STANDARD REFERENCE GLOSSARY

This reference section is an alphabetical listing of the "Standard ZBasic Commands". The following paragraphs describe the information layout and syntax of this section.

TYPE OF INFORMATION CONTAINED IN THIS REFERENCE SECTION

function	Returns a value; used wherever an expression is used
statement	Executed by itself
command	Used from the standard line editor mode; EDIT, SAVE...
operator	Like AND, OR, XOR or NOT

COMPATIBLE COMMANDS

BLACK BAR Indicates the command is the same on all versions of ZBasic.

SPECKLED BAR Indicates the command may not be available on all versions.
Check to see if your system does not support that command.

PAGE LAYOUT

The pages are layed out in the same way. Whenever possible descriptions are kept to one page. The header has the command type and description. Paragraph layout is:

FORMAT	Correct syntax for that statement, function or command
DEFINITION	Definition or explanation of usage
EXAMPLE	Program example or direct example of usage. Note that linenumbers are usually omitted. Add linenumbers if needed.
REMARK	Other information of importance and usually a reference to other related sections that will aid the understanding of that item.

IMPORTANT NOTE ABOUT DIVIDE

ZBasic compiles divide symbols based on configuration.

If the default expression evaluator; "Optimize Expressions as Integer?" is YES;
/=integer divide \=floating point divide
If the expression evaluator; "Optimize Expressions as Integer?" is NO;
/=floating point divide \=integer divide
See "Configure" and "Converting Old Programs" and "Math expressions" for more information about the options offered for expression types and how they are evaluated.

continued next page...

STANDARD REFERENCE

CROSS REFERENCE

These commands work the same way on almost every version of ZBasic. There is an extensive cross-reference to other commands and how a command works on specific machines. The reference section uses a computer icon to bring attention to a specific version of ZBasic. The following icons are used:

Apple // DOS 3.3 and ProDOS versions.

MSDOS and IBM PC and compatible versions.

The Macintosh versions (all except the 128k machine).

Z80 machines; Amstrad, CP/M-80 2.x and higher, Kaypro Graphics versions and TRS-80 model 1, 3 and 4 versions.

SYNTAX GLOSSARY

GLOSSARY	DEFINITION
RUN or COMMAND	What follows is program or command output.
[brackets]	Items within the brackets are optional (may be omitted)
{ A|B|C }	Any one of A, B or C may be used
... repeats	Three periods following items indicates a repeating sequence
Courier text	Something you type in, a program example, or program output
expression or expr	Numeric: Any; including integer and floating point
byte expression	Numeric: 0-255
word expression	Numeric: 0 to 65,535 or +-32,767
long expression	Numeric: 0 to 4,294,966,293 or +-2,147,483,647
variable or var	Any Variable
var$, var%, var&, var!, var#	String, integer, LongInteger, single or double precision variable types, respectively
"string"	Quoted strings (string constants)
simplestring or string	String variable, string constant, BIN$, CHR$, HEX$, INDEX$, OCT$, PSTR$, STR$, SPACE$, STRING$ or UNS$.
filenumber	File number: An expression 1-99. See "Configure"
filename	A legal filename for that operating system filename
filespec	Drive or storage volume specifier
line	A line number from 0 to 65,534 or a "label"
number	Requires a number. No variable or expression allowed
var name	A valid variable name

Be sure to take note when you see this hand. It is pointing out important information about using that command. If there is the message "Important Note" with the hand it is even more critical that you read the notes.

FORMAT **ABS** (expression)

DEFINITION Returns the absolute value of an expression. The absolute value is the value without regard to the sign (negative, zero or positive).

The result of ABS will always be a positive number or zero.

EXAMPLE

```
A=-15: B=15
PRINT ABS(A), ABS(B), ABS(-555)
X=ABS(0)
PRINT X

RUN

15, 15, 555
0
```

REMARK The SGN function will return the sign of an expression.

AND operator

FORMAT expression$_1$ **AND** expression$_2$

DEFINITION Used to determine if BOTH conditions are true. If both expression1 AND expression2 are true (non-zero), the result is true. Returns -1 for true, 0 for false.

Also used to compare bits in binary number operations. 1 AND 1 return a 1, all other combinations of 0's and 1's produce 0. See truth tables below.

EXAMPLE

```
IF 30>20 AND 20<30 THEN PRINT "TRUE "
IF "Hi"="hello" AND 6-5=1 THEN PRINT "TRUE TOO!"

RUN

TRUE

--------------------------------------------------------------------------------

PRINT BIN$( &X00001111 AND &X11111111)
PRINT 4 AND 255

RUN

0000000000001111
4
```

REMARK See OR, XOR and NOT.

AND TRUTH TABLE

condition AND condition TRUE(-1) if both conditions TRUE, else FALSE(0)

AND
1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0

BOOLEAN "16 BIT" LOGIC

```
   00000001      00000111
AND 00001111  AND 00001111
=   00000001  =   00000111
```

LongInteger will function with this operator in 32 bits.

FORMAT

APPEND line or label ["] filename["]
APPEND* line or label ["] filename["]

DEFINITION

Used to append or insert a program segment or subroutine (saved with SAVE+) into the present program in memory.

A non-line numbered ASCII program file is required to append a subroutine into the present program in memory at the specified line number. Line numbers will be assigned in increments of one.

APPEND* will strip REM(arks) and spaces to free up more memory for the program as the program is inserted.

EXAMPLE

```
10 "TEST ROUTINE"
20 FOR I = 1 TO 10
30   PRINT I
40 NEXT I
50 RETURN

SAVE+ TEST.APP

APPEND 31 TEST.APP

LIST

00010 "TEST ROUTINE"
00020 FOR I = 1 TO 10
00030   PRINT I
00031   "TEST ROUTINE"    <----Subroutine inserted here
00032   FOR I = 1 TO 10   <----(Example only, program will not run)
00033     PRINT I
00034   NEXT I
00035   RETURN
00040 NEXT I
00050 RETURN
```

REMARK

The program to be appended must be in ASCII format and not contain line numbers. Use the SAVE+ command to save programs without line numbers.

If any line number being used in APPEND already exists, it will overwrite the existing line. Also see MERGE, LOAD, SAVE, SAVE*, SAVE+.

ASC function

FORMAT **ASC**(string)

DEFINITION Returns the ASCII code value (a number between 0 and 255) of the first character in a string. ASCII stands for American Standard Code for Information Interchange.

EXAMPLE

```
PRINT ASC("A"), ASC("B")
PRINT CHR$(65), CHR$(66)
PRINT ASC("America")
```

RUN

```
65            66
A             B
65
```

REMARK ASC returns 0 if the length of string is zero or the ASCII code of the string is zero. Use this logic to determine the true status if an ASCII zero is the result:

```
LONG IF ASC(A$)=0 AND LEN(A$)>0
  PRINT "ASCII code of A$ =0"
XELSE
  PRINT"A$ is an empty string"
END IF
```

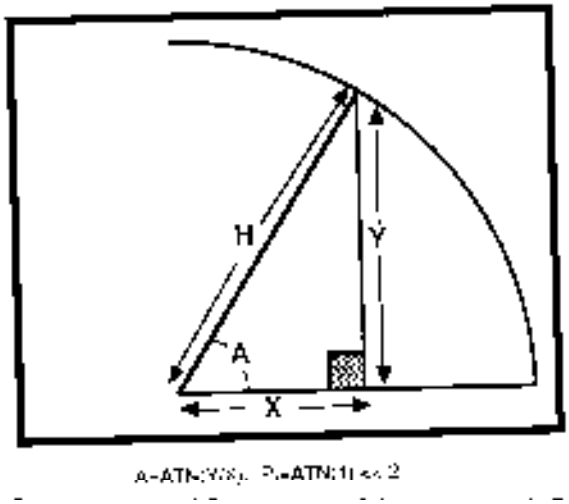
The inverse function of ASC is CHR$. To return the character represented by the ASCII code, use CHR$(ASCII number)

ASCII codes may vary from machine to machine.

ASCII codes 32 through 127 are usually the same for all microcomputers. See CHR$ with example ASCII listing.

FORMAT **ATN**(expression)

DEFINITION Returns the angle, in radians, for the inverse tangent of expression.



EXAMPLE

```
Pi#=ATN(1) << 2
PRINT Pi#
```

RUN

```
3.141592... <---Based on digits of accuracy set in configuration.
```

REMARK ATN is a scientific function. Using ATN in an expression will force ZBasic to calculate that part of an expression in Double Precision.

ZBasic allows you to configure the accuracy for scientific functions separately for both Double and Single Precision. See "Configure".

Also see "Expressions" and "Derived math functions" in the "MATH" section of this manual.

AUTO command

FORMAT **AUTO**
AUTO starting line
AUTO starting line, increment
AUTO , increment

DEFINITION This command automatically generates line numbers in the Standard Line editor to save time. The two optional parameters are:

starting line	Starting line number	(default is 10)
increment	Line spacing	(default is 10)

To end AUTO line numbering press either <BREAK> or <CTRL C> at the first line number you will not use.

EXAMPLE **AUTO**

```
10                    <--- Type in text then <ENTER> to go to next line.
20
30 <BREAK>
```

AUTO 100,20

```
100
*120                  <---- Careful, this line already exists!!
130 <BREAK>
```

REMARK An asterisk appearing before a line number indicates an occupied line. Pressing <ENTER> will skip that line leaving the original contents intact and resume auto line numbering with the next line. To remove the line type a space and <ENTER>.

Also see LIST, EDIT

FORMAT **BEEP**

DEFINITION Sounds the speaker.

EXAMPLE

```
FOR X=1 TO 10
  BEEP
NEXT

RUN

BEEP, BEEP...
```

REMARK Also see SOUND.

BEEP is not supported with Apple // or Z80 computers. For Apple // and most CP/M computers use `PRINT CHR$(7)` instead. See your SOUND and your computer appendix for other ways of creating audio output.

BASE OPTION configuration

FORMAT **Array Base 0 or 1?**

DEFINITION An option in the ZBasic configuration routine to set the array BASE to either zero or 1. The default is zero.

EXAMPLE See "Configure" in the beginning of this manual for an explanation of configuring your version of ZBasic to your preferences.

ARRAY BASE ZERO

```
DIM A(100)          <-- elements 0-100 (101 elements)
DIM Tables(22)      <-- elements 0-22 (23 elements)
```

ARRAY BASE ONE

```
DIM A(100)          <-- elements 1-100 (100 elements)
DIM Tables (22)     <-- elements 1-22 (22 elements)
```

REMARKS See DIM and "Array Variables".

FORMAT **BIN$** (expression)

DEFINITION Returns a 16 character string which represents the binary (BASE 2) value of the result of the integer expression. Some typical binary numbers:

```
0000000000000001     =     1
0000000000000011     =     3
0000000000000111     =     7
0000000011111111     =     255
0000000100000000     =     256
1111111111111111     =     -1 (65,535 unsigned)
```

EXAMPLE The following program will convert a decimal number to binary or a binary number to decimal:

```
"Binary Conversion"
CLS
DO
  INPUT"Decimal number to convert: ";Decimal%
  PRINT BIN$(Decimal%)
  INPUT"Binary number to convert: ";Binary$
  Binary$="&X"+Binary$
  PRINT VAL(Binary$)
UNTIL Decimal% = 0
```

RUN

```
Decimal number to convert: 255
0000000011111111

Binary number to convert: 0000000000000011
3
```

REMARK Note that conversions are possible from any base to any other base that ZBasic supports. &X is the inverse function of BIN$.

Also see HEX$, OCT$, UNS$ and "Numeric Conversions".

Use DEFSTR LONG to set BIN$ and &X to work in LongInteger (32bits).

BOX statement

FORMAT

BOX [TO] expr $_{x1}$, expr $_{y1}$ [**TO** expr $_{x2}$,expr $_{y2}$ ...]
BOX FILL [TO] expr $_{x1}$, expr $_{y1}$ [**TO** expr $_{x2}$,expr $_{y2}$ ...]

DEFINITION

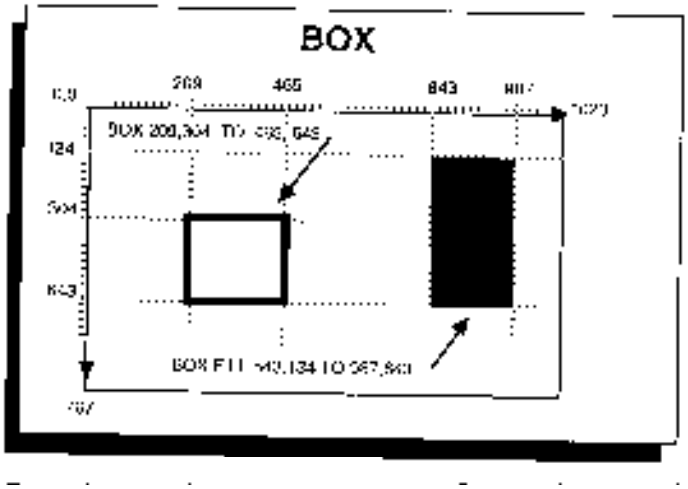
Draws a BOX from the coordinates defined by the first corner (x1,y1) to the coordinates defined by the opposite corner (x2,y2) in the current COLOR.

If BOX TO x,y is used the first corner will be the last graphic point used. If undefined then 0,0 will be the default.

If the optional FILL appears directly after the command, the BOX will be painted as a solid BOX in the current color.

The default screen positions are given using Device Independent Coordinates of 1024 across by 768 down.

EXAMPLE



REMARK

The output will vary depending on the graphic capability of the host computer. Also see CIRCLE, MODE, FILL, PLOT, RATIO and COLOR.

FORMAT **CALL** number
CALL LINE line or label

DEFINITION **CALL** will execute a machine language subroutine at the address specified by number or the address of the compiled line.

EXAMPLE Use these examples only if you understand machine language.

```
REM TRS80 I & III, CALL DEBUG
CALL &H440D
:
REM CPM 80, CALL WARM START (Exits to DOS)
CALL 0
:
REM APPLE CALL TO SOUND BELL TONE
CALL -198
:
10 REM CALL LINE examples
20 CALL LINE 40
30 CALL LINE "LABEL"
40 MACHLG 34, 21, x%, 255, 9: RETURN
50 "LABEL": MACHLG . . .  : RETURN
```

REMARK CALL is useful for transferring program control to a machine language subroutine from which a return to the ZBasic program is desired. The routine to be called must be terminated by that machine's instruction for RETURN.

Also see MACHLG, USR, LINE and DEFUSR.

WARNING: Use of this command requires an understanding of machine language programming and the computer hardware being used. Porting of this code may not be possible without re-writing the machine language routines.

See CALL in your appendix for enhancements.

CASE statement

FORMAT

SELECT [CASE] [expression]
CASE [IS] relational condition [, relational condition] [,...]
statement [:statement:...]]
CASE [IS] condition [, condition] [,...]
statement [:statement:...]]
CASE boolean expression
statement [:statement:...]]
CASE ELSE
statement [:statement:...]]
END SELECT

DEFINITION

When SELECT/CASE is encountered, the program checks the value of the controlling expression or variable, finds the CASE that compares true and executes the statements directly following the CASE statement. After these statements are performed, the program continues at the line after the END SELECT statement:

CASE relational,...

If the expression after SELECT compares true to any one of a number of relational conditions, the statements following the CASE are executed and the program continues after the END SELECT:

```
SELECT 12
  CASE >10
    PRINT "This is the right answer"
  CASE >20, <10
    PRINT "This is not true"
END SELECT
program continues here...
```

CASE condition,...

If the expression following SELECT equals any one of a number of conditions the statements following the CASE are executed (program continues after the END SELECT).

```
A=23
SELECT A
  CASE 10
    PRINT "This is the wrong answer"
  CASE 10,23,11,10
    PRINT "This would be true"
END SELECT
```

CASE boolean

If and expression after SELECT is omitted, you may use a boolean or TRUE/FALSE condition. The statements after the first TRUE (non-zero) CASE condition will be executed. Only one boolean statement is allowed following CASE.

```
A=10:B=20
SELECT
  CASE (A=10 AND A>20)
    PRINT "This is the correct answer"
  CASE (A>B OR A=B)
    PRINT "This is the wrong answer"
END SELECT
```

CASE ELSE

If all of the CASE statements in the SELECT CASE structure are false the statements following the CASE ELSE are executed.

```
"Start"
A$="Maybe"
SELECT A$
  CASE "Yes"
    PRINT "Thank you for saying Yes"
  CASE "No"
    PRINT "Thank you for saying No"
  CASE ELSE
    PRINT "You smart aleck!"<--Does this one
END SELECT
```

REMARK

This is a powerful structured way of doing complicated IF-THEN-ELSE or LONG IF statements especially when there are multiple lines of complicated comparisons.

This structure is also much easier to read than complicated IF statements.

See SELECT for more information.

