 4 = 2
23.68 MOD 6.99 = 3

Если во время вычисления выражения в Quick Basic происходит деление на ноль, выдается сообщение об ошибке "Division by zero", результату деления присваивается максимально возможное число со знаком числителя и выполнение программы продолжается. Если ноль возводится в отрицательную степень, выдается сообщение об ошибке "Division by zero", результату присваивается максимально возможное положительное число и выполнение программы продолжается. При возникновении переполнения выдается сообщение об ошибке "Overflow", результату присваивается максимально возможное число с соответствующим знаком и выполнение продолжается.

В Turbo Basic при таких операциях после выдачи соответствующего сообщение часто происходит "зависание" программы и требуется перезагружать компьютер.

Операции отношения

Операции отношения используются для сравнения двух величин. Результатом сравнения является "истина" (-1) или "ложь" (0):

1. = - знак равно, определяющий равенство величин - X=Y.
2. < > - знак не равно - X<>Y.
3. < - знак меньше - X<Y - величина X меньше, чем Y.
4. знак больше - X>Y - величина X больше, чем Y.
5. <= - меньше, чем или равно - X<=Y.

6. >= - больше, чем или равно - X>=Y.

Если арифметические операции и операции отношения объединены в одно выражение, то арифметические операции всегда выполняются первыми.

Например, выражение:

X ^ Y < (T - 1) / Z

является истинным, если величина X в степени Y меньше чем величина T - 1 деленная на Z.

Логические операции

Результатом логической операции является либо "истина" (не ноль), либо "ложь" (ноль). В выражениях логические операции выполняются после арифметических операций и операций отношения.

При выполнении логических операций операнды преобразовываются в шестнадцати битовое целое число в пределах от -32768 до +32767. Если значение операнда не находится в этих пределах, то результат будет ошибочным.

Строковые операции

Строки могут быть соединены с помощью операции конкатенации (обозначается знаком "+").

Пример:

```
10 A$ = "FILE"
15 B$ = "NAME"
20 PRINT A$ + B$
30 PRINT "NEW " + A$ + B$
```

RUN - запуск программы и вывод на экран результата.

```
FILENAME
NEW FILENAME
```

Для сравнения строк используются обычные операции отношения: =, <>, <, >, <=, >= .

Сравнение строк осуществляется путем последовательного сравнения (слева направо) кодов символов строк. Если все коды совпадают, то строки равны. Если коды разные, то та строка, чей код символа больше, считается большей. Если при сравнении строк обнаружен конец одной из строк, то это означает, что короткая строка меньше. Следует помнить, что пробелы в начале и в конце строк являются значащими и влияют на результаты сравнение.

Примеры:

```
"AA" < "AB"
"FILENAME" = "FILENAME"
"X&" > "X#"
"AB" < "ABC"
```

Все строковые константы, используемые в операциях отношения, должны быть заключены в кавычки. Можно предварительно текстовым переменным присвоит их значение и операции сравнения проводить с самими переменными.

Примеры:

```
A$="AA"
B$="AB"
```

Тогда при сравнении:

```
A$<B$
```

Если:

```
A$="AC"
B$="AB"
```

Тогда при сравнении будет обратный результат:

```
A$>B$
```

Если:

```
A$= "FILE"
B$= "FILE"
```

Тогда при сравнении получим равенство величин:

A$=B$

ОПЕРАТОРЫ BASIC

Написание любой программы на языке Basic выполняется с помощью последовательности определенных операторов, выполняющих те или иные математические и логические действия. Программа языка состоит также из операторов и операций присваивания переменным, и массивам их значений. Приведем далее краткие описания основных операторов, которые расположены в алфавитном порядке и некоторые примеры их использования.

Операторы BLOAD и BSAVE

Операторы BLOAD и BSAVE предназначены для загрузки/сохранения образа области оперативной памяти из/в файл(а) на диске.

Синтаксис:

BLOAD <имя файла>[,<смещение>]

и

BSAVE <имя файла>,<смещение>,<длина>

где:

1. Параметр <имя файла> - имя файла на диске куда записывается (SAVE) или откуда читается (LOAD) образ области оперативной памяти.
2. Параметр <смещение> - смещение, определяющее адрес оперативной памяти подлежащей сохранению или восстановлению.
3. Параметр <длина> - количество байтов сохраняемой области памяти.

Оператор CALL

Оператор CALL используется для вызова подпрограмм на Basic, находящихся в том же файле.

Синтаксис:

CALL <имя>[(<список аргументов>)]

где:

1. Параметр <имя> - задает имя подпрограммы.
2. Параметр <список аргументов> - представляет собой имена переменных или констант, разделенных запятыми, которые передаются в вызываемую подпрограмму.

Если необходимо получить результаты работы подпрограммы в основную программу, следует включить в <список аргументов> имена переменных, которым в подпрограмме будут присвоены возвращаемые значения.

Оператор CHAIN

Оператор CHAIN вызывает программу или подпрограмму из другого файла и передает ей переменные из текущей программы.

Синтаксис:

CHAIN <"имя файла">

где параметр <имя файла> - задает имя файла для вызываемой подпрограммы, заключенное в кавычки.

Имя файла есть допустимое для MS-DOS алфавитно - цифровое имя с расширением. Если расширение опущено - предполагаются программы с расширением .BAS, которая должна быть написана на Basic в компактном или текстовом формате. Если задано расширение .EXE - программа должна быть скомпилирована Basic в исполняемый файл. В Turbo Basic допустим вызов подпрограмм, находящихся в другом файле только из исполняемых файлов.

Пример:

CHAIN "PROG1"

Вызывающая программа обращается к файлу PROG1, в котором должна находиться подпрограмма, написанная на Basic.

Эта подпрограмма проводит необходимые, запрограммированные в ней вычисления и передает их результат в основную программу.

Оператор CLEAR

Оператор CLEAR обнуляет все переменные, закрывает все открытые файлы, а если заданы параметры, определяет максимальный адрес памяти и стековую область компилятора.

Синтаксис:

CLEAR [,<выражение 1>][,<выражение 2>]

где:

1. Параметр <выражение 1> (целое) - устанавливает максимальный адрес памяти, доступный компилятору языка Basic. Максимальным адресом памяти, устанавливаемым по умолчанию, является максимальный адрес доступной на компьютере памяти.
2. Параметр <выражение 2> - устанавливает область стековой памяти для Basic. По умолчанию эта область равна 256 байтам или одной восьмой части доступной памяти, если она меньше 256 байт.

Задание параметров в этом операторе допустимо только для программы Quick Basic.

Примеры:

CLEAR: CLEAR, 32768
CLEAR,,2000: CLEAR, 32768, 2000

Оператор CLOSE

Оператор CLOSE предназначен для завершения действий по вводу/выводу при работе с файлами на диске и закрывает файлы с указанными номерами.

Синтаксис:

CLOSE [[#]<номер файла>[,[#]<номер файла>...]]

где параметр <номер файла> - представляет собой номер, под которым файл был открыт (определен для работы с ним) с помощью оператора OPEN.

Оператор CLOSE без параметров закрывает все файлы. Связь между файлом и его номером прерывается после выполнения оператора CLOSE. Это значит, что под тем же номером с помощью оператора OPEN может быть открыт другой файл. Оператор END всегда автоматически закрывает все файлы на диске. Оператор STOP файлы не закрывает.

Операторы COMMON и SHARED

Оператор COMMON передает переменные программе, вызываемой оператором CHAIN. Оператор SHARED передает переменные программе, вызываемой оператором CALL. Переменным приписывается статус "общие".

Синтаксис:

COMMON [/имя-группы]<переменная>[(размерность)][AS тип][,...]

или

SHARED [/имя-группы]<переменная>[(размерность)][AS тип][,...]

где:

1. Параметр "имя-группы" - определяет группу переменных (длина имени до 40 байтов).
2. Конструкция "AS тип" - обязывает, чтобы переменная была одного из следующих типов: INTEGER, LONG, SINGLE, DOUBLE, STRING или типа, определенного пользователем.

Операторы могут находиться в любом месте программы, хотя рекомендуется размещать их в начале. Не допускается использование одной и той же переменной в нескольких операторах COMMON.

SHARED означает, что переменные используются совместно во всех процедурах - подпрограммах программного модуля, находящегося в одном файле. SHARED может использоваться только внутри процедур - подпрограмм FUNCTION (только для Quick Basic) и SUB. Обозначения переменных, стоящих в операторе должны совпадать с использованными с основной программе. Массивы переменных обозначаются добавлением символа () - скобки к имени переменной.

Операторы CHDIR, MKDIR и RMDIR

Эти операторы предназначены для работы с каталогами операционной системы MS DOS.

Синтаксис:

CHDIR <спецификация пути>
MKDIR <спецификация пути>
RMDIR <спецификация пути>

где <спецификация пути> - строка, заключенная в " " - кавычки, представляет собой полную спецификацию пути нужного каталога.

CHDIR изменяет текущий каталог на указанный. MKDIR создает указанный каталог на диске. RMDIR уничтожает указанный каталог на диске, если он пустой.

Оператор CONST

Оператор CONST применяется для определения числовых и строковых констант посредством произвольного идентификатора (только Quick Basic).

Синтаксис:

CONST <идентификатор константы>=<константа> [,...]

В одном операторе CONST можно определить несколько идентификаторов констант. Следует иметь в виду, что нельзя

переопределять значения констант, используя их идентификаторы в операции присваивания.

Оператор DECLARE

Оператор DECLARE предназначен для описания ссылок на процедуры и функции написанные, как на языке Basic, так и на других языках (только для Quick Basic).

Синтаксис:

DECLARE FUNCTION <имя>[CDECL][ALIAS "алиасное имя"][([<список параметров>])]

- только для Quick Basic или

DECLARE SUB <имя>[CDECL][ALIAS "алиасное имя"] [(<список параметров>)]

где:

1. Параметр CDECL - определяет порядок передачи аргументов в соответствии со стандартом языка Си. Эта опция может использоваться при объявлении процедуры, написанной на языках ассемблер, Си, FORTRAN или PASCAL.
2. Параметр ALIAS - определяет псевдоним процедуры в библиотеке (LIB-файле) или в объектном файле OBJ .
3. Параметр <список параметров> - описывается для процедур на других языках:

[Тип переменной может быть любым из допустимых: INTEGER, LONG, SINGLE, DOUBLE, STRING или ANY.

Если указано FUNCTION, то описываемая функция должна возвращать в себе значение, и может использоваться в качестве члена выражения, а SUB определяет процедуру, которая вызывается с помощью оператора CALL.

Оператор DATA

Оператор DATA заносит в память числовые и строковые константы, которые считываются оператором READ.

Синтаксис:

DATA <список констант>

где параметр <список констант> - может содержать числовые константы любого типа: с фиксированной точкой, плавающей точкой или целые. Строковые константы в операторе DATA должны заключаться в кавычки только в том случае, если они содержат запятые, двоеточия или пробелы в начале или в конце строки. В других случаях кавычки необязательны. Тип переменной (числовой или строковой), задаваемой в операторе READ, должен согласовываться с соответствующей константой в операторе DATA.

Оператор DATA может располагаться в любом месте программы. Количество констант в операторе DATA ограничивается только длиной строки (256 символов). Константы должны разделяться запятыми. В программе может быть любое количество операторов DATA. Оператор READ считывает данные последовательно в соответствии с номерами строк операторов DATA. Таким образом, константы, задаваемые несколькими операторами DATA, можно рассматривать, как последовательный список данных, формируемый в зависимости от длины <списка констант> операторов DATA и от того, какие номера имеют строки этих операторов.

Константы, задаваемые операторами DATA, могут быть прочитаны вновь с начала оператором READ после применения оператора RESTORE.

Оператор DEF FN

Оператор DEF FN определяет функцию, описываемую пользователем.

Синтаксис:

DEF FN<имя>[(<список параметров>)]=<описание функции>

где:

1. Тип любой из перечисленных переменных может быть INTEGER, LONG, DOUBLE или STRING.

2. Параметр <имя> - должен быть допустимым в Basic именем переменной. Это имя с предшествующими буквами FN и будет именем функции.

3. Параметр <список параметров> - включает имена переменных для описания функции, которые при вызове функции будут заменены соответствующими значениями. Параметры в списке должны разделяться запятыми.

4. Параметр <описание функции> - представляет собой выражение, описывающее функцию.

Имена переменных, которые включаются в выражение, используются только для описания функции и не влияют на переменные в программе, имеющие такие же имена. Переменные, используемые в описании функции, могут либо указываться, либо не указываться в списке параметров. Если переменная включена в список, то ее значение приводится во время вызова функции. Если же переменной в списке параметров нет, то используется текущее значение переменной. Функция, описываемая пользователем, может быть числовой или строковой. Если тип указан в имени функции, то значение выражения, описывающего функцию, преобразуется к этому типу. Если тип, указанный в имени функции, и тип аргумента не соответствуют, то выдается сообщение об ошибке "Type mismatch".

Оператор DEF FN должен располагаться прежде, чем произойдет вызов функции, определяемой им. Если функция вызывается раньше, то выдается сообщение об ошибке "Undefined user function".

Пример:

```
410 DEF FNAB(X,Y)=X^3/Y^2
.
.
.
.
450 T = FNAB(I,J)
```

Оператор DEF SEG

Оператор DEF SEG устанавливает адрес сегмента, который может использоваться в последствии операторами CALL ABSOLUTE, POKE, BLOAD, BSAVE и PEEK.

Синтаксис:

DEF SEG [=<адрес>]

где параметр <адрес> - представляет собой числовое выражение, значение которого должно быть целым беззнаковым числом в диапазоне от 0 до 65535.

Если же значение выражения выходит за этот диапазон, то выдается сообщение об ошибке "Illegal Function Call" и сохраняется прежнее значение адреса сегмента.

DEF и SEG должны быть разделены пробелом. Basic будет интерпретировать оператор DEFSEG=100, как присвоение величины 100 переменной DEFSEG.

Пример:

10 DEF SEG=&HB800.

Операторы DEFINT/LNG/ SNG/DBL/STR

Операторы DEFINT/LNG/SNG/DBL/STR объявляют тип переменных, как целый, длинный, обычной точности, двойной точности или строковый.

Синтаксис:

DEF<тип> <диапазон(ы) букв>

Здесь <тип> - это INT, LNG, SNG, DBL или STR.

Оператор DEF<тип> объявляет, что имена переменных, начинающихся с определенных букв, соответствуют указанному типу переменных. Если тип переменных не объявлен, то все переменные имеют обычную точность представления.

Примеры:

1. DEFDBL L-P - все переменные, начинающиеся с букв L, M, N, O и P, будут иметь двойную точность представления.
2. DEFSTR A - все переменные, начинающиеся с буквы A, будут строковыми переменными.

3. DEFINT I-N, W-Z - все переменные, начинающиеся с букв I, J, K, L, M, N, W, X, Y, Z, будут целыми переменными.

Оператор DO ... LOOP

Операторы DO ... LOOP организуют циклическое выполнение группы операторов ограниченных телом цикла DO ... LOOP пока (WHILE) или до тех пор, пока (UNTIL) условие продолжение цикла истинно.

Синтаксис:

```
DO
….
<операторы>
….
LOOP {WHILE | UNTIL} <условие продолжения цикла>
```

или

```
DO {WHILE | UNTIL} <условие продолжения цикла>
…..
<операторы>
…..
LOOP
```

где <условие продолжения цикла> - организуется оператором IF, а <операторы> - список выполняемых операторов программы.

Пример:

```
DO
X=X+1
PRINT X;
LOOP WHILE X<10
END
```

RUN - запуск программы и выдача на экран.

1 2 3 4 5 6 7 8 9 10

Операторы DIM и REDIM

Оператор DIM определяет максимальное значение индексов переменных массива и отводит необходимую массиву память.

Синтаксис:

[RE]DIM <список переменных с индексами [AS тип]>

Для каждой переменной списка в скобках указывается верхняя граница индекса. Сами переменные записываются через запятую. Тип массива может быть любым допустимым в Basic.

Пример для одномерных массивов:

DIM A(20), B(50), C(100)

Пример для двумерных массивов:

DIM A(20,10), B(20,50), C(5,100)

Если переменная массива не была описана оператором DIM, то максимальное значение индексов принимается равным 10. Если значение индекса превышает максимальное установленное значение, то выдается сообщение "Subscript out of range". Минимальное значение индекса всегда равняется нулю, кроме того случая, когда оно изменяется с помощью оператора OPTION BASE.

Оператор DIM устанавливает все элементы описываемого массива равными нулю. Приставка RE переопределяет ранее определенный массив.

Оператор END

Оператор END завершает выполнение программы, закрывает все файлы и возвращает пользователя на командный уровень компилятора языка Basic. Также оператор END определяет окончание различных синтаксических конструкций, таких как DEF, FUNCTION, IF, SELECT, SUB или TYPE.

Синтаксис:

END [{DEF | FUNCTION | IF | SELECT | SUB | TYPE}]

Операторы END могут располагаться в любом месте программы с целью завершения выполнения программы. Оператор END в конце программы является необязательным. Basic всегда возвращается на командный уровень после выполнения оператора END.

Пример:

```
520 IF K>1000 THEN END ELSE GOTO 20
```

Оператор ERASE

Оператор ERASE предназначен для удаления из программы массивов или их очистки. Если массив имеет атрибут STATIC, то все его элементы обнуляются. Массивы с атрибутом DYNAMIC уничтожаются и освобождают занимаемую ими память (только для Quick Basic).

Синтаксис:

ERASE <список переменных массивов>

После удаления массивов с помощью оператора ERASE имеется возможность заново задать их размерность или использовать отведенную ранее под массив память для других целей. В случае если производится попытка заново задать размерность массива, который не был удален оператором ERASE, выдается сообщение об ошибке: "Redimensioned array".

Пример:

```
50 ERASE A, B
60 DIM B(99)
```

Оператор EXIT

Оператор осуществляет выход из программных блоков, определенных в конструкциях DEF, циклах DO...LOOP и FOR...NEXT, а также из определенных пользователем функций FUNCTION и процедур SUB до их логического завершения.

Синтаксис:

EXIT {DEF | DO | FOR | FUNCTION | SUB}

В случае циклов управление передается на следующий оператор непосредственно после цикла. В случае процедур и функций управление передается на следующий оператор после точки вызова.

Оператор FIELD

Оператор FIELD предназначен для выделения памяти под переменные в файлах с прямым доступом.

Синтаксис:

FIELD [#]<номер файла>,<размер поля>AS<строковая переменная>[,<размер поля>AS<строковая переменная>]...

где:

1. Параметр <номер файла> - представляет собой номер, под которым файл был открыт с помощью оператора OPEN.
2. Параметр <размер поля> - это количество символов, отводимых под <строковую переменную>.

Для того чтобы получить данные из буфера после выполнения оператора GET, или для того, чтобы занести данные в буфер прямого доступа перед выполнением оператора PUT, необходимо выполнить оператор FIELD.

Пример:

```
100 FIELD 1,20 AS N$, 10 AS ID$
```

Оператор FIELD в приведенном примере отводит 20 байт в буфере прямого доступа строковой переменной N$ и 10 следующих байт строковой переменной ID$. Оператор FIELD не помещает данные в буфер файлов с прямым доступом.

Общее количество байт, отводимых переменным оператором FIELD не должно превышать длину записи, которая установлена к моменту открытия файла. Иначе выдается сообщение об ошибке "Field overflow".

Операторы FOR и NEXT

Операторы FOR и NEXT предназначены для организации циклического выполнения последовательности операторов языка Basic заданное число раз.

Синтаксис:

```
FOR <переменная>=x TO y [STEP z]
…..
<операторы>
…..
NEXT
```

где <переменная> - используется в качестве счетчика. Первое числовое выражение (x) задает начальное значение счетчика. Второе числовое выражение (y) определяет конечное значение счетчика.

Строки программы, следующие за оператором FOR, будут выполняться до тех пор, пока не будет встречен оператор NEXT. После этого к значению счетчика будет прибавлена величина, определяемая выражением (z) опции STEP - шаг. Затем осуществляется проверка с целью определения того, не превысило ли значение счетчика его конечного значения (y). Если этого не случилось, компилятор Basic осуществляет переход назад на оператор, который располагается сразу за оператором FOR и процесс вычисления повторяется. В случае если значение счетчика превысило конечное значение, выполнение программы продолжается с оператора, следующего непосредственно за оператором NEXT. В случае если опция STEP не задана, приращение принимается равным единице. Если в опции STEP задано отрицательное значение, то конечное значение счетчика цикла должно быть меньше начального. При этом значение счетчика каждый раз будет уменьшаться, и цикл будет выполняться до тех пор, пока значение счетчика не станет меньше конечного значения.

В случае если начальное значение счетчика, умноженное на знак приращения, превышает его конечное значение, умноженное на знак приращения, цикл игнорируется.

Циклы FOR ... NEXT могут быть вложенными, т.е. один цикл FOR ... NEXT может быть помещен в контексте другого цикла FOR ... NEXT.

