LEFT$** **LEFT$

Format
LEFT$(string$,numvar)

Cross Reference
SEG$, RIGHT$, POS

Description
LEFT$() returns the leftmost portion of the string represented by string$ of length numvar.

The LEFT$ function creates a new string but does not destroy the original string.

LEFT$(A$,5) is equivalent to SEG$(A$,1,5) if A$ is at least 5 characters long.

If the string is shorter then the length specified, the string LEFT$ function will pad the string with blank spaces rather than return an error condition

LEFT$ can be used with numerical data if the number is first converted to a string using the STR$(n) function.

Example

```
100 B$=LEFT$("1234",3)
110 PRINT B$
120 C$=VAL(LEFT$(STR$(-1234),4)
130 PRINT C$
RUN
123
-123
```

LEFT$ can also be used to make a program user friendly by separating first from last names, checking the first character of a response etc.

Example

```
100 INPUT "What is your full name please ":NAME$
110 SP=POS(NAME$," ",1)
120 FIRST$=LEFT$(NAME$,SP-1)
130 INPUT FIRST$&" IS THE CAPITOL OF THE UNITED STATES BROOKLYN ?":ANSWER$
135 A$=LEFT$(ANSWER$,1)
140 IF A$="Y" OR A$="y" THEN PRINT "I'M SORRY ";FIRST$;" that is not
    correct"::GOTO 170
150 if A$="N" OR "n" THEN PRINT "* THAT IS RIGHT"::STOP
160 PRINT "TYPE YES or NO as a response please "::GOTO 130
RUN
What is your full name please ? ABRAHAM LINCOLN
ABRAHAM IS THE CAPITOL OF THE UNITED STATES BROOKLYN ? NO
THAT IS RIGHT
```

LEN --Function--Length LEN

Format
LEN(string-expression)

Type
DEFINT

Description
The LEN function returns the number of characters in the string specified by the string-expression.

If the string-expression is a null string, LEN returns a zero.

Remember that a space is a valid character and is considered to be part of the length of a string.

Examples

```
100 PRINT LEN("ABCDE")
```
Prints 5.

```
100 X=LEN("THIS IS A SENTENCE.")
```
Sets X equal to 19.

```
100 DISPLAY LEN("")
```
Displays 0.

```
100 DISPLAY LEN(" ")
```
Displays 1.

```
100 A$="DAVID"
110 DISPLAY LEN(A$)
```
Displays 5 when A$ equals DAVID.

LET

LET

Format
[LET]variable-list=expression

Description
The LET instruction, often called the "assignment" instruction, enables you to assign values to variables.

You can use LET as either a program statement or a command.

The variable-list consists of one or more variables separated by commas. Do not mix numeric and string variables in the same variable-list. However, you can include both DEFINT and REAL numeric variables in the same variable-list.

The value of expression is assigned to all variables in the variable-list. If the variable-list contains numeric variables, the expression must be a
numeric-expression. If the variable-list contains string variables, the expression must be a string-expression.

The word LET can be optionally omitted from instruction.

Examples

```
100 T=4
```
Assigns to T the value 4.

```
100 X,Y,Z=12.4
```
Assigns to X, Y, and Z the value 12.4.

```
100 A=3<5
```
Assigns -1 to A because it is true that 3 is less than 5.

```
100 B=12<7
```
Assigns 0 to B because it is not true that 12 is less than 7.

```
100 L$,D$,B$="B"
```
Assigns to L$, D$, and B$ the string constant "B".

Program

The following program illustrates a use of LET.

```
100 K=1
110 K,A(K)=3
120 PRINT K;A(1)
130 PRINT A(3);A(K)
RUN
3 3
0 0
```

In line 100, the variable K is assigned the value 1.

In line 110, the variable K and the array element A(K) are assigned the value of 3. Note that when line 110 is executed, the subscript K is not assigned a new value, but has the same value it had before the line was executed. Therefore, A(K) is an expression equivalent to A(1), referring to the same element of the array.

In line 120, the values of K and A(1) are printed.

When line 130 is executed, K has a value of 3; therefore, A(K) is now an expression equivalent to A(3). Both expressions have a value of 0 (the default value) because no value has been assigned to this element of array.

LINK --Subprogram **LINK**

Format
CALL LINK(subprogram-name[,parameter-list])

Cross Reference
INIT, LOAD, SUB

Description
The LINK subprogram enables you to transfer control from a MYARC Advanced BASIC program to an assembly-language subprogram.

The subprogram-name is an entry point in an assembly-language subprogram that you have previously loaded into memory with the LOAD subprogram. The subprogram-name is a string-expression; if you use a string constant, it must be enclosed in quotation marks.

The optional parameter-list consists of one or more parameters, separated by commas, that are to be passed to the assembly-language subprogram. The contents of the parameter-list depend on the particular subprogram you are accessing.

The rules for passing parameters to an assembly-language subprogram are the same as the rules for passing parameters to a MYARC Advanced BASIC subprogram (see SUB).

Example

```
100 CALL LINK("START",1,3)
```
Links the MYARC Advanced BASIC program to the assembly-language subprogram START, and passes the values 1 and 3 to it.

LINPUT --Line Input **LINPUT**

Format
Keyboard Input
```
LINPUT [input-prompt:]string-variable
```
File Input
```
LINPUT #file-number[,REC record-number]:string-variable
```

Cross Reference
ACCEPT, EOF, INPUT, OPEN, TERMCHAR

Description
The LINPUT statement suspends program execution to enable you to enter a line of unedited data from the keyboard. LINPUT can be used also to retrieve an unedited record from an external device.

LINPUT assigns an entire line, a file record, or the remaining portion of a file record (if there is an input-pending condition) to the string-variable.

See INPUT for an explanation of keyboard- and file-input, and input options.

No editing is performed on the input data. All characters (including commas, quotation marks, colons, semicolons, and leading and trailing spaces) are assigned to the string-variable as they are encountered.

The maximum value that can be input from the keyboard is 255 characters.

LINPUT is frequently used instead of INPUT when the input data may include a comma. (A comma is not accepted as input by the INPUT statement, except as part of a string enclosed in quotation marks.)

To use LINPUT for file input the file must be in DISPLAY format.

You normally press ENTER to complete keyboard input; however, you can also use AID, BACK, BEGIN, CLEAR, PROC'D, DOWN ARROW, or UP ARROW. You can use the TERMCHAR function to determine which of these keys was pressed to exit from the previous ACCEPT, INPUT, or LINPUT instruction.

Note that pressing CLEAR during keyboard input normally causes a break in the program. However, if your program includes an ON BREAK NEXT statement, you can use CLEAR to exit from an input field.

Examples

```
100 LINPUT L$
```
Assigns to L$ anything typed before ENTER is pressed.

```
100 LINPUT "NAME: "NM$
```
Displays NAME: and assigns to NM$ anything typed before ENTER is pressed.

100 LINPUT #1,REC M:L$(M)
Assigns to L$(M) the value that was in record M of the file that was opened as #1 with RELATIVE DISPLAY file organization.

Program

The following program illustrates a use of LINPUT. It reads a previously existing file and displays only the lines that contain the word "THE."

```
100 OPEN #1:"DSK1.TEXT1",INPUT,FIXED 80,DISPLAY
110 IF EOF(1) THEN CLOSE #1 :: STOP
120 LINPUT #1:A$
130 X=POS(A$,"THE",1)
140 IF X>0 THEN PRINT A$
150 GOTO 110
```

NOTE:
Remember to press the two keys, Control + Break whenever the Manual refers to "CLEAR".

LIST LIST

Format
List to the screen
    LIST [line-number-range]
List to a File (or Device)
    LIST "file-specification"[:line-number-range]

Cross Reference
LLIST

Description
The LIST command displays the program (or a portion of it) currently in memory. You can also use LIST to output the program listing to an external device.

The optional line-number-range specifies the portion of the program to be listed. If you do not enter a line-number-range, the entire program is listed. The program lines are always listed in ascending order.

If you enter a file-specification, the program listing is output to the specified file or device. The file-specification, a string constant, must be enclosed in quotation marks.

The program listing is output as a SEQUENTIAL file in DISPLAY format with VARIABLE records (see OPEN); the file-specification option can be used only with devices that accept these options. For more information about listing a program on a particular device, refer to the owner's manual that comes with that device. If you do not enter a file-specification, the program listing is displayed on the screen.

You can stop the listing at any time by pressing CLEAR. Pressing any other key (except SHIFT, ALT, or CTRL) causes the listing to pause until you press a key again.

The LIST command only works with peripherals that support DISPLAY/VARIABLE type records.

The Line-Number-Range

A line-number-range can consist of a single line number, a single line number followed by a hyphen, a single line number preceded by a hyphen, or a range of line numbers.

COMMAND	LINES LISTED
LIST	All lines.
LIST X	Line number X only.
LIST X-	Lines from number X to the highest line number, inclusive.
LIST -X	Lines from the lowest line number to line number X, inclusive.
LIST X-Y or LIST X Y	All lines from line number X to line number Y, inclusive.

If the line-number-range does not include a line number in your program, the following conventions apply:

If line-number-range is higher than any line number in the program, the highest-numbered program line is listed.

If line-number-range is lower than any line number in the program, the lowest-numbered program line is listed.

If line-number-range is between lines in the program, the next higher numbered program line is listed.

Examples

LIST
Lists the entire program in memory on the display screen.

LIST 100
Lists line 100.

LIST 100-
Lists line 100 and all after it.

LIST -200
Lists all lines up to and including line 200.

LIST 100-200
Lists all lines from 100 through 200.

LLIST LLIST

Format
LLIST[linenum1][-][linenum2][,width]

Cross Reference
LIST, LPRINT

Description
Same line format as LIST except that LLIST automatically sends list to default print device.

COMMAND	LINES LISTED
LLIST	All lines.
LLIST X	Line number X only.
LLIST X-	Lines from number X to the highest line number, inclusive.
LLIST -X	Lines from the lowest line number to line number X, inclusive.
LLIST X-Y or LLIST X Y	All lines from line number X to line number Y, inclusive.
LLIST X-Y,132	All lines from line number X to line number Y, inclusive are printed using a page width of 132 characters.

If the line-number-range does not include a line number in your program, the following conventions apply:

If line-number-range is higher than any line number in the program, the highest-numbered program line is listed.

If line-number-range is lower than any line number in the program, the lowest-numbered program line is listed.

If line-number-range is between lines in the program, the next higher numbered program line is listed.

Width is the number of characters across the page the default is 80.

If the page width depends upon an escape code or control code sequence, then that sequence must be sent to the print device using the LPRINT command.

The default device can be changed by changing the name of LPT. (See defaults.)

```
LPT="PIO"
```

Examples

LLIST
Prints the entire program in memory on the display screen.

LLIST 100
Prints line 100.

LLIST 100-
Prints line 100 and all after it.

LLIST -200
Prints all lines up to and including line 200.

LLIST 100-200
Prints all lines from 100 through 200.

LLIST 100-200,132
Prints all lines from 100 through 200 on a page width of 132 characters.

LOAD --Subprogram LOAD

Format
File Only
```
     CALL LOAD(file-specification-list)
```
Data Only
```
     CALL LOAD(address,byte-list[,"",address,byte-list[,...]])
```
File and Data
```
     CALL LOAD(file-specification-list,address,byte-list[,...])
     CALL LOAD(address,byte-list,file-specification-list[,...])
```

Cross Reference
INIT, LINK, PEEK, PEEKV, POKEV, VALHEX

Description
The LOAD subprogram enables you to load assembly-language subprograms into memory. You can also use LOAD to assign values directly to specified CPU (Central Processing Unit) memory addresses. You can use the POKEV subprogram to assign values to VDP (Video Display Processor) memory.

To load an assembly-language subprogram, specify a file-specification-list; to assign values to CPU memory, specify an address and a byte-list (an address must always be followed by a byte-list).

You must enter at least one parameter. The first parameter you specify can be either a file-specification-list or an address.

If you wish to follow an address and byte-list with another address and byte-list, enter a file-specification-list or a null string (two-adjacent quotation marks) as a separator.

The optional file-specification-list consists of one or more file-specifications separated by commas. A file-specification is a string-expression; if you use a string constant, you must enclose it in quotation marks.

Each file-specification names an assembly-language object (program) file to be loaded into memory. The specified file can include subprogram names, so that the subprograms can be executed by the LINK subprogram.

The object file to be loaded must be in DISPLAY format with FIXED records (see OPEN). For more information about the file options available with a particular device, refer to the owner's manual that comes with that device.

You can optionally load bytes of data to a specified CPU memory address. The address specifies the first address where the data is to be loaded; if the byte-list specifies more than one byte of data, the bytes are assigned to sequential memory addresses starting with the address you specify.

The numeric-expression address must have a value from -32768 to 32767 inclusive.

You can specify an address from 0 to 32767 inclusive by specifying the actual address.

You can specify an address from 32768 to 65535 inclusive by subtracting 65536 from the actual address. This will result in a value from -32768 to -1 inclusive.

If you know the hexadecimal value of the address, you can use the VALHEX function to convert it to a decimal numeric-expression, eliminating the possible need for calculations.

If necessary, the address is rounded to the nearest integer.

The byte-list consists of one or more bytes of data, separated by commas, that are to be loaded into CPU memory starting with the specified address.

Each byte in the byte-list must be a numeric-expression with a value from 0 to 32767. If the value of a byte is greater than 255, it is repeatedly reduced by 256 until it is less than 256. If necessary, a byte is rounded to the nearest integer.

Note that you must use the INIT subprogram to reserve memory space before you use LOAD to load a subprogram.

If you call the LOAD subprogram with invalid parameters or load an object file with absolute (rather than relocatable) addresses, the computer may function erratically or cease to function entirely. If this occurs, turn off the computer, wait several seconds, then turn the computer back on again.

The Loader

LOAD uses a "relocatable linking" loader.

Because it is "relocatable," you cannot use LOAD to specify a memory address at which you want to load a file. However, the file you are loading may specify an absolute load address if it includes an AORG directive.