Important Note: Never exit a SELECT CASE structure using GOTO. This will introduce problems into the stack and cause unpredictable system errors. Always exit the structure at the END SELECT. Be sure to enclose loops and other constructs completely within the SELECT-CASE and CASE ELSE constructs.

The Z80 versions do not support SELECT CASE. See LONG IF and IF for ways of doing the same thing.

The Apple DOS 3.3 and ProDOS versions does not support SELECT CASE. See LONG IF and IF for ways of doing the same thing.

CHR$ function

FORMAT **CHR$** (expression)

DEFINITION Returns a single character string with the ASCII value of the result of expression. The range for the value of expression is 0 to 255.

The inverse function of CHR$ is ASC;

EXAMPLE

```
"Print ASCII character set for this computer"
CLS
REM  Use ROUTE 128 here to send output to printer.
FOR I=32 TO 127 STEP  8
  FOR J= 0 TO 7: X =I+J
    PRINT USING "###=";X;CHR$(X);"  ";
  NEXT J  :PRINT
NEXT I
```

```
RUN
```

```
32=    33=!   34="   35=#   36=$   37=%   38=&   39='
40=(   41=)   42=*   43=+   44=,   45=-   46=.   47=/
48=0   49=1   50=2   51=3   52=4   53=5   54=6   55=7
56=8   57=9   58=:   59=;   60=<   61==   62=>   63=?
64=@   65=A   66=B   67=C   68=D   69=E   70=F   71=G
72=H   73=I   74=J   75=K   76=L   77=M   78=N   79=O
80=P   81=Q   82=R   83=S   84=T   85=U   86=V   87=W
88=X   89=Y   90=Z   91=[   92=\   93=]   94=^   95=_
96=`   97=a   98=b   99=c   100=d  101=e  102=f  103=g
104=h  105=i  106=j  107=k  108=l  109=m  110=n  111=o
112=p  113=q  114=r  115=s  116=t  117=u  118=v  119=w
120=x  121=y  122=z  123={  124=|  125=}  126=~  127=#
```

```
PRINT CHR$(64)
PRINT ASC("A")
```

```
RUN
```

```
A
64
```

REMARK When the program above is run, the character set for that computer will be displayed. Some of the characters above may differ from what you get on your system. Try changing the range above from 127 to 255. Some computers have extra characters or graphic symbols for these codes.

Characters in the range of 0-31 are usually reserved for control codes like linefeed (10), carriage return (13)...

If the PRINT statement is changed to LPRINT the printer's character set will be printed. If expression is less than 0 or greater than 255, only the low order byte will be used.

CHR$(256) = CHR$(0)
CHR$(257) = CHR$(1)

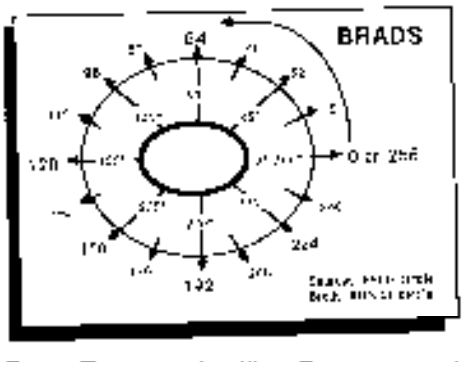
FORMAT

CIRCLE [FILL]	*expr*$_1$,	*expr*$_2$,	*expr*$_R$				
CIRCLE	*expr*$_1$,	*expr*$_2$,	*expr*$_R$	**TO**	*expr*$_S$,	*expr*$_B$	
CIRCLE	*expr*$_1$,	*expr*$_2$,	*expr*$_R$	**PLOT**	*expr*$_S$,	*expr*$_B$	

DEFINITION Draws a CIRCLE in the current COLOR.

If the optional FILL is used directly after the command, the CIRCLE will be filled with the current COLOR. If TO is used, a PIE segment will be displayed (shaped like pie slices). If PLOT is used, only the ARC segment will be displayed (a segment of the circumference).

expr1	horizontal center
expr2	vertical center
exprR	radius (diameter of circle) in graphic coordinates
exprs	start of angle in brads (zero starts at 3:00 o'clock)
exprB	Number of brads to draw ARC or PIE (counter clockwise).



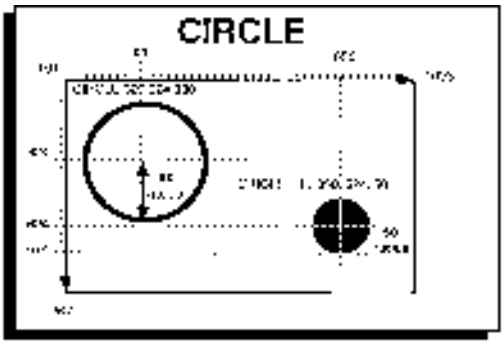
EXAMPLE SEE ILLUSTRATIONS OF FOLLOWING PAGE.

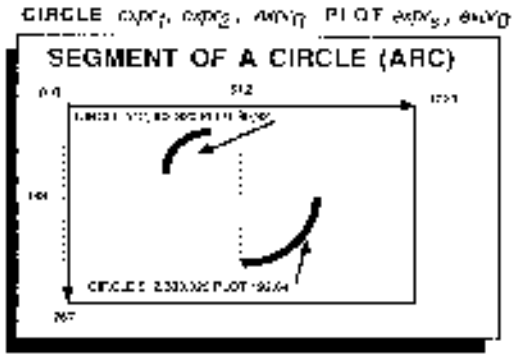
REMARK CIRCLE uses the ZBasic Device Independent Graphic Coordinates of 1024 x 768. For more details see the CIRCLE in the "Graphics" section in this manual. Also see RATIO,MODE,PLOT,COLOR,FILL and BOX.

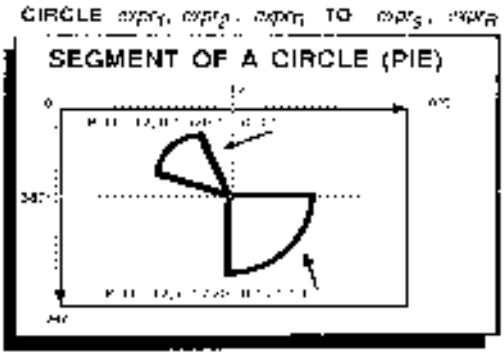
Macintosh: See COORDINATE WINDOW for pixel coordinates and toolbox for ways of using QuickDraw for creating boxes. MSDOS: See COORDINATE WINDOW for converting to pixel coordinates. Apple: See appendix for ways of converting to pixel graphics.

CIRCLE statement

EXAMPLE **CIRCLE** *expr*$_1$, *expr*$_2$, *expr*$_R$
CIRCLE FILL *expr*$_1$, *expr*$_2$, *expr*$_R$







FORMAT

CLEAR
CLEAR *number*
CLEAR END
CLEAR INDEX$

DEFINITION Used to reserve memory or clear all or specified variables (sets the values of the variables to null or zero).

CLEAR — Sets all variables and INDEX$ to zero or null.

CLEAR number — Sets aside number bytes for the INDEX$ array.

CLEAR END — CLEARS all variables which have not yet been assigned in the program. This form of CLEAR is normally used to clear all variables not being used when chaining. See "Chain" in the front section for more information.

CLEAR INDEX$ — Sets all elements of the INDEX$ array to null.

EXAMPLE

```
INPUT"Name: ";Name$
PRINT Name$
CLEAR
PRINT Name$

RUN

Fred
        <-----Nothing printed here since Name$ was cleared at line 3.
```

REMARK Only one CLEAR number is allowed in a program and must appear before any variables are encountered. Be sure to CLEAR one extra byte for each element in the INDEX$ array. Also see "Special INDEX$ Array" and "CHAIN".

A CLEAR is performed at the beginning of each program created with RUN or RUN*. RUN+ or warm start programs will not CLEAR variables at startup.

See INDEX$ in Mac appendix for added enhancements available on this version.

CLOSE statement

FORMAT **CLOSE** [[#] *expression*$_1$[, [#] *expression*$_2$...]]

DEFINITION This statement is used to CLOSE one or more OPEN files or other devices.

The parameter expression indicates a device number or file number.

If no file or device numbers are declared all OPEN devices will be closed.

EXAMPLE

```
OPEN"I",1,"FILE1",10
OPEN"I",2,"FILE2",100
READ#1, A$;10
READ#2, B$;10
CLOSE#1,2                    <---File1 and 2 are closed
OPEN"R",1,"FILE3"            <---File1 may now be used again
CLOSE                        <---All files are closed
```

REMARK All files should be closed before leaving a program to insure that data will not be lost or destroyed. If a program exit is through END or STOP, all files will be closed.

FORMAT

CLS
CLS *expression*
CLS LINE
CLS PAGE

DEFINITION These statements will clear all, or portions, of the screen of text and graphics.

CLS	Clears the entire screen of text and graphics. Cursor ends up at the top left corner of screen.
CLS expression	In TEXT mode this fills screen with the ASCII character specified by expression and places the cursor at the top left corner of the screen*.
CLS expression	In GRAPHICS mode this will fill the screen with the color specified by expression.
CLS LINE	Clears from the cursor position to the end of the line. Cursor will remain where it was.
CLS PAGE	Clears from the cursor position to the end of the screen. Cursor will remain where it was.

EXAMPLE

```
CLS
CLS 65                  <----Fills screen with A's
CLS ASC("*")            <----Fills screen with *'s
LOCATE 0,10
CLS LINE                <----Clears line 10 of text and graphics
LOCATE 0,12
CLS PAGE                <----Clears screen from line 12 down.
```

REMARK See LOCATE,PRINT@,PRINT%,FILL and MODE. See your computer appendix for possible variations.

CLS clears the current window (not the entire screen). CLS expression will clear the screen with white if expression=0 and black if expression><0.

COLOR statement

FORMAT **COLOR** [=] *expression*

DEFINITION Sets the COLOR to be used by all graphic drawing commands. Color values will vary from one computer to the next. See your computer appendix for specifics. For most computers 0 is the background color and -1 is the foreground color.

If you have a black and white monitor, 0 is Black, -1 is white.

If your computer is incapable of graphics or your are using one of the character modes, the expression will determine the ASCII character to be used. (With some graphics modes, zero=space, all others=asterisk "*").

EXAMPLE

```
CLS: MODE  6          <----even modes are character graphics with some versions
COLOR ASC("*")        <----Uses asterisks for graphics (not all versions)
PLOT 0, 256
MODE=7                <----odd modes are actual graphics
CIRCLE 768,200,50
COLOR=6               <----Sets COLOR to 6
BOX 0,0 TO 10,10
END
```

REMARK Also see MODE,PLOT,CIRCLE,BOX,POINT and FILL. Colors vary by mode, graphic type, monitors and other hardware criteria. Check hardware manual and the ZBasic appendix for your computer for specific color codes.

Macintosh: NOT(0) =black, 0=white. See appendix for variations especially with Macintosh II which supports a number of colors and grey levels.

MSDOS: COLOR is also used to change text color, background color, blinking, underline etc. See appendix for specifics. See CGA colors below.

Apple: Color chart below and the Apple appendix.

TRS-80 and **Kaypro**: Black=0, -1=white.

EXAMPLE COLORS CODES

IBM PC and compatibles **CGA MODE 5**		**Apple** // **ProDOS** and **DOS 3.3** **MODE 5**		**MODES 1,3 and 7**	
0= BLACK	8=GRAY	0=BLACK1		0=BLACK	8=BROWN
1=BLUE	9=LT BLUE	1=GREEN		1=MAGENTA	9=ORANGE
2=GREEN	10=LT GREEN	2=VIOLET		2=DARK BLUE	10=GREY
3=CYAN	11=LT CYAN	3=WHITE1		3=PURPLE	11=PINK
4=RED	12=LT RED	4=BLACK2		4=DARK GREEN	12=GREEN
5=MAGENTA	13=LT MAGENTA	5=ORANGE		5=GREY	13=YELLOW
6=BROWN	14=YELLOW	6=BLUE		6=MED. BLUE	14=AQUA
7=WHITE	15=Bright WHITE	7=WHITE2		7=LIGHT BLUE	15=WHITE

statement COMMON

FORMAT COMMON variable list...

DEFINITION Identical to the ZBasic DIM statement. It is used to allocate memory for variables and for declaring variables common to chained programs.

The order of the variables declared in COMMON is important when chaining programs. The COMMON statement in one program must be exactly the same and in exactly the same order in other programs being chained.

EXAMPLE See DIM.

REMARK See DIM and "Chaining" in this manual.

This statement is added to make ZBasic compatible with other versions of BASIC.

Not available on the Apple // or Z80 version of ZBasic. Use DIM.

COMPILE command

FORMAT **[L] COMPILE**

DEFINITION Compiles a program and lists all of the compile time errors that are encountered.

If optional "L" is used, the error listings are sent to the printer.

This command is essentially the same as RUN except the compiler does not stop at the first error.

EXAMPLE

```
PWINT "Hello"
X=X+1
INPUT "Yes or No:"A$
GOSUB "Routine"
END

COMPILE

Syntax Error in Stmt 01 at Line 00001
00001 PWINT "Hello"

";" Expected Error in Stmt 01 at line 00003
00003 INPUT "Yes or No:"_A$

Line# Error in Stmt 01 at Line 00004
00004 GOSUB "Routine"
```

REMARK See RUN and the section in the front of the manual called "Errors".

Not supported. Use RUN.

Not supported. Use RUN.

FORMAT **CONFIG**

DEFINITION Invokes the configuration prompts that allow you to set preferences for a number of items including:

Digits of precision
Default variable types
Integer or floating point expression evaluation
Spaces between keywords
Convert to uppercase
Number of files that can be opened
The Rounding factor for PRINT USING
Test Array bounds

and a number of special options for your computer.

EXAMPLE See "Configure" in the front of this manual and the section in your appendix for specific configuration options available for your version of ZBasic.

REMARK This command is not available on all versions. See below.

The Z80 versions of ZBasic do not offer this command. The option to configure is offered only when you first load ZBasic.

CONFIG is not offered as a command but "Configure" is always available as a menu item. See appendix for the options specific to this version.

COORDINATE statement

FORMAT **COORDINATE [[WINDOW]** *horizontal*, *vertical*]

DEFINITION Allows you to change the coordinate system used for graphic functions and statements.

ZBasic defaults to a coordinate system of 1024 x 768. This allows programs created on one computer work on other computers with different graphic hardware.

COORDINATE *horiz*,*vert*	Set the relative coordinate system to the specified limits minus one. COORDINATE 100,100 would allow setting the coordinates from 0 to 99 for both the horizontal and vertical.
COORDINATE WINDOW	Sets the system to pixel coordinates. This allows you calculate the graphic positions by the actual resolution of the screen. While this is not recommended for programs that will be ported to other computers, some people prefer it for certain applications.

EXAMPLE

```
PLOT 1023, 767          <--- Puts a graphic dot at the ZBasic
:                                 default coordinates (lower right corner)
COORDINATE WINDOW
PLOT 100,100            <--- Puts a graphic dot at the pixel coordinate
:
COORDINATE 1000,500
PLOT 100,100            <--- Puts a graphic dot at the relative coordinate
```

REMARK Some versions do not support this statement. See below for alternatives to changing coordinate systems.

Not supported on Z80 versions although COORDINATE WINDOW may be emulated by using this instruction: `POKE&xx3F,&C9` to enable pixel graphics and POKE&xx3F,&C3 to return to the default coordinates of 1024x768. The value of xx varies by version type: CP/M-80=01, TRS-80 1,3=52 and TRS-80 model 4=30.

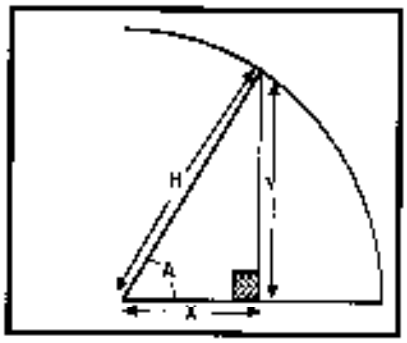
Not supported on these versions although COORDINATE WINDOW may be emulated using the statements below:

Apple ProDOS: `POKEWORD &85,0` for pixel coordinates for that mode of graphics. Use MODE to set back to regular coordinates.

Apple DOS 3.3: `POKE &F388,&60` for pixel coordinates of that mode. `POKE &F88,&A9` to set back to the default coordinates of 1024x768.

FORMAT **COS** (expression)

DEFINITION Returns the Cosine of the expression in radians.



EXAMPLE Using COS in an expression will force ZBasic to calculate that expression in floating point. COS is a scientific function. You may configure BCD scientific accuracy separately for both Double and Single Precision immediately after loading ZBasic.

Integer Cosine may be accomplished with the predefined ZBasic USR function; USR9(angle in Brads). This returns the integer cosine of an angle in the range +-255 (corresponding to +-1). The angle must be in Brads. This example program will draw a sine wave using USR9:

```
MODE7 :CLS
FOR I=0 TO 255
  PLOT I<<2,-USR9(I)+384
NEXT I
```

For more information about scientific functions and derived math functions see the "Math" section of this manual. See CIRCLE for more about BRADS. Also see ATN, SIN,TAN,EXP,SQR.