При вложении циклов каждый цикл должен иметь собственное, отличное от других имя счетчика цикла. Оператор NEXT вложенного цикла должен располагаться в программе до оператора NEXT, соответствующего внешнему циклу. В случае если вложенные циклы имеют одну и ту же точку окончания, для них можно задавать один оператор NEXT.

Переменные в операторе NEXT могут быть опущены. В этом случае оператор NEXT будет соответствовать последнему оператору FOR. Если оператор NEXT располагается в программе раньше соответствующего ему оператора FOR, то выдается сообщение об ошибке "NEXT without FOR", и выполнение программы прекращается.

Пример:

```
10 J=5
20 FOR I=1 TO J+5
30 PRINT I;
40 NEXT
```

RUN - запуск программы и выдача на экран результата.

```
1 2 3 4 5 6 7 8 9 10
```

В примере цикл выполняется десять раз и на печать выводится значение счетчика цикла.

Оператор FUNCTION

Этот оператор в Quick Basic (в Turbo Basic такого оператора нет) объявляет функцию, определенную пользователем.

Синтаксис вызова функции:

```
….
A=<имя >[(<список переменных>)]
….
```

Синтаксис функции:

```
FUNCTION <имя>[(<переменная 1>[()] [AS <тип>] [,...])] [STATIC]
....
<имя>=<выражение>
```

....
END FUNCTION

где:

1. Параметр <имя> - любое допустимое для Basic имя, которое будет именем функции.
2. Параметр STATIC - позволяет сохранить значения локальных переменных функции между вызовами. Количество переменных - аргументов неограниченно и могут быть использованы любые допустимые в языке типы данных.

В теле функции обязательно должен быть оператор присваивания значения имени функции, которое передается в вызывающую программу.

Оператор GET

Оператор GET считывает запись из файла с прямым доступом, расположенного на диске, в буфер прямого доступа.

Синтаксис:

GET [#]<номер файла>[,[<номер записи>],[переменная]]

где <номер файла> - представляет собой номер, который присваивается файлу во время его открытия с помощью оператора OPEN.

В случае если <номер записи> опущен, выполняется считывание в буфер следующей записи (после прочитанной предыдущим оператором GET). Максимальный допустимый номер записи равняется 32767. После выполнения оператора GET считывание символов из буфера прямого доступа может быть осуществлено с помощью операторов INPUT и LINE INPUT.

Операторы GOSUB и RETURN

Операторы GOSUB и RETURN предназначены для перехода на подпрограмму и возвращения из нее:

Синтаксис вызова подпрограммы:

GOSUB <номер строки 1>|<метка 1>

Например, номер метки - 1.
Синтаксис самой подпрограммы:

```
1 REM Подпрограмма для …..
…..
<операторы>
…..
RETURN [<номер строки 2>|<метка 2>]
```

где <номер строки 1> или <метка 1> - представляют собой номер первой строки подпрограммы, или ее метку.

Подпрограмма может быть вызвана из программы произвольное число раз. Кроме того, подпрограмма может быть вызвана из другой подпрограммы. Такая вложенность подпрограмм ограничивается только объемом доступной памяти.

Оператор (операторы) RETURN (без метки) находится в подпрограмме и вызывает переход на оператор, следующий за последним выполненным оператором GOSUB. Подпрограмма может содержать несколько операторов RETURN, каждый из которых будет осуществлять выход из подпрограммы в определенной точке. Подпрограммы могут располагаться в различных местах программы, однако рекомендуется, чтобы подпрограммы были легко отличимы от основной программы. Для того чтобы предотвратить случайный вход в подпрограмму, перед ней можно расположить оператор STOP, END, GOTO или расположить ее в конце программы. Если применяется оператор RETURN <метка|номер строки> то управление передается на указанный оператор.

Пример:

```
10 GOSUB 40
20 PRINT "STOP"
30 END
40 PRINT "1234567890"
50 RETURN
```

RUN - запуск программы со следующей выдачей на экран.

1234567890
STOP

Оператор GOTO

Оператор GOTO осуществляет безусловный переход из обычной последовательности операторов программы на строку программы с заданным номером.

Синтаксис:

GOTO <номер строки>|<метка>

В случае если <номер строки> или <метки> относится к выполняемому оператору, то выполняется этот оператор и последующие за ним. Если же это невыполняемый оператор, то выполнение программы продолжается с первого выполняемого оператора, встреченного после <номера строки>.

Операторы IF...THEN [...ELSE] и IF...GOTO

Операторы IF...THEN [...ELSE] и IF...GOTO предназначены для изменения порядка выполнения программы в зависимости от значения выражения, следующего за опцией IF.

Синтаксис:

IF <выражение> THEN <оператор(ы)> [ELSE <оператор(ы)>]

или

IF < выражение > GOTO < номер строки|метка > [ELSE <оператор(ы) > | GOTO <номер строки|метка>]

Если значение <выражения> истинно, то выполняется предписание THEN или GOTO. За THEN может следовать либо номер строки, на которую следует осуществить переход, либо один или более выполняемых операторов. За GOTO всегда следует номер строки. Если значение <выражения> ложно, т.е. условие не выполняется, то предписания THEN или GOTO игно-

рируются и выполняется предписание ELSE, если оно задано. Перед THEN допускается наличие запятой. Допускается вложенность операторов IF ... THEN ... ELSE. Вложенность ограничивается только длиной строки.

Например, оператор:

```
IF X>Y THEN PRINT "GREATER" ELSE IF Y>X THEN PRINT "LESS THAN" ELSE PRINT "EQUAL"
```

является допустимым оператором. Если оператор состоит из разного количества предписаний ELSE и THEN, то каждому предписанию ELSE соответствует ближайшее THEN, у которого нет пары ELSE.

Например, оператор:

```
IF A=B THEN IF B=C THEN PRINT "A=C"  ELSE PRINT "A<>C"
```

не будет печатать "A<>C", когда A<>B.

Когда в операторе IF анализируется значение выражения, являющегося результатом вычислений в формате с плавающей запятой, следует помнить о том, что внутреннее представление величины может быть не точным. Следовательно, проверка должна производиться в пределах точности, в которых может изменяться величина. Например, для того, чтобы сравнить вычисленное значение переменной A с числом 1.0, воспользуемся оператором:

```
IF ABS(A-1.0)<1.0E-6 THEN...
```

Эта проверка возвратит значение "истина" в случае, если значение A совпадает с 1.0 с относительной ошибкой, меньшей, чем 1.0E-6.

Другой пример:

```
100 IF(I<20)*(I>10) THEN D=100-I GOTO 300
110 PRINT "OUT OF RANGE"
```

В этом примере проверяется, имеет ли переменная I значение, большее 10 и меньшее 20. В случае если значение I находится в этом интервале, то производится вычисление D и осуще-

ствляется переход на строку 300. В противном случае выполнение программы продолжается со строки 110.

Оператор INPUT

Оператор INPUT позволяет в процессе выполнения программы вводить данные с клавиатуры.

Синтаксис:

INPUT[;][<"наводящая строка">{; | ,}]<список переменных>

При достижении оператора INPUT выполнение программы приостанавливается и на экране выводится знак вопроса, который указывает на то, что программа ожидает ввода данных. Если в оператор включена <"наводящая строка">, то она печатается перед знаком вопроса. После этого с клавиатуры производится ввод необходимой информации. Вместо точки с запятой после <"наводящей строки"> можно использовать запятую, которая подавляет печать вопросительного знака.

Введенные данные присваиваются переменным, заданным в <списке переменных>. Число вводимых элементов данных должно соответствовать числу переменных в списке. Элементы данных в списке переменных разделяются запятыми. В качестве имен переменных в списке могут быть заданы переменные с индексами и строковые переменные. Тип каждого вводимого элемента данных должен соответствовать типу, задаваемому именем переменной. Строковые данные для оператора INPUT не нужно заключать в кавычки.

В случае если в режиме оператора INPUT было задано меньше или больше данных, чем необходимо, или тип данных неверный (числовые вместо строковых и т.п.), выдается сообщение "?Redo from start". До тех пор, пока не будет дан правильный ответ на запрос оператора INPUT, никаких присвоений входных значений не выполняется.

Пример:

```
10 INPUT "ЗНАЧЕНИЕ X";X
20 PRINT X
30 END
```

RUN - запуск программы для работы и выдача на экран.

ЗНАЧЕНИЕ X ? 7

Здесь число 7 вводится пользователем с клавиатуры.

Оператор INPUT

Оператор INPUT# предназначен для считывания элементов данных из последовательного файла на диске и присвоения их переменным программы.

Синтаксис:

INPUT# <номер файла>, <список переменных>

где:

1. Параметр <номер файла> - представляет собой номер, присвоенный файлу оператором OPEN, который открыл файл для ввода/вывода.
2. Параметр <список переменных> - состоит из имен переменных, которым будут присвоены элементы данных, хранящиеся в файле. (Тип переменной должен соответствовать типу, который задается именем переменной.) В отличие от оператора INPUT оператор INPUT# не выводит на экран знак вопроса.

Элементы данных в файле должны иметь такой же порядок, как если бы они вводились в ответ на запрос оператора INPUT.

Для числовых значений лидирующие коды пробела, возврата каретки и перевода строки игнорируются. Первый встречный символ, отличный от указанных выше считается первым символом числа. Число отделяется от других данных пробелом, возвратом каретки, переводом строки или запятой.

В случае если компилятор Basic осуществляет поиск строкового элемента в последовательном файле, лидирующие коды пробела, возврата каретки и перевода строки также игнорируются. Первый встречный символ, отличный от указанных считается началом строки. Если этот первый символ является кавычками ("), то строковый элемент данных будет состоять из всех символов, расположенных между первым и вторым символами

кавычек. Следовательно, заключенная в кавычки строка символов не может содержать символа кавычек.

В случае если первый символ строки не является символом кавычек, то это означает, что строка не заключена в кавычки, и она отделяется от других элементов данных запятой, возвратом каретки или переводом строки (кроме того, считывание строкового элемента прекращается после того, как были прочитаны 255 символов.)

Оператор KILL

Оператор KILL удаляет файл с диска.

Синтаксис:

KILL <"имя файла">

Если оператор KILL задан для файла, который открыт в этот момент времени, то возникает ошибка "File already open". Оператор KILL используется для всех типов дисковых файлов, программных файлов, файлов данных с прямым и последовательным доступом.

Оператор LINE INPUT

Оператор LINE INPUT вводит с клавиатуры целую строку до 254 символов и присваивает ее строковой переменной без использования ограничителей.

Синтаксис:

LINE INPUT[;][<"наводящая строка">;] < строковая переменная>

где:

1. Параметр <наводящая строка> - представляет собой строку символов, которая печатается на экране дисплея перед вводом данных. Знак вопроса не печатается, если только он не является частью <наводящей строки>.
2. Параметр <строковая переменная> - имя переменной, которой присваивается вся введенная информация, располагаю-

щаяся от конца наводящей строки до возврата каретки - нажатия клавиши Enter.

Тем не менее, если была встречена последовательность кодов перевода строки, возврата каретки (именно в этом порядке), то оба эти символа повторяются на экране дисплея, при этом возврат каретки игнорируется, а перевод строки заносится в <строковую переменную>, после чего вывод данных продолжается.

Оператор LINE INPUT#

Оператор LINE INPUT# предназначен для считывания из последовательного файла, расположенного на диске, полной строки до 254 символов и присвоения ее строковой переменной.

Синтаксис:

LINE INPUT#<номер файла>,<строковая переменная>

где:

1. Параметр <номер файла> - представляет собой номер, под которым этот файл был открыт с помощью оператора OPEN.
2. Параметр <строковая переменная> - имя переменной, которой присваивается строка.

Оператор LINE INPUT# считывает из последовательного файла все символы вплоть до кода возврата каретки - Enter. Затем осуществляется пропуск последовательности кодов возврата каретки и перевода строки, а при выполнении следующего оператора LINE INPUT# будут считаны все символы вслед до следующего кода возврата каретки. В случае если будет встречена последовательность кодов перевода строки, возврата каретки, то эта последовательность сохраняется.

Особенно удобно использовать оператор LINE INPUT# в тех случаях, когда каждая строка файла данных разбита на поля или, когда программа на языке Basic в коде ASCII считывается, как данные другой программой.

Пример:

```
10 OPEN "O",1,"LIST"
20 LINE INPUT"CUSTOMER INFORMATION?";C$
30 PRINT#1,C$
40 CLOSE 1
50 OPEN "I",1,"LIST"
60 LINE INPUT#1, A$
70 PRINT A$
80 CLOSE 1
```

RUN - запуск программы и результат выдачи на экран.

CUSTOMER INFORMATION?

Ввод с клавиатуры:

JOHN JONES 234,4 MEMPHIS

Вывод на экран:

JOHN JONES 234,4 MEMPHIS

Оператор MOD

Арифметическая операция MOD дает в качестве результата остаток от деления двух числовых выражений.

Синтаксис:

<числовая переменная>=<делимое> MOD <делитель>

Пример:

A = 10 MOD 4

Результат действия оператора MOD: A = 2

Операторы LPRINT и LPRINT USING

Оператор LPRINT и LPRINT USING предназначены для вывода данных на устройство печати.

Синтаксис:

LPRINT [<список выражений>] [{;|,}]

или

LPRINT USING <строковое выражение>; <список выражений> [{;|,}].

Эти операторы аналогичны операторам PRINT и PRINT USING за исключением того, что выдача информации производится на печатающее устройство, а не на экран монитора. Оператор LPRINT подразумевает, что ширина печатающего устройства составляет 132 символа.

Операторы LSET и RSET

Операторы LSET и RSET позволяют передавать данные из памяти компьютера в буфер файлов с прямым доступом (для дальнейшего использования оператора PUT).

Синтаксис:

LSET<строковая переменная>=<строковое выражение>

или

RSET<строковая переменная>=<строковое выражение>

Если <строковое выражение> занимает меньше байт, чем было выделено под <строковую переменную> оператором FIELD, то оператор LSET будет осуществлять левостороннее выравнивание строки в отведенном поле, а оператор RSET - правостороннее выравнивание. Оставшиеся в поле позиции заполняются пробелами.

В случае если строка длиннее поля, то усекаются символы, расположенные с правого края строки. Числовые значения должны быть преобразованы в строки, если их требуется передать с помощью операторов LSET или RSET. Для этого предназначены функции MKI$, MKS$, MKD$. Операторы LSET и RSET могут также использоваться со строковыми переменными, которые не обрабатываются оператором FIELD, с целью распо-

ложения строки, начиная с левого или правого края заданного поля.

Например, следующий оператор:

```
120 RSET A$=SPACE$(20)
```

заносят строку A$, начиная с правого края поля, состоящего из 20 символов.

Оператор MID$

Оператор MID$ выполняет замену части строки другой строкой.

Синтаксис:

MID$(<строковое выражение 1>, n[,m])=<строковое выражение 2>

где n и m - представляют собой целые выражения. Символы в <строковом выражении 1>, начиная с позиции n, заменяются символами <строкового выражения 2>.

Необязательный параметр m указывает число символов из <строкового выражения 2>, которые необходимо использовать в процессе замещения. В случае если m отсутствует, используется все <строковое выражение 2>. Тем не менее, независимо от того, присутствует m или нет, в процессе замены символов никогда не происходит выход за пределы исходной длины <строкового выражения 1>.

Пример:

```
10 A$="ВЫЧИСЛЕНИЕ SIN"
20 MID$(A$,12)="COS"
30 PRINT A$
```

RUN - запуск программы и результат.

```
ВЫЧИСЛЕНИЕ COS
```

Кроме того, MID$ можно использовать, как функцию, которая возвращает часть заданной строки.

Директива NAME

Директива NAME изменяет имя дискового файла.
Синтаксис:

NAME <старое имя файла>AS<новое имя файла>

где <старое имя файла> должно быть именем существующего файла, а <новое имя файла>, наоборот, не должно существовать. В противном случае, в результате выполнения директивы возникнет ошибка.

После выполнения директивы NAME файл располагается на том же диске и в той же области, но имеет новое имя.

Директива NULL

Директива NULL устанавливает число нулевых символов, которые будут печататься в конце каждой строки.
Синтаксис:

NULL <целое выражение>

Оператор ON ERROR GOTO

Оператор ON ERROR GOTO позволяет обнаруживать ошибки и задает номер первой строки подпрограммы обработки ошибок.
Синтаксис:

ON ERROR GOTO <номер строки>

После того, как было разрешено обнаружение ошибок, все встреченные ошибки, включая ошибки в режиме непосредственного выполнения (например, синтаксические ошибки), будут вызывать переход на заданную подпрограмму обработки ошибок. Если строка программы с указанным <номером строки> не

существует, то возникает ошибка "Undefined line". Для того, чтобы запретить обработку ошибок, следует выполнить оператор ON ERROR GOTO 0.

Все последующие ошибки будут приводить к выдаче сообщения об ошибке и прекращению выполнения программы. В случае если оператор ON ERROR GOTO 0 введен в процедуру обнаружения и обработки ошибок, он вызовет остановку компилятора Basic и печать сообщения о встреченной ошибке. Рекомендуется, чтобы для всех ошибок, которые не могут быть устранены, процедуру обработки ошибок выполнял оператор ON ERROR GOTO 0.

В случае если ошибка возникает при выполнении подпрограммы обработки ошибок, выдается сообщение об ошибке, и выполнение программы прекращается.

Пример:

```
10 ON ERROR GOTO 1000
```

Операторы ON...GOSUB и ON...GOTO

Операторы ON...GOSUB и ON...GOTO осуществляют переход на одну из заданных строк, номер которой определяется в зависимости от значения вычисленного выражения.

Синтаксис:

ON <выражение> GOSUB <список номеров строк>

или

ON <выражение> GOTO <список номеров строк>

Значение <выражения> определяет, какой из номеров строк в списке, записанном через пробелы, будет использован при переходе. Например, в случае, если значение <выражения> равно трем, в качестве адреса перехода будет взят третий номер в списке. Если значение <выражения> не является целым числом, производится округление дробной части.

В операторе ON...GOSUB каждый номер строки в списке должен представлять собой номер первой строки подпрограм-

мы. В случае если значение <выражения> равно нулю или превышает число элементов списка (но не больше 255), то выполнение программы будет продолжено со следующего оператора. Если значение <выражения> больше 255, то выдается сообщение об ошибке "Illegal function call".

Оператор OPEN

Оператор OPEN обеспечивает возможность выполнения операций ввода/вывода в расположенный на диске файл.

Синтаксис:

OPEN <режим 1>,[#]<номер файла>,<имя файла>[<длина записи>]

или

OPEN <имя файла> [FOR <режим 2>][ACCESS <доступ>] [<использование> AS [#]<номер-файла>] [LEN=<длина записи>]

где:

1. Параметр <режим 1> - представляет собой строковое выражение, первым символом которого является один из перечисленных ниже символов:

1) O - определяет последовательный метод доступа при выводе.
2) I - определяет последовательный метод доступа при вводе.
3) R - определяет прямой метод доступа при вводе и выводе.
4) B - определяет доступ к двоичному файлу, который может содержать любые из 256 символов кода ASCII.
5) A - последовательный доступ на вывод после последней записи.

2. Параметр <Режим 2> - также представляет собой строковое выражение, определяющее режим ввода/вывода, который

совпадает с <режимом 1>, но в отличие от него должен записываться полностью, как приведено ниже:

1) O - OUTPUT.
2) I - INPUT.
3) R - RANDOM.
4) B - BINARY.
5) A - APPEND.

3. Параметр <Номер файла> - является целым выражением, значение которого лежит в диапазоне от 1 до 15. В дальнейшем этот номер связан с файлом до тех пор, пока файл открыт, и используется для обращения к файлу при помощи других операторов ввода/вывода.

4. Параметр <Имя файла> - представляет собой строковое выражение, содержащее имя файла (заключенное в кавычки), которое согласуется с правилами относительно имен файлов, принятыми в MS DOS.

5. Параметр <Доступ> - строковое выражение, определяющее условие доступа к файлу:

1) READ - только на чтение.
2) WRITE - только на запись.
3) READ WRITE - и на чтение и на запись (только для RANDOM, BINARY и APPEND).

6. Параметр <Использование> - строковое выражение, определяющее использование файла другими (параллельными) процессами:

1) SHARED - все процессы могут иметь любой доступ к файлу.
2) LOCK READ - доступ к файлу по чтению не доступен процессам.
3) LOCK WRITE - доступ к файлу по записи не доступен процессам.
4) LOCK READ WRITE - файл вообще недоступен другим процессам.

7. Параметр <Длина записи> - является целым выражением, которое (если оно задано) устанавливает длину записи для файлов с прямым методом доступа. По умолчанию длина записи принимается равной 128 байтам. Файл может быть открыт для организации последовательного ввода или для прямого доступа под несколькими номерами одновременно. Однако для последовательного вывода файл может быть открыт только под одним номером.