Because it is "linking", the object files specified in the file-specification-list can reference each other.

LOCATE --Subprogram **LOCATE**

Format
CALL LOCATE(#sprite-number,pixel-row,pixel-column[,...])

Cross Reference
DELSPRITE, SPRITE

Description
The LOCATE subprogram enables you to change the location of one or more sprites.

The sprite-number is a numeric-expression whose value specifies the number of a sprite as assigned by the SPRITE subprogram.

The pixel-row and pixel-column are numeric-expressions whose values specify the screen pixel location of the pixel at the upper-left corner of the sprite.

LOCATE can cause a sprite that has been deleted with DELSPRITE sprite-number to reappear.

Program

The following program illustrates a use of the LOCATE subprogram.

```
100 CALL CLEAR
110 CALL SPRITE(#1,33,7,1,1,25,25)
120 YLOC=INT(RND* 150+1)
130 XLOC=INT(RND* 200+1)
140 FOR DELAY=1 TO 300 :: NEXT DELAY
150 CALL LOCATE(#1,YLOC,XLOC)
160 GOTO 120
(Press CLEAR to stop the program.)
```

Line 110 creates a sprite as a fairly quickly moving red exclamation point.

Line 140 locates the sprite at a location randomly chosen in lines 120 and 130.

Line 150 repeats the process.

Also see the third example of the SPRITE subprogram.

LOG --Function--Natural Logarithm **LOG**

Format
LOG(numeric-expression)

Type
REAL

Cross Reference
EXP

Description
The LOG function returns the natural logarithm of the value of the numeric-expression. LOG is the inverse of the EXP function.

The value of the numeric-expression must be greater than zero.

Examples

```
100 PRINT LOG(3.4)
```
Prints the natural logarithm of 3.4, which is 1.223775432.

```
100 X=LOG(EXP(7.2))
```
Sets X equal to the natural logarithm of e raised to the 7.2 power, which is 7.2.

```
100 S=LOG(SQR(T))
```
Sets S equal to the natural logarithm of the square root of the value of T.

Program

The following program returns the logarithm of any positive number in any base.

```
100 CALL CLEAR
110 INPUT "BASE: ":B
120 IF B =1 THEN 110
130 INPUT "NUMBER: ":N
140 IF N =0 THEN 130
150 LG=LOG(N)/LOG(B)
160 PRINT "LOG BASE";B;"OF";N;"IS";LG
170 PRINT
180 GOTO 110
```
(Press CLEAR to stop the program.)

LPR

Format
CALL LPR(x,y)

Cross Reference
POINT, DRAW, DRAWTO, LINE, PSET or preset or the current position of the mouse cursor.

Description
Last Point Referenced returns the coordinates of the last point referenced by the graphics commands.

LPT

Syntax
LPT=device name string

Cross Reference
DOS Manual, DEFAULTS, LCOPY, LTRACE, LLIST

Description
You can use LPT either as a program statement or a command.

LPT is used to modify the name of the default print device.

Example

```
LPT="PIO"
LPT="RS232.BA=9600,DA=8"
```

The default print device is accessed from BASIC in the command mode or within a program by use of the following commands: LCOPY, LTRACE, LLIST.

LTRACE

LTRACE

Cross Reference
TRACE, BREAK

Description
LTRACE is used exactly as TRACE except the output is directed towards the default print device rather than the screen.

LTRACE is a valuable aid because it is not affected by screen clearing commands such as:

CALL CLEAR, CLS, DISPLAY, ERASE ALL, CALL GRAPHICS( ) etc.

MAGNIFY --Subprogram **MAGNIFY**

Format
CALL MAGNIFY(numeric-expression)

Cross Reference
CHAR, SPRITE

Description
The MAGNIFY subprogram enables you to specify whether all sprites are single- or double-sized and whether they are unmagnified or magnified.

The value of the numeric-expression specifies the size and magnification "level" of all sprites. (You cannot specify the level of an individual sprite.)

LEVEL	CHARACTERISTICS
1	Single-sized, unmagnified
2	Single-sized, magnified
3	Double-sized, unmagnified
4	Double-sized, magnified

The screen position of the pixel in the upper-left corner of a sprite is considered to be the position of that sprite. That pixel remains in the same screen position regardless of changes to the magnification level.

When you enter MYARC Advanced BASIC, sprites are single-sized and unmagnified (level 1). When your program ends (either normally or because of an error), stops at a breakpoint, or changes graphics mode, the sprite magnification level is restored to 1.

Single-Sized Sprites

A single-sized sprite is defined only by the character you specify when the sprite is created.

Double-Sized Sprites

A double-sized sprite is defined by four consecutive characters, including the character that you specify when the sprite is created.

If the number of the character you specify is a multiple of 4, that character is the first of the four characters that comprise the sprite's definition. If the character number is not a multiple of 4, the next lower character that is a multiple of four is the first character of the sprite.

The first of the four characters defines the upper-left quarter of the sprite, the second character defines the lower-left quarter of the sprite, the third defines the upper-right quarter of the sprite, and the last of the four characters defines the lower-right quarter of the sprite.

Unmagnified Sprites

An unmagnified sprite occupies only the number of characters on the screen specified by the characters that define it.

A single-sized unmagnified sprite occupies 1 character position on the screen; a double-sized unmagnified sprite occupies 4 character positions.

Magnified Sprites

A magnified sprite expands to twice the height and twice the width of an unmagnified sprite. The expansion occurs down and to the right; the pixel in the upper-left corner of the sprite remains in the same screen position.

A magnified sprite has 4 times the area of an unmagnified sprite. When you magnify a sprite, each pixel of the unmagnified sprite expands to 4 pixels of the magnified sprite.

A single-sized magnified sprite occupies 4 character positions on the screen; a double-sized magnified sprite occupies 16 character positions.

Program

The following program illustrates a use of the MAGNIFY subprogram.

A little figure (single-sized, unmagnified) appears near the center of the screen. In a moment, it becomes twice as big (single-sized, magnified), covering four character positions. In another moment, it is replaced by the upper-left corner of a larger figure (single-sized, magnified), still covering four character positions. Then the full figure appears (double-sized, magnified), covering sixteen character positions. Finally it is reduced in size to four character positions (double-sized, unmagnified).

```
100 CALL CLEAR
110 CALL CHAR(148,"1898FF3D3C3CE404")
120 CALL SPRITE(#1,148,5,92,124)
130 GOSUB 230
140 CALL MAGNIFY(2)
150 GOSUB 230
160 CALL CHAR(148,"0103C3417F3F07070707077E7C40000080C0C080
FCFEE2E3E0E0E06060606070")
170 GOSUB 230
180 CALL MAGNIFY(4)
190 GOSUB 230
200 CALL MAGNIFY(3)
210 GOSUB 230
220 STOP
230 REM DELAY
240 FOR DELAY=1 TO 500
250 NEXT DELAY
260 RETURN
```

Line 110 defines character 148.

Line 120 sets up sprite using character 148. By default the magnification factor is 1.

Line 140 changes the magnification factor to 2.

Line 160 redefines character 148. Because the definition is 64 characters long, it also defines characters 149, 150, and 151.

Line 180 changes the magnification factor to 4.

Line 200 changes the magnification factor to 3.

MARGIN --Subprogram **MARGIN**

Format
CALL MARGIN(left,right,top,bottom)

Cross Reference
ACCEPT, CLEAR, DISPLAY, DISPLAY USING, GRAPHICS, INPUT, LINPUT, PRINT, PRINT USING

Description
The MARGIN subprogram enables you to define screen margins. The margins you specify define a screen window that affects the operation of several instructions.

Left, right, top, and bottom are numeric-expressions whose values specify the margins.

The margins cannot "overlap"; that is, the position of the top margin must be higher on the screen than the bottom margin, and the position of the left margin must be farther left on the screen than the right margin.

When creating a screen window, you must leave the window large enough to allow entry of a command.

The valid range for margin location varies according to the graphics mode. Acceptable values for the margins in each mode are found in Appendix K.

The upper-left corner of the window defined by the margins is considered to be the intersection of row 1 and column 1 by the ACCEPT, DISPLAY, and DISPLAY USING instructions that use the AT option.

The lower-left corner of the window is considered to be the beginning of the input line by the ACCEPT, INPUT, and LINPUT instructions.

The lower-left corner of the window is considered to be the beginning of the print line by the DISPLAY, DISPLAY USING, PRINT, and PRINT USING instructions.

When the ACCEPT, INPUT, LINPUT, or PRINT USING instructions cause scrolling, scrolling occurs only in the window.

The CLEAR, GCHAR, HCHAR, and VCHAR subprograms are not affected by the margin settings.

In all modes, the margins can extend to the edges of the screen.

When you enter MYARC Advanced BASIC, the left margin is set to 3 and the right margin to 30. The top and bottom margins are set to 1 and 24 respectively. When a program running in High-Resolution Mode ends, these default margin settings are restored.

Examples

```
100 CALL MARGIN(3,30,1,24)
```
Sets all four margins to the default value in Pattern and High-Resolution Modes.

```
100 CALL MARGIN(1,40,1,24)
```
Sets the left, right, top and bottom margins to the extreme edges of the screen in the 40 column Text Mode (Graphics(2,1)).

MAX --Function--Maximum **MAX**

Format
MAX(numeric-expression1,numeric-expression2)

Type
Numeric (REAL or DEFINT)

Cross Reference
MIN

Description
The MAX function returns the larger value of two numeric-expressions.

MAX is the opposite of the MIN function.

If the values of the numeric-expressions are equal, MAX returns that value.

Examples

```
100 PRINT MAX(3,8)
```
Prints 8.

```
100 F=MAX(3E12,1800000)
```
Sets F equal to 3E12.

```
100 G=MAX(-12,-4)
```
Sets G equal to -4.

```
100 A=7::B=-5
110 L=MAX(A,B)
```
Sets L equal to 7 when A=7 and B=-5.

MEMSET

MEMSET

Format
CALL MEMSET(array-variable(),expression)

Cross Reference
DIM, SWAP

Description
The MEMSET statement will set all elements of the designated numeric or string array to the value of the expression.

Example

```
100 DIM A$(2,2),C(400)
110 CALL MEMSET(A$(),"B")
120 PRINT A$(2,1)
130 CALL MEMSET(C(),234)
140 PRINT C(0);C(400)
RUN
B
234 234
```

MERGE **MERGE**

Format
MERGE["]file-specification["]

Cross Reference
SAVE

Description
The MERGE command combines a program from an external storage device with the program currently in memory. MERGE is frequently used to combine several previously written program segments into one program.

> The file-specification is a string constant that indicates the name of the program on the external device. The file-specification can optionally be enclosed in quotation marks.

The lines of the external program are inserted in line-number order among the lines of the program in memory. If a line number in the external program duplicates a line number in the program in memory, the new line replaces the old line.

The MERGE command does not clear breakpoints.

A program on an external device can be merged only if it was saved with the MERGE option of the SAVE command.

Example

MERGE DSK1.SUB
Merges the program SUB into the program currently in memory.

Program

Listed below is an example of how to merge programs. If the following program is saved on DSK1 as BOUNCE with the merge option, it can be merged with other programs.

```
100 CALL CLEAR
110 RANDOMIZE
140 DEF RND50=INT(RND* 50-25)
150 GOSUB 10000
10000 FOR AA=1 TO 100
10010 QQ=RND50
10020 LL=RND50
10030 CALL MOTION(#1,QQ,LL)
10040 NEXT AA
10050 RETURN
SAVE "DSK1.BOUNCE",MERGE
NEW
```

Place the following program into the computer's memory.

```
120 CALL CHAR(96,"18183CFFFF3C1818")
130 CALL SPRITE(#1,96,7,92,128)
150 GOSUB 500
160 STOP
```

Now merge BOUNCE with the above program.

```
MERGE DSK1.BOUNCE
```

The program that results from merging BOUNCE with the above program is shown here.

```
LIST
100 CALL CLEAR
110 RANDOMIZE
120 CALL CHAR(96,"18183CFFFF3C1818")
130 CALL SPRITE(#1,96,7,92,128)
140 DEF RND50=INT(RND* 50-25)
150 GOSUB 10000
160 STOP
10000 FOR AA=1 TO 100
10010 QQ=RND50
10020 LL=RND50
10030 CALL MOTION(#1,QQ,LL)
10040 NEXT AA
10050 RETURN
```

Note that line 150 is from the program that was merged (BOUNCE), not from the program that was in memory.

MIN --Function--Minimum **MIN**

Format
MIN(numeric-expression1,numeric-expression2)

Type
Numeric

Cross Reference
MAX

Description
The MIN function returns the smaller value of two numeric-expressions. MIN is the opposite of the MAX function.

If the values of the numeric-expressions are equal, MIN returns that value.

Examples

```
100 PRINT MIN(3,8)
```
Prints 3.

```
100 F=MIN(3E12,1800000)
```
Sets F equal to 1800000.

```
100 G=MIN(-12,-4)
```
Sets G equal to -12.

```
100 A=7::B=-5
110 L=MIN(A,B)
```
Sets L equal to -5 when A=7 andB=-5.

MOD --Function **MOD**

Format
MOD(numvar1,numvar2)

Description
MOD computes the arithmetic remainder (MODulo) from the expression numvar1,numvar2. The remainder is then rounded up or down to the nearest integer.

Example

```
10 FOR I=1 TO 1000
20 R = MOD(I,20)
30 PRINT I,R
40 NEXT I RUN
```

The above program prints to the screen the modulo base 20 of all integers between 1 and 1000.

MOTION --Subprogram **MOTION**

Format
CALL MOTION(#sprite-number,vertical-velocity,horizontal-velocity[,...])

Cross Reference
SPRITE

Description
The MOTION subprogram enables you to change the velocity of one or more sprites.

The sprite-number is a numeric-expression whose value specifies the number of a sprite as assigned by the SPRITE subprogram.

The vertical- and horizontal-velocity are numeric-expressions whose values range from -128 to 127. If both values are zero, the sprite is stationary. The speed of a sprite is in direct linear proportion to the absolute value of the specified velocity.