CSRLIN function

FORMAT **CSRLIN**

DEFINITION Returns the line where the cursor is positioned.

EXAMPLE

```
CLS
PRINT
PRINT
PRINT CSRLIN
```

```
RUN

2
```

REMARK See POS to determine the horizontal cursor position.

Not supported with the Apple // or Z80 versions of ZBasic. For Apple // use `PEEK(37)` to get the current cursor line.

FORMAT **CVB** (string)

DEFINITION Returns the binary floating point value of the first n characters of the condensed number in *string* (depending on whether Single or Double Precision is used).

Double Precision	Returns the digits of accuracy defined in configure for double precision. (default is 8 digits i.e. the first 8 string characters.)*
Single Precision	Returns the digits of accuracy defined in configure for single precision. (default is 4 digits i.e. the first 4 string characters.)

This function is the compliment of MKB$.

EXAMPLE

```
A#=12345.678: B!=12345.678
:
A$=MKB$(A#): B$=MKB$(B!)
PRINT LEN(A$), LEN(B$)
:
C#=CVB(A$): D!=CVB(B$)
PRINT C#, D!

RUN

8             8
12345.678     12345.7
```

REMARK This function is used with some versions of BASIC to save space on disk when storing large amounts of numeric data in strings with FIELD. ZBasic does this automatically but CVB is still useful for string packing, etc. Also see MKI$,CVI,MKB$, READ# AND WRITE#. This command is not compatible with CVS or CVD.

A few things to remember concerning CVB:

Null strings or 1 character strings return 0
Two character strings will return 2 digits of accuracy. Four character strings will return four digits. See "Floating Point Variables" for more information.

*See "Floating Point Variables" for detailed information on how extended double precision variables are stored and the added range of this precision for the Mac.

CVI function

FORMAT **CVI** (string)

DEFINITION Returns the binary integer value of the first 2 characters of string.

This function is the compliment of MKI$.

EXAMPLE

```
A$=MKI$(30000)
PRINT LEN(A$)
:
Z%=CVI(A$)
PRINT Z%
END

RUN

2
30000
```

REMARK Also see MKI$,CVB,MKB$,READ# AND WRITE#.

A few things to remember concerning CVI:

- Null string returns 0
- One character strings will return the ASCII value.
- Two character strings will return an integer value.
- ASC(second character)*256 + ASC(first character)

This function was used with MBASIC to save space on disk when storing large amounts of numeric data. ZBasic does this automatically when using WRITE# and READ# but CVI is still useful for string packing, etc.

See DEFSTR LONG in the Mac appendix for using this function with LongIntegers. When LongIntegers are used the memory requirements are four bytes instead of two bytes. MSB and LSB are stored in reverse order for regular integers with this version.

FORMAT **DATA** *data item* [,*data item*[,…]]

DEFINITION The DATA statement is used to hold information that may be read into variables using the READ statement. DATA items are a list of string or numeric constants separated by commas and may appear anywhere in a program.

No other statements may follow the DATA statement on the same line.

Items are read in the order they appear in a program. RESTORE will set the pointer back to the beginning of the first DATA statement. RESTORE n will set the pointer to the nth DATA item.

EXAMPLE

```
DATA Tom, Dick, Harry, 12.32, 233
READ A$, B$, C$, A#, B%
:
DEF TAB 6
PRINT "DATA items are: ";A$,B$,C$,A#,C%
```

RUN

```
 DATA items are: Tom  Dick  Harry  12.32  233
```

```
DATA Tom, Dick, Harry, 12.32, 233
:
RESTORE 3
READ Name$
:
PRINT "Third DATA item is: ";Name$
```

RUN

```
Third DATA item is: Harry
```

REMARK Alphanumeric string information in a DATA statement need not be enclosed in quotes if the first character is not a number, math sign or decimal point.

Leading spaces will be ignored (unless in quotes). DATA statements can be included anywhere within a program and will be read in order.

Typical storage requirements for DATA items:

Number with zero value	2 bytes
Non-zero integer	3 bytes
Strings	Length of string + 2
Floating Point BCD	"See Floating Point Constants"
Floating Point Binary	"See Floating Point Constants"

See READ, PSTR$ DIM and RESTORE for common statements used with DATA.

Note: See PSTR$ for extremely efficient way of retrieving strings in DATA statements.

DATE$ function

FORMAT **DATE$**

DEFINITION Returns an eight character string containing the system date using the format MM/DD/YY,where MM=month, DD=day and YY=year.

EXAMPLE

```
DATA January, February, March, April, May, June
DATA July, August, September, October, November, December
:
A$=DATE$
:
Day$=MID$(A$,4,2)
REM  If leading zero; peel off on next line
If ASC(DAY$)=ASC("0") THEN DAY$=RIGHT$(DAY$,1)
:
Month%=VAL(A$)
RESTORE Month%
READ Month%                    <---Get month name from DATA
:
Year$="19"+RIGHT(A$,2)
:
PRINT "Computer date: ";TAB(20);DATE$
PRINT "Human date: ";TAB(20);Month$;" ";Day$;", ";Year$

RUN

Computer date:08/03/88
Human date:          August 3, 1988
```

REMARK If the system does not support a date function, 00/00/00 will be returned. See your computer appendix for more information.

Also see TIME$ and DELAY

Macintosh: Date can only be changed from the "Control Panel DA"

MSDOS: Date may be set in program: DATE$="MM/DD/YY"

Apple: Date must be set from the system.

CP/M-80 3.0 and Plus: DATE$ supported. CP/M 2.x does not support date.

FORMAT

DEFINT	*letter* [- *letter*] [, *letter* [- *letter*],...]
DEFSNG	*letter* [- *letter*] [, *letter* [- *letter*],...]
DEFDBL	*letter* [- *letter*] [, *letter* [- *letter*],...]
DEFSTR	*letter* [- *letter*] [, *letter* [- *letter*],...]
***DEFDBL INT**	*letter* [- *letter*] [, *letter* [- *letter*],...]

DEFINITION These statements define which variable type ZBasic will assume when encountering a variable name with letter as a first character and not followed by a type declaration symbol (% integer, ! single, # double, $ string, & double integer).

DEFINT	Integer
DEFSNG	Single Precision
DEFDBL	Double Precision
DEFSTR	String
*DEFDBL INT	LongInteger (Macintosh only)

ZBasic will assume that all variables are integers unless followed by a type declaration symbol or defined by a DEF type statement.

See "Configure" for another way of defining the default variable type.

letter	Letter from A to Z. Case is not significant.
letter - letter	Defines an inclusive range of letters.

EXAMPLE

```
DEFSNG A                 <--- A and A! are the same variable (A$ is still a string).
DEFDBL B                 <--- B and B# are the same variable (B% is still an integer).
DEFINT F                 <--- F and F% are the same variable (F! is still single prec.).
DEFSTR B-D, X,Y,Z        <--- B,C,D,X,Y and Z all strings
DEFDBL A, F-J, T         <--- A,F,G,H,I,J and T all Double precision
DEFSGL A, G, B-E         <--- A,G,B,C,D and E all Single Precision
```

REMARK Other versions of BASIC may assume all numeric variables are single precision unless otherwise defined. See the sections on "Floating Point Variables", "Math" and "Converting Old Programs" in the front of this manual for more information.

*Also see DEFSTR LONG in appendix for way of forcing HEX$, OCT$, UNS$, CVI and MKI$ to default to LongInteger instead of regular integer.

DEF FN statement

FORMAT **DEF FN** *name* [(*variable*[, *variable*[,...]])] = *expression*

DEFINITION This statement allows the user to define a function that can thereafter be called by FN name. This is a handy way of adding functions not provided in the language.

The expression may be a numeric or string expression and must match the type the FN name would assume if it was a variable name.

The name must adhere to variable name syntax.

The variable used in the definition of the function is a dummy variable. When using FN the dummy variables, other variables or expressions may be used to pass the values to the function. The variable should be of the right type used in the function.

EXAMPLE

```
DEF FN e# = EXP(1.)
DEF FN Pi#= ATN(1)<<2
DEF FN Sec#(x#) = 1.\COS(x#)
DEF FN ArcSin#(x#) = ATN (x# \ SQR( 1 - x# * x#))
:
PRINT FN Pi#
I#=4.2312
Planet#= FN ArcSin#(Sin(I#))* FN e#+ FN Sec# (Elipse#)
```

RUN

```
3.14159...
```

```
REM    A Handy rounding function
REM    Send the routine the number and places to round
:
DEF FN Round#(num#, places)=INT(num#*10^places+.5)/10^places
:
PRINT FN Round#(823192.12345675676,5)
X#=202031.12332
PRINT FN Round#(X#,2)
END
```

RUN

```
823192.12457
202031.12
```

REMARK One function may call another function as long as the function was defined first.

LONG FN is another form of DEF FN that allows multiple lines of code. It is very powerful for creating reusable subroutines.

See "Derived Math functions", "Functions and Subroutines", LONG FN, END FN and FN.

FORMAT **DEF LEN**[=] number

DEFINITION The DEF LEN statement is used to reset the default length of string variables until the next DEF LEN statement is encountered. The number must be from 1 to 255.

If DEF LEN is not used string length default is 255 characters each. Each string will consume 256 bytes; 1 byte for length byte, the rest for characters.

Since strings will consume so much memory if their length is not defined; it is imperative that thought be given to string length, especially if memory is at a premium.

EXAMPLE

```
C$="Welcome"            <---Length of C$ defaults to 255 characters.
:
DEF LEN 20
DIM A$(10)              <---A$() allocated 20 characters per element.
Greeting$="Hello"       <---Greeting$ allocated 20 characters
:
DEF LEN 200
B$="Goodbye"            <---B$ allocated 200 characters
:
DIM 50 Z$               <---Z$ allocated 50 characters. See DIM
```

REMARK DEF LEN will allocate the specified amount of memory to every string that is defined after it (unless defined differently in DIM or another DEF LEN).

Strings that appear before the DEF LEN statement are not affected. For example, in the above program, C$ is allocated the default length of 255 characters because it appeared BEFORE the DEF LEN statement.

DIM may also be used to set the length of string variables. See DIM.

Also see "String Variables" and "Converting Old Programs" in the front section for important information about strings and how they use memory.

Important Note: Always allocate one extra character for strings used with INPUT. Never use a one character string for INPUT. The extra character position is needed for the carriage return.

DEF MOUSE statement

FORMAT **DEF MOUSE** [=] *expression*

DEFINITION The DEF MOUSE statement is used to define the device to be used with the MOUSE functions and statements, or the type of mouse commands to use with the program.

DEF MOUSE=0 Regular ZBasic MOUSE commands for a mouse device. See MOUSE in this reference section.

- MSDOS: Uses MicrosoftTM compatible mouse devices. Be sure to "Configure" ZBasic for a mouse.
- Apple //: Assumes a mouse is connected.
- Macintosh: Standard MOUSE commands in this section of the reference manual. See DEF MOUSE=1 to do MSBASIC type mouse commands.
- Z80: NOT SUPPORTED.

DEF MOUSE=n Tells ZBasic that other devices are to be used instead of a MOUSE (in the case of the Macintosh it tells ZBasic to use MSBASIC mouse syntax).

- MSDOS: n=1 defines joystick/paddle A*
  n=2 defines joystick/paddle B*
  n=3 defines a lightpen device
- Apple //: n=1 defines a joystick/paddle device*

  *Mouse(3) function returns button status:
  0=No button pressed
  1=Button zero pressed
  2=Button one pressed
  3=Both buttons pressed
- Macintosh: n= non-zero sets commands to MSBASIC mouse commands. See Macintosh appendix for specifics.
- Z80: NOT SUPPORTED.

EXAMPLE See the appendix for your computer for specifics.

REMARK See MOUSE in this reference section and in your appendix for specifics.

MOUSE or DEF MOUSE is not supported with any Z80 versions of ZBasic. This is due to the fact that most Z80 computers do not offer this hardware device.

FORMAT **DEF TAB** [=] *expression*

DEFINITION The DEF TAB statement is used to define the number of characters between tab stops for use in PRINT,PRINT# or LPRINT statements

Tab stops are the number of spaces to move over when the comma is encountered in a PRINT statement.

The expression must be a number from 1 to 255. TAB default is 16.

EXAMPLE

```
PRINT 1,2,3            <---Tab stop default is 16, 32, 48...
DEF TAB = 8            <---Tab stops now set to 8, 16, 24...
PRINT 1,2,3: PRINT
:
FOR X=1 TO 5
  DEF TAB=X
  PRINT 1,2,3
NEXT X
```

RUN

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

REMARK Also see TAB,WIDTH,WIDTH LPRINT and PAGE.

DEF USR statement

FORMAT **DEF USR** *digit = expression*

DEFINITION The DEF USR statement is used to define the addresses of up to 10 machine language user subroutines; USR0 to USR9.

EXAMPLE *Examples only. Do Not Use!*

```
REM Calls graphic routine at memory address 5000
DEFUSR1=5000
X=USR0(45)
:
DEFUSR2=23445
PRINT USR2(x)
```

REMARK A machine language return is needed at the end of the routine to return program control to ZBasic.

See USR,MACHLG,CALL,LINE,VARPTR,BIN$,HEX$,OCT$,UNS$,PEEK, PEEKWORD,POKE,POKEWORD and the chapter "Machine Language".

Some other default USR functions are included in the appendix for your computer.

Warning: Use of this command requires a knowledge of machine language and a computer's hardware. Porting of programs with this statement may not be possible without re-writing the routines.

FORMAT **DELAY** *expression*

DEFINITION The DELAY statement will cause a program to pause a specified amount of time.

The expression sets the delay in milliseconds; thousandths of a second.

EXAMPLE

```
CLS
FOR I = 1 TO 5
  PRINT "DELAYING ";I;"SECONDS"
  DELAY I  *  1000
NEXT I
END
```

RUN

```
DELAYING 1 SECONDS
DELAYING 2 SECONDS  (after 1 second)
DELAYING 3 SECONDS  (after 2 second)
DELAYING 4 SECONDS  (after 3 second)
DELAYING 5 SECONDS  (after 4 second)
```

```
FOR X=1000 TO 0 STEP -50
  PRINT X
  DELAY X
NEXT
```

(try it)

REMARK The <BREAK> key is not scanned during DELAY. Any negative expression will cause delays in excess of 32 seconds (the unsigned value). Note that DELAY -1 will delay over 65 seconds (unsigned -1 = 65,535).

There may be a slight time variation from machine to machine due to processor speed, interrupts, hardware differences, etc.

Also see DATE$ and TIME$.

Also see TIMER.

DELETE command

FORMAT **DEL** [ETE] *line*
DEL [ETE] *-line*
DEL [ETE] *line - line*
DEL [ETE] *line-*

DEFINITION This command will remove a line or range of lines from a program in memory.

DELETE is used from the Standard Line Editor.

EXAMPLE

```
10  CLS
20  FOR I = 1 TO 10
30  PRINT  "NUMBER "; I
40  NEXT I
50  END

DEL  10-20

LIST

30  PRINT  "NUMBER "; I
40  NEXT I
50  END
```

```
10  "FRED" PRINT  "NUMBER ";I
20  PRINT "Fred was here"
30  END

DELETE "FRED"

LIST

20  PRINT "Fred was here"
30  END
```

REMARK Use this command with care as recovery of deleted lines is not possible.

FORMAT **DIM** [*len*] *var* [*type*] [(*number* [,*number* ..])][,…]

DEFINITION The DIM statement is used to allocate memory for variables and array variables and to define common variables for chained programs.

len Defines the length a of a string (how many characters it may hold). This is optional and defines the length of all the following string variables in that DIM statement or until a new length is encountered in that statement. The default is 255 characters unless changed by a previous DEFLEN.

var The name of a variable (any variable type).

type Forces the *variable* to be of that *type*.

%=Integer
&=LongInteger (Macintosh only)
!=Single Precision
#=Double Precision
$=String

Also see "Variables" in the front section of this manual.

number The maximum number of elements that a dimension may contain from 1 to 32,767 elements (add one if array BASE option is set to zero. default=0). Only numbers may be used, not variables.

EXAMPLE See the following page for more information and examples.

REMARK Use care when allocating memory with the DIM statement.

See BASE OPTION,DEFLEN,"Array Variables","String Variables",INDEX$ and RUN+ for more important information about using DIM.

Macintosh: This version is limited to 2,147,483,648 elements in an array.
MSDOS: In order to optimize performance; integer variables and integer array variables are limited to one 64k segment. String and BCD arrays may cross segment borders to use up to available memory.

continued next page...

DIM statement

DIM continued

DETERMINING THE MEMORY NEEDS OF DIMMED ARRAYS

```
DIM A%(10,10,10), A#(5), A!(9,7), B$(10), 5Cool$(20
DIM Long&(10):  REM Macintosh Only
```

The following chart shows how to calculate the memory requirements of the arrays dimensioned above with a BASE OPTION of zero.