Для того чтобы стало возможным выполнение операций ввода/вывода с расположенным на диске файлом, этот файл должен быть открыт с помощью оператора OPEN. Этот оператор назначает буфер для выполнения операций ввода/вывода в файл и определяет режим доступа к буферу.

Пример:

```
10 OPEN "I", 2, "DEMO.DAT"
```

Оператор OPTION BASE

Оператор OPTION BASE предназначен для установки минимального значения индексов массивов.

Синтаксис:

```
OPTION BASE n
```

где n принимает значение 0 или 1. По умолчанию минимальное значение индексов равняется нулю.

В случае если выполняется оператор

```
OPTION BASE 1
```

минимальным значением индексов будет единица.

Оператор POKE

Оператор POKE служит для записи байта данных в ячейку памяти.

Синтаксис:

POKE I, J ,

где I и J являются целыми выражениями:

1. I - представляет собой адрес ячейки памяти, в которую необходимо поместить однобайтную величину.
2. J - определяет величину, заносимую в память.

Выражение J должно принимать значение от 0 до 255, а выражение I от 0 до 65535. Дополнительной по отношению к оператору POKE является функция PEEK. Аргументом ее является адрес, по которому производится считывание информации.

Оператор POKE и функция PEEK применяются для хранения данных, загрузки подпрограмм на языке ассемблера, а также для передачи аргументов этим подпрограммам и получения от них результатов.

Пример:

10 POKE &H5A00, &HFF

Оператор PRINT

Оператор PRINT предназначен для вывода информации на дисплей - экран компьютера.

Синтаксис:

PRINT [<список выражений>]

В случае если <список выражений> отсутствует, печатается пустая строка. Если <список выражений> задан, то на дисплее распечатываются значения выражений. Выражения в списке могут быть числовыми и (или) строковыми. Строки должны быть заключены в кавычки. Позиция каждого печатаемого элемента определяется пунктуацией, используемой для разделения элементов в списке. Компилятор Basic разделяет строку на зоны печати, каждая из которых состоит из 14 позиций. Задание запятой в списке выражений приводит к тому, что следующее значение будет напечатано в начале следующей зоны. Если же задана точка с запятой, то следующее значение выражения будет распечатано непосредственно после предыдущего значения. Зада-

ние одного или нескольких пробелов между выражениями приводит к такому же результату, что и задание точки с запятой.

Если запятая или точка с запятой стоят в конце списка выражений, то следующий оператор PRINT будет осуществлять печать значений выражений в той же строке, что и предыдущий оператор PRINT. В случае если список выражений не заканчивается запятой или точкой с запятой, то в конце каждой строки на печать будет выдаваться код возврата каретки. Если печатаемая строка длиннее, чем длина строки дисплея, то осуществляется переход на следующую физическую строку и печать продолжается. Перед положительным числом и после любого числа всегда помещается пробел. Перед отрицательным числом ставится знак минус.

Числа с обычной точностью, которые могут быть представлены семью и менее цифрами в формате без показателя степени с такой же точностью, что и в формате с показателем степени, выводятся на печать в формате без показателя степени.

Числа с двойной точностью, которые могут быть представлены шестнадцатью и менее цифрами в формате без показателя степени с такой же точностью, что и в формате с показателем степени, выводятся на печать в формате без показателя степени.

В операторе PRINT вместо ключевого слова PRINT можно использовать вопросительный знак.

Пример:

```
10 X=5
20 PRINT X+5,X-5,X^5
30 END
```

RUN - запуск программы и результат на экране.

```
10          0          25
```

В приведенном примере запятые в операторе PRINT указывают на то, что каждое значение должно печататься в начале следующей зоны печати.

Пример:

```
10 INPUT X
20 PRINT X "SQUARED IS" X^2 "AND";
```

```
30 PRINT X "CUBED IS" X^3
```

RUN - запуск программы и выдача результата на экран.

```
? 9
```

Число 9 вводится с клавиатуры. Тогда на экране будет показан следующий результат:

```
9 SQUARED IS 81 AND 9 CUBED IS 729
```

В этом примере точка с запятой в конце строки 20 привела к тому, что оба оператора PRINT распечатывают данные в одной строке.

Пример:

```
10 FOR X=1 TO 3
20 J=J+5
30 K=K+10
40 ?J;K;
50 NEXT X
```

RUN - запуск программы и результат.

```
5  10  10  20  15  30
```

В этом примере две точки с запятой в операторе PRINT (строка 40) указывают на то, что каждое значение переменных должно выводиться на печать непосредственно за предыдущим значением. Не надо забывать, что за числом всегда следует пробел, а перед положительными числами также ставится пробел.

Оператор PRINT USING

Оператор PRINT USING осуществляет печать строк или чисел в определенном формате.

Синтаксис:

PRINT USING <строковое выражение>;<список выражений>

где:

1. Параметр <список выражений> - состоит из строковых или числовых выражений, которые необходимо вывести на печать, разделенных точками с запятой или запятыми.
2. Параметр <строковое выражение> - представляет собой строковую константу (или переменную), которая включает в себя специальные символы задания формата.

Эти символы задания формата (смотрите ниже) определяют поле и формат выводимых на печать строк или чисел.

В случае если оператор PRINT USING применяется для вывода строк, то для задания формата поля строки может быть использован один из следующих форматов:

1. Знак "!" - указывает, что на печать необходимо вывести только первый символ заданной строки.
2. Знак "\n пробелов" - указывает, что следует распечатать 2+n символов заданной строки. Если между обратными косыми чертами нет пробелов, на печать выводятся два символа строки. Один пробел приведет к выводу трех символов и т.д. В случае если строка длиннее поля, все дополнительные символы игнорируются. Если поле длиннее строки, то выполняется левостороннее выравнивание строки в поле и дополнение ее пробелами справа.
3. Знак "&" - указывает на то, что строка должна быть распечатана точно в том же виде, в каком она была введена.

Пример:

```
10 A$="FIVE"
15 B$="SEVEN"
20 PRINT USING"!";A$;B$
30 PRINT USING"\ \";A$;B$
40 PRINT USING"&";A$;B$
```

RUN - запуск программы и результат на экране.

```
FS
FIVSEV
FIVESEVEN
```

Если оператор PRINT USING применяется для печати чисел, то для задания формата поля числа используются следующие специальные символы:

1. Знак номера (#) - обозначает позицию каждой цифры. Если выводимое на печать число имеет меньше цифр, чем задано цифровых позиций, то осуществляется правостороннее выравнивание числа в поле (числу будут предшествовать пробелы).
2. Точка (.) - может размещаться в любой позиции поля. В случае если строка формата указывает на то, что перед десятичной точкой должна находиться цифра, то эта цифра всегда выводится на печать (ноль, если необходимо). В случае необходимости производится округление числа.
3. Знак плюс (+), помещенный в начале или в конце строки задания формата - указывает на то, что перед или после числа необходимо напечатать его знак (плюс или минус);
4. знак минус (-), расположенный в конце поля задания формата - указывает на то, что отрицательные числа должны печататься со знаком минус, расположенным после числа.
5. Два знака звездочка (**), помещенные в начале строки задания формата - указывают на то, что предшествующие пробелы в поле числа должны быть заполнены звездочками. Кроме того, звездочки резервируют позиции для двух дополнительных символов.
6. Два знака $$ - приводят к тому, что непосредственно перед числом будет напечатан знак $. Эти знаки резервируют место для двух символов, одним из которых является $. Экспоненциальный формат не может быть использован совместно с $$. Отрицательные числа могут использоваться только в том случае, если знак минус располагается справа от числа.
7. Символы **$ в начале строки задания формата - приводят к тому, что предшествующие числу пробелы будут заполнены звездочками и перед числом будет напечатан знак $. Символы **$ задают три дополнительные позиции, одну из которых занимает $.
8. Запятая (,), расположенная слева от десятичной точки в строке задания формата - приводит к тому, что слева от десятичной точки после каждой третьей цифры слева будет печататься запятая. Запятая, расположенная в конце строки задания

формата, выводится на печать, как часть строки. Запятая задает еще одну цифровую позицию. В случае если запятая используется вместе с экспоненциальным форматом (^^^^), она игнорируется.

9. Четыре стрелки вверх (^^^^), размещенные после символов цифровых позиций - используются для задания экспоненциального формата. Они отводят место для печати в виде E+xx. Может быть задана произвольная позиция десятичной точки. Осуществляется левостороннее выравнивание значащих цифр и определяется экспонента. Если не задан предшествующий числу знак плюс или следующие за числом знаки плюс или минус, то одна цифровая позиция слева от десятичной точки выделяется для печати числа или знака минус.

10. Знак подчеркивания (_) в строке определения формата - указывает на то, что следующий символ будет выведен на печать, как символьная константа. Для задания символа подчеркивания в качестве строковой константы необходимо в строке определения формата указать два этих символа.

Если число, которое требуется распечатать, превышает размеры отведенного для него числового поля, то перед числом будет напечатан знак процента (%). В случае, если в результате округления число выходит за пределы числового поля, перед округленным числом также будет напечатан знак процента.

Если число цифровых позиций в строке задания формата превышает 24, то выдается сообщение об ошибке "Illegal function call" ("Недопустимое обращение к функции").

Примеры использования оператора PRINT USING для печати чисел:

```
1) PRINT USING "##.##";.78
RUN - запуск программы и результат.

0.78

2) PRINT USING "###.##";987.654

RUN - запуск программы и результат.

987.65
```

3) PRINT USING "##.## ";10.2,5.3,66.789,.234

RUN - запуск программы и результат.

10.20 5.30 66.79 0.23

В последнем примере в конце строки задания формата помещены три пробела, которые предназначены для разделения печатаемых значений в строке.

Дополнительные примеры:

1) PRINT USING "+##.## ";-68.95,2.4,55.6,-.9

RUN - запуск программы и результат.

-68.95 +2.40 +55.60 -0.90

2) PRINT USING "##.##- ";-68.95,22.449,-7.01

RUN - запуск программы и результат.

68.95- 22.45 7.01-

3) PRINT USING "**#.# ";12.39,-0.9,765.1

RUN - запуск программы и результат.

*12.4 * -0.9 765.1

4) PRINT USING "$$###.##";456.78

RUN - запуск программы и результат.

-$456.78

5) PRINT USING "**$##.##";2.34

RUN - запуск программы и результат.

***$2.34

6) PRINT USING "####,.##";1234.5

RUN - запуск программы и результат.

1,234.50

7) PRINT USING "####.##,";1234.5

RUN - запуск программы и результат.

1234.50,

8) PRINT USING "##.##^^^^";234.56

RUN - запуск программы и результат.

2.35E+02

9) PRINT USING ".####^^^^-";888888

RUN - запуск программы и результат.

.8889E+06

10) PRINT USING " !##.## !";12.34

RUN - запуск программы и результат.

!12.34!

11) PRINT USING "##.##";111.22

RUN - запуск программы и результат.

%111.22

12) PRINT USING ".##";.999

RUN - запуск программы и результат.

%1.00

Операторы PRINT# и PRINT# USING

Операторы PRINT# и PRINT# USING предназначены для записи данных в последовательный файл, расположенный на диске.

Синтаксис:

PRINT#<номер файла>, [USING<строковое выражение>;] <список выражений>

где:

1. Параметр <номер файла> - представляет собой номер, присвоенный файлу в тот момент, когда файл был открыт для вывода с помощью оператора OPEN.
2. Параметр <строковое выражение> - состоит из символов задания формата.
3. Выражения в <списке выражений> - представляют собой числовые и (или) строковые выражения, которые будут записаны в файл.

Оператор PRINT# не производит уплотнения данных на диске. Представление данных записывается на диск в таком же виде, в каком данные распечатываются на экране дисплея с помощью оператора PRINT.

Поэтому необходимо разграничивать данные на диске таким образом, чтобы было обеспечено их правильное считывание с диска. В списке выражений числовые выражения должны быть разделены при помощи точек с запятой.

Например:

PRINT#1, A; B; C; X; Y; Z

В случае если в качестве разделителей используются запятые, то дополнительные пробелы, которые помещаются между печатаемыми числами, также будут записываться на диск.

Строковые выражения в списке должны быть разделены точками с запятой. Для того чтобы на диске строковые выражения были представлены в требуемом формате, следует использовать в списке выражений явно заданные ограничители.

Например, пусть:

```
A$="CAMERA"
B$="93604-1"
```

тогда оператор

```
PRINT #1, A$; B$
```

запишет на диск строку

```
CAMERA93604-1.
```

В связи с тем, что ограничители отсутствуют, эта информация не может быть введена обратно, как две отдельные строки. Для того чтобы такая ситуация не возникла, следует ввести в оператор PRINT# явно заданные разделители:

```
PRINT#1, A$; ","; B$
```

В результате выполнения этого оператора на диск будет записана следующая информация:

```
CAMERA, 93604-1
```

которая может быть прочитана обратно, как значения двух строковых переменных.

В случае если строки сами содержат запятые, точки с запятой, значащие предшествующие пробелы, коды возврата каретки или перевода строки, то при записи на диск они должны быть заключены в явно заданные кавычки (с помощью функции CHR$(34)).

Например, пусть:

```
A$="CAMERA, AUTOMATIC"
B$=" 93604-1".
```

тогда оператор

PRINT#1, A$; B$

запишет на диск следующую информацию:

CAMERA, AUTOMATIC 93604-1

и оператор

INPUT#1, A$, B$

прочитает данные и назначит переменной A$ символы:

"CAMERA"

а переменной B$ символы:

"AUTOMATIC 93604-1".

Для того, чтобы правильно разделить эти строки на диске, необходимо записать на диск символы кавычек, используя CHR$(34). Оператор:

PRINT#1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)

запишет на диск следующую информацию:

"CAMERA, AUTOMATIC"" 93604-1"

и оператор

INPUT#1, A$, B$

считает данные и присвоит переменной A$ символы:

"CAMERA, AUTOMATIC"

а переменной B$ символы:

" 93604-1".

Оператор PRINT# может также использоваться с опцией USING для управления форматом файла на диске.

Например:

PRINT#1,USING"$$###.##,";J;K;L

Оператор PUT

Оператор PUT предназначен для занесения записи из буфера прямого доступа в расположенный на диске файл с прямым доступом.

Синтаксис:

PUT[#]<номер файла>[,<номер записи>][,<переменная>]

где параметр <номер файла> - представляет собой номер, присвоенный файлу оператором OPEN во время его открытия.

Если <номер записи> не задан, то в качестве его принимается номер, следующий за номером записи последнего выполненного оператора PUT. Максимально возможным номером записи является 32767, а минимальным номером записи - единица.

Любая попытка прочитать или записать информацию после достижения конца буфера вызовет ошибку "Field overflow" ("Переполнение поля").

Оператор RANDOMIZE

Оператор RANDOMIZE инициирует генератор случайных чисел.

Синтаксис:

RANDOMIZE [<выражение>]

Если <выражение> опущено, компилятор языка Basic приостанавливает выполнение программы и запрашивает ввод значения с клавиатуры, печатая при этом на экране дисплея сле-

дующее сообщение: Random Number Seed (-32768 to 32767)? ("Начальное значение для генератора случайных чисел (от -32768 до 32767)?").

В случае если генератор случайных чисел не инициирован, функция RND всякий раз при выполнении программы будет возвращать одну и ту же последовательность случайных чисел. Для того чтобы изменить последовательность случайных чисел, генерируемых при каждом выполнении программы, необходимо поместить в начало программы оператор RANDOMIZE и при каждом запуске программы на выполнение изменять его аргумент.

Оператор READ

Оператор READ предназначен для считывания данных, задаваемых оператором DATA, и присваивания их переменным.

Синтаксис:

READ <список переменных>

Оператор READ должен всегда использоваться совместно с оператором DATA. Операторы READ устанавливают взаимно однозначное соответствие между переменными и данными, задаваемыми операторами DATA. Переменные в операторе READ могут быть, как числовыми, так и строковыми, но считываемые значения должны соответствовать заданным типам переменных. Если же такого соответствия нет, то выдается сообщение "Syntax error". Один оператор READ может обращаться к одному или нескольким операторам DATA, и наоборот, несколько операторов READ могут обращаться к одному и тому же оператору DATA. В случае если число переменных в <списке переменных> превышает число элементов в операторе (операторах) DATA, выдается сообщение “Out of data” ("Данные исчерпаны"). Если заданное число переменных меньше, чем число элементов в операторе (операторах) DATA, то последующие операторы READ начнут считывание данных с первого непрочитанного элемента. Если же последующих операторов DATA нет, то непрочитанные данные игнорируются.

Для того чтобы повторить сначала считывание данных, заданных операторами DATA, следует воспользоваться оператором RESTORE.

Пример:

```
10 DATA "ABC,", XYZ, 802111
20 READ B$,C$,Z
30 PRINT B$,C$,Z
```

RUN - запуск программы и результат.

```
ABC, XYZ 802111
```

Оператор REM

Оператор REM позволяет вводить в программу комментарии.

Синтаксис:

REM <примечание>

или

' <примечание>

Операторы REM не выполняются, однако при выдаче листинга программы воспроизводятся точно в том же виде, в котором они были введены в программу. Допускается переход на операторы REM с помощью операторов GOTO и GOSUB. После этого выполнение программы будет продолжено с первого выполняемого оператора, следующего за оператором REM. Комментарии могут добавляться в конце программной строки. Вместо оператора REM перед комментарием можно помещать знак апострофа ('). Однако, комментарии нельзя размещать в конце операторов DATA, поскольку они будут восприниматься как данные.

Пример:

```
120 REM ВЫЧИСЛЕНИЕ СРЕДНЕЙ СКОРОСТИ
130 FOR I=1 TO 20
```

```
135 SUM=SUM+V(I)
140 NEXT I
```

или

```
120 'ВЫЧИСЛЕНИЕ СРЕДНЕЙ СКОРОСТИ
130 FOR I=1 TO 20
135 SUM=SUM+V(I)
140 NEXT I
```

Оператор RESET

При выполнении этого оператора производится закрытие всех активных (открытых) файлов.

Синтаксис:

```
RESET
```

Оператор RESTORE

Оператор RESTORE обеспечивает повторное считывание данных из операторов данных DATA, начиная с указанной строки.

Синтаксис:

```
RESTORE [<номер строки>]
```

После выполнения оператора RESTORE следующий оператор READ считывает первый элемент первого оператора DATA программы. В случае если задан <номер строки>, следующий оператор READ обращается к первому элементу оператора DATA в указанной строке.

Оператор RESUME

Оператор RESUME продолжает выполнение программы после завершения подпрограммы обработки ошибок.

Синтаксис:

```
RESUME
RESUME 0
```

RESUME NEXT
RESUME <номер строки>

В зависимости от того, с какого места нужно продолжить выполнение программы, используется одна из приведенных выше форм записи оператора RESUME.

Если задана первая или вторая форма представления оператора RESUME, выполнение программы возобновляется с оператора, в котором была обнаружена ошибка. Если используется RESUME NEXT, то выполнение программы будет продолжаться с оператора, который непосредственно следует за оператором, вызвавшим ошибку. При использовании оператора RESUME <номер строки> выполнение программы возобновляется со строки с указанным номером.

Если оператор RESUME находится не в процедуре обработки ошибки, то печатается сообщение "RESUME without error".

Пример:

```
10 ON ERROR GOTO 900
900 IF (ERR=230) AND (ERL=90) THEN PRINT "AGAIN"
RESUME 80
```

Оператор RUN

Синтаксис:

RUN [{<метка строки> | <имя файла>}]

Оператор RUN выполняет программу, обрабатываемую в данный момент компилятором, начиная с оператора, заданного <меткой строки> или исполняет программу на языке Basic, находящуюся в файле, специфицированном <именем файла>. Если <имя файла> указано без расширения то подразумевается .BAS.

Оператор SELECT CASE

Синтаксис:

SELECT CASE <выражение> [CASE {<выражение>, [,...] |<выражение a> TO <выражение b>| IS <выражение с операто-

ром отношения>}[блок исполняемых операторов]] [CASE ELSE [блок исполняемых операторов]] END SELECT

Эта конструкция позволяет исполнять различные блоки операторов в зависимости от истинности выражений указанных в операторах CASE.

Оператор SHELL

Оператор SHELL позволяет исполнить любую команду MS DOS определенную в <командной строке>. Под командой понимается выполнение файлов с расширением COM, EXE или BAT.

Синтаксис:

SHELL [<"командная строка">]

Если <командная строка> не задана, то осуществляется выход в среду MS DOS путем запуска новой копии COMMAND. COM. Возврат в программу производится после ввода команды EXIT.

Оператор SETMEM

Оператор SETMEM позволяет изменить объем памяти отведенной под область хранения данных (только для Quick Basic).

Синтаксис:

SETMEM (<числовое выражение>)

где <числовое выражение> - определяет, на сколько байтов нужно увеличить или уменьшить область памяти в зависимости от знака. Если <числовое выражение> равно нулю, то область не изменяется.