A positive vertical-velocity causes the sprite to move toward the bottom of the screen; a negative vertical-velocity causes the sprite to move toward the top of the screen.

A positive horizontal-velocity causes the sprite to move to the right; a negative horizontal-velocity causes the sprite to move to the left.

If neither the vertical- nor horizontal-velocity are zero, the sprite moves at an angle in a direction and at a speed determined by the velocity values.

When a moving sprite reaches an edge of the screen, it disappears. The sprite reappears in the corresponding position at the opposite edge of the screen.

Program

The following program illustrates a use of the MOTION subprogram.

```
100 CALL CLEAR
110 CALL SPRITE(#1,33,5,92,124)
120 FOR XVEL=-16 TO 16 STEP 2
130 FOR YVEL=-16 TO 16 STEP 2
140 DISPLAY AT(12,11):XVEL;YVEL
150 CALL MOTION(#1,YVEL,XVEL)
160 NEXT YVEL
170 NEXT XVEL
```

Line 110 creates a sprite.
Lines 120 and 130 set values for the motion of the sprite.
Line 150 sets the sprite in motion.
Lines 160 and 170 complete the loops that set the values for the motion of the sprite.

MOUSE --Commands **MOUSE**

The MYARC 9640 supports the industry standard MS mouse interface. Software within the operating system is used to position the mouse on the screen and detect mouse key depressions. The mouse itself is implemented as sprite 1 and therefore sprite 1 should not be used elsewhere in the program when using the mouse. In order to easily interface to these low level routines, MYARC Advanced BASIC implements a standard set of mouse commands. An example program is given in Appendix L illustrating the use of these commands.

MOUSE ON

Turns ON mouse interrupt. Mouse buttons are checked at the start of each BASIC statement.

If a mouse button is pressed, program execution is branched to an "ON MOUSE" subroutine or subprogram if the particular mouse key pressed was "armed".

MOUSE OFF

Turns OFF mouse interrupt checking.

MOUSE STOP

Delays action of the mouse button until MOUSE ON statement is encountered. The MOUSE ON interrupts is put on hold until a MOUSE ON command is later executed. Branching then takes place immediately if a mouse button was depressed.

ON MOUSE GOSUB

Line number 1, line number 2, or line number 3 is executed to start the event trapping. The program line number of a sub routine is executed when its corresponding button is pressed. Mouse button number 1 (the left most) corresponds to line number 1. Line number 2 corresponds to button 2, and line number 3 to button 3.

CALLS MKEY(button#,status)

Status monitors the status of mouse button #.

```
-1 button was pressed only once
 0 button is not being or has not been pressed
 1 button was pressed once since last mousep(n) call
```

CALL MKEYLAST(r,c)

Returns value of row and column during last MKEY(,).

CALL MLOC(dr,dc)

Returns the location of the row and column when a mouse button was last pressed.

CALL MREL(dr,dc)
Returns information of row and column when mouse button was released.

CALL MOUSE(Y,X)
Monitors status of mouse button X. If button X is pressed, Y equals 1. If button X is not not pressed, Y equals 0.

CALL MOUSEDRAG(ON)
Solid line showing motion between mouse presses.

CALL MOUSEDRAG(OFF)
Reverse mouse drag ON command.

CALL HIDEMOUSE
Eliminates mouse cursor.

CALL SEEMOUSE
Displays mouse cursor.

NEW NEW

Format
NEW

Description
The NEW command erases the program currently in memory, so that you can enter a new program.

The NEW command restores the computer to the condition it was in when you selected MYARC Advanced BASIC from the main selection list with the following exceptions:

> Memory allocated by the INIT subprogram is not returned to the memory area available to MYARC Advanced BASIC.
>
> Assembly-language subprograms loaded by the LOAD subprogram remain in memory.

NEW restores all other default values, closes any open files, erases all variable values and names, and cancels any BREAK or TRACE commands in effect.

NEXT **NEXT**

Format
NEXT control-variable

Cross Reference
FOR TO

Description
The NEXT instruction marks the end of a FOR-NEXT loop.

You can use NEXT as either a program statement or a command.

> The control-variable is the same control-variable that appears in the corresponding FOR TO instruction.

The NEXT instruction is always paired with a FOR TO instruction to form a FOR-NEXT loop (see FOR TO).

A NEXT statement cannot be part of an IF THEN statement.

If NEXT is used as a command, it must be part of a multiple-statement line. A FOR TO instruction must precede it on the same line.

Program

The following program illustrates a use of the NEXT statement in lines 130 and 140.

```
100 TOTAL=0
110 FOR COUNT=10 TO 0 STEP -2
120 TOTAL=TOTAL+COUNT
130 NEXT COUNT
140 FOR DELAY=1 TO 100::NEXT DELAY
150 PRINT TOTAL,COUNT;DELAY
RUN
30           -2  101
```

NUMBER **NUMBER**

Format
NUMBER [initial-line-number][,increment]
NUM

Description
The NUMBER command puts the computer in Number Mode, so that it automatically generates line numbers for your program.

If you enter an initial-line-number, the first line number displayed is the one you specify. If you do not specify an initial-line-number, the computer starts with line number 100.

Succeeding line numbers are generated by adding the value of the numeric- expression increment to the previous line number. To specify increment only (without specifying an initial-line-number), you must precede the increment with a comma. The default increment is 10.

If a line number generated by the NUMBER command is the number of a line already in the program in memory, the existing program line is displayed with the line number. To indicate that the displayed line is an existing program line, the prompt symbol (>) that normally appears to the left of the line number is not displayed. When the computer displays an existing program line, you can either edit the line or press ENTER to leave the line unchanged.

If you enter a program line that contains an error, the appropriate error message is displayed, and the same line number appears again, enabling you to retype the line correctly.

If the next line number to be generated is greater than 32767, the computer leaves Number Mode.

To leave Number Mode, press ESC. If the computer is displaying only a line number (that is, a line number not followed by any characters), you can leave Number Mode by pressing ENTER, UP ARROW, DOWN ARROW.

Special Editing Keys in Number Mode

In Number Mode, you can use the editing keys whether you are changing existing program lines or entering new ones.

LEFT ARROW --Pressing LEFT ARROW moves the cursor one character position to the left. When the cursor moves over a character, it does not change or delete it.

RIGHT ARROW --Pressing RIGHT ARROW moves the cursor one character position to the right. When the cursor moves over a character, it does not change or delete it.

INS --Pressing INS enables you to insert characters at the cursor position. Characters that you type are inserted until you press one of the other special editing keys. The character at the cursor position and all characters to the right of the cursor move to the right as you type. You may lose characters if they move so far to the right that they are no longer in the program line.

DEL --Pressing DEL deletes the character in the cursor position. All characters to the right of the cursor move to the left.

ERASE (Ctrl C) --Pressing ERASE erases the program line currently displayed (including the line number). The program line is erased only from the screen, not from memory.

REDO (Alt + F8) --Pressing REDO causes the program line or other text most recently input to be displayed. This line can be especially helpful if you make an error while editing a program line, causing the computer not to accept it. Pressing REDO displays the original line so that you can make corrections without having to retype the entire line. When you press REDO, the computer leaves Number Mode and enters Edit Mode.

ESC (Ctrl +Break) --Pressing ESC causes the computer to leave Number Mode. If you were entering a new program line, it is not accepted. If you were changing an existing program line, any changes that you made are ignored.

ENTER --If you press ENTER when the computer is displaying only a line number (that is, a line number not followed by any characters), the computer leaves Number Mode. If the line number is the number of an existing program line, that program line is not changed or deleted.

If you press ENTER when the computer is displaying a line number followed by a program line, that line is accepted and the next line number is generated. The displayed line may be a new line that you have entered, an existing program line that you have not changed, or an existing program line that you have edited.

UP ARROW --UP ARROW works exactly the same as ENTER in Number Mode.

DOWN ARROW --DOWN ARROW works exactly the same as ENTER in Number Mode.

Example

In the following, what you type is UNDERLINED. Press ENTER after each line. NUM instructs the computer to number starting at 100 with increments of 10.

```
NUM
100 X=4
110 Z=10
120
NUM 110
110 Z=11
120 PRINT (Y+X)/Z
130
NUM 105,5
105 Y=7
110 Z=11
115
LIST
100 X=4
105 Y=7
110 Z=11
120 PRINT (X+Y)/Z
```

NUM 110 instructs the computer to number starting at 110 with increments of 10. Change line 110 to Z=11.

NUM 105,5 instructs the computer to number starting at line 105 with increments of 5. Line 110 already exists.

OLD **OLD**

Format
OLD ["]file-specification["]

Cross Reference
SAVE

Description
The OLD command loads a program from an external storage device into memory.

The file-specification indicates the name of the program to be loaded from the external device. The file-specification, a string constant, can optionally be enclosed in quotation marks.

The program to be loaded can be one of the following:

A saved MYARC Advanced BASIC program.

A file in DISPLAY VARIABLE 80 format, created by the LIST command or a text editing or word processing program.

A specially prepared assembly-language program that executes automatically when it is loaded.

Before the program is loaded, all open files are closed. The program currently in memory is erased after the program begins to load. For more information see "Loading an Existing Program".

Protected and Unprotected Programs

To execute an unprotected MYARC Advanced BASIC program that has been loaded into memory, enter the RUN command when the cursor appears. You can use the LIST command to display the program or any portion of the program.

If the program was saved using the PROTECTED option of the SAVE command, it starts executing automatically when it is loaded. When the program ends (either normally or because of an error) or stops at a breakpoint, it is erased from memory.

Examples
OLD CSI
Displays instructions and then loads into the computer's memory a program from a cassette recorder.

OLD "DSK1.MYPROG"
Loads into the computer's memory the program MYPROG from diskette in disk drive one.

OLD DSK.DISK3.UPDATE85
Loads into the computer's memory the program UPDATE85 from the diskette named DISK3.

ON BREAK

ON BREAK

Format
ON BREAK STOP
ON BREAK NEXT

Cross Reference
BREAK

Description
The ON BREAK statement enables you to specify the action you want the computer to take when either a breakpoint is encountered or CLEAR is pressed.

If you enter the STOP option, or if your program does not include an ON BREAK statement, program execution stops when a breakpoint is encountered or CLEAR is pressed.

If you enter the NEXT option, program execution continues normally (with the next program statement) when a breakpoint is encountered or CLEAR is pressed. If you press CLEAR while the computer is performing an input or an output operation with certain external devices, an error condition occurs, causing the program to halt. When the NEXT option is in effect, pressing CTL-ALT-DEL is the only way to interrupt your program. However, by doing so, you perform a "reboot" of the system therefore erasing the program in memory and causing you to exit from MYARC Advanced BASIC without closing any open files, possibly causing the loss of data in those files.

ON BREAK does not affect a breakpoint that occurs when a BREAK statement with no line-number-list is encountered in a program.

Program

The following program illustrates the use of ON BREAK.

```
100 CALL CLEAR
110 BREAK 150
120 ON BREAK NEXT
130 BREAK
140 FOR A=1 TO 50
150 PRINT "CLEAR IS DISABLED."
160 NEXT A
170 ON BREAK STOP
180 FOR A=1 TO 50
190 PRINT "NOW IT WORKS."
200 NEXT A
```

Line 110 sets a breakpoint at line 150.

Line 120 sets breakpoint handling to go to the next line.

A breakpoint occurs at line 130 despite line 120, because no line number has been specified after BREAK. Enter CONTINUE.

No breakpoint occurs at line 150 because of line 120; CLEAR has no effect during the execution of lines 140 through 160 because of line 120. Line 170 restores the normal use of CLEAR.

ON ERROR **ON ERROR**

Format
ON ERROR STOP
ON ERROR line-number

Cross Reference
ERR, GOSUB, RETURN

Description
The ON ERROR statement enables you to specify the action you want the computer to take if a program error occurs.

If you enter the STOP option, or if your program does not include an ON ERROR statement, program execution stops when a program error occurs.

If you enter a line-number, a program error causes program control to be transferred to the subroutine that begins at the specified line-number. A RETURN statement in the subroutine returns control to a specified program statement.

When an error transfers control to a subroutine, the line-number option is cancelled. If you wish to restore it, your program must execute an ON ERROR line-number statement again.

The ON ERROR line-number statement does not transfer control when the error is caused by a RUN statement.

Program

The following program illustrates a use of ON ERROR.

```
100 CALL CLEAR
110 DATA "A","4","B","C"
120 ON ERROR 190
130 FOR G=1 TO 4
140 READ X$
150 X=VAL(X$)
160 PRINT X;"SQUARED IS";X*X
170 NEXT G
180 STOP
190 REM ERROR SUBROUTINE
200 ON ERROR 230
210 X$="5"
220 RETURN
230 REM SECOND ERROR
240 CALL ERR(CODE,TYPE,SEVER,LINE)
250 PRINT "ERROR";CODE;" IN LINE";LINE
260 RETURN 170
```

Line 120 causes any error to pass control to line 190.

Line 130 begins a loop. An error occurs in line 150 and control passes to line 190.

Line 200 causes the next error to pass control to line 230.

Line 210 changes the value of X$ to an acceptable value. Line 220 returns control to the line in which the error occurred (line 150).

The second time an error occurs, the SECOND ERROR subroutine is called because of line 200. Line 240 obtains specific information about the error by using CALL ERR. Line 250 reports the nature of the error, and line 260 returns control to line 170 of the main program, which begins the next iteration of the loop.

When the third error occurs, the message Bad Argument in 150 is displayed because the program does not specify what action to take if another error occurs. Program execution ceases.

ON GOSUB **ON GOSUB**

Format
ON numeric-expression;GOSUB; line-number-list
GOSUB

Cross Reference
GOSUB, RETURN

Description
The ON GOSUB statement enables you to transfer conditional program control to one of several subroutines.

The value of the numeric-expression determines to which of the line numbers in the line-number-list program control is transferred.

If the value of the numeric-expression is 1, program control is transferred to the subroutine that begins at the program statement specified by the first line number in the line-number-list; if the value of the numeric-expression is 2, program control is transferred to the subroutine that begins at the program statement specified by the second line number in the line-number-list; and so on.