ARRAY	TYPE	Bytes per Element	How to Calculate**	Memory Required
A%(10,10,10)	Integer	2	11*11*11*2	2662
A#(5)	Double Precision	8	6*8	48
A!(9,7)	Single Precision	4	10*8*4 320	
B$(10)	String	256	11*256 2816	
Cool$(20)	String	6	21*6	126
Long&(10)	LongInteger	4	11*4	44

DEFINING STRING LENGTHS WITH DIM

```
DIM X$(10), 20A$, Z$(5), 45TEST$, 10MD$(20,20)
```

In the example above the maximum character capacities are:

X$	255 (default is 255)
A$	20
Z$ (5)	each element if Z$ as 20* (21*5=105 total bytes)
TEST$	45
MD$(20,20)	each element of MD$(20,20) as 10. (20*20*11=4400 total bytes of memory used)

* If no length is defined, the last given length in that DIM statement is used. In the example each element of Z$(n) gets a length of twenty. If no length is defined in that DIM statement then 255 characters is the default (or the last length used in DEF LEN).

**If you configure BASE OPTION 1 you will not need to add one to the dimension. To calculate the memory required for A%(10,10,10): 10*10*10*2. See "Configure".

Note: Add one to the defined length of each string for the length byte to determine the actual memory requirement of the string. This extra byte is the "Length byte" and it is the first byte in the string. It is wheat is pointed at by VARPTR(var$).

Important Note: Unpredictable system errors may result if an attempt is made to assign a string variable a string longer then its allocated length. It is also important to define the length of a string at least one greater than the maximum number of characters received in an INPUT or LINEINPUT statement.

FORMAT **DIR**[*drivespec*]

DEFINITION **DIR** will display the directory of the disk drive specified by *drivespec*.

The drivespec will vary from one computer to the next. See your Computer's Disk Operating System reference manual for syntax.

EXAMPLE **DIR** <ENTER>

```
LEDGER.COM     MAY.LEDJUN.LED
JUL.LEDAUG.LED

ZBasic Ready
```

REMARK The appearance of the directory layout will vary by computer. See appendix for further information. This is a command so it does not operate during runtime.

See below, or your appendix, for possible ways of getting directories at runtime.

Macintosh: Syntax is DIR "rootname or foldername". To get a directory during runtime see FILES$ in the appendix. LDIR will output the directory to a printer.

MSDOS: Use DIR *.BAS to see all the .BAS files or DIR Z*.* to see all the files starting with Z. To get a directory during runtime see FILES.

Apple ProDOS: To get a directory during runtime; OPEN"I" the directory pathname. Example: OPEN"I",1,"ZBASIC". See directory layout in ProDOS reference manual for more information about directory file layout. This version also supports LDIR to list the directory to the printer. CAT may be used as well as DIR.

Apple DOS 3.3: To get a directory during runtime:

```
LONG FN DIR (slot,drive)
  POKE &AA6A,slot
  POKE &AA68, drive
  CALL &A56E
END FN
```

Z-80: See appropriate section in appendix for your computer and DOS. Some Z80 versions do not allow getting a directory at runtime.

DO statement

FORMAT **DO**
.
.
UNTIL *expression*

DEFINITION The DO statement is used to define the beginning of a loop with the UNTIL statement defining the end.

Program functions and statements appearing between the DO and UNTIL will be executed over and over again until the expression defined at the UNTIL statement is TRUE.

EXAMPLE

```
DO
  PRINT"Hi!"
UNTIL LEN(INKEY$)
END

RUN

Hi!
Hi!
Hi!
Hi!             <-----You press a key and it stops
```

```
DO
  X=X+1
UNTIL X=2492
PRINTX
END

RUN

2492
```

REMARK The statements in a DO loop will be executed at least once. See WHILE-WEND for a loop type that ends immediately if the condition is false.

ZBasic automatically indents text appearing between a DO and UNTIL two spaces. This is helpful in debugging and documenting programs.

See the "Structure" and "Loops" sections of this manual for more information.

Also see FOR-NEXT-STEP and WHILE-WEND.

FORMAT **E** *line*
EDIT *line*

DEFINITION EDIT is used from the Standard Line Editor to specify the line you wish to edit.

EDIT may be abbreviated to E. A comma will start editing at the line currently selected by ZBasic's line pointer. List of the EDIT sub-commands:

SUB-COMMAND	DEFINITION
[n]**<SPACE>**	- MOVE CURSOR RIGHT (n characters)
[n]**<BACKSPACE>**	- MOVE CURSOR LEFT (n characters)
I	- Begin INSERT mode at cursor position
X	- Goto the end of the line and EXTEND it
<ESC>	- Exit INSERT mode (you will still be in line edit mode)
[n]**D**	- DELETE characters (if n is used deletes n characters)
[n]**C** key	- CHANGE character to <key> [n] times
H	- HACK to end of line and enter INSERT
[n]**S** key	- SEARCH for [n]the occurrence of <key>
L	- LIST line being edited, home cursor
A	- ABORT changes, restore original line
[n]**K** key	- KILL text to [n]the occurrence of <key>
<ENTER>	- EXIT editing with changes intact
<BREAK>	- ABORT EDIT SESSION (no changes made)

Note: n is a number from 1 to 255. If n is not used, one is assumed.

EXAMPLE

```
10  FOR I = 1 TO 20
20    PRINT I
30  NEXT I
```

```
EDIT 20          <---- or E20 (comma if 20 was the last line used.)

20 _             <---- Press spacebar or backspace to move cursor.
                       Use keys above to edit this line.
```

REMARK If you want to edit the current line, press the comma key <,> in command mode. It will do the same as E <ENTER>.

Line numbers may be edited in ZBasic. The line being edited will remain unchanged, the edited line with the new line number will be created.

See the "Standard Line Editor" section in the beginning of this manual.

Also see FIND,DELETE,AUTO and LIST.

These versions offer full screen editors as well as the Standard Line Editor. See "Full Screen Editor" in the appropriate appendix for details.

ELSE statement

FORMAT IF-THEN-**ELSE** *line or label*
IF-THEN-**ELSE** *statement(s)*

DEFINITION ELSE is used with an IF statement to route control on a false condition.

ELSE may refer to a linenumber or label or it may be followed by one or more statements that will be executed if the condition in the IF statement is FALSE.

EXAMPLE

```
X=99
IF X = 100 THEN STOP ELSE PRINT X
END

RUN

99
```

```
IF X=100 THEN STOP ELSE "End"
END
:
PRINT"Stopped here."
END

RUN

Stopped here.
```

REMARK All statements on a line following an ELSE are conditional on that ELSE.

See "Structure",IF-THEN,LONG IF,XELSE and ENDIF.

Also see SELECT CASE.

FORMAT **END**

DEFINITION END is used to stop the execution of a program.

END will return control to the Standard Line Editor if program was executed using RUN, or to the operating system if the program was compiled using RUN* or RUN+.

EXAMPLE

```
PRINT "HELLO"
END
PRINT "THERE"
```

```
RUN

HELLO
```

REMARK END will close all open files.

Also see STOP and TRONB.

See SHUTDOWN.

END FN statement

FORMAT

LONG FN
.
.
END FN [= *expression*]

DEFINITION

Marks the end of a LONG FN statement.

The optional expression MUST be numeric for numeric functions (#,%,&,!) and MUST be a string ($) for string functions.

EXAMPLE

```
REM Removes spaces from the end of a string
LONG FN RemoveSpace$(x$)
  WHILE ASC(RIGHT$(x$,1)=32
    x$= LEFT$(x$, LEN(x$)-1)
  WEND
END FN= x$
Name$="ANDY        "
PRINT "Before:";Name$;"*"
PRINT" After:"; FN RemoveSpace$(Name$);"*"
```

RUN

```
ANDY      *        ANDY*
```

```
REM Example of a simple Matrix Multiplication
DIM A%(1000)
:
LONG FN MatrixMult%(number%, last%)
  FOR temp%= 0 TO last%
    A%(temp%)=A%(temp%)*number%
  NEXT
END FN
:
A%(0)=1: A%(1)=2:A%(2)=3
FN MatrixMult%(10,3)
PRINT A%(0), A%(1), A%(2)
```

RUN

```
10            20            30
```

REMARK

If an END FN is omitted in a LONG FN construct, a structure error will occur. You must exit a function from and END FN otherwise problems will occur internally.

Also see "Functions and subroutines","Structure",LONG FN,FN statement,FN function and DEF FN.

Important Note: Loops like FOR-NEXT, DO-UNTIL or WHILE-WEND must be entirely contained within a LONG FN-END FN. Do not exit a function except at the END-FN.

FORMAT LONG IF *expression*
.
[XELSE]
.
END IF

DEFINITION This is an end marker for the LONG IF statement.

Program execution will continue normally at the END IF after completion of a LONG IF or XELSE.

EXAMPLE
```
Love$="Forever"
LONG IF Love$="Forever"
  PRINT "How Romantic!"
XELSE
  PRINT "How heartbreaking!"
END IF
END

RUN

How Romantic!
```

REMARK If an END IF is omitted in a LONG IF construct, a structure error will occur.

See "Structure",LONG IF,IF-THEN,ELSE and XELSE.

Also see SELECT CASE.

END SELECT statement

FORMAT

```
SELECT [CASE] [expression]
  CASE [IS] relational condition1[,relational condition][,...]
    statement(s)
  CASE [IS] condition[,condition][,...]
    statement(s)
  CASE [IS] boolean expression
    statement(s)
  CASE ELSE
    statement [:statement:...]]
END SELECT
```

DEFINITION

END SELECT is the end marker for the SELECT/CASE structure.

When SElECT/CASE is encountered, the program checks the value of the controlling expression or variable, finds the CASE that compares true and executes the statements directly following the CASE statement. After these statements are performed, the program continues at the line after the END SELECT statement:

EXAMPLE

```
A=100
SELECT A
  CASE >100
    PRINT "A>100"
  CASE 100
    PRINT "A=100"
  CASE ELSE
    PRINT"None of the above"
END SELECT
PRINT "Program continues..."
END

RUN

A=100
Program continues...
```

REMARK

Also see SELECT and CASE.

SELECT CASE is not supported with the Z80 versions. See IF and LONG IF for accomplishing the same thing.

SELECT CASE is not supported with this version. See IF and LONG IF for accomplishing the same thing.

FORMAT **EOF** (*filenumber*)

DEFINITION Returns true if end-of-file condition exists for filenumber, returns zero if the end-of-file has not yet been reached. This function is only available on the Macintosh and MSDOS versions of ZBasic.

EXAMPLE

```
OPEN"I",1,"FILE.TXT"
DO
  LINEINPUT#1, A$
  PRINT A$
UNTIL EOF(1)
CLOSE#1
END
```

What to do if you don't have EOF on your computer.

```
ON ERROR GOSUB 65535 <--- Enable disk error trapping
OPEN"I",1,"FILE.TXT"
IF ERROR GOSUB"Error message"
DO
  LINEINPUT#1, A$
  PRINT A$
UNTIL ERROR <>0
IF ERROR <> 257 THEN GOSUB "Error messsage"
ERROR=0 <---Error 257 is an end-of-file error. Reset Error here then continue.
CLOSE#1
END
:
"Error message"
PRINT "A disk error occured: "; ERRMSG$(ERROR)
INPUT"<C>ontinue or <S>top? ";temp$
If temp$="C" THEN ERROR=0:RETURN
STOP
```

REMARK Some versions of ZBasic do not support EOF because of system reasons. Also see ERROR function and statement, ON ERROR and ERRMSG$

EOF is not supported on Z80 versions of ZBasic. Use the second example above to accomplish the same thing.

EOF is not supported on the Apple // ProDOS or DOS 3.3 versions of ZBasic. Use the second example above to accomplish the same thing.

ERRMSG$ function

FORMAT **ERRMSG$** (*expression*)

DEFINITION Returns the error message string for the error number specified by expression. In most cases you will use the number returned by the ERROR function when a disk error has occured.

EXAMPLE

```
OPEN "I",1, "OLDFILE"
ON ERROR GOSUB "Error message"
.
.
.
"Error message"
PRINT "A disk error has occurred!!"
PRINT "The error was: ";ERRMSG$(ERROR)
ERROR=0:REM ALWAYS SET ERROR TO ZERO AFTER ERROR OCCURS!
RETURN

RUN

A disk error has occured!!
The error was: File Not Found Error in File #1
```

```
FOR X=0 TO 255
  PRINT ERRMSG$(X)
NEXT X

RUN

PRINTS ALL THE ERROR MESSAGES FOR THAT COMPUTER.
```

REMARK ZBasic will display disk errors for you unless you use the ON ERROR disk trapping options.

The ERROR function is commonly used for error trapping and display purposes. The expression is stored as follows:

The low byte is used for the ERROR number	(ERROR AND 255)
The high byte is used for the file number	(ERROR>>8) or (ERROR/256)

See "Disk Errors", ON ERROR GOSUB and ERROR functions and statements.

FORMAT **ERROR**

DEFINITION Returns the number of an ERROR condition, if any.

Zero (0) is returned if no error has occured.

This function is available to programmers who wish to trap disk errors using the ON ERROR statement.

EXAMPLE

```
ON ERROR GOSUB 65535:REM User disk trapping enabled
OPEN "I",1,"OLDFILE"
IF ERROR=259 GOSUB"NOT FOUND"" GOTO 20
ON ERROR RETURN: REM Let ZBasic do the error checking now!
.
.
.
"NOT FOUND"
REM ERROR 259 is: File Not Found error in Filenumber 1
PRINT" The file is not on that disk!"
PRINT" Please insert the correct disk"
PRINT" and press <ENTER>"
INPUT A$:ERROR=0:RETURN
```

REMARK ERROR may also be used as a statement. See ERROR statement, ERRMSG$ and ON ERROR GOSUB.

Important Note: If you do the disk error trapping, ERROR must be reset to zero after a disk error occurs or ERROR function will continue to return an error value.

Macintosh: Also see SYSERROR in appendix.

MSDOS: See appendix for ways of doing critical error handling.

Apple ProDOS: See appendix for additional ways of trapping ProDOS errors.

ERROR statement

FORMAT **ERROR** [=] *expression*

DEFINITION Allows the programmer to set or reset ERROR conditions for the purpose of disk error trapping.

Important Note: If you do the disk error trapping, ERROR must be reset to zero after a disk error occurs or ERROR function will continue to return an error value.

EXAMPLE

```
REM This routine checks to see if a file exists. If it
REM does exist it is opened as random, if it doesn't
REM exist an error message is returned.
:
LONG FN Openfile%(files$, filenum%, reclen%)
  ON ERROR GOSUB 65535: REM Disk error trapping on
  "Open file"
  OPEN"I",filenum%,file$
  LONG IF ERROR
    LONG IF (ERROR AND 255) <>3
      PRINT@(0,0);"Could not find: ";file$;" Check drive"
      INPUT"and press <ENTER> when ready";temp%
      ERROR=0: GOTO "Open file"
    END IF
  XELSE
    CLOSE# filenum%
  END IF
ON ERROR RETURN: REM Give error checking back to ZBasic
OPEN"R",filenum%, file$, reclen%
END FN
```

REMARK ERROR may also be used as a function. See "Disk Error Trapping",ERROR function, ERRMSG$ and ON ERROR.

Macintosh: Also see SYSERROR in appendix.

MSDOS: See appendix for ways of doing critical error handling.

Apple ProDOS: See appendix for additional ways of trapping ProDOS errors.

FORMAT **EXP** (*expression*)

DEFINITION Returns e raised to the power of expression. This function is the compliment of LOG. The BCD internal constant of the value of e is:

2.71828182845904523536028747135266249775724709369995957

The result will be rounded to the digits of precision configured for Double Precision accuracy.

EXAMPLE

```
DEFDBL A-Z
DO
  INPUT "ENTER A NUMBER ";X
  PRINT "e RAISED TO X =" ; EXP(X)
UNTIL X=0
END
```

RUN

```
ENTER A NUMBER  _ 1
e RAISED TO X = 20718281828459 <--- 14 digit accuracy
```

REMARK This is a scientific function. See "Configure" for information about configuring scientific accuracy.

For more information about scientific functions see "Math","Math expressions", "Floating Point Variables", COS,SIN,ATN,TAN,SQR and raise to the power"A".

FILL statement

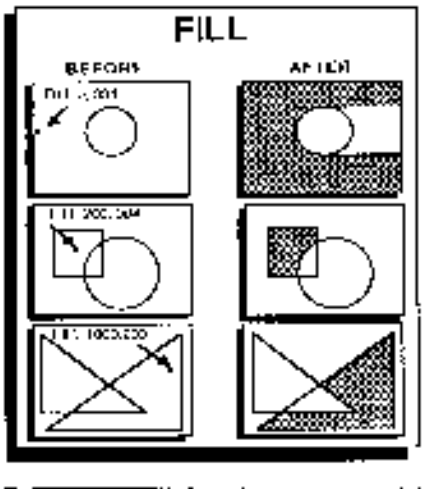
FORMAT **FILL** *expression*$_X$, *expression*$_Y$

DEFINITION The purpose of FILL is to paint an area of the screen in the current COLOR. The point defined by the two expressions are:

expression$_X$ (horizontal position) and *expression*$_Y$ (vertical position).