После выполнения, оператор SETMEM возвращает в своем значении новый объем памяти под область хранения данных программы.

Оператор STATIC

Оператор STATIC обеспечивает сохранение значений указанных переменных между вызовами процедур или функций из основной программы.

Синтаксис:

STATIC <имя переменной> [AS <тип>] [,...]

Область использования - внутри блоков FUNCTION, SUB и DEF FN.

Оператор STOP

Оператор STOP предназначен для прекращения выполнения работающей в данный момент программы и возвращения на командный уровень компилятора.

Синтаксис:

STOP

Операторы STOP могут размещаться в любых местах программы с целью прекращения ее выполнения. В отличие от оператора END, оператор STOP не закрывает файлы. После выполнения оператора STOP компилятор Basic всегда возвращается на командный уровень. Выполнение программы может быть продолжено при помощи директивы CONT (только Quick Basic).

Оператор SYSTEM

Оператор закрывает все открытые файлы, завершает программу и передает управление компилятору Basic.

Синтаксис:

SYSTEM

Оператор SUB

Оператор описывает процедуру - подпрограмму, которая может быть откомпилирована отдельно или вместе с головным

модулем. В случае раздельной компиляции имеется возможность поместить процедуру в общую библиотеку.

Синтаксис:

```
SUB <имя процедуры> [(<список параметров>)] [STATIC]
 ....
END SUB
```

Чтобы использовать библиотечную процедуру созданную ранее и помещенную в библиотеку с расширением QLB в интегрированном окружении компилятора следует запустить, например, компилятор Quick Basic с ключом: QB/L и на запрос: "QB QLB file not found" ввести полное имя требуемой библиотеки.

Оператор SWAP

Оператор SWAP осуществляет обмен значений двух переменных.

Синтаксис:

```
SWAP <переменная 1>,<переменная 2>
```

Обмен значениями с помощью оператора SWAP может выполняться для любых типов переменных (целых, с обычной точностью, с двойной точностью, строковых), однако обе переменные должны иметь один и тот же тип. В противном случае возникает ошибка "Type mismatch".

Пример:

```
10 A$="ONE "
11 B$="ALL "
12 C$="FOR"
20 PRINT A$ C$ B$
30 SWAP A$, B$
40 PRINT A$ C$ B$
```

RUN - запуск программы и результат.

```
ONE FOR ALL
ALL FOR ONE
```

Операторы TRON и TROFF

Операторы TRON и TROFF соответственно устанавливают и отменяют трассировку при выполнении программы.

Синтаксис:

TRON

и

TROFF

Оператор TRON, используемый в качестве средства отладки устанавливает режим трассировки, при котором печатается номер каждой выполненной строки программы. Номера строк заключаются в квадратные скобки.

Пример:

```
TRON
10 K=10
20 FOR J=1 TO 2
30 L=K+10
40 PRINT J;K;L
50 K=K+10
60 NEXT J
70 END
```

RUN - запуск программы и результат.

```
[10][20][30][40] 1 10 20
[50][60][30][40] 2 20 30
[50][60][70]
```

Оператор TYPE

Этот оператор позволяет описывать структуры данных, определяемые пользователем.

Синтаксис:

TYPE <имя структуры> <имя элемента> AS <тип>
....
END TYPE

В качестве типа могут использоваться INTEGER, LONG, SINGLE, DOUBLE, STRING * n (строка фиксированной длины n) и ранее определенные пользователем структуры данных. Имена должны соответствовать соглашениям языка и не должны определять имя массива.

Операторы WHILE ...WEND

Операторы WHILE...WEND предназначены для циклического повторения последовательности операторов до тех пор, пока заданное условие истинно.

Синтаксис:

WHILE<выражение>
...
[<операторы цикла>]
...
WEND

В случае если <выражение> не равно нулю (т.е. истинно), осуществляется выполнение <операторов цикла> до тех пор, пока не будет встречен оператор WEND. Затем происходит возврат к оператору WHILE и выполняется проверка <выражения>. Если оно, по - прежнему, истинно, то <операторы цикла> выполняются вновь. Если же оно не истинно, выполнение программы продолжается с оператора, который непосредственно следует за оператором WEND.

Допускается произвольное число уровней вложенности циклов WHILE…..WEND. Каждый оператор WEND соответствует последнему оператору WHILE. Если у оператора WHILE нет соответствующего ему оператора WEND, выдается сообщение об ошибке "WHILE without END". Если же наоборот оператору WEND не соответствует оператор WHILE, то выдается сообщение "WEND without WHILE".

Оператор WIDTH

Оператор WIDTH устанавливает длину строки в символах для экрана дисплея и устройства печати.

Синтаксис:

WIDTH <размер>
или

WIDTH <номер файла>, <размер>

или

WIDTH <устройство>, <размер>

где:

1. Параметр <размер> - числовое выражение, значение которого должно быть целым числом в диапазоне от 0 до 255 задает длину строки.
2. Параметр <устройство> - строковое выражение, определяющее имя устройства. Именем устройства может быть SCRN или LPT1.
3. Параметр <номер файла> - числовое выражение, значение которого должно быть целым числом в диапазоне от 1 до 4, которое определяет номер открытого файла.

Операторы:

WIDTH <размер>

и

WIDTH <колонки>, <строки>

устанавливают ширину экрана дисплея. Она может равняться 40 или 80 символам и до 43 строк.

Если дисплей находится в режиме со средней разрешающей способностью (SCREEN 1), то оператор WIDTH 80 переводит его в режим с высокой разрешающей способностью (SCREEN 2). Если же дисплей находится в режиме с высокой разрешаю-

щей способностью (SCREEN 2), то оператор WIDTH 40 переводит его в режим со средней разрешающей способностью (SCREEN 1).

Оператор:

WIDTH LPRINT, <размер>

устанавливает длину строки для устройства печати. Причем, в действительности длина строки изменяется лишь после выполнения оператора:

OPEN "LPT1:" FOR OUTPUT AS <номер>

Оператор:

WIDTH <номер файла>, <размер>

устанавливает длину строки для устройства печати в файл. Оператор позволяет изменять длину строки уже открытого файла, которая может иметь длину от 1 до 255 символов. Любое значение, выходящее за указанные пределы, приведет к ошибке "Illegal Function Call".

Пример:

10 WIDTH LPRINT, 75

При изменении режима работы дисплея с помощью оператора SCREEN длина строки экрана меняется только в случае перехода от режима SCREEN 2 к SCREEN 1 или SCREEN 0, или наоборот.

Оператор WRITE

Оператор WRITE предназначен для вывода информации на дисплей.

Синтаксис:

WRITE [<список выражений>]

Если <список выражений> не задан, то выводится пустая строка. В случае если <список выражений> задан, на дисплей выводятся значения выражений. В списке могут быть строковые и числовые выражения, которые должны быть разделены запятыми.

Каждое выводимое с помощью оператора WRITE значение выражения отделяется от предыдущего значения запятой. Выводимые строковые выражения заключается в кавычки. После того, как был распечатан последний элемент <списка выражений>, выдается последовательность кодов возврата каретки и перевода строки.

Оператор WRITE выводит числовые значения в том же формате, что и оператор PRINT.

Пример:

```
10 A=80
11 B=90
12 C$="THAT'S ALL"
20 WRITE A,B,C$
```

RUN - запуск программы и результат.

```
80, 90,"THAT'S ALL"
```

Оператор WRITE#

Оператор WRITE# осуществляет запись информации в последовательный файл.

Синтаксис:

WRITE# <номер файла>, <список выражений>

где <номер файла> - представляет собой номер, под которым файл был открыт с помощью оператора OPEN для работы в режиме последовательного доступа при вводе.

Входящие в список выражения представляют собой строковые или числовые выражения, разделенные запятыми. Различие между операторами WRITE# и PRINT# заключается в том, что оператор WRITE# по мере записи отдельных элементов на диск

вставляет между ними запятые и заключает строки в кавычки. Следовательно, пользователю не надо вводить в список явно заданные ограничители. После записи на диск последнего элемента списка осуществляется запись кодов возврата каретки и перевода строки.

Например, пусть:

```
A$="CAMERA"
B$="93604-1"
```

Тогда оператор

```
WRITE#1, A$, B$
```

запишет на диск следующую информацию:

```
"CAMERA", "93604-1"
```

Последующий оператор
```
INPUT#1, A$, B$
```

присвоит переменной A$ символы

```
"CAMERA"
```

а переменной B$ - символы

```
"93604-1"
```

ВСТРОЕННЫЕ ФУНКЦИИ

В этом разделе описываются встроенные функции компилятора языка Basic. К этим функциям можно обращаться из любой программы, написанной на языке Basic без их предварительного определения. Аргументы функций всегда заключаются в круглые скобки. При описании синтаксиса функций используются следующие обозначения их аргументов:

1. X и Y - произвольные числовые выражения.
2. I и J - целые выражения.
3. X$ и Y$ - строковые выражения.

В случае если вместо целого выражения задано выражение, представленное в формате с плавающей точкой, то его значение округляется и затем используется полученное целое число. Функция возвращает - выдает в качестве результата работы только целые значения или значения обычной точности.

Функция ABS

Функция ABS возвращает беззнаковое абсолютное значение числа X.

Синтаксис:

ABS (X)

Пример:

```
PRINT ABS(7*(-5))
```

RUN - запуск программы и результат.

35

Функция ASC

Функция ASC возвращает числовое значение, которое является кодом ASCII первого символа строки X$.

Синтаксис:

ASC(X$)

В случае, если строка пустая, выдается сообщение об ошибке "Illegal function call."

Пример:

```
10 X$="TEST": PRINT ASC(X$)
```

RUN - запуск программы и результат.

```
84
```

Для обратного преобразования используется функция CHR$.

Функция ATN

Функция ATN возвращает заданное в радианах значение арктангенса X, которое находится в пределах от -pi/2 до pi/2.

Синтаксис:

```
ATN(X)
```

Выражение X может иметь любой числовой тип. Вычисление арктангенса выполняется с обычной точностью для INTEGER и SINGLE, и с двойной точностью для DOUBLE.

Пример:

```
10 INPUT X: PRINT ATN(X)
```

RUN - запуск программы.

```
? 3
```

Ввод числа 3 с клавиатуры и результат.

```
1.249046
```

Функция CDBL

Функция CDBL преобразует значение X в число двойной точности.

Синтаксис:

CDBL(X)

Пример:

```
10 A=454.67: PRINT A; CDBL(A)
```

RUN - запуск программы и результат.

```
454.67  454.6700134277344
```

Функция CHR$

Функция CHR$ возвращает строку, символом которой является код ASCII для заданного числа I.

Синтаксис:

CHR$(I)

Чаще всего функция CHR$ используется для выдачи на экран дисплея специальных символов.

Пример:

```
PRINT CHR$(66)
```

RUN - запуск программы и результат.

```
B
```

Функция CINT

Функция CINT преобразует значение X в целое число путем округления дробной части.

Синтаксис:

CINT(X)

Если значение X не лежит в пределах от -32768 до 32767, то возникает ошибка "Overflow".

Пример:

```
PRINT CINT(45.67)
```

RUN - запуск программы и результат.

```
46
```

Функция COS

Функция COS возвращает значение косинуса X, заданного в радианах.

Синтаксис:

```
COS(X)
```

Вычисление значения COS(X) выполняется с обычной точностью.

Пример:

```
10 X=2*COS(0.4): PRINT X
```

RUN - запуск программы и результат.

```
1.842122
```

Функция CSNG

Функция CSNG преобразовывает значение X в число обычной точности представления.

Синтаксис:

```
CSNG(X)
```

Пример:

```
10 A#=975.34213
20 PRINT A#; CSNG(A#)
```

RUN - запуск программы и результат.

975.34213 975.3421

Функция EXP

Функция EXP возвращает результат возведения числа e=2.718282 в степень X.

Синтаксис:

EXP(X)

Значение X не должно превышать 87.3365. В случае если при возведении в степень происходит переполнение, выдается сообщение об ошибке "Overflow" и в качестве результата принимается максимально возможное число, и выполнение программы продолжается. В Turbo Basic после выдачи сообщения об ошибке работа программы останавливается.

Пример:

10 X=5: PRINT EXP(X-1)

RUN - запуск программы и результат.

54.59815

Функция FIX

Функция FIX усекает значение X до целого числа.

Синтаксис:

FIX(X)

Результат действия функции FIX(X) эквивалентен значению следующего выражения: SGN(X)*INT(ABS(X)). Основное различие между функциями FIX и INT заключается в том, что функция FIX просто отбрасывает дробную часть числа не зависимо от его знака.

Пример:

PRINT FIX(58.75)

RUN - запуск программы на счет и результат.

58

PRINT FIX(-58.75)

RUN - запуск программы на счет и результат.

-58

Функция FRE

Функция FRE возвращает число байт памяти, которые не использованы компилятором языка Basic.

Синтаксис:

FRE({-1 или -2 или число})

или

FRE(X$)

Функция FRE("") выполняет удаление ненужной информации в памяти машины, а затем возвращает число свободных байт. Удаление ненужной информации может длиться от одной до полутора минут. Компилятор языка Basic не выполняет удаления ненужной информации до тех пор, пока не будет использована вся свободная память. Таким образом, периодическое использование FRE("") сократит задержки, связанные с удалением ненужной информации. Функция FRE(-1) возвращает число байт памяти, которые не использованы наибольшим числовым массивом. Функция FRE(-2) возвращает число байт памяти, которые не использованы в стековой памяти. Функция FRE(<"строка">) возвращает число байт памяти, которые не использованы в области размещения строк (после первого освобождения занимаемой ими памяти в обычном блоке).

Функция HEX$

Функция HEX$ возвращает строку, которая является шестнадцатеричным представлением десятичного аргумента.

Синтаксис:

```
HEX$(X)
```

Перед вычислением HEX$(X) выполняется округление значения X до целого.

Пример:

```
10 INPUT X: A$=HEX$(X)
20 PRINT X "ДЕС-ЧНОЕ РАВНЯЕТСЯ" A$  "ШЕСТ-МУ"
```

RUN - запуск программы и результат.

```
? 32
```

Число 32 задается с клавиатуры. А результат выглядит на экране следующим образом:

```
32 ДЕС-ЧНОЕ РАВНЯЕТСЯ 20 ШЕСТ-МУ
```

Функция INKEY$

Функция INKEY$ возвращает либо строку, состоящую из одного символа, считанного с клавиатуры, либо нулевую строку, если с клавиатуры не было передано ни одного символа.

Синтаксис:

```
INKEY$
```

Дублирование символов на экране дисплея не производится, и символы в программу не передаются, за исключением команды Ctrl-Break, которая прекращает выполнение программы.

Пример:

```
10 RESPONSE$=""
20 FOR I%=1 TO TIMELIMIT%
30 A$=INKEY$
35 IF LEN(A$)=0 THEN 60
```

```
40 IF ASC(A$)=13 THEN RETURN
50 RESPONSE$=RESPONSE$+A$
60 NEXT I%
70 RETURN
```

Функция INPUT$

Функция INPUT$ возвращает строку, состоящую из X символов, которые считываются с клавиатуры или из файла с номером Y.

Синтаксис:

```
INPUT$(X[,[#]Y])
```

Если клавиатура используется для ввода, символы на экране не отображаются и все управляющие символы игнорируются.

Пример 1:

```
10 OPEN"I",1,"DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

Пример 2:

```
100 PRINT "ВВЕДИТЕ P ДЛЯ ПРОДОЛЖЕНИЯ ИЛИ S ДЛЯ ПРЕКРАЩЕНИЯ"
110 X$=INPUT$(1)
120 IF X$="P" THEN 140
130 IF X$="S" THEN END ELSE 100
140 ….
```

Функция INSTR

Функция INSTR определяет первое вхождение строки Y$ в строку X$ и возвращает номер позиции, начиная с которой обнаружено совпадение.

Синтаксис:

INSTR([I,]X$,Y$)

Необязательный параметр I определяет позицию, с которой начинается поиск вхождения строки Y$ в строку X$. Значение I должно быть в пределах от 1 до 255. В случае если I превышает длину строки X$, то функция INSTR возвращает ноль. Если Y$ представляет собой пустую строку, функция INSTR возвращает значение I или 1. X$, Y$ могут быть строковыми переменными, строковыми выражениями или строковыми константами.

Пример:

```
10 X$="ABCDEB": Y$="B"
30 PRINT INSTR(X$,Y$); INSTR(4,X$,Y$)
```

RUN - запуск программы и результат.

```
2         6
```

Если I=0, выдается сообщение об ошибке "Illegal argument in <номер строки>".

Функция INT

Функция INT возвращает максимальное целое число, не превышающее X, т.е. округляет X в меньшую сторону.

Синтаксис:

INT(X)

Примеры:

1) PRINT INT(99.89)

RUN - запуск программы и результат.

99

2) PRINT INT(-12.11)

RUN - запуск программы и результат.

-12

Функция LEFT$

Функция LEFT$ возвращает строку, состоящую из I символов, располагающихся с левого края строки X$.

Синтаксис:

LEFT$(X$,I)

Значение I должно быть в пределах от 0 до 255. Если значение I больше, чем LEN(X$), функция LEFT$ возвращает всю строку X$. В случае, если I=0, функция возвращает нулевую строку (строку, у которой длина равна нулю).

Пример:

```
10 A$="INTERPRETER": B$=LEFT$(A$,5): PRINT B$
```

RUN - запуск программы и результат.

INTER

Функция LEN

Функция LEN возвращает число символов в строке X$, при этом учитываются символы, не имеющие графического представления, и пробелы.

Синтаксис:

LEN(X$)

Пример:

```
10 X$="ЯЛТА КРЫМСКОЙ"
20 PRINT  LEN(X$)
```

RUN - запуск программы и результат.

14

Функции LOC и LOF

Для файлов с прямым доступом функция LOC возвращает номер последней записанной или прочитанной записи операторами PUT или GET.

Синтаксис:

LOC (<номер файла>)

Если файл был открыт, но дисковые операции ввода/вывода не выполнялись, то функция LOC возвращает ноль. Для файлов с последовательным доступом функция LOC возвращает число секторов (128-байтных блоков), считанных или записанных в файл с момента, когда он был открыт при помощи оператора OPEN. Для двоичных файлов возвращает номер последнего считанного или записанного байта.

Пример:

```
200 IF LOC(1)>50 THEN STOP
```

Функция LOF возвращает количество байт в файле.

Синтаксис:

LOF(<номер файла>)

Пример:

```
10 OPEN "FILE.BIG" AS #1
20 GET #1, LOF(1)/128
```

Функция LOG

Функция LOG возвращает значение натурального логарифма числа X.

Синтаксис:

LOG(X)

Значение X должно быть больше нуля.

Пример:

```
PRINT LOG(45/7)
```

RUN - запуск программы и результат.

```
1.860752
```

Функция MID$

Функция MID$ возвращает строку длиной в J символов, которая извлекается из строки X$ и начинается с I-го символа строки X$.

Синтаксис:

```
MID$(X$,I[,J])
```

Значения I и J должны представлять собой целые числа в диапазоне от 1 до 255. Если J не задано или справа от I-го символа строки X$ располагается меньше символов, чем задано значением J, то функция MID$ в качестве результата возвращает все символы строки X$, начиная с I-го символа. В случае если значение I превышает длину строки X$ (I>LEN(X$)), функция MID$ возвращает нулевую строку.

Пример:

```
10 A$="GOOD "
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$;MID$(B$,9,7)
```

RUN - запуск программы и результат.

```
GOOD EVENING
```

Если I=0, выдается сообщение об ошибке "Illegal argument in <номер строки>".

Функции MKI$, MKS$, MKD$, MKL$

Функции MKI$, MKS$, MKD$ предназначены для преобразования числовых величин в строковые.

Синтаксис:

MKI[MBF]$(<целое выражение>)
MKL[MBF]$(<длинное целое выражение>)
MKS[MBF]$(<выражение обычной точности>)
MKD[MBF]$(<выражение двойной точности>)

Всякое числовое значение, размещаемое в буфере файла прямого доступа при помощи операторов LSET и RSET, должно быть преобразовано в строковое выражение. Функция MKI$ преобразовывает целое число в 2-байтную строку. Функция MKS$ переводит число обычной точности в 4-байтную строку. Функция MKD$ преобразовывает число двойной точности в 8-байтную строку.

Пример:

```
100 FIELD #1,8 AS D$,20 AS N$: LSET D$=MKS$(Z)
120 LSET N$=A$: PUT #1
```

Функция OCT$

Функция OCT$ возвращает строку, которая представляет собой восьмеричное значение десятичного аргумента.

Синтаксис:

OCT$(X)

Перед вычислением OCT$(X) производится округление значения X.

Пример:

```
PRINT OCT$(24)
```

RUN - запуск программы и результат.

30

Функция PEEK

Функция PEEK возвращает байт (целое десятичное число в диапазоне от 0 до 255), прочитанный из ячейки памяти с адресом I.

Синтаксис:

PEEK(I)

Значение I должно находиться в пределах от 0 до 65535. Функция PEEK является дополнительной по отношению к оператору POKE.

Пример:

```
A = PEEK(&H5A00)
```

Функция RIGHT$

Функция возвращает I самых правых символов строки X$. В случае, если I=LEN(X$), возвращается вся строка X$.

Синтаксис:

RIGHT$(X$,I)

Если I=0, функция RIGHT$ возвращает нулевую строку (строку нулевой длины).

Пример:

```
10 A$="1234567890"
20 PRINT RIGHT$(A$,3)
```

RUN - запуск программы и результат.

890

Функция RND

Функция RND возвращает случайное число в диапазоне от 0 до 1.

Синтаксис:

RND[(X)]

Всякий раз при запуске программы на выполнение функция RND будет генерировать одну и ту же последовательность случайных чисел, если только не будет инициирован генератор случайных чисел RANDOMIZE. Если аргумент X имеет отрицательное значение, функция RND будет всегда выдавать одну и ту же последовательность случайных чисел при различных X. Если аргумент X не задан или X>0, будет выполнена генерация следующего случайного числа из последовательности случайных чисел. В случае если X=0, происходит повторение последнего сгенерированного случайного числа.

Пример:

```
10 FOR I=1 TO 5
20 PRINT INT(RND*100);
30 NEXT
```

RUN - запуск программы и результат.