If necessary, the value of the numeric-expression is rounded to the nearest integer. The value of the numeric-expression must be greater than or equal to 1 and less than or equal to the number of line numbers in the line-number-list.

The line-number-list consists of one or more line numbers separated by commas. Each line number specifies a program statement at which a subroutine begins.

Use a RETURN statement to return program control to the statement immediately following the ON GOSUB statement that called the subroutine.

To avoid unexpected results, it is recommended that you exercise special care if you use ON GOSUB to transfer control to or from a subprogram or into a FOR-NEXT loop.

Examples

100 ON X GOSUB 1000,2000,300
Transfers control to 1000 if X is 1, 2000 if X is 2, and 300 if X is 3.

100 ON P-4 GOSUB 200,250,300,800,170
Transfers control to 200 if P-4 is 1 (P is 5), 250 if P-4 is 2, 300 if P-4 is 3, 800 if P-4 is 4, and 170 if P-4 is 5.

Program

The following program illustrates a use of ON GOSUB.

```
100 CALL CLEAR
110 DISPLAY AT(11,1):"CHOOSE ONE OF THE FOLLOWING:"
120 DISPLAY AT(13,1):"1 ADD TWO NUMBERS."
130 DISPLAY AT(14,1):"2 MULTIPLY TWO NUMBERS."
140 DISPLAY AT(15,1):"3 SUBTRACT TWO NUMBERS."
150 DISPLAY AT(16,1):"4 EXIT PROGRAM."
160 DISPLAY AT(20,1):"YOUR CHOICE:"
170 DISPLAY AT(22,2):"FIRST NUMBER."
180 DISPLAY AT(23,1):"SECOND NUMBER."
190 CALL MARGIN(3,30,1,24)
200 ACCEPT AT(20,14)VALIDATE(DIGIT):CHOICE
210 IF CHOICE<1 OR CHOICE>4 THEN 200
220 IF CHOICE=4 THEN STOP
230 ACCEPT AT(22,16)VALIDATE(NUMERIC):FIRST
240 ACCEPT AT(23,16)VALIDATE(NUMERIC):SECOND
250 CALL MARGIN(3,30,1,8)
260 ON CHOICE GOSUB 280,300,320
270 GOTO 190
280 DISPLAY AT(3,1)ERASE ALL:FIRST;"PLUS";SECOND;"EQUALS";FIRST+SECOND
290 RETURN
300 DISPLAY AT(3,1)ERASE ALL:FIRST;"TIMES";SECOND;"EQUALS";FIRST*SECOND
310 RETURN
320 DISPLAY AT(3,1)ERASE ALL:FIRST;"MINUS";SECOND;"EQUALS";FIRST-SECOND
330 RETURN
```

Line 260 determines where to go according to the value of CHOICE.

ON GOTO **ON GOTO**

Format
ON numeric-expression GOTO line-number-list
GOTO

Cross Reference
GOTO

Description
The ON GOTO statement enables you to transfer unconditional program control to one of several program statements.

The value of the numeric-expression determines to which of the line numbers in the line-number-list program control is transferred. If the value of the numeric-expression is 1, program control is transferred to the program statement specified by the first line number in the line-number-list; if the value of the numeric-expression is 2, program control is transferred to the program statement specified by the second line number in the line-number-list; and so on.

If necessary, the value of the numeric-expression is rounded to the nearest integer. The value of the numeric-expression must be greater than or equal to 1 and less than or equal to the number of line numbers in the line-number-list.

The line-number-list consists of one or more line numbers separated by commas. Each line number specifies a program statement.

To avoid unexpected results, it is recommended that you exercise care if you use ON GOTO to transfer control to or from a subroutine or a subprogram or into a FOR-NEXT loop.

Examples

100 ON X GOTO 1000,2000,300
Transfers control to 1000 if X is 1, 2000 if X is 2, and 300 if X is 3. The equivalent statement using an IF-THEN-ELSE statement is IF X=1 THEN 1000 ELSE IF X=2 THEN 2000 ELSE IF X=3 THEN 300 ELSE PRINT "ERROR!"::STOP.

100 ON P-4 GOTO 200,250,300,800,170
Transfers control to 200 if P-4 is 1 (P is 5), 250 if P-4 is 2, 300 if P-4 is 3, 800 if P-4 is 4, and 170 is P-4 is 5.

Program

The following program illustrates a use of ON GOTO. Line 260 determines where to go according to the value of CHOICE.

```
100 CALL CLEAR
110 DISPLAY AT(11,1):"CHOOSE ONE OF THE FOLLOWING:"
120 DISPLAY AT(13,1):"1 ADD TWO NUMBERS."
130 DISPLAY AT(14,1):"2 MULTIPLY TWO NUMBERS."
140 DISPLAY AT(15,1):"3 SUBTRACT TWO NUMBERS."
150 DISPLAY AT(16,1):"4 EXIT PROGRAM."
160 DISPLAY AT(20,1):"YOUR CHOICE:"
170 DISPLAY AT(22,2):"FIRST NUMBER:"
180 DISPLAY AT(23,1):"SECOND NUMBER:"
190 CALL MARGIN(3,30,1,24)
200 ACCEPT AT(20,14)VALIDATE(DIGIT):CHOICE
210 IF CHOICE<1 OR CHOICE>4 THEN 200
220 IF CHOICE=4 THEN STOP
230 ACCEPT AT(22,16)VALIDATE(NUMERIC):FIRST
240 ACCEPT AT(23,16)VALIDATE(NUMERIC):SECOND
250 CALL MARGIN(3,30,1,8)
260 ON CHOICE GOTO 270,290,310
270 DISPLAY AT(3,1)ERASE ALL:FIRST;"PLUS";SECOND;"EQUALS";FIRST+SECOND
280 GOTO 190
290 DISPLAY AT(3,1)ERASE ALL:FIRST;"TIMES";SECOND;"EQUALS";FIRST*SECOND
300 GOTO 190
310 DISPLAY AT(3,1)ERASE ALL:FIRST;"MINUS";SECOND;"EQUALS";FIRST-SECOND
320 GOTO 190
```

ON WARNING ON WARNING

Format
```
ON WARNING PRINT
           STOP
           NEXT
```

Description
The ON WARNING statement enables you to specify the action you want the computer to take if a warning condition occurs during the execution of your program.

A warning, a condition caused by invalid input or output, does not normally cause program execution to be terminated.

> If you enter the PRINT option, or if your program does not include an ON WARNING statement, the computer displays a warning message when a warning condition occurs during program execution.
>
> If you enter the STOP option, program execution stops when a warning condition occurs during program execution.
>
> If you enter the NEXT option, program execution continues normally when a warning condition occurs and no warning message is displayed. Normally, execution continues beginning with the next program statement; however, if the cause of the warning is an invalid response to an INPUT statement, program execution continues beginning with that same INPUT statement.

You may have multiple ON WARNING statements in the same program.

Program

The following program illustrates a use of ON WARNING.

```
100 CALL CLEAR
110 ON WARNING NEXT
120 PRINT 120,5/0
130 ON WARNING PRINT
140 PRINT 140,5/0
```

```
150 ON WARNING STOP
160 PRINT 160,5/0
170 PRINT 170
RUN
120          9.99999E+**
140
* WARNING
  NUMERIC OVERFLOW IN 140
             9.99999E+**
160
* WARNING
  NUMERIC OVERFLOW IN 160
```

Line 110 sets warning handling to go to the next line. Line 120 therefore prints the result without any message.

Line 130 sets warning handling to the default, printing the message and then continuing execution. Line 140 therefore prints 140, then the warning, and then continues.

Line 150 sets warning handling to print the warning message and then stop execution. Line 160 therefore prints 160 and the warning message and then stops.

OPEN **OPEN**

Format
OPEN #file-number:file-specification[file-organization[size]]
[,file-type][,open-mode][,record-type[record-length]]

Cross Reference
CLOSE, INPUT, PRINT

Description
The OPEN instruction establishes an association between the computer and an external device, enabling you to store, retrieve, and process data.

The file-number is a numeric-expression having a value between 1 and 255. The file-number is assigned to the external file or device indicated by the file-specification so that input/output processing instructions may refer to the file by its file-number. While a file is open, its file-number cannot be assigned to another file. However, you may have more than one file open to a device at one time. File-number 0 always refers to the keyboard and screen of your computer, and is always open. You cannot open or close file-number 0.

If necessary, the file-number is rounded to the nearest integer.

The file-specification is a string-expression; if you use a string constant, you must enclose it in quotation marks.

Options

The following options may be entered in any order.

The file-organization specifies whether records are to be accessed sequentially or randomly. Enter SEQUENTIAL for sequential access, or RELATIVE for random access. Records in a sequential-access file are read or written in sequence from beginning to end. Records in a random-access (relative-record) file can be accessed in any order (they can be processed randomly or sequentially.)

If you do not specify a file-organization, it is assumed to be SEQUENTIAL.

You can optionally specify the initial size of the file. Size is a numeric- expression, the value of which specifies the initial number of records in the file. Note: The size option cannot be used with all peripherals.

The file-type specifies the format of data in the file.

INTERNAL--The computer transfers data in binary format. This is the most efficient method of sending data.

DISPLAY--The computer transfers data in ASCII format. DISPLAY

files can only use FIXED records of 64 or 128. If no file-type is specified in OPEN, the default is DISPLAY.

DISPLAY type files require a special kind of output record. Each element in the PRINT field must be separated by a comma enclosed in quotation marks. The comma serves as a field separator in the file. The omission of this comma causes an I/O error. Note: This is not the same as a print separator, which must be inserted between an element in the PRINT field and the field separator.

The open-mode specifies the input/output operations that can be performed on the file.

INPUT--The computer can only read data from the file.

OUTPUT--The computer can only write data to the file.

UPDATE--The computer can both read from and write to the file.

APPEND--The computer can only write data and only at the end of the file; records already in the file cannot be accessed.

If you open an existing file for OUTPUT, the data items you write to the file replace those currently in the file.

If you do not specify an open-mode, it is assumed to be an UPDATE.

The record-type specifies whether the records in the file are FIXED (all of the same length) or VARIABLE (of various lengths).

SEQUENTIAL files can have FIXED or VARIABLE records. If you do not specify the record-type of a SEQUENTIAL file, it is assumed to be VARIABLE.

RELATIVE files must have FIXED records. If you do not specify the record-type of a RELATIVE file, it is assumed to be FIXED.

You can optionally specify the length of records in the file. Record-length is a numeric-expression, the value of which specifies the fixed size (for FIXED records) or maximum size (for VARIABLE records) of each record.

If you do not specify a record-length, its value is supplied by the peripheral.

If you open a file that does not exist, a file is created with the options you specify. If you open a file that does exist, the options you specify must be the same as the options that you specified when you created the file, except that a file with FIXED records can be opened as either SEQUENTIAL or RELATIVE, regardless of the file-organization that you specified when you created the file.

For more information about the options available with a particular device, refer to the owner's manual that comes with that device.

Examples

```
100 OPEN #1:"CS1",OUTPUT,FIXED
```
Opens a file on cassette. The file is SEQUENTIAL, with data stored in DISPLAY format. The file is opened in OUTPUT mode with FIXED length records of 64 bytes.

```
300 OPEN #23:"DSK.MYDISK.X",RELATIVE 100,INTERNAL,FIXED,UPDATE
```
Opens a file named "X". The file is on the diskette named MYDISK in whichever drive that diskette is located. The file is RELATIVE, with data kept in INTERNAL format with FIXED length records of 80 bytes. The file is opened in UPDATE mode and starts with 100 records made available for it.

```
100 OPEN #234:A$,INTERNAL
```
Where A$ equals "DSK2.ABC", assumes a file on the diskette in drive 2 with a name of ABC. The file is SEQUENTIAL, with data kept in INTERNAL format. The file is opened in UPDATE mode with VARIABLE length records that have a maximum length of 80 bytes.

Program

The following program illustrates a use of the SIZE option in an OPEN statement.

```
100 OPEN #1:"DSK1.LARGE",RELATIVE
110 PRINT #1,REC 100:0
120 CLOSE #1
130 OPEN #1:"DSK1.LARGE",SEQUENTIAL,FIXED
200 CLOSE #1
```

Line 100 opens a RELATIVE file on diskette.

Line 110 writes to the 100th record, thereby reserving space for 100 contiguous records.

Line 120 closes the file.

Line 130 reopens the file, this time with SEQUENTIAL file organization.

Line 200 closes the file.

OPTION BASE **OPTION BASE**

Format
OPTION BASE 0
1

Cross Reference
DIM

Description
The OPTION BASE statement enables you to set the lower limit of array subscripts.

You can use the OPTION BASE statement to specify a lower array-subscript limit of either 0 or 1. If your program does not include an OPTION BASE statement, the lower limit is set to 0.

The OPTION BASE statement applies to every array in your program. You can have only one OPTION BASE statement in a program.

If you do not set the lower array-subscript limit to 1, the computer reserves memory for element 0 of each dimension of each array. To avoid reserving unnecessary memory, it is recommended that you set the lower limit to 1 if your program does not use element 0.

The OPTION BASE statement must have a lower line number than any DIM statement or any reference to an array in your program. The OPTION BASE statement is evaluated during pre-scan and is not executed.

The OPTION BASE statement cannot be part of an IF THEN statement.

Example

100 OPTION BASE 1
Sets the lowest allowable subscript of all arrays to one.

OUT **OUT**

Format
CALL OUT(port,databyte[,databyte...])

Cross Reference
INP, INPUT$

Description
You may use CALL OUT either as a program statement or a command.

Sends a databyte to a port.

Port may be an integer from 1 to 65535.

The dataByte may be any integer between 0 and 255.

Data is received and sent internally through various components within the computer, known as ports.

The OUT statement is used to obtain direct control of a device such as the keyboard, sound, etc.

OUT is the complement function to the INP command.

PATTERN --Subprogram **PROGRAM**

Format
CALL PATTERN(#sprite-number,character-code[...])