Fill will search for the uppermost point in the contained area that has the background color, then start filling from left to right and down. For this reason irregular shapes may not fill completely with one fill command. It may be necessary to use a fill statement for each appendage.

EXAMPLE




```
COLOR=1
FILL 0,284
```

RUN

See chart.

REMARK FILL may not be available on machines without the capability of seeing pixels on the screen. See computer appendix. Also see CIRCLE FILL,BOX FILL,POINT and PLOT.

BOX FILL,CIRCLE FILL and the QuickDraw routines like FILLPOLY,FILLRGN, FILLRECT etc. are much faster ways of filling areas.

FORMAT

FIND	*commands or keywords*
FIND #	*line*
FIND "	*quoted string text or labels*
FIND REM	*items in REM statements*
FIND DATA	*items in DATA statements*

DEFINITION FIND is used in the Standard Line Editor to locate text in a program.

To FIND additional occurrences, press semi-colon (;) or FIND <ENTER>.

EXAMPLE

YOU TYPE	ZBASIC FINDS
`FIND "HELLO`	`01010 A=20:PRINT"HELLO THERE"`
`FIND A$`	`01022 Z=1:A$=B$:PRINTA$+B$`
`or...`	`01222 BA$="hello"`
`or...`	`01333 ABA$="goodbye"`
`FIND 99`	`05122 F=2:X=X+2+F/999`
`FIND #12345 (line#)`	`08000 GOTO 12345`
`FIND X(C)`	`03050 A=1:T=ABS(X(C)/9-293+F)`
`or...`	`03044 ZX(C)=4`
`FIND PRINT`	`00230 A=92:PRINTA`
`FIND "SUB5`	`00345 "SUB500": CLS`
`or...`	`03744 GOSUB "SUB500"`
`FIND OPEN`	`03400 OPEN"R",1,"FILE54",23`
`FIND CLOSE`	`09900 CLOSE#2`
`FIND REM This`	`02981 REM This is a remark`
`FIND DATA 123, 232`	`09111 DATA 123, 232`
`FIND DATA "Fred"`	`10233 DATA "Tom", "Dick", "Fred"`

REMARK When finding a string inside quotes, you must supply all of the characters up to the point that will insure the uniqueness of the string.

See "Standard Line Editor" in the beginning of this manual.

See "Full Screen Editor" in the appropriate appendix for other FIND commands.

FIX function

FORMAT **FIX** (*expression*)

DEFINITION Truncates the digits on the right side of the decimal point.

EXAMPLE

```
PRINT FIX (123.456),
A#=1293.21
PRINT FIX(A#),
PRINT FIX (.12340),
PRINT FIX (999999.455)  + 0.

RUN

123           1293          0             999999
```

REMARK FIX works the same as INT in ZBasic. They are both included to maintain compatibility with other forms of BASIC. FIX will consider an expression floating point.

FRAC is the opposite of FIX. It returns the fraction part of the number.

See FRAC and INT.

FORMAT **FN** *name* [(*expression*$_1$ [,*expression*$_2$ [,...]])]

DEFINITION FN calls a function by name which was previously defined by DEF FN or LONG FN. The name of the function must follow the syntax of variable names, that is, a string FN must have a name with a $, and integer FN must have a name with a %, etc.

The expressions must match the variable types as defined by the DEF FN or LONG FN. Numeric expressions are not a problem, string expressions allow only simple strings.

FN may not be used before it is defined with DEF FN or LONG FN.

EXAMPLE

```
DEF FN e# = EXP(1.)
DEF FN Pi#= ATN(1) << 2
DEF FN Sec#(x#) = 1.\ COS(x#)
DEF FN ArcSin#(x#) = ATN (x# \ SQR(1-x# * x#))
:
PRINT FN Pi#
```

RUN

```
3.14159... <---Returned in the current digits of accuracy
```

```
REM Round number to the number of places indicated.
LONG FN ROUND#(number#, places)
  number#=INT(number#*10^places+.5)/10^places
END FN=number#
:
PRINT FN ROUND#(43343.327, 2)
```

RUN

```
43343.33
```

REMARK This function is useful for saving program space and for making a program easier to read.

Also see "Functions and Subroutines","Structure",LONG FN,END FN,DEF FN, APPEND and FN statement.

FN statement

FORMAT **FN** *name* [(*expression*$_1$ [,*expression*$_2$ [,...]])]

DEFINITION FN calls a function by name which has previously been defined by a DEF FN or a LONG FN.

The expressions must match the variable types as defined by DEF FN or LONG FN.

EXAMPLE

```
DEF FN LastChr%(x) = PEEK( x + PEEK(x))
LONG FN RemoveSpace$(x$)
  WHILE FN LastChr$(VARPTR(x$)) = ASC("  ")
  x$= LEFT$(x$, LEN(x$)-1)
  WEND
END FN= x$
Name$="ANDY       "
PRINT Name$;"*", FN RemoveSpace$(Name$);"*"

RUN

ANDY          *                ANDY*
```

REMARK Also see "Functions and Subroutines","Structure",LONG FN,END FN,DEF FN, APPEND and FN function

FORMAT **FOR** *variable* = *expression*$_1$ **TO** *expression*$_2$ [**STEP** *expression*$_3$]

.
.
.
NEXT[*variable*][,*variable* ...]

DEFINITION Permits the repeated execution of commands within the loop.

A FOR/NEXT loop will automatically increment variable by the amount set by STEP and compare this to the end value, expression2, exiting the loop when var exceeds this value after adding STEP. Default STEP = 1.

Note the loop will be executed at least once with the value of expression1.

EXAMPLE

```
FOR Counter = 0 TO 100 STEP 2
  PRINT Counter;
NEXT
```

RUN

```
0 2 4 6 8 10 12 ...  100
```

```
FOR Counter = 100 TO 0 STEP -2
  PRINT Counter;
NEXT Counter
```

RUN

```
100 98 96 94 92 90 88 ... 0
```

```
FOR Counter# = 0.0 TO 1.0 STEP .01
  PRINT Counter#;
NEXT Counter#
```

RUN

```
0 .01 .02 .03 .04 ... 1
```

REMARK ZBasic will automatically indent all of its loop structures in listings. This is helpful in debugging and documenting programs.

See chapter called "Loops" and WHILE-WEND and DO-UNTIL.

Note: If STEP is set to zero, the program will enter an endless loop. If the variable is an integer, do not allow the loop to exceed 32,767 or you will enter an endless loop (unsigned integer).

FRAC function

FORMAT **FRAC** (*expression*)

DEFINITION FRAC returns the fractional part of expression. The digits to the left of the decimal point will be truncated.

This function is the compliment of INT and FIX.

EXAMPLE

```
A#=123.456
B#=99343.999
C#=3.5
:
PRINT A#, FRAC(A#)
PRINT B#, FRAC(B#)
PRINT C#, FRAC(C#)
PRINT 2.321, FRAC(2.321)

RUN

123.456.456
99343.999      .999
3.5            .5
2.321          .321
```

REMARK This function will automatically set floating point calculation.

FIX and INT are the opposite. They return the whole part of the number.

See FIX and INT.

FORMAT **GET** (*x1,y1*)-(*x2,y2*), *variable*[*array*(*index*[,*index*…,])]

DEFINITION Stores a graphic image from the screen into a variable or variable array so that it may be retrieved later and put to the screen with PUT.

GET and PUT are extremely fast and useful for sophisticated graphic animation.

x1,y1 Coordinates of the upper-left-corner of the graphic image on the screen.
x2,y2 Coordinates of the lower-right-corner of the image.

Coordinates are pixel coordinates; use with COORDINATE WINDOW.

The image is normally stored in memory specified by an integer array since it is easier to calculate how much memory is required this way (although other variables may also be used as long as the memory set aside is correct).

To calculate the amount of bytes to DIM for a graphic image, use this equation. Bits-per-pixel (bpp) has to do with colors or grey levels available. See next page for specifics:

```
6+ (( y2-y1)+1) * ((x2-x1+1) * bpp +7) / 8)
```

Failure to DIM enough memory for an image will cause unpredictable system errors so be sure to carefully calculate the memory needed.

EXAMPLE

```
DIM A(750)                    <---Bytes above divided by two for integer array
MODE 7                        <---Not needed on the Macintosh version
COORDINATE WINDOW             <---Pixel coordinates
:
CIRCLE 100,100,80
GET (0,0)-(100,100), A1
:
FOR x= 1 TO 200 STEP 3
  PUT (x, 90), A(1)   <---Does twice to move the image across
  PUT (x, 90), A(1)       the screen without disturbing the background
NEXT x
:
END
```

This routine moves a section of a circle across the screen. It is PUT to the screen twice so the item doesn't repeat and it will appear to move across the screen without disturbing the background (default PUT mode is XOR).

continued...

GET statement

REMARK **Important Note:** Failure to DIM enough memory for the variables storing the graphic images may result in unpredictable system problems.

Also see DIM and PUT.

Macintosh: With this version of ZBasic, PUT has another, optional, parameter: `PUT (x1,y1) [-(x2,y2)]`, var. The second parameter allows you to scale the image, making it either larger or smaller by giving the rectangle size in which it is to appear. The x2,y2 parameter is the lower-right corner of the image.

Bits-per-pixel (bpp) will vary by the type of Macintosh you have. The standard black and white Macintoshes have one bit per pixel.

The Macintosh II may have up to 32 bits-per-pixel. Sixteen colors is 4 bpp, 256 colors is 8 bpp. Check addendum or "Inside Macintosh Volume V (Color Quickdraw)" for the specifics of your color board.

MSDOS: Bits per pixel (bpp) will vary by the graphics adaptor board being used:

TYPE	MODE(s)	COLORS	BITS PER PIXEL (bpp)
CGA	5	4	2
CGA	7	2	1
EGA	16-19	3-16	2 (64K or less on EGA card)
EGA	16-19	16	4 (More than 64K on card)
HERCULES	20	1	1

Z80: GET and PUT are not supported with these versions of ZBasic.

Apple // ProDOS and DOS 3.3: GET and PUT are not supported with these versions. See DRAW example on ProDOS disk and the BLOAD and BSAVE functions for possible alternatives.

FORMAT **GOSUB** *line or label*

DEFINITION GOSUB will call that part of a program starting with line or label and return to the next statement following the GOSUB when RETURN is encountered.

EXAMPLE

```
10 GOSUB 40: PRINT "All Done!"
20 END
30 :
40 PRINT"Hello"
50 RETURN

RUN

HELLO
All Done!
```

```
GOSUB "Hello Routine"
PRINT "All Done!"
END
:
"Hello Routine"
PRINT "Hello"
RETURN

RUN

HELLO
All Done!
```

REMARK On multiple statement lines, a RETURN will return control to the next statement on the line following the originating GOSUB.

To avoid errors, be certain there is a line with the number or label that you GOSUB. All subroutines must be terminated with a RETURN statement.

Note: If ZBasic encounters a RETURN without a matching GOSUB, it will return to the operating system or the editor. ZBasic does not check for stack overflow which may cause errors if subroutines do not end with a RETURN.

See RETURN LINE,GOTO, ON GOTO and ON GOSUB.

See SEGMENT RETURN in appendix.

GOTO statement

FORMAT **GOTO** *line or label*

DEFINITION **GOTO** will transfer control to a line or label in a program.

Note that excessive use of this statement is considered inappropriate for structured code because in complex programs it becomes extremely hard to read.

In most programming situations GOSUB, DO-UNTIL, WHILE-WEND, FOR-NEXT or other programming structures are much easier to follow.

EXAMPLE

```
10 X=X+1
PRINT X,
20 IF X<5 THEN GOTO 10
```

RUN

```
1       2       3       4
```

```
"Loop"
X=X+1
PRINT X,
IF X<5 THEN GOTO "Loop"
```

RUN

```
1       2       3       4
```

REMARK A line error will occur during compile if the destination line or label cannot be found.

See "Structure", GOSUB,ON GOTO,ON GOSUB,LONG FN,FN statement,WHILE, DO,FOR,LONG IF.

FORMAT **HELP** [*number*]

DEFINITION HELP without a number prints the HELP menu to the screen. This menu will give you corresponding numbers to the help topics available. This command is used from the Standard Line Editor.

Type HELP and a number to get answers to a specific topic.

Press the SPACE BAR to continue when you see "MORE".

EXAMPLE

```
HELP
```

A menu for your version of ZBasic will be printed to the screen. To get help for an item in the menu, type HELP and the number corresponding to that item.

REMARK HELP will return control to the Standard Line Editor upon completion of the listing.

If the help file has been deleted from the disk a File Not Found Error will occur. Check your computer appendix for the filename of the HELP file.

The HELP window is brought up when you type this command or select "About ZBasic" under the menu. The command does not work exactly as above. Just double click the appropriate item with the mouse.

HEX$ function

FORMAT **HEX$**(*expression*)

DEFINITION The HEX$ function converts a numeric expression to a four character HEXadecimal string (BASE 16). The following program will convert a Decimal number to HEX or HEX to Decimal. Some sample HEX numbers:

Decimal	Hexadecimal
0-9	0-9
10	A
11	B
12	C
13	D
14	E
15	F

EXAMPLE

```
DO
  INPUT"Decimal number to convert: ";Decimal%
  PRINT "Decimal";Decimal%;"= HEX ";HEX$(Decimal%)
  PRINT
  :
  INPUT"HEX number to convert: ";Hx$
  Hx$="&H"+Hx$
  PRINT"Decimal value of ";Hx$;"="VAL(Hx$)
  PRINT"The unsigned Decimal value of "Hx$"=" UNS$(VAL(Hx$))
UNTIL (Decimal% =0) OR (LEN(Hx$)=2)

RUN

Decimal number to convert: 255
Decimal 255= HEX FF

HEX number to convert: F9CD
Decimal value of  F9CD = -1587
The unsigned Decimal value of F9CD = 63949
```

REMARK Floating point numbers will be truncated to integers.

See "Numeric Conversions",VAL,OCT$,BIN$ and UNS$.

See DEFSTR LONG in the appendix for doing LongInteger conversions in Hex, Octal,CVI and MKI$. In this case HEX$ would return an eight character string.

FORMAT **IF** *expression* **THEN** *line* [*or label*][**ELSE** *line* [*or label*]]
IF *expression* **THEN** *statement* [:*statement*: …]**[ELSE** *statement* [:*statement*: …]]

DEFINITION The IF statement allows a program to do a number of things based on the result of expression:

1.Branch to a line or label after the THEN if a condition is true; expression /=0
2.Execute statement(s) after the THEN if a condition is true; expression /=0
3.Branch to a line or label after the ELSE if a condition is false; expression=0
4.Execute statement(s) after the ELSE if a condition is false;expression=0

EXAMPLE

```
X=99
IF X=99 THEN PRINT"X=99":PRINT"HELLO: ELSE STOP
:
IF X=99 THEN "CHECK AGAIN"
END
:
"CHECK AGAIN"
IF X=100 THEN PRINT"YEP" ELSE PRINT"NOT TODAY!";:PRINT X
END

RUN

X=99
HELLO
NOT TODAY! 99
```

REMARK Complex strings will generate an error if used in an IF statement.

Improper	IF LEFT$(A$,2)="HI"THEN STOP
Proper	B$=LEFT$(A$,2):IF B$="HI" THEN STOP

See LONGIF,ELSE,XELSE,WHILE-WEND and DO-UNTIL for more ways of doing program comparisons.

Note: In many cases LONG IF is easier to read.

Also see SELECT CASE

INDEX$ statement

FORMAT

INDEX$ (*expression*) = *string expression*
INDEX$I (*expression*) = *string expression*
INDEX$D (*expression*)

DEFINITION INDEX$ is a special array unique to ZBasic. Expression indicates an element number.

Statement	Definition
INDEX$(n)=simple string	Assigns a value to INDEX$(n)
INDEX$**I**(n)=simple string	Move element n (and all consecutive elements) up and **INSERT** simple string at INDEX$ element n
INDEX$**D**(n)	**DELETE** element n and move all consecutive elements down to fill the space.