```
24 30 31 51 5
```

Функция SGN

Синтаксис:

SGN(X)

1. Если X>0, функция SGN возвращает 1.
2. Если X=0, функция SGN возвращает 0.
3. Если X<0, функция SGN возвращает -1.

Например, оператор:

```
ON SGN(X)+2 GOTO 100,200,300
```

осуществит переход на строку 100 в случае, если X отрицательное, на строку 200, если X=0 и на строку 300 в случае, если X имеет положительное значение.

Функция SIN

Функция SIN возвращает значение синуса аргумента X, заданного в радианах.

Синтаксис:

SIN(X)

Функция SIN(X) вычисляется с обычной точностью. Кроме того:

COS(X)=SIN(X+3.14159265/2).

Пример:

```
PRINT SIN(1.5)
```

RUN - запуск программы и результат.

```
0.9974951
```

Функция SPACE$

Функция SPACE$ возвращает строку пробелов длиной X.
Синтаксис:

SPACE$(X)
Значение X округляется до целого и должно находиться в пределах от 0 до 255.
Пример:

```
10 FOR I=1 TO 5
20 X$=SPACE$(I)
30 PRINT X$;I
40 NEXT I
```

RUN - запуск программы и результат.

```
 1
  2
   3
    4
     5
```

Функция SPC

Функция SPC выдает на экран дисплея I пробелов.
Синтаксис:

SPC(I)

Эта функция может быть использована только с операторами PRINT и LPRINT. Значение аргумента I должно быть в пределах от 0 до 255.
Пример:

```
PRINT "РАСПОРЯДОК" SPC(5) "ДНЯ"
```

RUN - запуск программы и результат.

```
РАСПОРЯДОК     ДНЯ
```

Функция SQR

Функция SQR возвращает значение квадратного корня аргумента X.
Синтаксис:

SQR(X)

Значение X должно быть положительным.
Пример:

```
10 FOR X=10 TO 25 STEP 5
20 PRINT X, SQR(X)
30 NEXT X
```

RUN - запуск программы и результат.

```
10   3.162278
15   3.872984
20   4.472136
25   5
```

Функция STR$

Функция STR$ возвращает строковое представление значения аргумента X.

Синтаксис:

STR$(X)

Пример:

```
10 INPUT "ВВЕДИТЕ ЧИСЛО";N
20 ON LEN(STR$(N)) GOSUB 100,200,300
```

Функция STRING$

Функция STRING$ возвращает строку длиной I, все символы которой имеют код ASCII, определяемый значением J, или равны первому символу строки X$.

Синтаксис:

STRING$(I,J)
или

STRING$(I,X$)

Пример:

```
10 X$=STRING$(7,45)
20 PRINT X$ "КВАРТАЛЬНЫЙ ОТЧЕТ" X$
```

RUN - запуск программы и результат.

------КВАРТАЛЬНЫЙ ОТЧЕТ-------

Функция TAB

Функция TAB осуществляет переход на I-ую позицию строки дисплея или устройства печати.

Синтаксис:

TAB(I)

Если текущая позиция печати находится дальше I-ой позиции, то функция TAB выполняет переход на I-ую позицию следующей строки. Самая левая позиция строки имеет номер 1. Значение аргумента I должно быть в пределах от 1 до 255. Функцию TAB можно использовать только в операторах PRINT и LPRINT.

Пример:

```
PRINT "ИМЯ" TAB(10) "ОТЧЕСТВО"
```

RUN - запуск программы и результат.

```
ИМЯ         ОТЧЕСТВО
```

Функция TAN

Функция TAN возвращает тангенс аргумента X, заданного в радианах.

Синтаксис:

```
TAN(X)
```

Значение TAN(X) вычисляется с обычной точностью. Если при вычислении функции TAN произошло переполнение, выдается сообщение об ошибке "Overflow", и в качестве результата возвращается максимально возможное число с соответствующим знаком, и выполнение программы продолжается. В Turbo Basic работа программы прекращается.

Пример:

```
10 Y=Q*TAN(X)/2
```

Функция VAL

Функция VAL возвращает числовое значение строки X$.

Синтаксис:

```
VAL(X$)
```

Функция VAL игнорирует лидирующие пробелы, коды табуляции и перевода строки в строке аргумента. Если первым символом строки X$ является символ, отличный от "=", "-", "&" или цифры, то VAL(X$)=0.

Пример:

```
10 X$="-345": PRINT VAL(X$)
```

RUN - запуск программы и результат.

```
-345
```

Функция VARPTR

Синтаксис:

VARPTR (<имя переменной>)

или

VARPTR (#<номер файла>)

или

VARPTR$(<имя переменной>)

Первая форма функции VARPTR возвращает адрес первого байта данных, которые относятся к <имени переменной>. Значение должно быть присвоено <имени переменной> до выполнения функции VARPTR, иначе возникает ошибка "Illegal function call". В качестве аргумента функции можно использовать имя переменной любого типа (числовой, строковой, массива). Возвращаемый адрес представляет собой целое число в диапазоне от 32767 до -32767. В случае если в качестве результата функция VARPTR возвращает отрицательное число, то для того, чтобы получить фактическое значение адреса, следует к этому числу прибавить 65536. Обычно функцию VARPTR используют для получения адреса переменной или массива, после чего он может быть передан подпрограмме на языке ассемблера.

Обращение к функции следующего вида: VARPTR(A(0)) обычно применяется для того, чтобы определить адрес первого элемента массива. Перед обращением к функции VARPTR с целью получения адреса массива необходимо выполнить присваивание значений всем простым переменным, т.к. адреса массивов изменяются всякий раз, когда осуществляется присваивание значения новой простой переменной.

Вторая форма функции VARPTR возвращает начальный адрес буфера, который соответствует файлу с заданным <номером файла>.

Пример:

```
100 X=USR(VARPTR(У))
```

Третья форма функции VARPTR$ возвращает строковое представление адреса переменной. Эта особенность может быть применена для формирования аргументов функций DRAW и PLAY.

ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

В этом разделе описываются дополнительные возможности компилятора языка Basic. К ним относятся операторы и функции, предназначенные для установления режима работы дисплея, работы с графикой и звуковым устройством, и других целей. Следует помнить, что операторы графики PSET, PRESET, POINT, LINE, CIRCLE, GET, PUT, PAINT и DRAW можно использовать только в графическом режиме работы монитора.

Оператор DATE$

Оператор DATE$ используется для установки или изменения текущей даты.

Синтаксис:

DATE$=<строковое выражение>

или

<строковое выражение>=DATE$

где <строковое выражение> представляет собой строковую константу или выражение.

Первая форма оператора DATE$ позволяет устанавливать текущую дату, а вторая форма предназначена для определения текущей даты. Текущая дата определяется и присваивается строковой переменной, если DATE$ входит в выражение, использующееся в операторе LET или PRINT. Если DATE$ находится слева от знака равенства в операторе присваивания, то происходит установка текущей даты.

Если используется первая форма оператора, то строковое выражение может быть представлено в одном из следующих форматов:

1. "ммм-дд-гг"
2. "мм/дд/гг"
3. "мм-дд-гггг"

Пример:

```
10 DATE$="01-01-81"
20 PRINT DATE$
```

RUN - запуск программы и результат.

```
01-01-81
```

При использовании второй формы оператора, переменной присваивается 10 символов в формате

"мм-дд-гггг"

где мм - месяц (от 01 до 12), дд - день (от 01 до 31) и гггг - год (от 1980 до 2099).

Если <строковое выражение> является неправильным, то выдается сообщение об ошибке "Type mismatch", и сохраняется прежнее значение.

Оператор TIME$

Оператор TIME$ используется для установки или изменения текущего времени.

Синтаксис:

TIME$=<строковое выражение>

или

<строковое выражение>=TIME$

где параметр <строковое выражение> - представляет собой строковую константу или переменную.

Первая форма оператора TIME$ применяется для установки текущего времени, а вторая форма - для определения текущего времени. Если TIME$ входит в выражение, использующееся в операторе LET или PRINT, то определяется текущее время и присваивается соответствующей строковой переменной.

При использовании оператора первой формы строковое выражение может быть задано в одном из следующих видов:

1. "чч" - установка часа. Минуты и секунды по умолчанию принимаются равными 00.
2. "чч:мм" - установка часа и минут. Секунды по умолчанию принимаются равными 00.
3. "чч:мм:сс" - установка часа, минут и секунд.

Пример:

```
10 TIME$="08:00"
20 PRINT TIME$
```

RUN - запуск программы и результат.

```
08:00:04
```

При использовании второй формы переменной присваивается 8 символов в формате:

"чч:мм:сс"

где чч - час (от 00 до 23), мм - минуты (от 00 до 59) и сс - секунды (от 00 до 59).

При неправильном задании значений выдается сообщение об ошибке "Illegal Function Call" и сохраняется установленное ранее время.

Оператор COLOR

Оператор COLOR устанавливает цвет для отображения информации, а также цвета фона и границ экрана дисплея. Синтаксис оператора зависит от текущего режима работы дисплея, который устанавливается оператором SCREEN.

В алфавитно-цифровых режимах работы дисплея синтаксис оператора COLOR:

COLOR [<отображение>][,[<фон>][,<границы>]],

где:

1. Параметр <отображение> - целое беззнаковое число от 0 до 31, устанавливающее цвет отображаемых символов.
2. Параметр <фон> - целое беззнаковое число от 0 до 15, определяющее цвет фона.
3. Параметр <границы> - целое беззнаковое число от 0 до 15, устанавливающее цвет границ.

Цвета задаются следующим образом:

1. 0 – черный.
2. 1 – синий.
3. 2 – зеленый.
4. 3 – бирюзовый.
5. 4 – красный.
6. 5 – сиреневый.
7. 6 – коричневый.
8. 7 - светло серый.
9. 8 - темно серый.
10. 9 - светло синий.
11. 10 - светло зеленый.
12. 11 - светло бирюзовый.
13. 12 - светло красный.
14. 13 - светло сиреневый.
15. 14 – желтый.
16. 15 - белый.

Любые значения параметров оператора COLOR, лежащие вне указанных пределов, приводят к ошибке "Illegal Function Call" и сохраняются прежние значения параметров.

Значения параметра отображения в диапазоне от 0 до 15 устанавливают цвет, а значения от 16 до 31 устанавливают соответствующие цвета от 0 до 15 и вводят режим мигания символов.

Любой из параметров оператора COLOR может быть опущен. В этом случае значения не заданных параметров сохраняются прежними.

Пример:

10 COLOR 7,0,0

Устанавливает белый цвет отображения, черный цвет границ и фона.

В графических режимах работы дисплея используются:

1. Цвет отображения - определяется палитрой 0 (зеленый/ красный/ желтый) или палитрой 1 (бирюзовый/ сиреневый/ белый).
2. Цвет фона - один из 16 цветов.
3. Цвет границ - совпадает с цветом фона.

Синтаксис:

COLOR [<C0>][,<C1>],

где:

1. Параметр <C0> - цвет фона и границ.
2. Параметр <C1> - палитра цветов для отображения графической информации - символов на экране.

Если значение C1 четное, то выбирается палитра 0 (зеленый/ красный/ желтый), если нечетное, то палитра 1 (бирюзовый/ сиреневый/ белый). Выбор цвета фона, границ и палитры имеет смысл только для режима со средней разрешающей способностью (цветная графика), так как режим с высокой разрешающей способностью позволяет отображать информацию на экране только в черно-белом режиме.

Попытка использовать оператор COLOR в режиме с высокой разрешающей способностью приводит к ошибке "Illegal Function Call".

Оператор CLS

Оператор CLS предназначен для очистки активной страницы экрана дисплея.

Синтаксис:

CLS [<{0|1|2}>]

Если дисплей в данный момент времени находится в алфавитно-цифровом режиме работы, то активная страница экрана после очистки будет иметь цвет фона.

Если дисплей находится в графическом режиме работы (со средней или высокой разрешающей способностью), то после очистки экран будет черным.

Выполнение операторов SCREEN и WIDTH может привести к очистке экрана, если устанавливаемый режим работы дисплея отличается от текущего режима работы. Совместно с оператором CLS могут быть использованы уточняющие параметры:

1. Параметр CLS 0 - полная очистка экрана (и текст и графика).
2. Параметр CLS 1 - если исполнялся оператор VIEW, то графическое окно стирается, иначе стирается весь экран.
3. Параметр CLS 2 - стирается только текстовое окно. Последняя строка экрана не изменяется.

Функции CSRLIN и POS

Функция CSRLIN возвращает номер строки, в которой располагается курсор.

Синтаксис:

X=CSRLIN,

где X - числовая переменная.

Возвращаемое значение находится в диапазоне от 1 до 24. Функция X=POS(0) возвращает номер колонки в строке, где располагается курсор. В зависимости от установленной ширины экрана, возвращаемое функцией POS значение находится в диапазоне от 1 до 40 или от 1 до 80.

Оператор LOCATE

Оператор LOCATE перемещает курсор в указанную позицию на экране дисплея.

Синтаксис:

LOCATE [<ряд>] [,[<колонка>] [,[<курсор>] [,[<старт>] [,<стоп>]]]],

где:

1. Параметр <ряд> - номер строки экрана - целое выражение, значение которого должно быть в диапазоне от 1 до 24.
2. <колонка> - номер колонки экрана - целое выражение, значение которого в зависимости от ширины экрана должно быть в диапазоне от 1 до 40 или от 1 до 80.
3. <курсор> - логическая величина, определяющая, будет ли курсор видимым или нет. Если эта величина нулевая, то курсор не отображается на экране дисплея.
4. <старт> - номер строки знакоместа, с которой будет начинаться отображение курсора - целое выражение со значением в диапазоне от нуля до 7.
5. <стоп> - номер строки знакоместа, на которой будет заканчиваться отображение курсора - целое выражение, значение которого должно быть в диапазоне от нуля до 7.

Оператор LOCATE перемещает курсор в указанную позицию, а последующие операторы PRINT будут выводить на экран символы, начиная с этого места.

Любой из параметров может быть опущен. Тогда значения этих параметров остаются прежними. Если задан параметр "старт", а параметр "стоп" нет, то предполагается, что значение отсутствующего параметра равно значению параметра "старт". Мигание курсора происходит с частотой 16 раз в секунду.

Если значения параметров оператора LOCATE выходят за пределы указанных диапазонов, то выдается сообщение "Illegal Function Call".

Оператор SCREEN

Оператор SCREEN устанавливает режим работы дисплея компьютера.

Синтаксис:

SCREEN [<режим>][,[<отображение>][,[<страница 1>] [,<страница 2>]]],

где:

1. Параметр <режим> - числовое выражение, значение которого должно быть равно 0, 1 или 2. Эти значения определяют режим работы дисплея:

1) 0 - алфавитно-цифровой режим с текущей шириной экрана (40 или 80 символов).
2) 1 - графический режим работы со средней разрешающей способностью 320x200 точек (цветная графика).
3) 2 - графический режим работы с высокой разрешающей способностью 640x350 точек.

2. <отображение> - числовое выражение. Если его значение равно нулю, то цвета подавляются, и информация отображается в черно - белом режиме. Ненулевое значение разрешает цветное отображение.

3. <страница 1> - используется только в алфавитно-цифровом режиме. Числовое выражение должно быть целым числом в диапазоне от нуля до 7 для экрана шириной в 40 символов или в диапазоне от нуля до 3 для экрана шириной в 80 символов. Этот параметр выбирает страницу экрана, на которую осуществляется запись информации.

4. <страница 2> - используется только в алфавитно-цифровом режиме.

Если все параметры заданы правильно, то устанавливается новый режим работы дисплея и происходит очистка экрана. В качестве цвета отображения выбирается белый цвет, а в качестве цвета фона и границ экрана - черный цвет.

Если установленный оператором SCREEN режим работы дисплея совпадает со старым, то ничего не происходит. Если значения параметров оператора SCREEN выходят за указанные границы, то выдается сообщение "Illegal Function Call", и сохраняются прежние значения.

Функция SCREEN

Функция SCREEN возвращает код символа, расположенного по указанному месту на экране дисплея.

Синтаксис:

SCREEN (<строка>,<колонка><[,Z]>),

где:

1. <строка> - числовое выражение, значение которого должно быть целым беззнаковым числом в диапазоне от 1 до 24.
2. <колонка> - числовое выражение, значение которого должно быть целым беззнаковым числом в диапазоне от 1 до 40 или от 1 до 80 в зависимости от ширины экрана.
3. Параметр <Z> - числовое выражение.

Если задан параметр Z и его значение не нулевое, то функция SCREEN возвращает атрибут цвета символа, расположенного по указанному месту на экране дисплея.

Если значения параметров функции SCREEN выходят за указанные пределы, выдается сообщение "Illegal Function Call".

Пример:

```
100 X=SCREEN(10,10)
```

Если в десятой строке и десятой колонке экрана размещается символ A, то X будет равен 65.

В следующем примере функция SCREEN возвратит атрибут цвета того же символа:

```
110 X=SCREEN(10,10,1)
```

Оператор PALETTE

Оператор PALETTE выбирает палитру цветов.

Синтаксис:

PALETTE [<атрибут>,<цвет>]

или

PALETTE USING <имя массива> [(<индекс>)]

где:

1. Параметр <атрибут> - определяет номер палитры - 1 (темная) или 2 (светлая).
2. Параметр <цвет> - номер цвета. Для VGA и MGA мониторов требуется использовать длинное целое число при режимах SCREEN 11-13.
3. Параметр <имя массива> - имя массива содержащего номера цветов.
4. Параметр <индекс> - определяет элемент массива цветов для установки палитры.

Оператор KEY

Оператор KEY позволяет присвоить одной или всем десяти функциональным клавишам 15-ти байтную строку. При нажатии клавиши эта строка вводится в Basic.

Несколько вариантов синтаксиса:

1. KEY <номер клавиши>, <строковое выражение>
2. KEY LIST
3. KEY ON
4. KEY OFF
5. KEY(n) <{ON | OFF | STOP}>

Первая форма оператора KEY позволяет присвоить строковое выражение указанной функциональной клавише. <Строковое выражение> может иметь длину от одного до 15 символов. Если его длина больше, символы после пятнадцатого игнорируются.

Оператор KEY ON распечатывает в 25-ой строке экрана дисплея значения клавиш. Когда ширина экрана равняется 40 символам, на экране распечатываются значения пяти клавиш. При ширине экрана в 80 символов распечатываются значения всех десяти клавиш.

При использовании оператора KEY OFF удаляется 25-ая строка на экране дисплея.

Оператор KEY LIST распечатывает на экране значения (по 15 символов) всех десяти функциональных клавиш.

Если значение выражения <номер клавиши> выходит за пределы диапазона от 1 до 10, то выдается сообщение "Illegal Function Call". При этом сохраняются прежние значения функциональных клавиш.

Примеры:

1. KEY(n) ON - задействует функциональную клавишу n.
2. KEY(n) OFF - исключает заданную клавишу.
3. KEY(n) STOP - приостанавливает реакцию на клавишу n до ее повторной активизации по KEY(n) ON.

При этом все нажатия этой клавиши в состоянии приостановки запоминаются и становятся последовательно доступными при активизации. Здесь n - условный номер клавиши:

1. 1-10 клавиши F1-F10.
2. 11 клавиша UP.
3. 12 клавиша LEFT.
4. 13 клавиша RIGHT.
5. 14 клавиша DOWN.
6. 15-25 клавиши определяемые пользователем.
7. 30-31 клавиши F11-F12 на 101-клавишной клавиатуре.

Оператор OPEN

В этом параграфе описывается вторая возможная конструкция оператора OPEN. Оператор OPEN устанавливает соответствие между физическим устройством и буфером ввода/вывода.

Синтаксис:

OPEN [<устройство>]<имя файла>[FOR<режим>] AS [#] <номер файла> [LEN=<длина записи>]

где:

1. Параметр <устройство> - необязательная часть строки и определяет имя устройства. Им может быть:

1) A: или B: - имя дискового устройства.
2) KYBD: - клавиатура (только ввод).
3) SCRN: экран (только вывод).
4) LPT1: - устройство печати.

2. Параметр <имя файла> - строковая константа, которая может содержать <устройство>.

3. Параметр <режим> - задает режим использования файла, позиционирование указателя в файле и определяет действия, которые следует выполнить, если такого файла не существует. Могут использоваться следующие режимы:

1) INPUT - размещение указателя в начале файла. Если указанного файла нет, то выдается сообщение "File not found".
2) OUTPUT - размещение указателя в начале файла. Если файл не существует, то он создается.
3) APPEND - размещение указателя в конце файла. Если указанного файла нет, то он создается. Если опция FOR <режим> не задана, то осуществляется размещение указателя в начале файла.

4. Параметр <номер файла> - целое выражение, значение которого должно быть числом в диапазоне от 1 до 15. Это число используется для установления соответствия между буфером ввода/вывода и файлом на диске или устройством. Это соответствие будет сохраняться до тех пор, пока не выполнится оператор CLOSE.

5. Параметр <длина записи> - целое выражение в диапазоне от 2 до 32768. Значение выражения устанавливает длину записи, используемую для файлов с прямым доступом. По умолчанию длина записи принимается равной 128 байтам.

Если файл открыт для использования в режиме APPEND, то номер записи устанавливается равным последней записи в файле. При попытке записи в файл, открытый в режиме INPUT, выдается сообщение об ошибке "Bad File Mode".

При попытке чтения из файла, открытого в режиме OUTPUT, выдается сообщение об ошибке "Bad File Mode". Если

значения параметров оператора OPEN выходят за указанные пределы, выдается сообщение об ошибке "Illegal Function Call".

Пример:

```
10 OPEN "PARTS.DAT" AS #1
```

В приведенном примере, в текущем каталоге открыт файл прямого доступа PARTS.DAT и ему присвоен номер 1. Другие примеры использования оператора OPEN:

```
10 OPEN "B:INVENT.DAT" FOR APPEND AS #1
20 OPEN "EXDATA" FOR OUTPUT AS #2
```

Оператор PSET

Оператор PSET предназначен для отображения точки на экране дисплея.

Синтаксис:

PSET (<X-координата,Y-координата>)[,<атрибут>]

(X-координата, Y-координата) - обозначает координаты отображаемой точки и может задаваться в одной из следующих форм:

PSET (<X-смещение, Y-смещение>)

или

PSET (<X-абсолютный, Y-абсолютный>)

где <атрибут> в операторе PSET - определяет цвет отображаемой точки. Его значение может равняться 0, 1, 2 или 3. Нулевое значение соответствует цвету фона, а при других значениях выбирается соответствующий цвет текущей палитры. По умолчанию значение атрибута в режиме со средней разрешающей способностью принимается равным 3, а в режиме с высокой разрешающей способностью равным 1.

При использовании первой формы оператора задаются относительные координаты точки, отсчитываемые от последней

точки, к которой было обращение. Вторая форма задает абсолютные координаты точки. Верхний левый угол экрана дисплея имеет координаты (0,0). Значения координат точек по вертикали обычно лежат в диапазоне от 0 до 199. В графическом режиме со средней разрешающей способностью значения координат по горизонтали (X) находятся в диапазоне от 0 до 319, а в режиме с высокой разрешающей способностью - в диапазоне от 0 до 639.

В следующем примере сначала рисуется, а затем удаляется диагональ от точки с координатами (0,0) до точки с координатами (100,100):

```
10 FOR I=0 TO 100
20 PSET (I,I)
30 NEXT
40 FOR I=100 T0 0 STEP -1
50 PSET (I,I),0
60 NEXT
```

Функция POINT

Функция POINT возвращает значение атрибута цвета указанной точки.

Синтаксис:

POINT(<X-координата,Y-координата>)

или

POINT(<число>)

Здесь <число> - определяет условия задания координат точки:

1. 0 - текущая физическая x - координата
2. 1 - текущая физическая y-координата
3. 2 - текущая логическая x-координата
4. 3 - текущая логическая y-координата

Если значения координат точки превышают размеры экрана, то функция POINT возвращает -1. В режиме со средней разре-

шающей способностью возвращаемые значения равны 0, 1, 2, 3, а в режиме с высокой разрешающей способностью - 0, 1.

Оператор LINE

Оператор LINE позволяет отображать на экране дисплея группу точек.

Синтаксис:

LINE [STEP] [(<X1,Y1>)]-(<X2,Y2>) [,[<атрибут>] [,b[f]], [,<стиль>]]

Простейшая форма оператора LINE:

LINE (<X2,Y2>)

Этот оператор обрисовывает линию, начиная от последней отображенной точки до точки с координатами (X2,Y2).

В операторе LINE можно указать и координаты начальной точки: LINE (X1,Y1)-(X2,Y2). Атрибут в операторе LINE задает цвет отображаемых точек. Параметр b указывает, что на экране будет обрисовываться прямоугольник с координатами противоположных углов (X1,Y1) и (X2,Y2). Таким образом, действие оператора:

LINE (X1,Y1)-(X2,Y2),,b

аналогично действию следующих четырех операторов:

LINE(X1,Y1)-(X2,Y1)
LINE(X1,Y1)-(X1,Y2)
LINE(X2,Y1)-(X2,Y2)
LINE(X1,Y2)-(X2,Y2)

Если кроме параметра b задан параметр f, то это означает, что прямоугольник будет закрашен цветом, определяемым атрибутом <стиль>. Параметр STEP определяет обрисовку относительно координат последней выведенной точки.

Пример:

```
LINE(0,0)-(100,100),2,bf
```

Когда значения координат выходят за пределы допустимых для точек экрана значений, то они принимаются равными ближайшему допустимому значению. Так, например, отрицательные значения принимаются равными нулю.

Оператор CIRCLE

Оператор CIRCLE предназначен для обрисовки на экране дисплея эллипса, координаты центра и радиус которого задаются его первыми параметрами.

Синтаксис:

CIRCLE (<X центр,Y центр>), <радиус> [, <атрибут> [, <начало>, <конец> [,<отношение>]]]

где:

1. Параметры <начало> и <конец> - задают углы в радианах (от 0 до 2□). С помощью этих параметров задается та часть эллипса, которая будет отображаться на экране. Если значения параметров отрицательные, то точки начала и конца обрисовки эллипса будут соединены с центром эллипса линиями, а значения этих параметров будут восприниматься, как положительные.
2. Параметр <отношение> - определяет отношение полуоси X к полуоси Y. По умолчанию это отношение принимается равным 5/6 для режима с цветной графикой или 5/12 для режима с черно-белой графикой. Эти величины дают изображение окружности при стандартном отношении масштабов сторон экрана 4/3. Если отношение меньше единицы, то радиус измеряется количеством точек по оси X. Если же отношение больше единицы, то радиус задается в точках по оси Y.

Оператор GET

В этом параграфе описывается вторая возможная конструкция оператора GET. Оператор GET предназначен для записи изображения экрана в указанный массив.

Синтаксис:

GET [STEP] (<X1,Y1>) - [STEP] (<X2,Y2>), <имя массива> (<индекс>)

где параметр <индекс> - определяет элемент массива, начиная с которого будет сохраняться изображение экрана.

Операторы GET и PUT используются для быстрого перемещения графических фигур по экрану.

Прямоугольник задается координатами противоположных углов (X1, Y1) и (X2, Y2). Тип массива может быть любым, кроме строкового. STEP определяет координаты относительно последней выведенной точки.

Оператор PUT

Здесь описывается вторая возможная конструкция оператора PUT. Оператор PUT предназначен для записи на экран дисплея изображения, хранящегося в указанном массиве.

Синтаксис:

PUT [STEP](<X1,Y1>), <имя массива> [, <действие>]

где:

1. (X1,Y1) - координаты точки экрана, которая будет соответствовать верхнему левому углу передаваемого изображения.
2. STEP - определяет относительные координаты после последней выведенной точки.
3. <имя массива> - имя числового массива, содержащего предназначенную для передачи информацию.
4. Параметр <действие> - определяет одну из следующих операций: PSET, PRESET, XOR, OR, AND. При записи изображения на экран с помощью оператора PUT учитывается текущее состояние точек экрана.

Если в качестве параметра <действие> выбрана операция PSET, то данные просто переписываются из массива на экран, и в этом случае оператор PUT будет по своим функциям обратным оператору GET.

При использовании PRESET формируется негативное изображение. Если в качестве параметра <действие> применяется

XOR, OR или AND, то выполняется соответствующая логическая операция над передаваемым изображением и текущим состоянием экрана. То есть, операндами этих операций являются атрибуты цвета налагаемых точек (значением атрибута может быть число 0, 1, 2 или 3).

Оператор PCOPY

Оператор PCOPY копирует одну экранную страницу в другую.

Синтаксис:

PCOPY <страница-источник>,<страница-цель>

Страницы определяются их номерами, которые лежат в диапазоне от 0 до n. Параметр n зависит от конкретного типа дисплейного адаптера (видеопамяти) и выбранного размера страницы.

Оператор VIEW

Синтаксис:

VIEW [[SCREEN](<X1,Y1>)-(<X2,Y2>)[,[<цвет>][,<цвет границы>]]]

или

VIEW PRINT [<номер верхней строки> TO <номер нижней строки>]

Первая форма оператора позволяет получить графическое прямоугольное окно с физическими размерами, определяемыми координатами верхнего левого угла (X1,У1) и нижнего левого - (X2,У2). Если задана опция SCREEN, то в созданном окне будут сохранены результаты предыдущих графических операторов (естественно в пределах графического окна), хотя все координаты точек были определены для абсолютных размеров экрана. Задание параметров <цвет> и <цвет границ> обеспечит закраску окна и обрисовку границ соответственно. Результаты работы

всех последующих графических операторов будут выводится в графическом окне.

Вторая форма оператора используется для определения текстового прямоугольного окна, представляющего собой область физического экрана заключенную между строками с указанными номерами. Вывод текста операторами PRINT будет производится в это окно.

Оператор WINDOW

Оператор WINDOW определяет логические окна (координаты прямоугольной области) внутри текущего графического окна.

Синтаксис:

WINDOW [[SCREEN] (<X1,Y1>)-(<X2,Y2>)]

Числа (X1,У1) и (X2,У2) определяют координаты левого нижнего и правого верхнего углов окна соответственно. Опция SCREEN означает, что значения координаты Y возрастают от верхней границы графического окна к нижней. Отсутствие этой опции означает возрастание координаты Y от нижней границы к верхней.

Оператор PAINT

Оператор PAINT предназначен для закрашивания произвольной графической фигуры заданным цветом.

Синтаксис:

PAINT [STEP] (<X,Y>)[,<атрибут> [,<граница>], [<фон>]],

где:

1. Параметры X,Y - координаты точки внутри области, которую необходимо закрасить.
2. Параметр STEP - координаты относительно последней точки.
3. Параметр <атрибут> - цвет, которым должна быть закрашена область.

4. Параметр <граница> - цвет, определяющий контур закрашиваемой фигуры.
5. Параметр <фон> - строковое значение, используемое при закраске уже окрашенного пространства.

Точка координаты, которую задают в операторе PAINT, должна находиться внутри закрашиваемой фигуры. Если параметр <атрибут> не задан, то его значение по умолчанию принимается равным 3 (для режима со средней разрешающей способностью) или 1 (для режима с высокой разрешающей способностью). Значение параметра <граница> по умолчанию равно значению параметра <атрибут>.

Пример:

```
10 SCREEN 1
20 LINE(0,0)-(100,150),2,B
50 PAINT(50,50),1,2.
```

Оператор DRAW

Оператор DRAW предназначен для обрисовки графической информации.

Синтаксис:

DRAW <строковое выражение>

В <строковом выражении> задаются команды, предназначенные для обрисовки графики. Команды перемещения в качестве начальной точки принимают последнюю отображенную точку. Первоначально такой точкой является центр экрана (160,100 или 320,100 в зависимости от режима работы дисплея). К командам перемещения относятся:

1. U[<n>] - перемещение вверх.
2. D[<n>] - перемещение вниз.
3. L[<n>] - перемещение влево.
4. R[<n>] - перемещение вправо.
5. E[<n>] - перемещение по диагонали вверх и направо.
6. H[<n>] - перемещение по диагонали вверх и налево.
7. G[<n>] - перемещение по диагонали вниз и налево.

8. F[<n>] - перемещение по диагонали вниз и направо.

Здесь <n> - расстояние, на которое осуществляется перемещение. Количество точек, на которое производится перемещение, определяется, как произведение n на масштабный коэффициент (задается командой S).

Команда M(X,Y) задает абсолютное или относительное перемещение. Если X предшествует знак "+" или "-", это означает, что задано относительное перемещение. В противном случае задано перемещение по абсолютным значениям координат.

Любой из перечисленных ниже команд может предшествовать один из следующих префиксов:

1. B - перемещение без обрисовки линии.
2. N - перемещение с возвратом в исходное положение.
3. A<n> - задание поворота на угол <n>, где n - целое число в диапазоне от 0 до 3 (0 - 0, 1 - 90, 2 - 180, 3 - 270 градусов).
4. C<n> - задает атрибут цвета, где n может принимать целые значения от 0 до 3 в режиме со средней разрешающей способностью или 0, 1 в режиме с высокой разрешающей способностью.
5. S<n> - задает масштабный коэффициент <n>, где n может принимать целые значения в диапазоне от 1 до 255.
6. V - использование логических координат.
7. X<строка> - выполнение подстроки. Эта команда позволяет выполнить строку внутри другой строки (подобно оператору GOSUB).

Числовой аргумент <n> может быть числовой константой или последовательностью символов "<имя переменной>.

Пример программы обрисовки прямоугольника:

```
10 SCREEN 1
20 A=20
30 DRAW "U=A;R=A;D=A;L=A;"
```

Для обрисовки треугольника можно воспользоваться следующими операторами:

```
10 SCREEN 1
```

```
20 DRAW "E15F15L30"
```

Оператор BEEP

Оператор BEEP позволяет генерировать звуковой сигнал частотой 800 Гц и длительностью 1/4 секунды.

Синтаксис:

```
BEEP
```

Пример:

```
IF X<20 THEN BEEP
```

Оператор SOUND

Оператор SOUND предназначен для генерации звуковых сигналов.

Синтаксис:

```
 SOUND <частота, длительность,>
```

где:

1. Параметр <частота> - частота в герцах, представляющее собой целое выражение, значение которого должно быть в диапазоне от 37 до 32767.
2. Параметр <длительность> - длительность в тактах, представляет собой целое выражение, значение которого должно быть в диапазоне от 0 до 65535.

В одной секунде 18.2 такта. Если задана нулевая длительность, то любой выполняемый оператор SOUND прекращает свою работу.

Пример:

```
SOUND RND*1000+37,2.
```

Оператор PLAY

Оператор PLAY предоставляет дополнительные возможности в использовании встроенного в ЭВМ звукового устройства.

Синтаксис:

PLAY <строковое выражение>

<Строковое выражение> состоит из следующих обозначающих команды символов:

1. A-G[#|+,-] - играть ноту. Символ "#" или "+" после буквы, определяющей ноту, означает диез, а символ "-" означает бемоль.
2. L<n> - длительность. Устанавливает длительность каждой ноты. L4 означает четверть ноты, L1 - целую ноту и т.д. Значение n должно быть в пределах от 1 до 64. Длительность может также следовать за определенной нотой, в случае, если необходимо изменить длительность только этой ноты. В этом случае A16 эквивалентно L16A.
3. MF - означает, что каждая последующая нота или звук, генерируемые операторами PLAY или SOUND, не будут звучать до тех пор, пока не закончится звучание предыдущей ноты или звука. Этот режим устанавливается по умолчанию.
4. MB - означает, что каждая нота или звук помещается в буфер, позволяя продолжать выполнение программы во время звучания музыки.

В буфер можно поместить до 32 нот (или пауз):

1. MN - каждая нота играется 7/8 времени, заданного числом L (длительность).
2. ML - каждая нота звучит полный период времени, заданный числом L.
3. MS - каждая нота играется 3/4 времени, заданного L.
4. N<n> - играть ноту n. Значение n должно быть в диапазоне от 0 до 84. В семи возможных октавах существует 84 ноты, причем n=0 означает отсутствие звучания.
5. O<n> - октава. Устанавливает текущую октаву. Существует 7 октав (0...6).

6. P<n> - пауза. Значение n может изменяться от 1 до 64.
7. T<n> - темп. Устанавливает число четвертей ноты (L4) в секунду. Значение n может изменяться от 32 до 255. По умолчанию оно равняется 120.
8. X<строка> - выполнить подстроку.

Точка после каждой ноты приводит к тому, что нота будет играться 3/2 периода времени, определяемого произведением L (длительность) и T (темп). За нотой может следовать многоточие. При этом период времени звучания ноты масштабируется соответствующим образом. Так A. приводит к увеличению периода звучания в 3/2 раза; A.. - в 9/4 раза, A... - в 27/8 раза и т.д.

Точки могут также помещаться после P (пауза) и при этом длительность паузы масштабируется, как описано выше. Числовой аргумент <n> может быть числовой константой или последовательностью символов "<имя переменной>.

Пример:

```
10 A$="BB-C"
20 B$="O4XA$;"
30 C$="L1CT50N3N4N5N6"
40 PLAY"P2XA$;XB$;XC$;"
```

Приведем и некоторые другие варианты записи оператора:

1. PLAY(n) - возвращает количество нот в музыкальной очереди.
2. PLAY ON - разрешает воспроизведение музыки.
3. PLAY OFF - запрещает воспроизведение музыки.
4. PLAY STOP- приостанавливает воспроизведение музыки.
5. ON PLAY (n) GOSUB <метка> - вызывает переход на метку, когда воспроизведены ноты из буфера с n по n-1.

ПРИМЕРЫ ПРОГРАММ

Вычисление значений функции

Выберем в качестве функции простое выражение:

$f(x)=ax+b$

где a и b параметры, определяющие вид кривой этой функции.

Выполним вычисление значений функции в 11 точках, включая нулевую на интервале 0 - 1. Примем для параметров следующие значения: a=2 и b=3.

Ниже приведен текст программы для вычисления такой функции. Операторы программы даны заглавными буквами, а пояснения к ним - строчными.

```
REM ---- Программа для вычисления значений функции ---

CLS  - оператор очистки экрана.
DEFDBL A-Z  - оператор двойной точности.
DIM V(100)  - оператор массива.

B=1  - конечное значение интервал.
A=0  - начальное значение интервал.
N=100  - число шагов.
H=(B-A)/N  - величина шага.
AA=2  - величина a.
BB=3  - величина b.

S=0
FOR I=0 TO N  - оператор цикла.
X=H*I  - текущее значение величины x.
V(I)=AA*X+BB  - текущее значение функции.
PRINT X,V(I)  - оператор печати.
NEXT  - конец цикла.
END  - конец программы.
```

В результате работы программы на экран выводится два столбика - в первом величина аргумента, во втором самой функции:

```
0.0   3.0
0.1   3.2
0.2   3.4
0.3   3.6
0.4   3.8
0.5   4.0
0.6   4.2
0.7   4.4
0.8   4.6
0.9   4.8
1.0   5.0
```

Не трудно проверить, что получены правильные значения функции. В частности, при x=0 функция равна 3, при x=1 функция равна 5.

Вычисление суммы значений функции

Процедура вычисления суммы значений функции в N точках записывается виде:

$$S = \sum_{i=0}^{N} f(x_i) \ ,$$

где N число шагов при вычислении суммы, $f(x_i)$ - значения функции в точке i и $x_i = h*i$ – величина аргумента в точке i. Шаг вычисления h определяется в виде (b-a)/N, где a и b верхняя и нижняя границы интервала вычисления функции.

В качестве функции выберем Sin(x) и будем считать сумму на интервале 0 - 1 с числом шагов 101, включая ноль. Ниже приведен текст программы для нахождения такой суммы:

```
REM ------ Программа для вычисления суммы значений функции ------
```

```
CLS
DEFDBL A-Z
DIM V(100)

B=1  - конечное значение интервала.
A=0  - начальное значение интервала.
N=100  - число шагов.
H=(B-A)/N  - шаг вычисления.

S=0
FOR I=0 TO N
X=H*I
V(I)=SIN(X)
S=S+V(I)
NEXT

PRINT S

END
```

В результате вычислений суммы получим величину:

46.39012182353973.

Запись и считывание информации на диске

Запись и считывание информации производится оператором OPEN с различными параметрам, определяющими его режим работы. Ниже приведена программа, в которой проводится вычисление массива значений функции с последующей записью в файл на диск и считывание этого массива с последующим выводом его на экран. В качестве функции выбран Sin(x), а вычисления проводятся в 11 точках на интервале от 0 до π.