Cross Reference
CHAR, MAGNIFY, SPRITE

Description
The PATTERN subprogram enables you to change the pattern of one or more sprites.

The sprite-number is a numeric-expression whose value specifies the number of the sprite as assigned in the SPRITE subprogram.

Character-code is a numeric-expression with a value from 0-255, specifying the character number of the character you want to use as the pattern for a sprite.

If you use the MAGNIFY subprogram to change to double-sized sprites, the sprite definition includes the character specified by the character-code and three additional characters (see MAGNIFY.)

Program

The following program illustrates a use of the Pattern subprogram.

```
100 CALL CLEAR
110 CALL COLOR(12,16,16)
120 FOR A=19 TO 24
130 CALL HCHAR(A,1,120,32)
140 NEXT A
150 A$="0107182121414 1FFFF4141212119070080E09884848282FFFF8282848498E000"
160 B$="01061820305C4681814246242C18070080601834246242818162 3A0C0418E000"
170 C$="0106182C2446428181465C3020180700806018040C3A628181426224 3418E000"
180 CALL CHAR(244,A$,248,B$,252,C$)
190 CALL SPRITE(#1,244,5,130,1,0,8)
200 CALL MAGNIFY(3)
210 FOR A=244 TO 252 STEP 4
220 CALL PATTERN(#1,A)
230 FOR DELAY=1 TO 15 :: NEXT DELAY
240 NEXT A
250 GOTO 210
(Press CLEAR to stop the program.)
```

Lines 110 through 140 build a floor.

Lines 150 through 180 define characters 244 through 255.

Line 190 creates a sprite in the shape of a wheel and starts it moving to the right.

Line 200 makes the sprite double-sized.

Lines 210 through 250 make the spokes of the wheel appear to move as the character displayed is changed.

PEEK --Subprogram--Peek at CPU RAM **PEEK**

Format
CALL PEEK(address,numeric-variable-list[,"",address,
numeric-variable-list[,...]])

Cross Reference
LOAD, PEEKV, POKEV, VALHEX

Description
The PEEK subprogram enables you to ascertain the contents of specified CPU memory addresses.

You can use the PEEKV subprogram to ascertain the contents of VDP memory.

The address is a numeric-expression whose value specifies the first CPU (Central Processing Unit) memory address at which you want to peek.

The address must have a value from -32768 to 32767 inclusive.

You can specify an address from 0 to 32767 inclusive by specifying the actual address.

You can specify an address from 32768 to 65535 inclusive by subtracting 65536 from the actual address. This will result in a value from -32768 to -1 inclusive.

If you know the hexadecimal value of the address, you can use the VALHEX function to convert it to a decimal numeric-expression, eliminating the need for manual calculations.

If necessary, the address is rounded to the nearest integer.

The numeric-variable-list consists of one or more numeric-variables separated by commas. Bytes of data starting from the specified CPU memory address are assigned sequentially to the numeric-variables in the numeric-variable-list.

One byte, with a value from 0 to 255 inclusive, is returned to each specified numeric-variable.

You can specify multiple addresses and numeric-variable-lists by entering a null string (two adjacent quotation marks) as a separator between a numeric-variable-list and the next address.

If you call the PEEK subprogram with invalid parameters, the computer may function erratically or cease to function entirely. If this occurs, turn off the computer, wait several seconds, and then turn the computer back on again.

Examples

```
100 CALL PEEK(8192,X1,X2,X3,X4)
```
Returns the values in memory locations 8192, 8193, 8194, and 8195 in the variables X1, X2, X3, and X4, respectively.

```
100 CALL PEEK(22433,A,B,C,"",-4276,X,Y,Z)
```
Returns the values in locations 22433, 22434, and 22435 in A, B, C, respectively; and the values in locations 61260, 61261, and 61263 in X, Y, and Z, respectively.

```
100 CALL PEEK(VALHEX("4F55"),V1,V2,V3)
```
Uses VALHEX to ascertain the decimal equivalent of the hexidecimal number 4F55, which is 20309. Then the values in locations 20309, 20310, and 20311 are returned in V1, V2, and V3, respectively.

Program

The following program returns in A the number of the highest numbered sprite (#15) currently in use. A zero is returned to B, because no sprites are defined after the DELSPRITE statement.

```
100 CALL CLEAR
110 CALL SPRITE(#15,33,7,100,100,0,0)
120 CALL PEEK(VALHEX("837A"),A)
130 CALL DELSPRITE(ALL)
140 CALL PEEK(VALHEX("837A"),B)
150 PRINT A,B
```

PEEKV --Subprogram--Peek at VDP RAM **PEEKV**

Format
CALL PEEKV(address,numeric-variable-list[,"",address,
numeric-variable-list[,...])

Cross Reference
LOAD, PEEK, POKEV, VALHEX

Description
The PEEKV subprogram enables you to ascertain the contents of specified VDP memory addresses. You can use the PEEK subprogram to ascertain the contents of CPU memory.

The address is a numeric-expression whose value specifies the first VDP (Video Display Processor) memory address at which you want to peek.

The address must have a value from 0 to 16383 inclusive.

If you know the hexadecimal value of the address (0000-3FFF), you can use the VALHEX function to convert it to a decimal numeric-expression.

If necessary, the address is rounded to the nearest integer.

The numeric-variable-list consists of one or more numeric-variables separated by commas. Bytes of data starting from the specified VDP memory address are assigned sequentially to the numeric-variables in the numeric-variable-list.

One byte, with a value from 0 to 255 inclusive, is returned to each specified numeric-variable.

You can specify multiple addresses and numeric-variable-lists by entering a null string (two adjacent quotation marks) as a separator between a numeric-variable-list and the next address.

If you call the PEEKV subprogram with invalid parameters, the computer may function erratically. If this occurs, turn off the computer, wait several seconds, then turn the computer back on.

Example

100 CALL PEEKV(6300,A1,A2,A3)
Returns the values in locations 6300, 6301, and 6302 in A1, A2, and A3, respectively.

Programs

The following program illustrates a use of the PEEKV subprogram.

```
100 CALL CLEAR
110 CALL POKEV(32* 16+12,66)
120 CALL PEEKV(32* 16+12,A)
130 PRINT A
```

Line 110 pokes a "B" into a location that causes it to appear in the middle of the screen. Line 120 peeks at that location, and assigns the value found there (66) to the variable A.

The next program starts a sprite moving diagonally across the screen. Line 120 assigns the values of the row and column coordinates of the sprite to Y and X, respectively.

```
100 CALL CLEAR
110 CALL SPRITE(#1,33,5,100,100,25,25)
120 CALL PEEKV(VALHEX("300"),X,Y)
130 DISPLAY AT(24,1):Y;X
140 GOTO 120
(Press CLEAR to stop the program.)
```

PI --Function--Pi PI

Format
PI

Type
REAL

Description
The PI function returns the value of pi.

The value of pi is 3.14159265359.

Example

```
100 VOLUME=4/3*PI*6^3
```

Sets VOLUME equal to four-thirds times pi times six cubed, which is the volume of a sphere with a radius of six.

POINT --Subprogram **POINT**

Format
CALL POINT(pixel-type,pixel-row,pixel-column[,pixel-row,pixel-column2[,...]])

Cross Reference
CIRCLE, DCOLOR, DRAW, DRAWTO, FILL, GCHAR, GRAPHICS, RECTANGLE

Description
The POINT subprogram enables you to place, or erase specific points (pixels) on the screen, one or more at a time.

Pixel-type is a numeric-expression whose value specifies the action taken by the POINT subprogram.

TYPE	ACTION
2	Reverses the status of the specified point (pixel). (If a pixel is on, it is turned off; if a pixel is off, it is turned on). This effectively reverses the color of the specified pixel.
1	Places a point, of the foreground-color specified by the DCOLOR subprogram, at a specified pixel-row and pixel-column. This is accomplished by turning on the pixel at the designated row and column.
0	Erases a point at a specified pixel-row and pixel-column. This is accomplished by turning on the pixel at the designated row and column.

Pixel-row and pixel-column are numeric-expressions whose values represent the screen position where the point will be placed (turned on or off).

You can optionally place more points by specifying additional sets of pixels.

Pixel-row and pixel-column must be within the range of the particular graphics mode of the screen.

The last pixel-row/pixel-column you specify becomes the current position used by the DRAWTO subprogram.

POINT cannot be used in Pattern or Text Modes.

Example

```
100 CALL POINT(1,96,128)
```
Turns on a single pixel in the center of the screen.

POKEV --Subprogram--Poke to VDP RAM

Format
CALL POKEV(address,byte-list[,"",address,byte-list[,...]])

Cross Reference
LOAD, PEEK, PEEKV, VALHEX

Description
The POKEV subprogram enables you to assign values directly to specified VDP memory addresses.

You can use the LOAD subprogram to assign values to CPU.

The address is a numeric-expression whose value specifies the first VDP (Video Display Processor) memory address where data is to be poked. If the byte-list specifies more than one byte of data, the bytes are assigned to sequential memory addresses starting with the address you specify.

The address must have a value from 0 to 16383 inclusive.

If you know the hexadecimal value of the address (0000-3FFF), you can use the VALHEX function to convert it to a decimal numeric-expression.

If necessary, the address is rounded to the nearest integer.

The byte-list consists of one or more bytes of data, separated by commas, that are to be poked into VDP memory starting with the specified address.

Each byte in the byte-list must be a numeric-expression with a value from 0 to 32767. If the value of a byte is greater than 255, it is repeatedly reduced by 256 until it is less than 256. If necessary, a byte is rounded to the nearest integer.

You can specify multiple addresses and byte-lists by entering a null string (two adjacent quotation marks) as a separator between a byte-list and the next address.

If you call the POKEV subprogram with invalid parameters the computer may function erratically. If this occurs, turn off the computer, wait several seconds, then turn the computer back on.

Examples

```
100 CALL POKEV(3333,233)
```
Pokes the value 233 into location 3333.

```
100 CALL POKEV(13784,273)
```
Pokes the value 17 (273 reduced by 256 once) into location 13784.

```
100 CALL POKEV(7343,246,"",VALHEX("2E4F"),433)
```

Pokes the value 246 into location 7343, and uses VALHEX to ascertain the decimal equivalent of the hexadecimal number 2E4F (11855). The value 177 (433 reduced by 256 once) is then poked into this location.

Program

The following program uses POKEV to display on the screen the characters that correspond to ASCII codes 65 through 208, at the location specified by the value of R* 32+C.

```
100 CALL CLEAR::X=65
110 FOR R=0 TO 23
120 FOR C=0 TO 31 STEP 6
130 CALL POKEV(R* 32+C,X)
140 X=X+1
150 NEXT C
160 NEXT R
```

POS --Function--Position POS

Format
POS(string-expression,substring,numeric-expression)

Type
DEFINT

Description
The POS function returns the position of the first occurrence of a substring within a specified string.

The string-expression specifies the string within which you are seeking the substring. If you use a string constant, it must be enclosed in quotation marks.

The substring is the segment (of the string-expression) you are trying to locate. The substring is a string-expression; if you use a string constant, it must be enclosed in quotation marks.

The value of the numeric-expression specifies the character position in the string-expression where the search for the substring begins.

If necessary, the value of the numeric-expression is rounded to the nearest integer.

If the substring is present within the string-expression, POS returns the number of the character position (within the string-expression) of the first character of the substring.

If the substring is not present, or if the value of the numeric-expression is greater than the number of characters in the string-expression, POS returns a zero.

Examples

```
100 X=POS("PAN","A",1)
```
Sets X equal to 2 because A is the second letter in PAN.

```
100 Y=POS("APAN","A",2)
```
Sets Y=3 because the A in the third position in APAN is the first occurrence of A in the portion of APAN that was searched.

```
100 Z=POS("PAN","A",3)
```
Sets Z equal to 0 because A was not in the part of PAN that was searched.

```
100 R=POS("PABNAN","AN",1)
```
Sets R equal to 5 because the first occurrence of AN starts with the A in the fifth position in PABNAN.

Program

The following program illustrates a use of POS. Input is searched for spaces, and is then printed with each word on a single line.

```
100 CALL CLEAR
110 PRINT "ENTER A SENTENCE."
120 LINPUT X$
130 S=POS(X$," ",1)
140 IF S=0 THEN PRINT X$::PRINT::GOTO 110
150 Y$=SEG$(X$,1,S)::PRINT Y$
160 X$=SEG$(X$,S+1,LEN(X$))
170 GOTO 130
(Press CLEAR to stop the program.)
```

POSITION --Subprogram **POSITION**

Format
CALL POSITION(#sprite-number,numeric-variable1,numeric-variable2[,...])

Cross Reference
SPRITE

Description
The POSITION subprogram enables you to ascertain the current position of one or more sprites.

The sprite-number is a numeric-expression whose value specifies the number of the sprite as assigned in the SPRITE subprogram.

The current screen position of a sprite is returned as two numeric-variables representing the pixel-row and pixel-column, respectively, specifying the position of a screen pixel.

The screen position of the pixel in the upper-left corner of a sprite is considered to be the position of that sprite.

Note that a sprite in motion continues to move during and following the execution of the POSITION subprogram. Remember to allow for this continued motion in your program.

Example

```
100 CALL POSITION(#1,Y,X)
```
Returns the position of the upper left corner of sprite #1. Also see the third example of the SPRITE subprogram.

PPR

PPR

Description
Prints current printer device-name to screen or current output device.

PRINT

PRINT

Format
Print to the Screen
PRINT [print-list]
Print to a File (or Device)
PRINT #file-number[,REC record-number][:print-list]

Cross Reference
DISPLAY, OPEN, PRINT USING, TAB

Description
The PRINT instruction enables you to display data items on the screen or print them to an external device. You can use PRINT as either a program statement or a command.

The print-list consists of one or more print items (items to be printed or displayed) separated by print separators. A PRINT instruction without a print-list advances the print position to the first position of the next record. This has the effect of printing a blank record, unless the preceding PRINT instruction ended with a print-separator.

The numeric- and/or string-expressions in the print-list can be constants and/or variables.