EXAMPLE

```
INDEX$(0)="FRED"                    <---Normal assignments
INDEX$(1)="TOM"
INDEX$(2)="FRANK"
:
GOSUB"Print INDEX$"
INDEX$I(1)="HARRY"                  <---HARRY INSERTED between FRED and TOM
GOSUB"Print INDEX$"
:
INDEX$D(0)                          <---FRED is DELETED here
GOSUB"Print INDEX$"
END
:
"Print INDEX$": REM Routine prints contents of INDEX$
FOR X=0 TO 4
  PRINT X; INDEX$ (X)
NEXT: PRINT
RETURN
```

```
RUN
0 FRED
1 TOM
2 FRANK

0 FRED
1 HARRY
2 TOM                   <--- Notice how values move from one element to another
3 FRANK                      as items are inserted and deleted with INDEX$I and D.

0 HARRY
1 TOM
2 FRANK
```

REMARK INDEX$ provides for memory efficient string array manipulation and lends itself very well to list management applications. See "Special INDEX$ Array",INDEX$ function, CLEAR,CLEAR INDEX$ and MEM.

Allows up to ten simultaneous INDEX$ arrays. See INDEX$ in your appendix.

FORMAT **INDEXF** (*string* [,*expression*])

DEFINITION INDEXF is a special INDEX$ array function used to FIND a leading string within an INDEX$ array quickly.

IF INDEX$(1000) equaled "Hello", then X=INDEXF("Hel") would return 1000.

IF X=INDEXF("llo") X would equal -1 since "llo" would not be found. The leading characters are significant.

EXAMPLE

```
INDEX$(0)="FRED"
INDEX$(1)="MARY"
INDEX$(2)="TOM"
:
X=INDEXF("TOM")              <--- Search for TOM
PRINT X
:
PRINT INDEXF("MARY")  <--- Search for MARY
:
PRINT INDEXF("RED")   <--- Search for RED
:
PRINT INDEXF("FRED",1)       <--- Search for FRED starting at element 1

RUN

2       <----- TOM found at element two
1       <----- MARY found at element one
-1      <----- RED not found. The first characters are significant
-1      <----- FRED not found because search started at element 1
```

REMARK INDEX$ provides for memory efficient string array manipulation and lends itself very well to list management and text editing applications.

See "Perpetual Sort" under "Special INDEX$ Array". Also see INDEX$,INDEX$I, INDEX$D,CLEAR,CLEAR INDEX$ and MEM.

Allows up to ten simultaneous INDEX$ arrays. See INDEX$ in your appendix.

INKEY$ function

FORMAT **INKEY$**

DEFINITION INKEY$ returns the character of the last key that was pressed or an empty string if no key was pressed.

EXAMPLE

```
WHILE A$<>"S": REM Press "S" to Stop
  DO
    A$=INKEY$
  UNTIL LEN(A$)
  A$=UCASE$(A$)
  PRINT A$;
WEND
END

RUN

GHUIJD,KEUG FAQCCQ OPU...S <---When <S> is pressed program stops
```

```
REM An easy function you can use to get a key
LONG FN Waitkey$(local$)
  DO
    local$=INKEY$
  UNTIL LEN(Local$)
END FN=local$
:
key$=FN Waitkey$(key$)
PRINT key$
END

RUN
```

(user presses "b")

```
b
```

REMARK When using INKEY$ for character entry, avoid having the TRON function active as this may cause pressed keys to be missed.

See INPUT,LINEINPUT,INPUT#,ASC and CHR$. See your computer appendix for variations or enhancements.

Macintosh: See DIALOG (16) for way of doing INKEY$ during event trapping.
MSDOS: INKEY$ returns two characters for function keys. ON INKEY$ does event checking for function keys. See appendix for specifics.

FORMAT **INP** (*expression*)

DEFINITION The INP function is used to read an input port. The function returns the value that is currently at the port specified by expression.

EXAMPLE

```
X=INP(1)
PRINT X
PRINT INP(G-1)

RUN

0
255
```

REMARK Note: This function requires a knowledge of your computer hardware and may not be portable to other computers (may not be available on your version of ZBasic or may have an unrelated function).

See your computer appendix for specifics.

Not supported with this version. See INSLOT.

Not supported with this version. See OPEN"C" and "Toolbox" in the appendix for accessing hardware ports.

INPUT statement

FORMAT **INPUT**[(@ or %)(*expr*^{X},*expr*^{Y})][;][!][&*expr*,]["*string*";] *var*[,*var* …]

DEFINITION The INPUT statement is used to input values (string or numeric) from the keyboard into variables.

Multiple variables must be separated by commas (this is bad form since users often forget commas). If no value in INPUT, a zero or null string will be returned.

@(*xpr*^{X},*expr*^{Y})	Places cursor at text coordinate horiz,vert.
%(*expr*^{X},*expr*^{Y})	Places cursor at graphic coordinate horiz,vert.
;	Suppress carriage return/line feed.
!	Automatic Carriage return after maximum characters entered. User doesn't have to press <ENTER>.
&*expr*,	Sets the maximum number of characters to be INPUT. Default is 255. Will not allow more than expr characters.
"*string*";	Optional user prompt will replace question mark. If a null string is used the question mark will be suppressed.
var	May be any variable type integer, single,double or string.

EXAMPLE *See examples on following pages…*

REMARK Differences in screen width may affect operation.

See LOCATE and PRINT for more information on cursor positioning. Also see INPUT#,LINEINPUT,LINEINPUT# and INKEY$ for others ways of getting input.

See "Keyboard input" in the technical section.

Important Note: String lengths MUST be one greater than maximum INPUT length since a CHR$(13) is temporarily added. Never define a string used in an INPUT or LINEINPUT as ONE.

In certain cases EDIT FIELD,MENU or BUTTON may be preferable. See appendix.

INPUT continued

EXAMPLES OF REGULAR INPUT

EXAMPLE	RESULT
INPUT A$	Wait for input from the keyboard and store the input in A$. Quotes, commas and control characters cannot be input. <ENTER> to finish. A carriage return is generated when input is finished (cursor moves to beginning of next line).
INPUT"NAME: ";A$	Prints "NAME: " before input. A semi-colon must follow the last quote. A carriage return is generated after input (cursor moves to next line).
INPUT;A$	Same as INPUT A$ above, only the semi-colon directly after INPUT disables the carriage return (cursor stays on the same line).

EXAMPLES OF LIMITING THE NUMBER OF CHARACTERS WITH INPUT

EXAMPLE	RESULT
INPUT &10,A$	Same as INPUT A$ only a maximum of ten characters may be input. (&10) A carriage return is generated after input (cursor moves to the beginning of the next line). The limit of input is set for ALL variables, not each.
INPUT ;&3,I%	Same as INPUT &10, except the SEMI-COLON following INPUT stops the carriage return (cursor stays on line).
INPUT !&10,A$	Same as INPUT & 10 except INPUT is terminated as soon as 10 characters are typed (or <ENTER> is pressed).
INPUT;!&10,"NAME: ",A$	Same as INPUT ;&10,A$ except no carriage return is generated (semi-colon). INPUT is terminated after 10 characters(&10 and Exclamation point). and the message "NAME: " is printed first.
LINEINPUT;!&5,"NAME: ";A$	LINEINPUT A$ until 5 characters or <ENTER> is pressed. (no carriage return after <ENTER> or after the 5 characters are input. Accepts commas and quotes.)

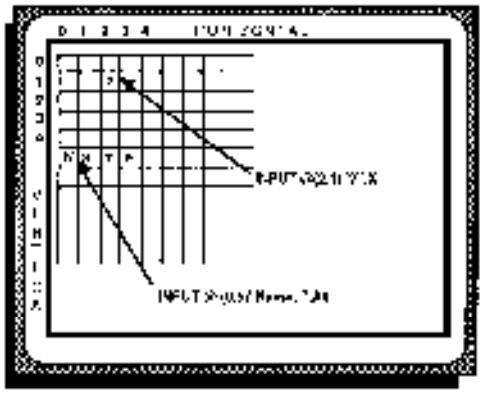
Note 1: Wherever INPUT is used, LINEINPUT may be substituted when commas, quotes or some other control characters need to be input (except with multiple variables).

Note 2: If more than one variable is INPUT, commas must be included from the user to separate input. If all the variables are not input, the value of those variables will be null.

INPUT statement

INPUT continued

INPUTTING FROM A SPECIFIC SCREEN LOCATION



INPUT@(H,V);A$Wait for input as TEXT screen POSITION defined by Horizontal and Vertical coordinates. No "?" is printed. A carriage return is generated.

INPUT%(gH,gV);A$ Input from a graphic coordinate. Syntax is the same as "@". Very useful for maintaining portability without having to worry about different screen widths or character spacing.

INPUT@(H,V);!10,"AMT: ";D# Prints "AMT: " at screen position H characters over by V characters down. D# is input until 10 characters, or <ENTER> are typed in, and the input is terminated without generating a carriage return (the cursor DOES NOT go to the beginning of the next line).

INPUT%(H,V);!10,"AMT: ";D# Prints "AMT: " at Graphic position H positions over by V positions down. D# is input until 10 characters, or <ENTER>, are typed in, and input is terminated without generating a carriage return (the cursor DOES NOT go to the beginning of the next line).

Note: Replace INPUT with LINEINPUT whenever there is a need to input quotes, commas and control characters (except with multiple variables).

FORMAT **INPUT** # *expression*,*var*[,*var*[,…]]

DEFINITION This statement will read INPUT from a disk or other device specified by expression until a carriage return, <COMMA>, End-Of-File or 255 characters are encountered.

Commas and leading spaces may be read into a string variable if the data on disk was enclosed in quotes, otherwise leading spaces and line feeds will be ignored.

See LINEINPUT# for ways of inputting commas, quotes and some control characters.

EXAMPLE

```
A$="HELLO"
B$="GOODBYE"
C$="WHAT?"
X#=12.345
:
OPEN"O",1"TEST.TXT":REM OPEN FOR OUTPUT
PRINT#1, A$","B$","C$","X#    <--- Quoted commas important with PRINT#
CLOSE#1
:
OPEN"I",1,"TEST.TXT":REM OPEN FOR INPUT
INPUT#1, X$,Y$,Z$,A#          <--- INPUT# in same order and type as PRINT#
END

RUN

HELLO          GOODBYEWHAT?          12.345
```

REMARK See OPEN,CLOSE,PRINT#, and LINEINPUT#.

See your computer appendix for available devices.

Compatibility Note: ZBasic and MSBASIC have almost the same syntax with the following exceptions:

MSBASIC ALLOWS	**ZBasic REQUIRES**
PRINT#n, A$,B$,X#,C%	PRINT#n, A$","B$","X#","C%
PRINT#n, A$ B$ C$	PRINT#n, A$","B$","C$

If you remember that ZBasic puts the image to the disk just as if it were going to the printer or to the screen you will see why the syntax is important.

INSTR function

FORMAT **INSTR**(*expression*, *string*$_1$,*string*$_2$)

DEFINITION Finds the first occurrence of string 2 in string 1, starting the search at the position specified by expression.

expression	Starting position of the search.
string1	String to be searched.
string2	String to search for.

EXAMPLE

```
Humble$="I am cool!"
PRINT INSTR(1,Humble$, "cool")
:
B$="am"
PRINT INSTR(1,Humble$, B$)
:
X=INSTR(1, Humble$, "FRED")
PRINT X
END
```

RUN

```
6               <---"Cool" started in the sixth position
3               <---"am" started at the third position
0               <---There was no "FRED" in the string.
```

```
Name$="Fred Smith"
Lastname$=RIGHT$(Name$,LEN(Name$)-INSTR(1,Name$, " "))
PRINT "Hello there Mr.";Lastname$
END
```

RUN

```
Hello there Mr. Smith
```

REMARK If the string is not found, zero (0) will be returned.

See LEFT$,RIGHT$,MID$ and INDEXF.

FORMAT **INT**(*expression*)

DEFINITION Truncates all digits to the right of the decimal point of expression.

EXAMPLE

```
DEFDBL A-Z
DEFTAB 8
PRINT"      X","ABS(X),"INT(X)","FRAC(X)","SGN(X)"
:
FOR X = -15.0 TO +15.0 STEP 3.75
  PRINT USING"-##.##";X,
  PRINT USING"-##.##";ABS(X),
  PRINT USING"-##.##";INT(X),
  PRINT USING"-##.##";FRAC(X),
  PRINT USING"-##.##";SGN(X)
NEXT X
END
```

RUN

```
X             ABS(X)        INT(X)        FRAC(X)       SGN(X)
-15.00        15.00         -15.00        .00           -1.00
-11.25        11.25         -11.00        -.25          -1.00
- 3.75        3.75           -3.00        -.75          -1.00
   .00        .00              .00        .00             .00
  3.75        3.75            3.00        .75            1.00
  7.50        7.50            7.00        .50            1.00
 11.25        11.25          11.00        .25            1.00
 15.00        15.00          15.00        .00            1.00
```

REMARK INT works the same as FIX in that expression will be restricted to the integer range of -32,768 to +32,767 only when the expression has not been defined as floating point.

INT is simply as a function that truncates an expression to a whole number.

To get the fractional part of a number use FRAC.

See FIX,SGN,ABS and FRAC.

INT range for the Macintosh is -2,147,483,648 to +2,147,483,647.

KILL statement

FORMAT **KILL** *simplestring*

DEFINITION KILL will erase a disk file specified by simplestring.

KILL functions either as a command or from within a program.

EXAMPLE

```
INPUT"File to erase:";A$
PRINT"Are you sure you want ";A$;" erased?";
INPUT B$
:
LONG IF B$<>"YES"
  PRINT"File not erased": STOP
XELSE
  KILL A$:PRINT A$;" is history."
END IF
:
END
```

RUN

```
File to erase: OldFile
Are you sure you want OldFile erased?
YES
Oldfile is history!
```

REMARK Use this statement with caution. When a file has been killed it is normally unrecoverable.

See RENAME,ERROR,ON ERROR,ERRMSG$ and the "Files" section of this manual for more information.

This page intentionally left blank.

LEFT$ function

FORMAT **LEFT$** (*string* , *expression*)

DEFINITION LEFT$ returns the left-most characters of string defined by expression. The string will not be altered.

EXAMPLE

```
Quote$="Early to Bed, Early to rise..."
:
PRINT LEFT$(Quote$, 5)
:
Part$= LEFT$(Quote$, 12)
PRINT Part$
:
PRINT LEFT$(Quote$, 50);
PRINT "Makes men healthy...at least"

RUN

Early
Early to Bed
Early to Bed, Early to rise... Makes men healthy...at least"
```

REMARK Also see RIGHT$,MID$,LEN,VAL,STR$,INSTR,INDEX$,SWAP and the "String Variable" section of this manual for more information about using strings.

FORMAT **LEN** (*string*)

DEFINITION Returns the number of characters that are stored in a string constant or string variable. If zero is returned it indicates a null (empty) string.

EXAMPLE

```
A$="FRED"
B$="SMITH"
:
PRINT A$;" has";LEN(A$);" characters."
PRINT B$;" has";LEN)B$);" characters."
:
PRINT LEN(A$)+LEN(B$)
:
PRINT LEN("Hello Fred")
```

RUN

```
FRED has 4 characters
SMITH has 5 characters
9
10
```

REMARK The maximum length of a string is 255 characters. You may set the length of strings in ZBasic. See DIM,DEF LEN and the chapter on "String Variables" for more information about defining string length.

Since the first character of a string stored in memory is the length byte, PEEK(VARPTR(var$)) will also return the length of a string.

The memory required for a string variable is the defined length + one for the length byte (256 bytes if not defined).

LET statement

FORMAT [**LET**] *variable = expression*

DEFINITION LET is an optional statement that may be used to assign an expression to a variable. Numbers, strings, numeric expressions, or other variables may be used to assign values to a variable if the types are compatible or convertable.

EXAMPLE

```
LET B=100
PRINT B
:
LET B=B+10
PRINT B
:
Z$="HELLO"+" THERE"   <---Notice "LET" is optional
PRINT Z$

RUN

100
110
HELLO THERE
```

REMARK See SWAP,"Optimize expressions for Integer", "Math Expressions" and "Conversions Between Variable Types" for more information about assignments.

FORMAT **LINE** *line number or label*

DEFINITION Returns the starting address of a compiled line in memory. Normally used with CALL to execute machine language subroutines created with MACHLG.

EXAMPLE

```
10 CALL LINE 30                          <--- Example only. DO NOT RUN!
20 END
30 MACHLG 23,323,11,232,A%, 2,1,0,0,1:RETURN
```

```
"Start"
PRINT"THIS IS A TEST ",1,2,3
"END"
A = LINE "END" - LINE "START"
PRINT  "The second line is ";A;" bytes long"
```

RUN

```
THIS IS A TEST              1             2             3
The second line is   36 bytes long
```

REMARK This statement is useful for calling machine language subroutines embedded in your program or for calculating the number of bytes used by program lines.

Also see MACHLG and CALL.

Macintosh: Use LongIntegers for addresses. See CALL in the appendix.

MSDOS: See CALL in appendix.

Apple ProDOS: See MLI in ProDOS appendix.

LINEINPUT statement

FORMAT **LINEINPUT**[(@ or %)(*expr1*,*expr2*)][;][!][&*expr*,]["*string*";]*var*$

DEFINITION The LINEINPUT statement is used to input characters from the keyboard into a string variable. It is different from INPUT in that quotes, commas and some control characters may also be entered. LINEINPUT is terminated when <ENTER> is pressed.