```
REM ------ Программа записи и считывания с диска ------
DEFDBL A-Z
DIM U(100), V(100)
CLS
```

```
G$="C:\BASICA\TB\FAIL.BAS"

A=0  - начальное значение интервала.
B=3.141592653589793  - конечное значение интервала.
N=10  - число шагов.
H=(B-A)/N  - шаг вычисления.

OPEN "O",1,G$

FOR L=0 TO N
R=L*H
U(L)=SIN(R)
PRINT#1, USING " +#.######^^^^  ";R,U(L)
NEXT
CLOSE

OPEN "I",1,G$

FOR I=0 TO N
INPUT#1, R,V(I)
PRINT USING "  +#.######^^^^  ";R,V(I)
NEXT

END
```

В результате на экран выводятся значения аргумента (левый столбец) и функции, которые были считаны с диска:

```
+0.000000E+00   +0.000000E+00
+3.141593E-01   +3.090170E-01
+6.283185E-01   +5.877853E-01
+9.424778E-01   +8.090170E-01
+1.256637E+00    +9.510565E-01
+1.570796E+00   +1.000000E+00
+1.884956E+00    +9.510565E-01
+2.199115E+00    +8.090170E-01
+2.513274E+00    +5.877853E-01
+2.827433E+00    +3.090170E-01
+3.141593E+00    +8.979318E-11
```

Не сложно проверить, что величина функции при R= π/2 = 1.570796E+00 равна 1, а при R= π = 3.141593E+00 должна быть равна нулю. Последнее условие выполняется примерно с точностью 10^{-10}.

Запись числа в виде 3.141593E+00 называется экспоненциальной формой и такое число равно 3.141593 10^0, а число 3.090170E-01 равно 3.090170 10^{-1} и т.д.

Вычисление определенного интеграла

Рассмотрим теперь метод и программу для вычисления интеграла функции, заданной массивом переменных в N точках с использованием метода Симпсона. Такой интеграл может быть представлен в виде суммы вида:

$$\int_a^b f(x)dx = \frac{b-a}{3N}\left[\{f(x_0)+f(x_N)\}+4\sum_{i=1}^{N-1} f(x_i)+2\sum_{i=2}^{N-2} f(x_i)\right],$$

где:

1. Величена *N* - число шагов интегрирования, а b и a - пределы интервала интегрирования.
2.
3. Запись $f(x_i)$ - обозначает значения функции на i-том шаге, т.е. при $x_i = hi$. Здесь *h* - шаг интегрирования, который определяется в виде $h=(b-a)/N$.

Первая сумма выражения вычисляется по нечетным, а вторая по четным значениям i. В качестве функции выбран $2Sin^2(x)$ на интервале 0 - π Вычисления проводятся в 101 точке, т.е. *N* =100.

Значение этого интеграла известно и равно π = 3,141592653589793.

Программа наглядно демонстрирует, как можно повысить точность вычислений при использовании оператора двойной точности. В этом случае вполне можно получить результат с

точностью до 16 значащих чисел, т.е. до 15 чисел после запятой при не очень большом значении *N*. А выключение этого оператора приводит к точности только в 7 значащих чисел.

```
REM ------ Программа для вычисления интеграла методом
Симпсона ------

CLS   - оператор очистки экрана перед выдачей результатов.
DEFDBL A-Z   - оператор задания двойной точности.
DIM V(1000)   - оператор задания размерности массива.

PI=3.141592653589793   - присвоение значения числу □.
N=100   - определение числа шагов интегрирования.
H=(PI-0)/N   - вычисление шага интегрирования.

FOR I=0 TO N   - цикл для вычисления значений функции.
 X=H*I   - вычисление значений аргумента.
V(I)=2*(SIN(X))^2   - вычисление значений функции.
NEXT   - конец цикла.

A=0; B=0   - обнуление начальных значений переменных.
FOR II=1 TO N-1 STEP 2   - цикл вычисления первой суммы.
B=B+V(II)
NEXT II   - конец цикла.

FOR JJ=2 TO N-2 STEP 2   - цикл вычисления второй суммы
A=A+V(JJ)
NEXT JJ   - конец цикла.

SINT=H*(V(0)+V(N)+2*A+4*B)/3 - окончательное нахождение интеграла.
PRINT SINT - печать результата.

END - конец программы.
```

Если вычисления проводить с двойной точностью, то в результате получим значение: 3.141592653589793, что полностью совпадает с приведенной выше величиной π. Если же убрать оператор двойной точности, то результат будет: 3.141592502593994. Тем самым видно, что совпадение наблюдается только у 6 чисел после запятой.

Конечно, на точность результатов влияет и величина N, но выбранное значение N и применение оператора DEFDBL позволяют правильно передать до 15 чисел после запятой.

Заметим, что без оператора двойной точности DEFDBL даже при N=1000 получаем: 3.141592741012573, т.е. и в этом случае правильно передаются только 6 знаков.

Вычисление произведения двух матриц

Для нахождения матричных элементов произведения квадратных матриц размерности N

A=BC

используется математическое выражение:

$$a_{ij} = \sum_{k=1}^{N} b_{ik} c_{kj}$$

где a_{ij}, b_{ik} и c_{kj} - матричные элементы.

Программа вычисления такого произведения приведена ниже:

```
REM ----- Программа вычисления произведения матриц -----

DEFDBL A-Z
DIM  A(20,20), B(20,20),C(20,20)

B(1,1)=1: B(1,2)=2: B(1,3)=3: B(1,4)=4
B(2,1)=2: B(2,2)=2: B(2,3)=2: B(2,4)=2
B(3,1)=3: B(3,2)=2: B(3,3)=3: B(3,4)=2
B(4,1)=4: B(4,2)=2: B(4,3)=2: B(4,4)=4

C(1,1)=1: C(1,2)=3: C(1,3)=5: C(1,4)=7
C(2,1)=3: C(2,2)=2: C(2,3)=2: C(2,4)=2
C(3,1)=5: C(3,2)=2: C(3,3)=3: C(3,4)=3
C(4,1)=7: C(4,2)=2: C(4,3)=3: C(4,4)=4
```

```
N=4

FOR I=1 TO N
FOR J=1 TO N
FOR K=1 TO N
A(I,J)= A(I,J)+B(I,J)*C(K,J)
NEXT K
NEXT J
NEXT I

PRINT "МАТРИЦА B"
FOR I=1 TO N
FOR J=1 TO N
PRINT B(I,J)
NEXT
NEXT

PRINT "МАТРИЦА C"
FOR I=1 TO N
FOR J=1 TO N
PRINT C(I,J)
NEXT
NEXT

PRINT "МАТРИЦА A"
FOR I=1 TO N
FOR J=1 TO N
PRINT A(I,J)
NEXT J
NEXT I
END
```

В результате работы программы получаем искомую матрицу:

```
МАТРИЦА B

 1  2  3  4
 2  2  2  2
 3  2  3  2
```

```
 4  2  2  4

МАТРИЦА C

 1  3  5  7
 3  2  2  2
 5  2  3  3
 7  2  3  4

МАТРИЦА A

 50  21  30  36
 32  18  26  32
 38  23  34  42
 48  28  42  54
```

Результат можно проверить вручную, используя известные правила - элемент, стоящий на I строке и J столбце искомой матрицы равен сумме произведения соответствующих элементов I строки матрицы B и элементов J столбца матрицы C.

Поиск корня функции

Корнем функции называется величина ее аргумента, при котором функция равна нулю. Возьмем простую функцию Cos(x) - известно, что ее первый корень при x>0 находится при x=π/2.

Равенство нулю какой-то величины в численной математике означает, что эта величина меньше заранее заданного числа π - точности вычисления корня. Ниже приведена программа для нахождения корня такой функции. На печать выводится не x, а величина 2x, равная $\tilde{\pi}$ Это позволяет легко сравнить полученную в результате расчетов величину с известным значение числа π, которое приводится в одном из предыдущих примеров и равно 3.141592653589793.

Пределы поиска корня (SKN - верхний и SKV - нижний) задаются в радианах. Начальный шаг поиска - HC выбран произвольно, поскольку его величина практически не влияет на результаты. Точность вычислений - EP задана максимальная даже для режима двойной точности.

```
REM -----Программа поиска корня функции -----

DEFDBL A-Z
CLS

SKN=0  - нижний предел поиска.
SKV=2  - верхний предел поиска.
HC=.01  - шаг поиска.
EP=1.0E-16  - точность поиска корня.

A2=SKN
DK=A2
GOSUB 1000
D12=DD

B2=A2+HC
51 DK=B2

GOSUB 1000
D11=DD

IF D12*D11>0 GOTO 4
3 A3=A2
B3=B2
11 C3=(A3+B3)/2
IF (ABS(A3-B3))<EP GOTO 151

DK=C3
GOSUB 1000
F2=DD

IF D12*F2>0 GOTO 14
B3=C3
D11=F2
GOTO 15
4 A3=C3
D12=F2
15 IF ABS(F2)>EP GOTO 11
```

```
151 CO=C3

GOTO 7
4 IF ABS(D11*D12)<EP GOTO 3
A2=A2+HC
B2=B2+HC
D12=D11
IF B2-SKV<+0.1 GOTO 51
YS=SKV
GOTO 8
7 YS=NC
8 REM

PRINT "2*KOR-FUN=";2*CO
PRINT "VEL-FUN=";F2

END

1000 REM
DD=COS(DK)  - подпрограмма вычисления функции.

RETURN
```

В результате работы программы на экран выдается следующий результат:

```
2*KOR-FUN=3.141592653589

VEL-FUN= 6.1257422745431E-017
```

Двойная величина корня функции найдена точно при значении π. Значит, корень находится при π/2. Величина самой функции, когда ее аргумент равен корню, оказывается меньше 10^{-16}.

Приведем пример дугой программы для поиска корня, который использует метод Ньютона:

```
REM ----- Программа поиска корня функции методом Ньютона -----

DEFDBL A-Z
CLS
```

```
X=2  - начальное значение аргумента.
E=1.0E-16  - точность вычислений.
30 CALL FUN(X,F)
L=F
X=X+E
CALL FUN(X,F)
L=E*L/(F-L)
X=X-L-E
IF ABS(L)>E THEN 30
PRINT  "2*X=";2*X,
CALL FUN(X,F)
PRINT "FUN= ";F
END
SUB FUN(X,F)
F=COS(X)
END SUB
```

В результате получаем:

```
2*X= 3.141592653589793
FUN=  6.1257422745431E-017
```

Двойное значение корня, как и в предыдущем случае точно равно числу π, а значение самой функции меньше $1.0\ 10^{-16}$. В данном случае и в предыдущем примере не нулевое значение функции обусловлено пределом точности вычислений, который обеспечивает только 16 значащих цифр.

Нахождение минимума функции

Ниже приведена программа для нахождения минимума функции одного аргумента. В качестве функции выбран Sin(x). Известно, что его минимум, равный -1, находится при 3/2 π. Эта величина выводится в первом столбце результатов. Во втором столбце дается значение аргумента x, при котором достигается минимум функции и в третьем - значение самой функции в минимуме.

Вычисление значений функции здесь организовано через подпрограмму GOSUB, но эта процедура может быть записана и через оператор DEF FN.

```
REM ----- Программа нахождения минимума функции -----
CLS
DEFDBL A-Z

H=.1  - шаг поиска минимума.
Z=.2  - начальное значение аргумента.
E=1E-16  - точность поиска.

115 X=Z
GOSUB 200
V=F

X=Z-H
GOSUB 200
W=F

X=Z+H
GOSUB 200
U=F

T=W*(2*Z+H)-4*V*Z+U*(2*Z-H)
T=T/(W-2*V+U)/2
IF ABS(T-Z)<E THEN 180
Z=T
GOTO 115

180 PRINT 3*3.141592653589793/2,T,V

STOP

200 REM
F=SIN(X)
RETURN
```

В результате работы программы на экране получаем:

4.71238898038469 4.71238898038469 -1

Видно, что при заданной точности удается правильно определить 15 значащих чисел, а величина функции в минимуме находится абсолютно правильно.

Определение детерминанта матрицы

Для расчета детерминанта выбрана треугольная матрица третьего порядка, в которой элементы выше главной диагонали равны нулю. В данном случае, вместо нулей используются очень маленькие величины.

```
REM ----- Программа для вычисления детерминанта матрицы методом Гаусса с выбором главного элемента -----

CLS
DEFDBL A-Z
DIM A(3,3),AA(3,3)

N=3

AA(1,1)=1: AA(1,2)=1E-16: AA(1,3)=1E-16
AA(2,1)=1: AA(2,2)=1:     AA(2,3)=1E-16
AA(3,1)=1: AA(3,2)=1:     AA(3,3)=1
PRINT " ПРОГРАММА ДЛЯ ВЫЧИСЛЕНИЯ ДЕТЕРМИНАНТА МАТРИЦЫ МЕТОДОМ ГАУССА С ВЫБОРОМ ГЛАВНОГО ЭЛЕМЕНТА "
PRINT
PRINT "ИСХОДНЫЙ ВИД МАТРИЦЫ ДЛЯ ВЫЧИСЛЕНИЯ ДЕТЕРМИНАНТА "
PRINT

FOR I=1 TO N
FOR J=1 TO N
A(I-1,J-1)=AA(I,J)
PRINT AA(I,J),
NEXT J
PRINT
NEXT I
```

```
Z=1
D=1
FOR K=0 TO N-2
E=1
FOR I=K TO N-1
FOR J=K TO N-1
IF ABS(E)=ABS(A(I,J)) THEN 90
E=A(I,J)
B=I
C=J
NEXT J
NEXT I

90 IF K=B THEN 120

FOR J=K TO N-1
S=A(K,J)
A(K,J)=A(B,J)
A(B,J)=S
NEXT J

Z=-Z
120 IF K=C THEN 150
FOR I=K TO N-1
S=A(I,K)
A(I,K)=A(I,C)
A(I,C)=S
NEXT I
Z=-Z

150 FOR I=K+1 TO N-1
G=A(I,K)/A(K,K)

FOR J=K TO N-1
A(I,J)=A(I,J)-G*A(K,J)
NEXT J
NEXT I
 NEXT K
```

```
FOR I=0 TO N-1
D=D*A(I,I)
NEXT I

D=D*Z
PRINT
PRINT " ВЕЛИЧИНА ДЕТЕРМИНАНТА =";D

STOP
```

В результате работы программы будем иметь на экране следующие результаты:

ПРОГРАММА ДЛЯ ВЫЧИСЛЕНИЯ ДЕТЕРМИНАНТА МАТРИЦЫ МЕТОДОМ ГАУССА С ВЫБОРОМ ГЛАВНОГО ЭЛЕМЕНТА

ИСХОДНЫЙ ВИД МАТРИЦЫ ДЛЯ ВЫЧИСЛЕНИЯ ДЕТЕРМИНАНТА

1	1E-016	1E-016
1	1	1E-016
1	1	1

ВЕЛИЧИНА ДЕТЕРМИНАНТА= 0.9999999999999998

Точная величина детерминанта такой матрицы известна и равна 1. Видно, что найденное значение отличается от точной величины только на $2.0\ 10^{-16}$.

Решение системы линейных уравнений

Система линейных уравнений вида:

$a_{11}x_1+a_{12}x_2+\ldots\ldots+a_{1N}x_N=b_1$

……..

$a_{N1}x_1+a_{N2}x_2\ldots\ldots..+a_{NN}x_N=b_N$

может быть записана в матричном виде

AX=B

Где B и X - матрицы столбцы, и A - квадратная матрица порядка N.

Ниже приведена программа решения такой системы методом Гаусса с выбором главного элемента. При работе программы надо ввести порядок системы и все матричные элементы для A и B. В результате решения получается столбец неизвестных элементов X. Затем проводится проверка правильности решения по уравнению:

AX-B=Y

Строго говоря, матрица Y должна быть нулевой, но в численных методах ставиться условие малости ее элементов. Чем меньше элементы Y, тем точнее получено решение.

```
REM ----- Программа решения системы линейных уравнений методом Гаусса -----

CLS
DEFDBL A-Z

INPUT "ВВЕДИТЕ ЧИСЛО УРАВНЕНИЙ N?";N
DIM A(N,N),B(N),X(N),C(N,N),D(N),Y(N)

PRINT "ВВОД КОЭФФИЦИЕНТОВ ЛЕВОЙ ЧАСТИ УРАВНЕНИЯ A(I,J)"

FOR I=1 TO N
FOR J=1 TO N
PRINT "A(";I;J;")=? ";
INPUT ,A(I,J)
NEXT J
NEXT I

PRINT "ВВОД КОЭФФИЦИЕНТОВ ПРАВОЙ ЧАСТИ УРАВНЕНИЯ B(I)"
```

```
FOR I=1 TO N
PRINT "B(";I;")=";
INPUT B(I)
D(I)=B(I)
NEXT I

PRINT "ПЕЧАТЬ МАТРИЦЫ КОЭФФИЦИЕНТОВ УРАВ-
НЕНИЯ "A" И "B""
FOR I=1 TO N
FOR J=1 TO N
PRINT USING " +#.#####^^^^ "; A(I,J);
C(I,J)=A(I,J)
NEXT J
PRINT USING "          +#.#####^^^^ ";B(I)
NEXT I

FOR I=1 TO N-1
FOR J=I+1 TO N
A(J,I)=-A(J,I)/A(I,I)
FOR K=I+1 TO N
A(J,K)=A(J,K)+A(J,I)*A(I,K)
NEXT K
B(J)=B(J)+A(J,I)*B(I)
NEXT J
NEXT I
X(N)=B(N)/A(N,N)
FOR I=N-1 TO 1 STEP -1
H=B(I)
FOR J=I+1 TO N
H=H-X(J)*A(I,J)
NEXT J
X(I)=H/A(I,I)
NEXT I

PRINT " ВЫЧИСЛЕННЫЕ КОРНИ СИСТЕМЫ  X(I)"

FOR I=1 TO N
PRINT  "X("I")=";
PRINT USING " +#.#####^^^^ "; X(I)
NEXT I
```

```
PRINT "ПРОВЕРКА ПРАВИЛЬНОСТИ НАЙДЕННОГО РЕШЕНИЯ. ЕСЛИ ВЕЛИЧИНЫ B(I) БЛИЗКИ К НУЛЮ - РЕШЕНИЕ ПРАВИЛЬНОЕ "

FOR I=1 TO N
S=0
FOR J=1 TO N
S=S+C(I,J)*X(J)
NEXT J
Y(I)=S-D(I)
PRINT USING " +#.#####^^^^ ";Y(I)
NEXT I

END
```

В результате решения системы на экране получим:

```
ВВЕДИТЕ ЧИСЛО УРАВНЕНИЙ N? 2

ВВОД КОЭФФИЦИЕНТОВ ЛЕВОЙ ЧАСТИ УРАВНЕНИЯ A(I,J)

A( 1  1 )=? 1
A( 1  2 )=? 2
A( 2  1 )=? 3
A( 2  2 )=? 4

ВВОД КОЭФФИЦИЕНТОВ ПРАВОЙ ЧАСТИ УРАВНЕНИЯ B(I)

B( 1 )=? 5
B( 2 )=? 6

ПЕЧАТЬ МАТРИЦЫ КОЭФФИЦИЕНТОВ УРАВНЕНИЯ "A" И "B"

 +1.00000E+00  +2.00000E+00          +5.00000E+00
 +3.00000E+00  +4.00000E+00          +6.00000E+00
```

ВЫЧИСЛЕННЫЕ КОРНИ СИСТЕМЫ X(I)

X(1)= -4.00000E+00
X(2)= +4.50000E+00

ПРОВЕРКА ПРАВИЛЬНОСТИ НАЙДЕННОГО РЕШЕНИЯ. ЕСЛИ ВЕЛИЧИНЫ Y(I) БЛИЗКИ К НУЛЮ - РЕШЕНИЕ ПРАВИЛЬНОЕ

+0.00000E+00
+0.00000E+00

Из результатов видно, что для такой системы получено абсолютно точное решение, поскольку элементы Y(I) строго равны нулю.

Приведем еще одно решение для другой системы 3 - го порядка:

ПЕЧАТЬ МАТРИЦЫ КОЭФФИЦИЕНТОВ УРАВНЕНИЯ "A" И "B"

+9.00000E+00 +7.00000E+00 +5.00000E+00 +5.00000E+00
+3.00000E+00 +1.00000E+00 +4.00000E+00 +3.00000E+00
+2.00000E+00 +6.00000E+00 +8.00000E+00 +7.00000E+00

ВЫЧИСЛЕННЫЕ КОРНИ СИСТЕМЫ

X(1)= -1.70455E-02
X(2)= +2.32955E-01
X(3)= +7.04545E-01

ПРОВЕРКА ПРАВИЛЬНОСТИ НАЙДЕННОГО РЕШЕНИЯ. ЕСЛИ ВЕЛИЧИНЫ Y(I) БЛИЗКИ К НУЛЮ - РЕШЕНИЕ ПРАВИЛЬНОЕ

+0.00000E+00
+0.00000E+00
-8.88178E-16

В данном случае один из элементов Y(I) точно не равен нулю, но его величина настолько мала, что решение можно считать правильным.

Обращение матрицы

Обратной O(I,J) для заданной матрицы A(I,J) будет матрица удовлетворяющая условию:

AO=E
или

$AA^{-1}=E$

где E - единичная матрица, у которой все элементы, кроме диагональных равны нулю, в диагонали стоят единицы.

Обратная матрица используется для решения системы линейных уравнений:

X=OB

где X - решение системы.

Ниже приведена программа нахождения обратной матрицы:

```
REM -----Программа обращения матрицы -----

DEFDBL A-Z
REM DEFINT I,J,K,L,N,M
DEFDBL A-Z
CLS

N=4
DIM A(N,N),P(N,N),B(N),C(N,N),G(N),X(N)

A(1,1)=2
A(1,2)=8
A(1,3)=1
A(1,4)=11
```

```
A(2,1)=1
A(2,2)=3
A(2,3)=6
A(2,4)=2

A(3,1)=1
A(3,2)=2
A(3,3)=9
A(3,4)=3

A(4,1)=3
A(4,2)=4
A(4,3)=1
A(4,4)=7

PRINT " ИСХОДНАЯ МАТРИЦА A(I,J)"
FOR I=1 TO N
FOR J=1 TO N
PRINT USING " +#.####^^^^ ";A(I,J);
NEXT J
PRINT
NEXT I
PRINT

FOR I=1 TO N
FOR J=1 TO N
P(I,J)=A(I,J)
NEXT J
NEXT I

FOR J2=1 TO N

FOR I=1 TO N
B(I)=0
NEXT I

B(J2)=1

FOR J3=1 TO N
```

```
FOR J4=1 TO N
A(J3,J4)=P(J3,J4)
NEXT J4
NEXT J3

GOSUB 100

FOR II=1 TO N
O(II,J2)=X(II)
REM PRINT X(II)
NEXT II
NEXT J2

SS=0
FOR I=1 TO N
FOR J=1 TO N
S=0
FOR K=1 TO N
S=S+P(I,K)*O(K,J)
NEXT K
E(I,J)=S
SS=SS+S
NEXT J
NEXT I

PRINT " ОБРАТНАЯ МАТРИЦА O(I,J)"
FOR I=1 TO N
FOR J=1 TO N
PRINT USING " +#.####^^^^ ";O(I,J);
NEXT J
PRINT
NEXT I
PRINT

PRINT " СУММА ЭЛЕМЕНТОВ ЕДЕНИЧНОЙ МАТРИЦЫ
S="
PRINT SS
PRINT " ЕДЕНИЧНАЯ МАТРИЦА E(I,J)"
FOR I=1 TO N
FOR J=1 TO N
```

```
PRINT USING " +#.####^^^^ ";E(I,J);
NEXT J
PRINT
NEXT I

END

100 N1=N-1
FOR K=1 TO N1
IF ABS(A(K,K))>0 GOTO 200
K1=K+1
FOR M=K1 TO N
IF ABS(A(M,K))>0 GOTO 150
GOTO 165
150 FOR L=1 TO N
V=A(K,L)
A(K,L)=A(M,L)
A(M,L)=V
NEXT L
165 NEXT M
V=B(K)
B(K)=B(M)
B(M)=V
200 G(K)=B(K)/A(K,K)
K1=K+1
FOR I=K1 TO N
B(I)=B(I)-A(I,K)*G(K)
FOR J1=K TO N
J=N-J1+K
C(K,J)=A(K,J)/A(K,K)
A(I,J)=A(I,J)-A(I,K)*C(K,J)
NEXT J1
NEXT I
NEXT K
M=N
X(M)=B(M)/A(M,M)

250 M=M-1

S=O
```

```
FOR L=M TO N1
S=S+C(M,L+1)*X(L+1)
NEXT L
X(M)=G(M)-S
IF M>1 GOTO 250
RETURN
```

В результате работы программы для поиска обратной матрицы на экране получаем:

```
ИСХОДНАЯ МАТРИЦА A(I,J)
 +2.0000E+00  +8.0000E+00  +1.0000E+00  +1.1000E+01
 +1.0000E+00  +3.0000E+00  +6.0000E+00  +2.0000E+00
 +1.0000E+00  +2.0000E+00  +9.0000E+00  +3.0000E+00
 +3.0000E+00  +4.0000E+00  +1.0000E+00  +7.0000E+00

ОБРАТНАЯ МАТРИЦА O(I,J)

 -3.2468E-01  +3.3766E-01  -2.4675E-01  +5.1948E-01
+6.4935E-02  +5.3247E-01  -3.5065E-01   -1.0390E-01
 -1.2987E-02   -6.4935E-03  +1.2013E-01  -2.9221E-02
+1.0390E-01   -4.4805E-01  +2.8896E-01  -1.6234E-02

СУММА ЭЛЕМЕНТОВ ЕДЕНИЧНОЙ МАТРИЦЫ

S= 3.999999999999999

ЕДЕНИЧНАЯ МАТРИЦА E(I,J)

 +1.0000E+00  -1.6653E-16   -1.1102E-16    -3.4694E-17
 +0.0000E+00   1.0000E+00 +0.0000E+00   -3.4694E-17
 +2.7756E-17   -1.6653E-16  +1.0000E+00  +6.9389E-18
 +2.4980E-16  +1.8319E-15    -2.3315E-15   1.0000E+00
```

Видно, что все элементы единичной матрицы, кроме диагональных практически равны нулю, а по диагонали стоят единицы, как и должно быть при правильном нахождении обратной матрицы. Величина S дает сумму всех элементов единичной матрицы, которая должна быть равна 4 при N=4.

Вычисление гамма функции

Гамма функция, часто используемая в различных математических вычислениях при Z от 0 до 1 может быть представлена в виде степенного ряда:

$$\Gamma(1+Z)=\left[\sum_{k=1}^{N} A_k Z^k\right]^{-1}$$

где a_k - заданные коэффициенты, величина которых известна до N=20 и приведена в самой программе:

```
REM ----- Программа вычисления Γ(Z+1) функции -----

DEFDBL A-Z
DIM A(20)
CLS

Z=0.46163  - аргумент функции

A(0)=1: A(1)=0.57721566490153286
A(2)=-0.65587807152025388: A(3)=-0.04200263503409523
A(4)=0.1665861138229148: A(5)=-0.04219773455554433
A(6)=-0.00962197152787697: A(7)=0.00721894324666309
A(8)=-0.00116516759185906: A(9)=-0.00021524167411495
A(10)=0.00012805028238811: A(11)=-0.00002013485478078
A(12)=-0.00000125049348214: A(13)=0.0000013302723198
A(14)=-0.00000020563384169: A(15)=0.00000000611609510
A(16)=0.00000000500200764: A(17)=-0.00000000118127457
A(18)=0.0000000001044267: A(19)=0.00000000000778226
A(20)=-0.0000000000069681

S=0
FOR I=0 TO 20
S=S+A(I)*Z^I
NEXT I
GA=1/S
PRINT "Γ=";GA
STOP
```

Результатом вычисления, при заданном Z=0.46163 будет величина:

Г= 0.8856015025262314

полностью совпадающая с известными табличными данными - 0.88560.

Пример программы для передачи массива в подпрограмму

На интервале от 0 до π/2 вычисляется значение Sin(x), затем, полученный массив передается в подпрограмму, которая вычисляет сумму всех его значений.

Для передачи массива в подпрограмму нужно ставить просто скобки в операторе вызова CALL в программе и любое целое число в самой подпрограмме SUB. Приведенная программа демонстрирует также использование функции DEF FNимя для вычисления синуса и возможность задания размерности массива через число N, которое предварительно вводится с клавиатуры. Определенную в программе функцию можно использовать и в подпрограммах при вычисляемых там значениях аргумента. Передача в подпрограмму и обратно обычных переменных и массива выполняется по расположению в списке переменных, а не по их имени.

В подпрограмме для Quick Basic нужно оставлять пустые скобки при массиве, а первым оператором программы будет описание подпрограммы - DECLARE SUB AA(M,B(),S).

```
REM  ---------- Программа использования массива ----------

INPUT N
DEF FNS(X)=SIN(X)
DIM A(N)

FOR I=0 TO N
X=3.141159265358979*I/N
A(I)=FNS(X/2)
```

```
NEXT

CALL AA(N,A(),D)
PRINT A(N),D
END

SUB AA(M,B(1),S)
S=0
FOR I=0 TO M
S=S+B(I)
NEXT
END SUB
```

В результате имеем:

```
? 1000
```

Ввод размерности массива с клавиатуры и результат вычислений:

```
 1            637.0699462890625
```

Известно, что $\mathrm{Sin}(\pi/2) = 1$.

Пример программы для использования многострочной функции

Программа набрана маленькими символами, что также допускается в Basic. Определяемая многострочная функция использует текущие значения переменных из самой программы после их вычисления.

```
dim c(100)
def fns=sin(x)

def fna
for i=1 to 10
s=s+c(I)
next
fna=s
```

```
end def

for i=1 to 10
x=3.1415*i/10
c(i)=fns
next

b=fna

print b
end
```

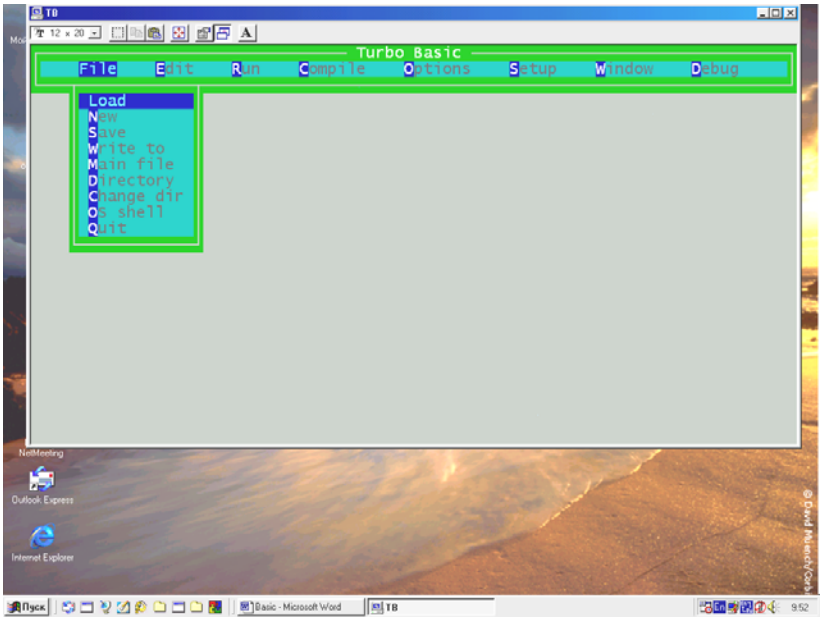
В результате получаем число 55, которое не трудно проверить даже вручную .

ПРИЛОЖЕНИЕ 1
Некоторые дополнительные различия между Turbo и Quick Basic

Turbo Basic	Quick Basic
EXIT LOOP	EXIT DO
Разрешены модули: SUB и DEF FNимя.	Разрешены модули: SUB, FUNCTION и DEF FNимя.
LBOUND(a$(размерность))	LBOUND(a$, размерность)
UBOUND(a$(размерность))	UBOUND(a$, размерность)
INCLUDE - файл может содержать модули SUB и DEF FNимя. Синтаксис: $INCLUDE "имя файла".	INCLUDE - файл не может содержать модули SUB, FUNCTION и DEF FNимя. Синтаксис: '$INCLUDE 'имя файла'.
Длинную строку операторов можно продолжить на другой строке, поставив в конце первой строки символ (_).	Все операторы записываются в одной строке без переносов.
Пауза на N секунд: DELAY N	Пауза на N секунд: SLEEP N
Повтор сигнала N раз: BEEP N.	Однократный сигнал: BEEP.
Функции удаления пробелов слева и справа из символьной строки отсутствуют.	LTRIM$, RTRIM$
В конструкции SELECT CASE при наличии операций сравнения дополнительные ключевые слова не используются - наличие слова IS приведет к невыполнению условия или к появлению ошибки.	В конструкции SELECT CASE при наличии операций сравнения необходимо применение ключевого слова IS, то есть CASE IS > или CASE IS <= или CASE IS <> и т.д.
В 80-ю колонку 24-й и 25-й строк экрана вывод операторами PRINT и LOCATE вызовет сдвиг экрана на строку вверх.	Можно выводить в любую позицию экрана без опасения его сдвига.

ПРИЛОЖЕНИЕ 2
Основное окно компилятора Turbo Basic

Программа Turbo Basic запускается файлом tb.exe, который обычно находится в директории BASIC или TB. После запуска на экране появляется начальное окно, в котором следует нажать клавишу Enter, что приводит к появлению основного окна программы, показанного ниже.



Вверху окна показаны пункты Главного меню. Слева приведено открытое подменю раздела File.

При работе с программой переход в Главное меню, если оно не видно на экране, осуществляется нажатием на клавишу Esc. Открытие подменю происходит при нажатии клавиши со стрелкой "вниз" и Enter. Переход между пунктами Главного меню выполняется нажатием клавиш со стрелками "влево - вправо", а выбор пунктов подменю производится клавишами со стрелками "вниз" или "вверх". После выбора нужного пункта подменю, для выполнения выбранной операции следует нажать Enter.

Приведем краткое описание разделов Главного меню:

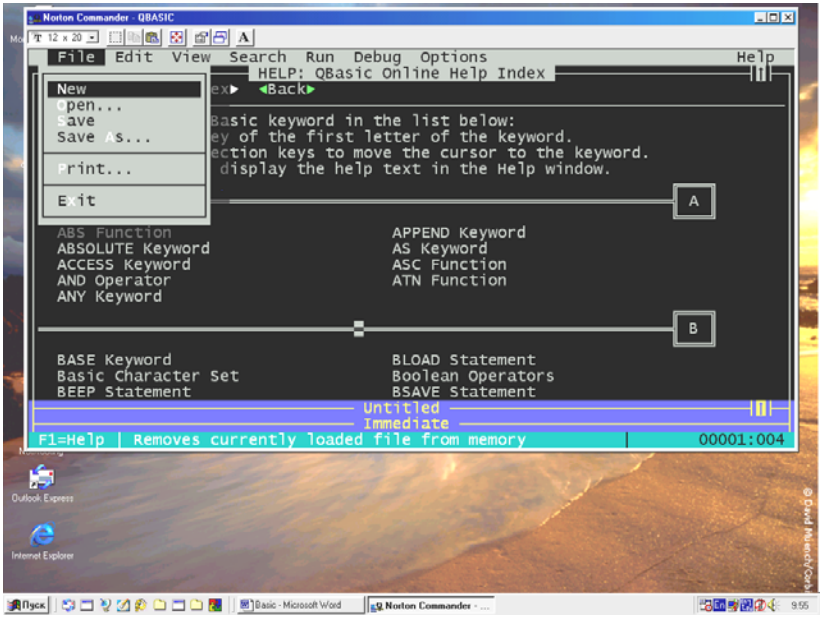
1. File - раздел предназначен для работы с файлами. Позволяет загружать уже существующий файл (Load), создавать новый (New), записывать в файл сделанные изменения (Save), изменять имя файла (Write to), создавать главный файл (Main file), переходить в другую директорию для поиска файлов (Directory), изменять имя директории, в которой находится сама программа (Change dir), выходить в командный режим операционной системы DOS (OS shell) и завершать работу с программой (Quit).
2. Edit - раздел предназначен для редактирования загруженного файла.
3. Run - раздел предназначен для компиляции и запуска программы на счет.
4. Compile - компиляция программы без запуска на счет.
5. Options - установка режимов работы программы Basic и согласование ее с имеющимися аппаратными средствами.
6. Setup - установка режимов показа результатов на экране и вида самого экрана программы.
7. Window - переход между разными экранами - окнами программы.
8. Debug - режим отладки с пошаговым выполнением задания и выдачей на экран результатов расчета каждого шага.

Список основных файлов, входящих в состав Turbo Basic:

tb.exe
tbconfig.tb
tbhelp.tbh
ibmbasic.ng

ПРИЛОЖЕНИЕ 3
Основное окно компилятора Quick Basic

Программа Quick Basic запускается файлом qbasic.exe, который обычно находится в директории BASIC или QB. После запуска программы на экране появляется начальное окно, показанное на рисунке ниже.



Для вывода на экран справки следует нажать клавишу Enter. Для перехода в режим набора новой программы нужно нажать Esc. Переход в Главное меню осуществляется нажатием Alt. Переход между пунктами меню и подменю проводится, как в предыдущем случае для программы Turbo Basic.

Приведем краткое описание пунктов Главного меню:

1. File - раздел позволяет работать с файлами. Можно сохранить сделанные в вашей программе изменения (Save), создать новый файл (New), переименовать имеющийся открытый файл (Save as), открыть уже существующий файл (Open), напе-

чатать содержимое файла на принтере (Print) и выйти из программы с завершением ее работы(Exit).

2. Edit - редактирование содержимого файла или набор новой программы.
3. View - просмотр содержимого файла с программой и подпрограммами.
4. Search - поиск в файле нужного объекта.
5. Run - выполнение, запуск программы на счет.
6. Debug - режим отладки с пошаговым выполнением задания и выдачей на экран результатов расчета каждого шага.
7. Options - установка режимов работы программы Basic и вида ее окна.

Список основных файлов, входящих в состав Quick Basic:

qbasic.exe
qbasic.hlp
qbasic.ini

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Смирнов Н.Н. - Программные средства персональных ЭВМ, Л., 1990, 270с.

2. Моррил Г. - Бейсик для ПК ИБМ, М., ФиС, 1987, 260с.

3. Пярнпу А.А. - Программирование на современных алгоритмических языках, М., Наука, 1990, 383с.

4. Лемешко Е.В., Романчук В.Г. - Основы программирования на языке Бейсик, М. МИУ, 1988, 49с.

5. Блэнд Г. - Основы программирования на языке Бейсик, М., ФиС, 1989, 207с.

6. Геворкян Г.Х., Семенов В.Н. - Бейсик - это просто, М., Радио и Связь, 1989, 143с.

7. Дьяконов В.П. - Применение персональных ЭВМ и программирование на языке Бейсик, М., Радио и Связь, 1989, 286с.

8. Дьяконов В.П. - Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ, М., Наука, 1989, 239с.

9. Кетков Ю.Л. - Диалог на языке Бейсик для мини - и микро ЭВМ, М., Наука, 1988, 367с.

10. Кетков Ю.Л. - Программирование на Бейсике, М., Статистика, 1978, 158с.

11. Кэтлин Э. - Программирование на языке Бейсик, М., Мир, 1990, 288с.

12. Вашкевич Ю.Ф. и др. - Справочник по программированию на Бейсике, М. Радио и Связь, 1987. 335с.

13. Уорт Т. - Программирование на языке Бейсик, М., Машиностроение, 1981. 255с.

14. Трояновский В.М., Шаньгин В.Ф. - Бейсик для начинающих и будущих профессионалов, М., Высшая школа, 1992. 239с.

15. Башмаков Е.С., и др. - Программирование микро ЭВМ на языке Бейсик, М., Радио и Связь, 1991. 239с.

16. Джордейн Р. - Справочник программиста персональных компьютеров типа IBM PC, XT и AT, Пер. с англ. и предисл. Н.И. Гайского, М., Финансы и статистика, 1991, 554с.

17. Очков В.Ф., Пухначев Ю.В. - 128 советов начинающему программисту, М., Энергоатомиздат, 1991, 256с.

18. Зельднер Г.А. - Quick BASIC для носорога, М., ABF, 1994, 461с.

19. Корчак А.Е. - Справочник. БЕЙСИК - версии для MS-DOS, М., МЦНТИ, 1996, в 2-томах, 463с.

20. Ясинский В. - Turbo, Quick и PDS Basic - алгоритмы и программы, М., 1997.

21. Данилина И.И. - Численные методы, М., Высшая школа, 1976, 367с.

22. Попов Б.А., Теслер Г.С. - Вычисление функций на ЭВМ, Киев, Наукова думка, 1984, 597с.

23. Дубовиченко С.Б. - Некоторые версии алгоритмического языка Бейсик, Издание второе, Алматы, УЭиП, 2001, 166с.

Сергей Борисович Дубовиченко
Член - корреспондент Казахстанской Международной Академии информатизации

Turbo и Quick Бейсик.
Издание третье, дополненное и исправленное
Учебное пособие

Подписано к печати 25.12.2000. Формат 60x84 1/16. Бумага офсетная. Печать офсетная. Уч. - изд. л. 10,0. Заказ № .
Цена договорная.