Print items are the numeric- and string-expressions to be printed. Any function is also a valid print item.

Print separators are the punctuation (commas, semicolons, and colons) between print items specifying the placement of the print items in the print record.

Printing to the Screen

Each print item is displayed in the row of the screen window defined by the margins, starting from the far left column of the window. Before a new line is displayed at the bottom of the window, the entire contents of the window (excluding sprites) scroll up one line to make room for the new line. The contents of the top line of the window scroll off the screen and are discarded.

Each line on the screen is treated as one print record. The record length of the screen is the width of the window.

Printing to a File

If you include an optional file-number, the print-list is sent to the specified device. The file-number is a numeric-expression whose value specifies the number of the file as assigned in its OPEN instruction. You cannot print to a file opened in INPUT mode.

If you do not specify a file-number (or if you specify file-number 0), the print-list is displayed on the screen.

If you use the REC option, the record-number is a numeric-expression whose value specifies the number of the record in which you want to print the print-list. The records in a file are numbered sequentially, starting with zero. The REC option can be used only with a file opened for RELATIVE access.

If you print to a file opened in INTERNAL format with FIXED records, each record is filled with trailing binary zeros, if necessary, to bring it to its specified length. If a record is longer than the record length of the file, it is truncated (extra characters are discarded).

For more information about printing to a particular device, refer to the owner's manual that comes with that device.

Printing Numbers: INTERNAL Files

The amount of memory space allocated to a number printed to a file opened in INTERNAL format varies according to its data-type. A DEFINT is always allocated 3 bytes, whereas a REAL number is always allocated 9 bytes.

Note that if you print a DEFINT value to a file, you cannot access that file on a Home Computer that does not support the INTEGER data-type. You can circumvent this by converting all DEFINT variables and functions to REAL variables before printing them to a file.

Printing Numbers: The Screen and DISPLAY Files

The format of a number printed to the screen or to a file opened in DISPLAY format varies according to the characteristics of the number.

Positive numbers and zero are printed with a leading space (instead of a plus sign); negative numbers are printed with a leading minus sign. All numbers are printed with a trailing space.

Numbers are printed in either decimal form or scientific notation, according to these rules:

All numbers with 10 or fewer digits are printed in decimal form.

REAL numbers with more than 10 digits are printed in scientific notation only if they can be presented with more significant digits in scientific notation than in decimal form. If printed in decimal form, all digits beyond the tenth are omitted.

If a number is printed in decimal form, the following rules apply:

DEFINT numbers and REAL numbers with no decimal portion are printed without decimal points.

REAL numbers are printed with decimal points in the proper position. If the number has more than 10 digits, it is rounded to 10 digits. A zero is not printed by itself to the left of the decimal point. Trailing zeros after the decimal point are omitted.

If number is printed in scientific notation, the following rules apply:

The format is mantissaEexponent.

The mantissa is printed with six or fewer digits, with one digit to the left of the decimal point.

Trailing zeros are omitted after the decimal point of the mantissa.

If there are more than five digits after the decimal point of the mantissa, the fifth digit is rounded.

The exponent is a two-digit number displayed with a plus or minus sign.

If you attempt to print a number with an exponent greater than 99 or less than -99, the computer prints two asterisks (**) following the sign of the exponent.

Printing Strings

A string constant in a print-list must be enclosed in quotation marks. A quotation mark within a string constant is represented by two adjacent quotation marks.

A string printed to a file opened in INTERNAL format has a length one greater than the length of the string.

When a string is printed to the screen or to a file opened in DISPLAY format, no leading or trailing spaces are added to the string.

Print Separators

At least one print separator must be placed between adjacent print items in the print-list. Valid print separators are the semicolon (;), the colon (:), and the comma (,).

A semicolon (;) print separator causes the next print item to print immediately after the current print item.

A colon (:) print separator causes the next print item to print at the beginning of the next record. Consecutive colons used as print separators must be divided by a space. Otherwise, they are treated as a statement separator symbol.

If you print to the screen or to a file opened in DISPLAY format, a comma (,) print separator causes the next print item to print at the beginning of the next "zone." Print records are divided into 14-character zones; the number of zones in a print record varies according to its record length.

If you print to a file opened in INTERNAL format, a comma print separator has the same effect as a semicolon print separator.

If a print separator would have the effect of splitting the next print item between two records, the print item is moved to the beginning of the following record. However, if discarding the trailing space from a numeric print item allows it to fit in the current record, the number is printed in the current record without its trailing space.

If the print-list ends with a print separator, the computer is placed in a print-pending condition. Unless the next PRINT instruction includes the REC option, it is considered to be a continuation of the current PRINT instruction. RESTORE #file-number terminates a print-pending condition.

If the print-list is not terminated by a print separator, the computer considers the current record complete when all the print items in the print-list are printed. The first print-item in the next PRINT instruction begins in the next record.

Examples

```
100 PRINT
```
Causes a blank line to appear on the display screen.

```
100 PRINT "THE ANSWER IS";A
```
Causes the string constant THE ANSWER IS to be printed on the display screen, followed immediately by the value of ANSWER. If ANSWER is positive, there will be a blank for the positive sign after IS.

```
100 PRINT X:Y/2
```
Causes the value of X to be printed on a line and the value of Y/2 to be printed on the next line.

```
100 PRINT #12,REC 7:A
```
Causes the value of A to be printed on the eighth record of the file that was opened as number 12 with RELATIVE file organization. (Record number 0 is the first record.)

```
100 PRINT #32:A,B,C,
```
Causes the values of A, B, and C to be printed on the next record of the file that was opened as number 32. The final comma creates a pending print-condition. The next PRINT statement directed to file number 32 will print on the same record as this PRINT statement unless it specifies a record, or a RESTORE #32 statement is executed, thereby closing the print-pending print condition.

```
100 PRINT #1,REC 3:A,B
150 PRINT #1:C,D
```
Causes A and B to be printed in record 3 of the file that was opened as number 1. PRINT #1:C,D causes C and D to be printed in record 4 of the same file.

Program

The following program prints out values in various positions on the screen.

```
100 CALL CLEAR
110 PRINT 1;2;3;4;5;6;7;8;9
120 PRINT 1,2,3,4,5,6
130 PRINT 1:2:3
140 PRINT
150 PRINT 1;2;3;
160 PRINT 4;5;6/4
RUN
1 2 3 4 5 6 7 8 9
1          2
3          4
5          6
1
2
3

1 2 3 4 5 1.5
```

PRINT USING **PRINT USING**

Format
Print to the Screen

```
PRINT USING format-string[:print-list]
            line-number
```

Print to a File (or Device)

```
PRINT #file-number[,REC record-number],USING format-string[print-list]
                                                    line-number
```

Cross Reference
IMAGE, PRINT

Description
The PRINT USING instruction enables you to define specific formats for numbers and strings you print.

You can use PRINT USING as either a program statement or a command.

The format-string specifies the print format. The format-string is a string expression; if you use a string constant you must enclose it in quotation marks. See IMAGE for an explanation of format-strings.

You can optionally define a format-string in an IMAGE statement, as specified by the line-number.

See PRINT for an explanation of the print-list print options.

The PRINT USING instruction is identical to the PRINT instruction with the addition of the USING option, except that:

You cannot use the TAB function.

You cannot use any print separator other than a comma (,), except that the print-list can end with a semicolon (;).

If you use PRINT USING to print to a file, the file must have been opened in DISPLAY format.

Examples

```
100 PRINT USING "###.##":32.5
```
Prints 32.50.

```
100 PRINT USING "THE ANSWER IS ###.#":123.98
```
Prints THE ANSWER IS 124.0.

```
100 PRINT USING 185:37.4,-86.2
185 IMAGE ###.#
```
Prints the values of 37.4 and -86.2 using the IMAGE statement in line 185.

RANDOMIZE **RANDOMIZE**

Format
RANDOMIZE[seed]

Cross Reference
RND

Description
The RANDOMIZE instruction varies the sequence of pseudo-random numbers generated by the RND function.

You can use RANDOMIZE as either a program statement or a command.

> The optional seed is a numeric-expression whose value specifies the random number sequence to be generated by RND functions. The first two bytes of the internal representation of the value of the seed determine the random number sequence generated by RND. If the first two bytes of the seed are identical each time you run your program, the same random number sequence is generated. If you do not enter a seed, a different and unpredictable sequence of random numbers is generated by RND each time you run your program.

Program

The following program illustrates a use of the RANDOMIZE statement. It accepts a value for the seed and prints the first 10 values obtained using the RND functon.

```
100 CALL CLEAR
110 INPUT "SEED: ":S
120 RANDOMIZE S
130 FOR A=1 TO 10::PRINT A;RND::NEXT A::PRINT
140 GOTO 110
(Press CLEAR to stop the program.)
```

READ

READ

Format
READ variable-list

Cross Reference
DATA, RESTORE

Description
The READ statement enables you to assign constants (stored within your program in DATA statements) to variables.

The variable-list, consisting of one or more variables separated by commas, specifies the numeric and/or string variables that are to be assigned values. When a READ statement is executed, the variables in its variable-list are assigned values from the data-list of a DATA statement. Unless you use a RESTORE statement to specify otherwise, DATA statements are read in ascending line-number order.

If a data-list does not contain enough values to assign to all the variables, the READ statement assigns values from subsequent DATA statements until all the variables have been assigned a value. If there are no more DATA statements, a program error occurs and the message Data error in line-number is displayed.

If a numeric variable is specified in the variable-list, a numeric constant must be in the corresponding position in the data-list of a DATA statement. If a string variable is specified in the variable-list, either a string or a numeric constant can be in the corresponding position in the DATA statement.

See the DATA statement for examples.

REC --Function--Record Number **REC**

Format
REC(file-number)

Type
DEFINT

Description
The REC function returns a record number reflecting the position of the next record in the specified file.

The file-number is a numeric-expression whose value specifies the number of the file as assigned in its OPEN instruction.

The REC function returns the number of the record in the specified file that is to be accessed by the next PRINT, INPUT, or LINPUT instruction (the next sequential record). (REC always treats a file as if it were being accessed sequentially, even if it has been opened for relative access.)

The records in a file are numbered sequentially starting with zero.

Example

```
100 PRINT REC(4)
```

Prints the position of the next record in the file that was opened as number 4.

Program

The following program illustrates a use of the REC function.

```
100 CALL CLEAR
110 OPEN #1:"DSK1.PROFILE",RELATIVE,INTERNAL
120 FOR A=0 TO 3
130 PRINT #1:"THIS IS RECORD",A
140 NEXT A
150 RESTORE #1
160 FOR A=0 TO 3
170 PRINT REC(1)
180 INPUT #1:A$,B
190 PRINT A$;B
200 NEXT A
210 CLOSE #1
RUN
```

```
0
THIS IS RECORD 0
1
THIS IS RECORD 1
2
THIS IS RECORD 2
3
THIS IS RECORD 3
```

Line 110 opens a file.

Lines 120 through 140 write four records on the file.

Line 150 resets the file to the beginning.

Lines 160 through 200 print the file position and read and print the values at that position.

Line 210 closes the file.

RECTANGLE --Subprogram **RECTANGLE**

Format
CALL RECTANGLE(line-type,pixel-row1,pixel-column1,
pixel-row2,pixel-column2,pixel-row3 ,pixel-column3[,...]])

Cross Reference
CIRCLE, DCOLOR, DRAW, DRAWTO, FILL, GRAPHICS, POINT

Description
The RECTANGLE subprogram enables you to place rectangles of various types and proportions on the screen.

Rectangles may be hollow (only the perimeter of the rectangle is drawn), or solid (both the perimeter and the entire area enclosed by the perimeter is drawn).

Line-type is a numeric-expression whose value specifies the action taken by the RECTANGLE subprogram.

TYPE	ACTION
5	Reverses the status of each pixel of the specified rectangle (solid). (If a pixel is on, it is turned off; if a pixel is off, it is turned on). This effectively reverses the color of the specified rectangle.
4	Draws a rectangle (solid), of the foreground-color specified by the DCOLOR subprogram. This is accomplished by turning on each pixel in the specified rectangle.
3	Erases a rectangle (solid). This is accomplished by turning off each pixel in the specified rectangle.
2	Reverses the status of each pixel in the perimeter of the specified rectangle. (If a pixel is on, it is turned off; if a pixel is off, it is turned on.) This effectively reverses the color of the perimeter.
1	Draws the perimeter of a rectangle, of the foreground-color specified by the DCOLOR subprogram. This is accomplished by turning on each pixel in the specified rectangle.
0	Erases the perimeter of a rectangle. This is accomplished by turning off each pixel in the specified rectangle.

Pixel-row(#), and pixel-column(#), are numeric-expressions whose values represent the screen positions of specific points of the rectangle. There are three points needed to define the rectangle, as shown below.

Pixel-row1 / pixel-column1 specify the TOP LEFT corner of the rectangle.

Pixel-row2 / pixel-column2 specify the TOP RIGHT corner of the rectangle.

Pixel-row3 / pixel-column3 specify the BOTTOM LEFT corner of the rectangle.

All pixel-rows must have a value from 1 to 192. All pixel-columns must have a value from 1 to 256.

Note that the first pixel set (pixel-row1 and pixel-column1) represents the top leftmost point of the rectangle and must have a lower column value than the second pixel set. The second pixel set represents the top rightmost point of the rectangle. In the same manner, the third pixel set, which represents the bottom leftmost point of the rectangle, must have a higher row value than set1 or set2.

If the procedure outlined above is not followed, an error is issued.

You can optionally draw more rectangles by specifying additional sets of pixels. You must specify three sets of pixels for each rectangle.

The bottom-rightmost point of the last rectangle drawn becomes the current position used by the DRAWTO subprogram.

RECTANGLE cannot be used in Pattern or Text Modes.