@(*expr1*,*expr2*) %(expr1,expr2)	Inputs from horizontal,vertical TEXT coordinate. Inputs from horizontal,vertical GRAPHIC coordinate.
;	Suppresses carriage-return/line-feed after input is complete. (disable inputs that cause scrolling or overwriting.)
!	Automatically executes a carriage return after the maximum number of characters are entered. The user doesn't have to press <ENTER>.
&*expr*,	Sets the maximum number of characters to be input.
"*string*";	Optional string prompt will replace the question mark "?" normally shown with LINEINPUT.
var$	Only string variables may be used with LINEINPUT.

EXAMPLE

```
INPUT"Last name <COMMA> First name";A$
PRINT A$
:
LINEINPUT"Last name <comma> First name";B$
PRINT B$
```

RUN

```
Smith
Smith, Fred
```

REMARK See the chapter on "Keyboard Input" in the front of this manual for more examples.

The advantage of using LINEINPUT over INPUT is its ability to receive most of the ASCII character set except:

<ENTER>	CARRIAGE RETURN
<CTRL C>	CONTROL "C"
<BACKSPACE>	DELETE or LEFT ARROW
<CANCEL>	DELETE CURRENT LINE
<NULL>	NO CHARACTER

Important Note: String lengths MUST be at least one greater than the number of characters being input, otherwise a string overflow condition will destroy subsequent variables. Never use a one character string with LINEINPUT.

FORMAT **LINEINPUT** # *expression* ,*variable*$

DEFINITION This statement will input ASCII or TEXT data from a disk file specified by expression until <ENTER>, End-Of-File or 255 characters are encountered.

Useful for accepting commas, quotes and other characters that INPUT# will not accept. A good example of using LINEINPUT would be for reading an ASCII or TEXT file a line at a time (as in the example below).

EXAMPLE

```
REM Read a text file and print it to the screen
REM Routine compatible with all versions of ZBasic
:
ON ERROR GOSUB 65535: REM Error trapping on to check for EOF
:
OPEN"I",1,"TEXT.TXT"
:
Counter=0
:
WHILE ERROR=0: REM Read file until an EOF error
  LINEINPUT#1, A$
  PRINT A$
WEND
IF ERROR <> 257 THEN PRINT ERRMSG$(ERROR): STOP
ERROR=0
:
ON ERROR RETURN: REM Give error trapping back to ZBasic
END
```

REMARK The advantage of using LINEINPUT# over INPUT# is its ability to receive most of the ASCII character set. Leading linefeeds will be ignored on some systems.

If a CHR$(0) or CHR$(26) is encountered as a leading character it may assume EOF and set ERROR = End Of File (varies by computer).

Also see INPUT#,LINEINPUT and "Keyboard Input" in the front section of the manual.

These versions support an EOF function that would simplify the error trapping techniques used above. See the appropriate appendix for details about EOF:

```
OPEN"I",1,"TEXT.TXT"
Counter=0
:
WHILE EOF=0: REM Read until EOF
  LINEINPUT#1, A$
  PRINT A$
WEND:CLOSE#1
```

LIST command

FORMATS

[L]L[IST] [+][*]
[L]L[IST] [+][*] *line or label*
[L]L[IST] [+][*] *- line or label*
[L]L[IST] [+][*] *line or label - line or label*

DEFINITION

LIST (or L) is used from the Standard Line Editor to list the current program to the screen. LLIST will list the current program to a printer.

+	Suppress line numbers
*	Highlight keywords on the screen (some versions)

EXAMPLE

YOU TYPE	**ZBASIC RESPONDS**
LIST or L	Lists complete program to the screen
LLIST	Lists complete program to the printer
LIST 100-200	Lists lines from 100-200
LLIST-100	Lists lines up to 100 to printer
LIST "SUBROUTINE"	Lists the line with that label
LIST 100- or L100-	Lists the lines from 100 on
<period>	Lists the last line listed or edited
<UP ARROW>	Lists previous line (or plus <+> key)*
<DOWN ARROW>	Lists next line (or minus <-> key)*
L+	Lists program without line numbers
LLIST+	Lists to printer without line numbers
L+-100	Lists up to line 100 without line numbers
<SPACE>	PAUSE. <ENTER> continues
</> (slash key)	PAGE AT A TIME: Lists 10 lines to the screen*

*See computer appendix for keyboard variations.

REMARK

LIST automatically indents program lines two spaces between FOR-NEXT,DO-UNTIL, WHILE-WEND,LONG IF-XELSE-END IF and LONG FN-END FN structures.

See PAGE, WIDTH,WIDTH LPRINT and the chapter; "Formatting Listings".

Note: Labels may be used in place of line numbers.

LLIST+* will format listings to an Imagewriter or Laserwriter with no line numbers and with keywords in bold. While the output in of this format is extremely attractive and easy to read, it should be noted that listings will take about twice as long to print.

FORMATS

LOAD ["] *filespec* ["]
LOAD * ["] *filespec* ["]

DEFINITION

LOAD is used from the Standard Line Editor to load a ZBasic tokenized or a regular ASCII text file into memory.

ZBasic does not load tokenized files from other languages; the file must first be saved in TEXT or ASCII format.

If the program does not have line numbers they are added in increments of one.

LOAD* will strip away remarks and unnecessary spaces from an ASCII file releasing more room for the source and object code in systems with limited memory.

EXAMPLE

```
LOAD    PROGRAM        <--- Loads a regular tokenized or text file
LOAD   "SOURCE"        <--- Double Quotes optional
LOAD*  THISONE         <--- Strips spaces and REM's while loading
```

REMARK

Each operating system may require specific syntax for a drivespec.

Line numbers are optional in ASCII files.

If a program was created using another form of BASIC it must be in ASCII format before the ZBasic editor can load it.

These version of ZBasic support a Full Screen Editor that may support other forms of LOAD. See appropriate appendix for information about full Screen Editors.

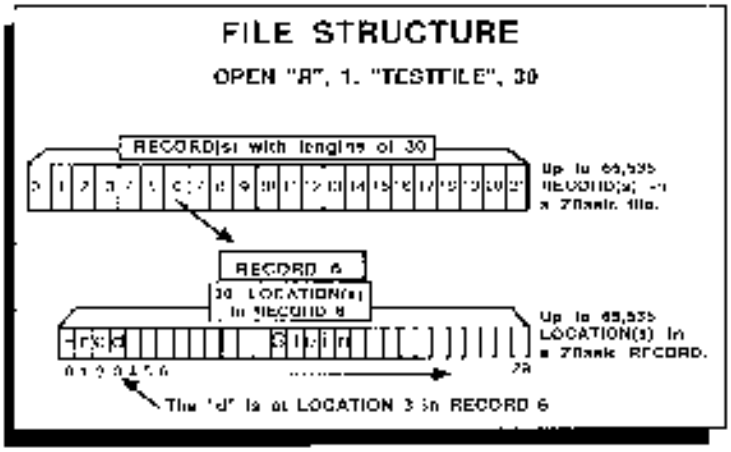
LOC function

FORMAT LOC (expression)

DEFINITION Returns the byte pointer position within the current RECORD of the filenumber specified by expression.

EXAMPLE

```
OPEN"R",1,"TESTFILE",30
RECORD#1, 6, 3        <---See illustration
PRINT LOC(1)
:
READ#1, Char$;1
PRINT LOC(1)
:
PRINT Char$
CLOSE#1

RUN

3
4
d
```



REMARK The LOC position is incremented to the next file position automatically when READ#,WRITE#,INPUT#,LINEINPUT# or PRINT# are used. REC(filenumber) returns the current RECORD. LOF returns the last record in the file. Also see "Files" section for more information.

The record length limits are different for these versions. See appendix.

FORMAT **LOCATE** *expr*$_X$,*expr*$_Y$,[*expr*$_{cursor}$]

DEFINITION Positions the cursor to the coordinates given by expry and optionally turns on or off the cursor character (zero=off, not zero=on).

expr$_X$	The horizontal coordinate (characters across)	
expr$_Y$	The vertical coordinate (lines down)	
expr$_{cursor}$	Zero=cursor OFF.	Non-zero = cursor ON

EXAMPLE

```
LOCATE 0,0          <---sets cursor in upper left corner
LOCATE 10,0         <---sets Cursor 10 char to right at top
LOCATE 0,10,0       <---sets Cursor 10th line down. Cursor OFF
LOCATE 0,12,1       <---sets Cursor 12th line down. Cursor ON
```

REMARK This function is also useful with CLS LINE and CLS PAGE for clearing the screen to the end of line and end of page.

See "Screen and Printer Control",PRINT@,PRINT%,INPUT@,LINEINPUT@, LINEINPUT% and INPUT% for other ways of controlling the cursor positioning.

The ability to turn the cursor on or off may be limited by the hardware or software of some computers.

These versions of ZBasic allow you swap the horizontal and vertical coordinates under "Configure". This is handy for converting other BASIC programs that use the vertical coordinate first (not Apple DOS 3.3).

LOF function

FORMAT **LOF** (*expression*)

DEFINITION Returns the last valid RECORD number for the file specified by expression. LOF stands for Last-Of-File.

Important Note: This function may not return the last record correctly on some systems, especially if the record length of the file is different from the operating system's internal record length or if a file is opened with a different record length then that which it was opened originally. This is often remedied by simply setting the record length to the system default record length or the record length of which it was opened originally.

EXAMPLE See "Opening files for Append" in the "Files" section in the front of this manual for methods of getting a pointer to the last position in a file.

REMARK LOF returns the last record in the file. The default record length is 256 and may need to be changed to make LOF function properly.

See LOC and REC for getting file pointer information. See "Files" and "Disk Errors" for more information. Some systems return one for both record zero and record one.

Note to better usage: If you need to keep track of the last byte position of a sequential file or the last record of a random file, you might consider storing the last REC and LOC of a file in record zero before it is closed. Examples:

```
OPEN"O",1,"Textfile.txt"
RECORD#1,1      <---Set file pointer to record one (zero will store last REC and LOC)
PRINT#1,A$","B$","X","Z# <---Save data
RECORD#1,0      <---- Position pointer to RECORD 0 to save last REC and LOC
R=REC(1):L=LOC(1)
WRITE#1, R,L    <---Save pointers for future use
CLOSE#1
```

To add data to the end of the file later:

```
OPEN"R",1,"Textfile.txt"
RECORD#1,0
READ#1, R, L    <--- Get last positions of file
RECORD#1, R,L   <---- Position pointer to append data to the end of the file.
PRINT#1, A$     <--- Now you can append new data to the file
```

Don't forget to store the LOC and REC before closing! You could do the same thing with random files by saving the last record.

Also supports: LOF(*filenumber*,[*recordlength*]). LOF(1,1) would return the length of filenumber one in bytes.

FORMAT **LOG** (*expression*)

DEFINITION Returns the natural logarithm of expression (LN). LOG is the compliment of EXP.

Common LOG10= LOG(n)\LOG(10)

EXAMPLE

```
PRINT LOG(2)
X#=LOG(3)
PRINT X#

RUN

.69314718056
1.09861228857
```

REMARK LOG is a scientific function. Scientific precision may be configured by the user differently from both single and double precision.

See "Configure" and "Math" in the beginning of this manual.

Also see COS,SIN,EXP,"^",ATN and TAN.

LONG FN statement

FORMAT **LONG FN** *name*[(*var*[,*var*[,…]])]
.
.
END FN[= *expression*]

DEFINITION LONG FN is similar to DEF FN but allows the function to span over several lines. This is useful for your own functions that you can use with ZBasic.

A re-usable, non-line-numbered function may be saved to the disk with SAVE+ and retrieved later for use in other programs with APPEND.

The variables being passed to the function must not be arrays. The *expression* must be numeric for numeric functions and string for string functions.

EXAMPLE

```
LONG FN RemoveSpace$(x$)
  WHILE ASC(RIGHT$(x$),1)=32
    x$=LEFT$(x$,LEN(x$)-1)
  WEND
END FN= x$
:
Name$="ANDY          "
:
PRINT Name$;"*"
:
Name$=FN RemoveSpace$(Name$)
PRINT Name$;"*"
```

RUN

```
ANDY          *
ANDY*
```

```
REM Wait until key press. Return key in key$
LONG FN WaitKey$(key$)
  DO
    key$=INKEY$
  UNTIL LEN(key$)
END FN=key$
:
Z$=FN WaitKey$(Z$)
PRINT Z$
```

RUN

(returns key that was pressed)

REMARK Also see APPEND,SAVE+,DEF FN,FN statement,FN function and "Structure".

FORMAT **LONG IF** *expression*
.
[XELSE]
.
ENDIF

DEFINITION LONG IF allows multiple line IF-THEN-ELSE structures. Very useful for breaking down complicated IF statements into more readable, logical structures. Two things happen based on the result of expression:

* If expression is TRUE: Executes all the statements up to the XELSE (if used) and then exits at the END IF.
* IF expression is FALSE: Executes all the statements between the XELSE and END IF and then exits at the END IF. If XELSE is not used it will simply exit at the END IF.

EXAMPLE

```
INPUT"How old are you: ";Age%
LONG IF Age% >=30
  PRINT "You are Old aren't you !?"
XELSE
  PRINT "You're just a baby!"
END IF
```

RUN

```
How old are you: 30
You are Old aren't you!?
```

```
LONG IF Name$="Fred"
  PRINT"Hello Fred...Long time no-see!"
  PRINT"The balance you owe is";USING"$####.##";Due#
  PRINT"Thanks for asking."
XELSE
  PRINT "I don't know you! Go away!"
END IF
```

RUN

```
Hello Fred...Long Time no-see!"
The balance you owe is $1234.56
Thanks for asking.
```

REMARK No loop may be executed within a LONG IF construct unless it is completely contained between a LONGIF and XELSE or between XELSE and ENDIF. The entire LONG IF construct must be completely contained within loops or nested loops in order to compile properly.

ZBasic will automatically indent program lines between LONG IF,XELSE and END IF two spaces. See the chapter about "Structure" for more information.

LPRINT statement

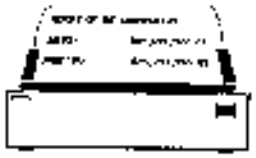
FORMAT **LPRINT** [*variables*, *constants*,...]

DEFINITION The LPRINT statement sends output to a printer.

To use LPRINT from the Standard Line Editor use a colon first (:LPRINT).

EXAMPLE

```
LPRINT "REPORT OF THE CORPORATION"
LPRINT
LPRINT
LPRINT "SALES:";TAB(50);USING"$##,###,###.##";Sales#(1)
LPRINT
LPRINT "PROFITS:";TAB(50);USING"$##,###,###.##";Profits#(1)

RUN
```



REMARK Some systems may lock up if a printer is not connected. See your hardware manual for required action.

See ROUTE 128,PRINT,LLIST,TAB,DEFTAB, PAGE, USING, WIDTH LPRINT and POS(1).

Macintosh: See DEF LPRINT,PRCANCEL,DEF PAGE,PRHANDLE, TEXT and ROUTE 128 in the appendix for more information about printing to the Imagewriter and Laserwriter printers. See appendix for specifics.

MSDOS: To use more than one printer you may also use OPEN"I",1,"LPT2:" and use PRINT#1,[variables,constants...]. Be sure to close the printer device when finished. See MSDOS reference manual for more information about LPT2:,LPT1: and any other devices you may have available for your hardware.

Apple ProDOS and DOS 3.3: See DEF LPRINT for setting the printer slot.

FORMAT **MACHLG**{[*bytes*,...]} -or- {[*words*,...]} -or-{[*variables*][,…]}

DEFINITION The MACHLG statement is used to insert bytes directly into a compiled program. These bytes may be machine language programs, variables or other items.

It may be used to insert machine language into memory without using POKE.

bytes	Numbers from 0 to 255
words	Numbers from 0 to 65535. They are stored in standard format
variables	Will create the address where the variable is located. See appendix for specifics. Note: ZBasic uses registers when calculating elements of an array variable. Contents of these registers may be destroyed.

EXAMPLE

```
X = LINE "Machine Language Routine"
FOR I = 0 TO 10
  PRINT PEEK(X+I);
NEXT I
END
:
"Machine Language Routine"
MACHLG 0,1,2,3,4,5,6,7,8,9,10
```

RUN

```
0 1 2 3 4 5 6 7 8 9 10
```

REMARK

See LINE,CALL,USR,DEFUSR,PEEK,POKE and the chapter about "Machine Language" in the technical section of this manual.

Important Note: Use of this statement requires knowledge of the machine language of the computer you are using. Machine language may not be portable to other computers.

Macintosh: Since the Macintosh is a 32 bit machine, MACHLG puts the code into word, not byte, positions.

MSDOS: See DEF SEG in appendix.

Apple ProDOS: See section entitled Machine Language Interface in appendix.