Program

```
100 CALL GRAPHICS(1,2)
110 CALL RECTANGLE(1,8,80,8,175,134,80)
120 FOR T=1 TO 8 :: CALL RECTANGLE(4,T* 16,100,T* 16,155,T* 16+T-1,100) ::
    NEXT T
130 FOR DELAY=1 TO 2000 :: NEXT DELAY
140 CALL RECTANGLE(3,16,100,16,155,128,100)
150 FOR DELAY=1 TO 2000 :: NEXT DELAY
160 END
```

Line 100 selects a usable graphics mode (and clears the screen).

Line 110 draws a large box on the screen.

Line 120 uses a for-next loop to fill the box with lines of different thickness. (This shows how RECTANGLE could be used to replace DRAW. RECTANGLE is slower, but more versatile.)

Line 130 uses a for-next loop to delay execution of the next statement.

Line 140 clears the lines, but leaves the box to illustrate how RECTANGLE can be used as an eraser.

Line 150 delays the execution of the next statement.
Line 160 ends the program.

REM --Remark **REM**

Format
REM remark
! remark

Description
The REM statement enables you to document your program by including explanatory remarks within the program itself.

You can use any character in a remark.

The length of a REM statement is limited only by the length of a program statement.

A REM statement encountered during program execution is ignored by the computer.

Trailing Remarks

In addition to the REM statement, trailing remarks can be added to the ends of lines in MYARC Advanced BASIC, allowing detailed internal documentation of programs. An exclamation mark (!) begins each trailing remark.

Example

100 REM BEGIN SUBROUTINE
Identifies a section beginning a subroutine.

100 FOR X=1 to 16 ! BEGIN LOOP
Identifies a section beginning a FOR-NEXT loop.

RESEQUENCE **RESEQUENCE**

Format
RESEQUENCE [initial-line-number][,increment]
RES

Description
The RESEQUENCE command assigns new line numbers to all lines in the program currently in memory.

If you enter an initial-line-number, the first line number assigned is one you specify. If you do not specify an initial-line-number, the computer starts with line number 100.

Succeeding line numbers are assigned by adding the value of the numeric-expression increment to the previous line number. Note that to specify an increment only (without specifying an initial-line-number), you must precede the increment with a comma. The default increment is 10.

To ensure that your program continues to function properly, all line-number references within your program are changed to reflect the newly assigned line numbers. (Line numbers mentioned in REM statements are not affected.) If an invalid line-number reference (a reference to a line number that does not exist in your program) is encountered, the computer changes the line-number reference to 32767, without displaying any error message or warning.

If the values you enter for the initial-line-number and increment would have the effect of creating a line number greater than 32767, the message Bad line number is displayed and the program is not resequenced.

Examples

RES
Resequences the lines of the program in memory to start with 100 and number by 10s.

RES 1000
Resequences the lines of the program to start with 1000 and number by 10s.

RES 1000,15
Resequences the lines of the program in memory to start with 1000 and number by 15s.

RES ,15
Resequences the lines of the program in memory to start with 100 and number by 15s.

RESTORE **RESTORE**

Format
Restore Data
RESTORE [line-number]
Restore a File
RESTORE #file-number[,REC record-number]

Cross Reference
DATA, INPUT, PRINT, READ

Description
The RESTORE instruction specifies either the DATA statement to be used with the next READ statement or the record to be accessed by the next file-processing instruction.

RESTORE with DATA and READ Statements

If you enter a line-number, the next READ statement executed assigns values beginning from the data-list in the specified DATA statement.

If the specified line-number is not the line-number of a DATA statement, the computer uses the first DATA statement with a line-number higher than the one you specified.

If there is no higher numbered DATA statement, a program error occurs and the message Data error in line-number is displayed (the line-number is the line number of the READ statement that caused the error).

If you do not enter a line-number or a file-number, the next READ statement executed assigns values beginning from the data-list of the first DATA statement in your program.

If there are no DATA statements in your program, the message Data error in line-number is displayed.

RESTORE with a File

If you enter a file-number, RESTORE repositions the specified file at its first record, record zero (unless you use the REC option). The file-number is a numeric-expression whose value specifies the number of the file as assigned in its OPEN instruction.

If you use the REC option, the record-number is a numeric-expression specifying the number of the record at which you want to position the file. The records in a file are numbered sequentially, starting with zero. The REC option can be used only with a file opened for RELATIVE access.

RESTORE terminates any print- or input-pending conditions.

Examples

```
100 RESTORE
```
Sets the next DATA statement to be used to the first DATA statement in the program.

```
100 RESTORE 130
```
Sets the next DATA statement to be used to the DATA statement at line 130 or, if line 130 is not a DATA statement, to the next DATA statement after line 130.

```
100 RESTORE #1
```
Sets the next record to be used by the next PRINT, INPUT, or LINPUT statement using file #1 to be the first record in the file.

```
100 RESTORE #4,REC H5
```
Sets the next record to be used by the next PRINT, INPUT, or LINPUT statement using file #4 to be record H5.

RETURN **RETURN**

Format
With GOSUB and ON GOSUB
     RETURN
With ON ERROR
     RETURN [NEXT
            line-number]

Cross Reference
GOSUB, ON GOSUB, ON ERROR

Description
The RETURN statement causes program control to return to the main program from a subroutine called by a GOSUB, ON GOSUB, or ON ERROR statement.

RETURN with GOSUB and ON GOSUB

When the computer encounters a RETURN statement in a subroutine called by a GOSUB or ON GOSUB statement, program control returns to the statement immediately following the GOSUB or ON GOSUB statement.

No options are allowed with a RETURN statement in a subroutine called by a GOSUB or ON GOSUB statement.

RETURN with ON ERROR

The action taken by the computer when it encounters a RETURN statement in a subroutine called by an ON ERROR statement depends on the RETURN option.

     If you specify the NEXT option, program control returns to the statement immediately following the statement that caused the error.

     If you specify a line-number, program control is transferred to the specified program statement.

     If you do not specify an option, program control returns to the statement that caused the error. The statement is re-executed.

RETURN "clears" the error, so that it can no longer be analyzed by the ERR subprogram.

Programs

The following program illustrates a use of RETURN as used with GOSUB. The program figures interest on an amount of money put into savings.

```
100 CALL CLEAR
110 INPUT "AMOUNT DEPOSITED: ":AMOUNT
120 INPUT "ANNUAL INTEREST RATE: ":RATE
130 IF RATE 1 THEN RATE=RATE* 100
140 PRINT "NUMBER OF TIMES COMPOUNDED"
```

```
150 INPUT "ANNUALLY: "COMP
160 INPUT "STARTING YEAR: ":Y
170 INPUT "NUMBER OF YEARS: ":N
180 CALL CLEAR
190 FOR A=Y TO Y+N
200 GOSUB 240
210 PRINT A,INT(AMOUNT* 100+.5)/100
220 NEXT A
230 STOP
240 FOR B=1 TO COMP
250 AMOUNT=AMOUNT+AMOUNT*RATE/(COMP* 100)
260 NEXT B
270 RETURN
```

The following program illustrates a use of RETURN with ON ERROR.

```
100 CALL CLEAR
110 A=1
120 ON ERROR 160
130 X=VAL("D")
140 PRINT 140
150 STOP
160 REM ERROR HANDLING
170 IF A>4 THEN 220
180 A=A+1
190 PRINT 190
200 ON ERROR 160
210 RETURN
220 PRINT 220 :: RETURN NEXT
RUN

190
190
190
190
220
140
```

Line 120 causes an error to transfer control to line 160. Line 130 causes an error.

Line 170 checks to see if the error has occurred four times and transfers control to 220 if it has. Line 180 increments the error counter by one. Line 190 prints 190. Line 200 resets the error handling to transfer to line 160. Line 210 returns to the line that caused the error and executes it again.

Line 220, which is executed only after the error has occurred four times, prints 220 and returns to the line following the line that caused the error.

Line 140, the next one after the one that causes the error, prints 140.

See also example of the ON ERROR statement.

RIGHT$** **RIGHT$

Format
RIGHT$(string-expression,length)

Cross Reference
LEFT$, POS, STR$

Description
RIGHT$ returns the right-most "length" of characters from the string expression. If the string-expression is shorter than the length, the actual string-expression will be returned.

Example

```
10 A$="MY NAME IS HARRY POTTER"
20 PRINT RIGHT$(A$,12)
RUN
HARRY POTTER
```

RND --Function--Random Number **RND**

Format
RND

Type
REAL

Cross Reference
RANDOMIZE

Description
The RND function returns a pseudo-random number.

RND returns the next pseudo-random number in the current series of pseudo-random numbers. The number returned is always greater than or equal to 0 and less than 1.

The numbers returned by RND are called "pseudo-random" because they are not generated strictly at random, but are generated as members of predefined series. You can use the RANDOMIZE instruction to make the numbers generated by RND more random.

The same sequence of random numbers is generated by RND each time you run a particular program unless the program includes a RANDOMIZE instruction.

Examples

```
100 COLOR16=INT(RND* 16)+1
```
Sets COLOR16 equal to some number from 1 through 16.

```
100 VALUE=INT(RND* 16)+10
```
Sets VALUE equal to some number from 10 through 25.

```
100 LL(8)=INT(RND*(B-A+1))+A
```
Sets LL(8) equal to some number from A through B.

RPT$** --Function--Repeat String **RPT$

Format
RPT$(string-expression,numeric-expression)

Type
String

Description
The RPT$ function returns a string consisting of a specified string repeated a specified number of times.

The string-expression specifies the string to be repeated. If you use a string constant, it must be enclosed in quotation marks.

The value of the numeric-expression specifies the number of repetitions of the string-expression.

If the length of the string-expression and the value of the numeric-expression would create a string longer than 255 characters, the excess characters are discarded and the following message is displayed:

```
*WARNING
 STRING TRUNCATED
```

Examples

```
100 M$=RPT$("ABCD",4)
```
Sets M$ equal to "ABCDABCDABCDABCD".

```
100 CALL CHAR(244,RPT$("0000FFFF",8))
```
Defines characters 244 through 247 with the string
"0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF".

```
100 PRINT USING RPT$("#",40):X$
```
Prints the value of X$ using an image that consists of 40 number signs.

Format
Execute Program in Memory
RUN [line-number]
Execute Program on External Device
RUN file-specification[,Continue]

Description
The RUN instruction causes the computer either to execute the program currently in memory or to both load and execute a program from an external. You can use RUN as either a program statement or a command.

When you use RUN as a program statement, one program can start the execution of another program. This enables you to divide a large program into smaller segments, each of which can be loaded into memory only as needed.

If you specify a line-number, your program starts running at the specified program line.

If you enter a file-specification, your program is first loaded into memory from the specified external device, and then executed starting from the lowest-numbered line in the program. The file-specification is a string expression; if you use a string constant, you must enclose it in quotation marks. If you additionally specify the Continue option, the new program loaded must contain only variables used in the previous program. A syntax error will occur when trying to use a variable not contained in the previous program.

If you do not enter either a line-number or a file-specification, the computer executes the program currently in memory starting with the lowest-numbered line in the program.

Before the program starts running, the computer:

Sets the values of all numeric variables to zero.

Sets the values of all string variables to null strings (strings containing no characters).

Closes all open files.

Restores the default screen color (cyan).

Deletes all sprites.

Resets the sprite magnification level to 1.

Checks for certain program errors.

RUN does not affect the graphics mode, margin settings, graphics colors (see DCOLOR), or current position (see DRAWTO).

Examples

RUN
Causes the computer to begin execution of the program in memory.

RUN 200
100 RUN 200
Causes the computer to begin execution of the program in memory starting at line 200.

RUN "DSK1.PRG3"
100 RUN "DSK1.PRG3"
Causes the computer to load and begin execution of the program named PRG3 from the diskette in disk drive 1.

100 A$="DSK1.MYFILE"
110 RUN A$
Causes the computer to load and begin execution of the program named MYFILE from the diskette in disk drive 1.

Program

The following program illustrates a use of the RUN command used as a statement. It creates a "menu" and lets the person using the program choose what other program he wishes to run. The other programs should RUN this program rather than ending in the usual way, so that the menu is given again after they are finished.

```
100 CALL CLEAR
110 PRINT "1 PROGRAM 1."
120 PRINT "2 PROGRAM 2."
130 PRINT "3 PROGRAM 3."
140 PRINT "4 END."
150 PRINT
160 INPUT "YOUR CHOICE: "C
170 IF C=1 THEN RUN "DSK1.PRG1"
180 IF C=2 THEN RUN "DSK1.PRG2"
190 IF C=3 THEN RUN "DSK1.PRG3"
200 IF C=4 THEN STOP
210 GOTO 100
```

SAVE

SAVE

Format
SAVE file-specification[,MERGE
PROTECTED]

Cross Reference
MERGE, OLD

Description
The SAVE command copies the program in memory to an external storage device. When you are using SAVE, your program remains in memory, even if an error occurs.

The saved program can later be loaded back into memory with the OLD command.

The file-specifications names the program to be stored The file-specification, a string constant, optionally can be enclosed in quotation marks.

To specify that your program is to be available for merging with other programs, use the MERGE option. If you use the MERGE option, the program is stored as a SEQUENTIAL file in DISPLAY format with VARIABLE records (see OPEN); MERGE can be used only with devices that accept these options.

For more information about using MERGE with a particular device, refer to the owner's manual that comes with that device.

If you do not use the MERGE option, your program cannot later be merged with another program.

If you use the PROTECTED option, you ensure that the program, when subsequentially loaded with the OLD command, cannot be listed, edited, or saved.

A protected program starts executing automatically when it is loaded; when the program ends (either normally or because of an error) or stops at a breakpoint, it is erased from memory. As the PROTECTED option is not reversible, it is recommended that you keep an unprotected version of the program. If you also wish to protect a diskette-based program from being deleted, use the protect feature of the Disk Manager

SAVE removes any breakpoints you have set in your program.

Examples

SAVE DSK1.PRG1
Saves the program in memory on the diskette in disk drive 1 under the name PRG1.

SAVE DSK1.PRG1,PROTECTED
Saves the program in memory on the diskette in disk drive 1 under the name PRG1. The program may be loaded into memory, but it may not be edited, listed, or resaved.

SAVE DSK1.PRG1,MERGE
Saves the program in memory on the diskette in disk drive 1 under the name PRG1. The program may later be merged with a program in memory by using the MERGE command.

SAY --Subprogram SAY

Format
CALL SAY(word-string[,direct-string][,...])

Cross Reference
SPGET

Description
The SAY subprogram enables you to instruct the computer to produce speech.

Word-string is a string-expression whose value is any of the words or phrases in the computer's resident vocabulary. If you use a string constant, you must enclose it in quotation marks. Alphabetic characters must be upper-case.

The computer substitutes "UHOH" for a word-string not in the vocabulary.

A speech phrase (more than one word) must be enclosed in pound signs (#). A speech phrase must be predefined; that is, it must be resident in the computer's vocabulary.

A compound is a new word formed by combining two words already in the vocabulary. For example, SOME+THING produces "something" and THERE+FOUR produces "therefore". A compound must not be enclosed in pound signs.

See Appendix H for a list of the computer's resident vocabulary.

Direct-string is a string-expression whose value is the computer's internal representation of a word or phrase. You can use or modify a direct-string returned by the SPGET subprogram.

See Appendix I for information on adding suffixes to direct-strings. You can specify multiple word-strings and direct-strings by alternating them. To specify two consecutive word-strings or direct-strings, enter an extra comma as a separator between them.

Examples
```
100 CALL SAY("HELLO, HOW ARE YOU")
```
Causes the computer to say "Hello, how are you".

```
CALL SAY(,A$,,B$)
```
Causes the computer to say the words indicated by A$ and B$, which must have been returned by SPGET.

The following program illustrates a use of CALL SAY with a word-string and three direct-strings.

```
100 CALL SPGET("HOW",X$)
110 CALL SPGET("ARE",Y$)
120 CALL SPGET("YOU",Z$)
130 CALL SAY("HELLO",X$,,Y$,,Z$)
```

SCREEN --Subprogram **SCREEN**

Format
CALL SCREEN([,foreground-color,]background-color)

Cross Reference
COLOR, DCOLOR, GRAPHICS

Description
The SCREEN subprogram enables you to change the screen color. The screen color is the color of the border and the color displayed when transparent is specified as the foreground- or background-color of a character or pixel.

In Text Mode, SCREEN enables you to change the color of the displayed characters, as well as the color of the screen.

Background-color is a numeric-expression whose value specifies a screen color from among the 16 available colors.

In Text Modes, foreground-color is a numeric-expression whose value specifies a color from among the 16 available colors, representing the foreground-color of all 256 characters.

If you specify a foreground-color and the computer is not in Text Mode, it has no effect. If the computer is in Text Mode and you do not specify foreground-color, the foreground-color remains unchanged.

When you enter MYARC Advanced BASIC, the background-color is cyan and the foreground-color is black. When your program ends (either normally or because of an error), stops at a breakpoint, or changes graphics mode, the default colors are restored.

The codes for the available colors are listed in Appendix F.

Examples

```
100 CALL SCREEN(8)
```
Changes the screen to cyan, which is the standard screen color.

```
100 CALL SCREEN(2)
```
Changes the screen to black.

Program

The following program uses CALL SCREEN with CALL VCHAR and PRINT in the Text Mode to change the color of a character.

```
100 CALL CLEAR
110 CALL GRAPHICS(2,1)
120 CALL VCHAR(12,12,33,3)
130 CALL SCREEN(5,16)
140 PRINT "DARK BLUE SCREEN WITH WHITE LETTERS"
150 GOTO 150
(Press CLEAR to stop the program.)
```

Line 130 changes the screen to dark blue and the characters to white.

SEG$** --Function--String Segment **SEG$

Format
SEG$(string-expression, start-position,length)

Type
String

Description
The SEG$ function returns a specified substring (segment of a string).

The string-expression specifies the string of which you want to specify a substring. If you use a string constant, it must be enclosed in quotation marks.

The start-position is a numeric-expression whose value specifies the character position in the string-expression where the substring begins. The value of the start-position must be greater than zero.

The length is a numeric-expression whose value specifies the length of the substring.

If the start-position is greater than the length of the string-expression, or if the length is zero, SEG$ returns a null string.

If the specified length is greater than the remaining length of the string-expression (starting from the specified start-position), SEG$ returns a substring consisting of all characters in the string-expression starting from the start-position to the end of the string-expression.

Examples

```
100 X$=SEG$("FIRSTNAME LASTNAME",1,9)
```
Sets X$ equal to FIRSTNAME.

```
100 Y$=SEG$("FIRSTNAME LASTNAME"11,8)
```
Sets Y$ equal to LASTNAME.

```
100 Z$=SEG$("FIRSTNAME LASTNAME",10,1)
```
Sets Z$ equal to " ".

```
100 PRINT SEG$(A$,B,C)
```
Prints the substring of A$ starting at the character at position B and extending for C characters.

SGN --Function--Signum (Sign) SGN

Format
SGN(numeric-expression)

Type
DEFINT

Description
The SGN function returns a number indicating the algebraic sign of the value of the numeric-expression.

If the value of the numeric-expression is negative, SGN returns a -1.

If the value of the numeric-expression is zero, SGN returns a 0.

If the value of the numeric-expression is positive, SGN returns a (+)1.

Examples

```
100 IF SGN(X2)=1 THEN 300 ELSE 400
```
Transfers control to line 300 if X2 is positive and to line 400 if X2 is zero or negative.

```
100 ON SGN(X)+2 GOTO 200,300,400
```
Transfers control to line 200 if X is negative, line 300 if X is zero, and line 400 if X is positive.

SIN --Function--Sine **SIN**

Format
SIN(numeric-expression)

Type
REAL

Cross Reference
ATN, COS, TAN

Description
The SIN function returns the sine of the angle whose measurement in radians is the value of the numeric-expression.

The value of the numeric-expression cannot be less than -1.5707963267944E10 or greater than 1.5707963267944E10.

To convert the measure of an angle from degrees to radians, multiply pi/180.

Program

The following program gives the sine for each of several angles.

```
100 A=.5235987755982
110 B=30
120 C=45*PI/180
130 PRINT SIN(A);SIN(B)
140 PRINT SIN(B*PI/180)
150 PRINT SIN(C)
RUN
.5   -.9880316241
.5
.7071067812
```

SOUND --Subprogram **SOUND**

Format
CALL SOUND(duration,frequency1,volume1[,frequency2,volume2]
[,frequency3,volume3][,frequency4,volume4])

Description
The SOUND subprogram enables you to instruct the computer to produce musical tones or noise.

The computer contains three music generators and one noise generator, enabling you to create up to four different sounds at once. You can specify the frequency and volume of each sound independently.

Duration is a numeric-expression whose absolute value specifies the length of the sound in milliseconds (thousanths of seconds). Duration can have an absolute value from 1 to 4250. (A value of 1000 will produce a sound for one second.)

The actual duration produced by the computer may vary by as much as one sixtieth (1/60) of a second from the value you specify.

You can enter only one duration, which applies to all specified sounds (music and noise).

Frequency is a numeric-expression that has different meanings depending on whether you use it to specify one of the music generators or the noise generator.

You must enter at least one frequency.

The frequency of a music generator specifies the frequency of the tone in Hertz (cycles per second). The acceptable values range from 110 to 44733; the upper limit exceeds the range of human hearing.

The actual frequency produced by the computer may vary by as much as ten percent from the value you specify.

See Appendix C for the frequencies of some commomly used tones.

The frequency of the noise generator has a value from -1 to -8, specifying the type of noise produced.

The frequencies from -1 to -3 produce different types of periodic noise. A frequency of -4 produces a periodic noise that varies depending on the frequency value of the third music generator.

The frequencies from -5 to -7 produce different types of white noise. A frequency of -8 produces a white noise that varies depending on the frequency value of the third music generator.

Volume is a numeric-expression whose value is inversely proportional to the loudness of the sound.

You must enter at least one volume.

The volume can be from 0 to 30. Zero is the maximum volume and 30 is silence.

If you call SOUND while the computer is still producing the tones specified in a previous call to the SOUND subprogram, the result depends on the algebraic sign of the duration of the previous call to SOUND. If the duration was positive, the new sound does not begin until the old sound is complete. If the duration was negative, the new sound begins immediately, interrupting the old sound.

Examples

```
100 CALL SOUND(1000,110,0)
```
Plays A below low C loudly for one second.

```
100 CALL SOUND(500,110,0,131,0,196,3)
```
Plays A below low C and low C loudly, and G below C not as loudly, all for half a second.

```
100 CALL SOUND(4250,-8,0)
```
Plays loud white noise for 4.250 seconds.

```
100 CALL SOUND(DUR,TONE,VOL)
```
Plays the tone indicated by TONE for a duration indicated by DUR, at a volume indicated by VOL.

Program

The following program plays the 13 notes of the first octave that is available on the computer.

```
100 X=2^(1/12)
110 FOR A=1 TO 13
120 CALL SOUND(100,110*X^A,0)
130 NEXT A
```

SPEED **SPEED**

Format
CALL SPEED(x)

Description
Commmand to set speed of execution of BASIC program.

SPEED = 5	Maximum speed
SPEED = 3	Approx. speed of MYARC Advanced BASIC
SPEED = 2	Approx. speed of TI Extended BASIC
SPEED = 1	Minimum speed - TI 99/4A emulation

SPGET --Subprogram--Get Speech **SPGET**

Format
CALL SPGET(word-string,string-variable[,...])

Cross Reference
SAY

Description
The SPGET subprogram enables you to assign the computer's internal representation of a speech word to a variable.

SPGET is especially useful if you want to add a suffix to a word in the computer's resident vocabulary.

Word-string is a string-expression whose value is any of the words or phrases in the computer's resident vocabulary. If you use a string constant, you must enclose it in quotes.

The computer substitutes "UHOH" for a word-string not in the vocabulary.

A speech phrase (more than one word) must be enclosed in pound signs (#).

See Appendix H for a list of the computer's resident vocabulary.

The internal representation of the word-string (the direct-string) is returned in the string-variable. See Appendix I for information on adding suffixes to direct-strings.

You can specify multiple word-strings and direct-strings by alternating them.

Program

The following program illustrates using CALL SPGET.

```
110 CALL SPGET("COMPUTER",Y$)
120 CALL SAY("I AM A",Y$)
```

SPRITE --Subprogram **SPRITE**

Format
CALL SPRITE(#sprite-number,character-code,foreground-color,
pixel-row,pixel-column[, vertical-velocity,horizontal-velocity][,...])

Cross Reference
CHAR, COINC, COLOR, DELSPRITE, DISTANCE, GRAPHICS, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION, SCREEN

Description
The SPRITE subprogram enables you to create sprites.

Sprites are graphics that can be assigned any valid color and placed anywhere on the screen. Sprites treat the screen as a grid 256 pixels high and 256 pixels wide. However, only the first 192 pixels are visible on the screen.

You can create up to 32 sprites in all graphics modes except Text Modes, which do not allow sprites (the SPRITE subprogram has no effect in Text Modes).

Sprites can be set in motion in any direction at a variety of speeds. A sprite continues its motion until it is specifically changed by the program or until program execution stops. Because sprites move from pixel to pixel, their motion can be smoother than that of characters, which can be moved only one character position (6 or 8 pixels) at a time.

Sprites "pass over" characters on the screen. When two or more sprites are coincident (occupying the same screen pixel position), the sprite with the lowest sprite-number covers the other sprite(s).

At any given time, only four sprites (in Graphics(1,1) and (1,2)) or eight sprites (in the other graphics modes) can be on the same horizontal pixel-row. Once this limit is exceeded the row of pixels in the sprite(s) with the highest sprite-number(s) disappears.

You can use the DELSPRITE subprogram to delete one or more sprites. All sprites are deleted when your program ends (either normally or because of an error), stops at a breakpoint, or changes graphics mode.

Sprite Specifications

The sprite-number is a numeric-expression with a value from 1 to 32. If you specify the value of a previously defined sprite, the old sprite is replaced by the new sprite. If the old sprite had a vertical- or horizontal-velocity and you do not specify a new velocity, the new sprite retains the old velocity.

Character-code is a numeric-expression with a value from 0-255, specifying the character that defines the sprite pattern.

If you use the MAGNIFY subprogram to change to double-sized sprites, the

sprite definition includes the character specified by the character-code and three additional characters (see MAGNIFY).

Once defined by the SPRITE subprogram, the character-code of a sprite can be changed by the PATTERN subprogram.

The foreground-color is a numeric-expression with a value from 1 to 16, specifying one of the 16 available colors. Once defined by the SPRITE subprogram, the foreground-color of a sprite can be changed by the COLOR subprogram.

The background-color of a sprite is always transparent.

The pixel-row and pixel-column are numeric-expressions whose values specify the screen pixel position of the pixel at the upper-left corner of the sprite.

Once defined by the SPRITE subprogram, the pixel-row and pixel-column of a sprite can be changed by the LOCATE subprogram, and the current pixel-row and pixel column of a sprite can be ascertained by the POSITION subprogram. Also, the distance between sprites or between a sprite and a specified screen pixel can be ascertained by the DISTANCE subprogram, and the COINC subprogram can be used to ascertain whether sprites are coincident with each other or with a specified screen pixel.

Sprite Motion

The optional vertical- and horizontal-velocity are numeric-expressions with values from -128 to 127. If both values are zero, the sprite is stationary. The speed of a sprite is in direct linear proportion to the absolute value of the specified velocity.

A positive vertical-velocity causes the sprite to move toward the top of the screen; a negative vertical-velocity causes the sprite to move toward the bottom of the screen.

A positive horizontal-velocity causes the sprite to move to the right; a negative horizontal-velocity causes the sprite to move to the left.

If neither the vertical- nor horizontal-velocity are zero, the sprite moves at an angle, in a direction and at a speed determined by the velocity values.

The velocity of a sprite can be changed by the MOTION subprogram.

When a moving sprite reaches an edge of the screen, it disappears. The sprite reappears in the corresponding position at the opposite edge of the screen.

The motion of a sprite may be affected by the computer's internal processing and by input to, and output from, external devices.