MAYBE function

FORMAT **MAYBE**

DEFINITION MAYBE is a random function that returns either a TRUE(-1) or FALSE(0) with equal probability.

MAYBE is faster than RND, convenient, and requires little program space.

EXAMPLE

```
DEFTAB = 8: DIM Coin$(1)
Coin$(0)="HEADS":Coin$(1)="TAILS"
:
"Flip a Coin"
DO
  X=X+1
  PRINT Coin$(MAYBE+1),
UNTIL X=25
END
```

RUN

```
HEADS   HEADS   TAILS   HEADS   TAILS
TAILS   TAILS   TAILS   HEADS   HEADS
TAILS   TAILS   HEADS   TAILS   TAILS
HEADS   HEADS   HEADS   HEADS   TAILS
HEADS   TAILS   TAILS   TAILS   HEADS
```

REMARK This function is useful anytime a 50% random factor is needed.

MAYBE with logical operators:

MAYBE	50% TRUE	50% FALSE
MAYBE AND MAYBE	25% TRUE	75% FALSE
MAYBE OR MAYBE	75% TRUE	25% FALSE

FORMAT **MEM**[ORY]

DEFINITION Typing either MEM or MEMORY in command mode will return information about system memory use.

TEXT — The number of bytes being used by the source code. The source code is that part of the program that you type in.

MEMORY — The number of bytes remaining for program use (varies; see your computer appendix for details).

OBJECT — The size of the object code after compiling. *Valid only immediately after RUN*.

VARIABLES — The number of bytes required for variables. INDEX$ array, and disk I/O buffers. This varies dramatically by version. See computer appendix. *Valid only immediately after RUN.*

EXAMPLE

```
MEM

00046  Text
41244  Memory
00039  Object
00388  Variable
(some versions may display more information)
```

REMARK These numbers are relative to that version of ZBasic being used. Varies significantly by computer.

See your computer appendix for more information.

Also see MEM function,CLEAR,CLEAR INDEX$,CLEAR END,LOAD* and the chapter about "Converting Old Programs".

MEM function

FORMAT **MEM**

DEFINITION Returns the number of bytes available in the INDEX$ array.

EXAMPLE

```
CLEAR 1000
PRINT MEM
A= MEM
INDEX$(0)  = STRING$(49,"*")
PRINT MEM

RUN

1000
950
```

REMARK See also INDEX$, MEM command, and CLEAR INDEX$. This function varies by version. See appendix for specifics.

MEM(index number) returns the memory available to that INDEX$ (there are ten available on the Macintosh).

MEM(-1): Returns the maximum amount of memory available for variables. Also forces unloading of all unlocked memory segments. Returns a LongInteger.

INDEX$ has many enhancements with this version. See appendix.

See appendix for various additions to the MEM function that return memory pointers to arrays, strings, BCD variables and more.

FORMATS

MERGE ["] *filespec* ["]
MERGE* ["] *filespec* ["]

DEFINITION

MERGE is used to overlay a line numbered TEXT/ASCII program from disk onto the current program text in memory. Program being merged must be in ASCII (saved with SAVE*).

Incoming txt with the same line number(s) as resident text will replace resident text.

The asterisk is used to strip spaces and REM's from the incoming program.

EXAMPLE

```
010 REM Program one
120 DO
130   I$=INKEY$
140 UNTIL LEN(I$)
SAVE* "PROG1"
NEW

10 REM Program two
20 PRINT "MAIN MENU"
30 PRINT
40 PRINT "1. Do Inventory"
50 PRINT "2. Print Inventory"
60 PRINT "3. Delete Inventory"
MERGE "PROG1"
LIST

00010 REM Program one          <---- Line from first program overwrote this line
00020 PRINT "MAIN MENU"
00030 PRINT
00040 PRINT "1. Do Inventory"
00050 PRINT "2. Print Inventory"
00060 PRINT "3. Delete Inventory"
00120 DO                       <---First program merged here
00130   I$=INKEY$
00140 UNTIL LEN(I$)
```

REMARK

MERGE has the same affect as manually typing in text.

Programs that were written in another BASIC must be in ASCII format before being MERGED into ZBasic.

Also see LOAD,SAVE*,RENUM,APPEND and DELETE

MID$ function

FORMAT **MID$** (*string* , *expr*1[,*expr*2])

DEFINITION Returns the contents of string starting at position expr1, and expr2 characters long.

string The string from which the copy will occur.

expr1 The distance from the left that the copy will begin.

expr2 Optional parameter that determines how many characters will be copied. If omitted, all characters from expr1 to the end of the string will be copied.

EXAMPLE

```
A$="The Sun Shines Bright"
:
PRINT MID$(A$,5,3)
:
Z$=MID$(A$,15)
PRINT Z$
:
FOR Pointer = 1 TO LEN(A$)
  PRINT MID$(A$,Pointer,1)
NEXT

RUN

Sun
Bright
T
h
e

S
u
n
.
.
.
```

```
INPUT"First and Last name please:";Name$
PRINT "Thank you Mr. ";MID$(Name$,INSTR(1,Name$," ")+1)

RUN

First and Last name please: Fred Smith
Thank you Mr. Smith
```

REMARK See LEFT$,RIGHT$,INSTR,LEN, STR$ and the MID$ statement.

FORMAT **MID$** (*string*1, *expr*1[,*expr*2]) = *string*2

DEFINITION Replace a portion of string1 starting at expr1, with expr2 characters of string2.

***string*1**	Target string. String2 will be inserted or layed over this string.
***string*2**	String to be inserted or layed over string 1.
***expr*1**	Distance from the left of string1 where overlay is to begin
***expr*2**	How many characters of string2 to insert into string1. Using 255 will assure that all characters are used.

EXAMPLE

```
A$  =  "SILLY BOY"
B$  =  "SMART"
:
MID$(A$,1,5) = B$
PRINT A$

RUN

SILLY BOY
SMART BOY
```

REMARK This function is very useful for altering selected portions of strings.

Also see RIGHT$,LEFT$,MID$ function,STR$,INSTR,VAL,LEN,SPACE$, STRING$.

MKB$ function

FORMAT **MKB$** (*expression*)

DEFINITION Returns a string which contains the compressed floating point value of a ZBasic BCD expression.

This function works with either single or double precision. The amount of string space used will vary depending on the digits of precision configured. See "Configure".

To return the floating point values stored in strings use the CVB function.

EXAMPLE

```
A$=MKB$(991721.645643)
PRINT "The length of A$=";LEN(A$)
X!=CVB(A$)
PRINT X!
:
PRINT
:
B$=MKB$(991721.645643)
PRINT "The length of B$=";LEN(B)
X#=CVB(B$)
PRINT X#

RUN

The length of A$=4     <--- Value returned depends on configured precision
991722

The length of B$=8     <--- Value returned depends on configured precision
991721.645643
```

REMARK Since ZBasic automatically compresses and decompresses BCD variables when using READ# and WRITE#, this function is of primary interest to those people that need to conserve memory for other reasons.

See also CVB,CVI,READ#,WRITE# and MKI$.

See your appendix for default accuracy and variations.

FORMAT **MKI$** (*expression*)

DEFINITION Returns a two character string which contains a two byte integer specified by expression.

To extract the integer stored in a string with MKI$, use the CVI function.

EXAMPLE

```
A$=MKI$(12345)
PRINT"Length of A$=";LEN(A$)
B%=CVI(A$)
PRINT B%
PRINT
:
A$=STR$(12345)
PRINT "Length of A$=";LEN(A$)
PRINT VAL(A$)
```

RUN

```
Length of A$=2
 12345                <--- MKI$ saves space...(4 bytes compared to below)

Length of A$=6
 12345                <--- Leading blank reserved for the "SIGN"
```

REMARK Used in older versions of BASIC to convert integers to strings for FIELD statements. ZBasic does this automatically when using READ# and WRITE#. Nevertheless, MKI$ and CVI are still useful for packing strings to save memory-- especially on systems with limited memory.

See also CVI,CVB,READ#, WRITE# and MKB$.

Use DEFSTR LONG to allow MKI$,CVI,HEX$,OCT$ and BIN$ to work with LongIntegers. Use DEFSTR WORD to set back to regular integer. Note that MKI$ returns a four byte string with LongIntegers.

MOD operator

FORMAT *expression*$_1$ **MOD** *expression*$_2$

DEFINITION MOD returns the remainder of an integer division with the sign of expression1.

EXAMPLE

```
PRINT "9 DIVIDED BY 2=";INT(9/2);"REMAINDER =";9 MOD 2
```

```
RUN
```

```
9 DIVIDED BY 2= 4 REMAINDER= 1
```

```
PRINT "-4 DIVIDED BY 2=";INT(-4/2);"REMAINDER=";-4 MOD 2
```

```
RUN
```

```
-4 DIVIDED BY 2= -2 REMAINDER= 0
```

REMARK MOD replaces the old BASIC routines for finding the remainder of a division and is also much faster:

OLD BASIC: X = (X - INT(X/N) * N)

ZBasic: X = X MOD N

FORMAT MODE expression

DEFINITION MODE is used to set the screen graphics or text format.

Most computers offer a number of different character and/or graphic modes. Use MODE to choose the mode most applicable to the program.

For most systems EVEN modes are character graphics and ODD modes are regular graphics. Not all machines have graphic capability. MODE for some popular microcomputers:

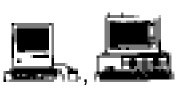
Mode number	MSDOS type		APPLE //e, //c		TRS-80 I, III	
	Text	Graphic	Text	Graphic	Text	Graphic
0	40x25	character	40x24	character	32x16	character
1	40x25	40x25	mono	40x48	64x16	128x48
2	80x25	character	80x24	character	32x16	character
3	80x25	80x25	mono	80x48	64x16	128x48
4	80x25	character	40x24	character	32x16	character
5	40x25	320x200	40x24	280x192	64x16	128x48
6	80x25	character	80x24	character	32x16	character
7	80x25	640x200	80x24	560x192	64x16	128x48
8	40x25	character	40x24	character	32x16	[illegible]
9	40x25	40x25	Bottom	40x48	64x16	128x48
10	80x25	character	40x24	character	32x16	character
11	80x25	80x25	Bottom	80x48	64x16	[illegible]
12	80x25	character	80x24	character	32x16	character
13	40x25	320x200	Bottom	280x160	64x16	128x48
14	80x25	character	80x24	character	32x16	character
15	80x25	640x200	Bottom	560x160	64x16	128x48

MACINTOSH	
Text	Graphic
[illegible]	SEE [illegible] APPENDIX

CP/M-80	
Text	Graphic
Normally 80x24	SEE [illegible] APPENDIX

Be sure to read the appropriate appendix for exact mode designations.

REMARK MODE will reset COLOR to the default, usually the darkest background and lightest foreground, and may clear the screen with some systems.



Macintosh: MODE is ignored with the Macintosh. See the TEXT statement for setting character styles and sizes. To emulate other computers you will probably want to use Monaco or Courier mono-spaced fonts. TEXT font, size, face, mode.

MSDOS: Modes 16-19 support EGA modes. Mode 20 supports Hercules graphics. See appendix for details.

MOUSE function

FORMAT **MOUSE** (*expression*)

DEFINITION Returns information concerning the position and status of a MOUSE or JOYSTICK if one is connected to the system. The following values are returned.

MOUSE(0)	Initializes the MOUSE on some systems (initialization is required on the Apple // ProDOS and DOS 3.3 versions).
MOUSE(1)	Returns the horizontal coordinate of the mouse.
MOUSE(2)	Returns the vertical coordinate of the mouse.
MOUSE(3)	Returns 0 if button not pressed. Non-zero if button pressed.

EXAMPLE

```
MODE 5 :REM GRAPHIC MODE
CLS
X= MOUSE  (0)                  <---Initialize mouse
:
WHILE LEN(INKEY$)=0            <--- Press any key to stop
 LONG IF MOUSE (3)             <--- If button down then ok to draw
   PLOT MOUSE (1), MOUSE (2) <--- Plot where mouse (or joystick) is.
  END IF
WEND
```

REMARK The above example uses a mouse to draw on the screen. A joystick may also be used (depending on the system). See your computer appendix for hardware device specifics that may apply to these functions.

Also see DEF MOUSE.

Macintosh Note: You may use the mouse functions above or configure ZBasic for MSBASIC Mouse compatibility using DEF MOUSE=1. See Mac Appendix.

MSDOS: Compatible with Microsoft Mouse. ZBasic has to be configured to support a mouse. See "Configure" in MSDOS appendix. If MOUSE(0) <> 0 then a mouse is installed. MOUSE(3) return 0-3; Zero if both buttons up, three if both buttons down, one or two if one button is pressed. MOUSE(4) and MOUSE(5) hide and show the mouse cursor. DEF MOUSE=0 for Mouse, 1 or 2 for joysticks, 3 for lightpens.

Apple ProDOS and DOS 3.3: Compatible with AppleMouse or joysticks. Use DEF MOUSE=0 for AppleMouse or DEF MOUSE=1 for Joysticks. If using a joystick MOUSE(3) returns 0-3. Zero if both buttons up, three if both buttons down, one or two if one button pressed. See appendix for specifics.

Z80: MOUSE IS NOT SUPPORTED with Z80 versions of ZBasic.

FORMAT **NAME** *string*1 **AS** *string*2

DEFINITION Renames a file with a filename of string1 to string2. Same as the RENAME statement except for syntax. This statement is provided to make ZBasic compatible with other BASIC languages.

EXAMPLE

```
DIR

FRED.BAS                TOM.BAS
DICK.BAS                HARRY.BAS

NAME FRED.BAS AS GEORGE.BAS

DIR

GEORGE.BAS              TOM.BAS
DICK.BAS                HARRY.BAS
```

REMARK See RENAME for more information.

Not available on Apple // or Z80 versions of ZBasic. See RENAME.

NEW command

FORMAT **NEW**

DEFINITION NEW is used to clear the text buffer of the current program.

Since programs that have been erased in this manner are impossible to recover, SAVE your program first!

EXAMPLE **LIST+**

```
CLS
PRINT"THIS IS A PROGRAM  ';
PRINT"WHICH IS ABOUT TO BE LOST FOREVER AND EVER..."
END
```

```
NEW
LIST
```

(Nothing listed...)

REMARK Use this command with care. See LOAD.

FORMAT FOR var = *expression*1 TO *expression*2 [STEP *expression*3]
.
.
NEXT [*variable* ,[*variable* …]]

DEFINITION The NEXT statement is used as the end marker of a FOR loop. There must be a matching NEXT for every FOR, otherwise a Structure Error will occur at compile time.

EXAMPLE

```
FOR Count1= 1 TO 2
  FOR Count2 = 2 TO 4 STEP 2
    PRINT Count1, Count2
NEXT Count2, Count1
```

```
RUN

1             2
1             4
2             2
2             4
```

```
FOR X= 1 TO 2
  FOR Y= 1 TO 2
    PRINT X,Y
  NEXT
NEXT
```

```
RUN

1             1
1             2
2             1
2             2
```

REMARK The variable(s) following the NEXT statement are optional; however, if used they must match the corresponding FOR variable(s).

A FOR-NEXT loop will execute AT LEAST ONCE!

A Structure Error will specify the line number if there is an extra NEXT, or will specify line 65535 if a NEXT is missing. ZBasic automatically indents all loop structures when you LIST your program. This may be used to find where the missing NEXT is located by simply following the program listing back to the point where the extra indent ends.

See "Loops" in the front of this manual and; WHILE-WEND, DO-UNTIL, LONGIF-XELSE-ENDIF for other loop and structure types.

NOT operator

FORMAT **NOT** *expression*

DEFINITION NOT returns the opposite of expression. True is False, False if TRUE. This is equivalent to changing a logical true (-1) to a logical false(0) and vice versa.

With Boolean (binary) operations, the NOT function will toggle all bits in expression. That is, all bits that are one will be changed to zero, and all bits that are zero will be changed to one.

EXAMPLE

```
A$="Hello"
IF NOT A$="Bye" THEN PRINT"True, it is False"
END
```

RUN

```
True, it is False
```

REMARK A logical true is -1 and logical false is 0. Also see XOR,OR,AND.

NOT condition TRUE(-1) if condition FALSE, else FALSE(0) if TRUE

NOT	**BOOLEAN "16 BIT" LOGIC**
NOT 1 = 0	NOT 11001100 NOT 01111011
NOT 0 = 1	= 00110011 = 10000100

Will also function with 32 bit LongIntegers.

FORMAT **OCT$** (*expression*)

DEFINITION OCT$ returns a 6 character string which represents the Octal value (base 8) of the result of expression truncated to an integer. Octal digits are from 0-7.

OCTAL	DECIMAL equivalent
0-7	0-7
10	8
11	9
12	10
13	11
14	12
15	13
16	14
17	15
20	16

EXAMPLE The following program will convert a decimal number to Octal or an Octal number to decimal:

```
CLS
DO
  INPUT"Decimal number: ";Decimal%
  PRINT "Octal Equivalent: ";OCT$(Decimal%)
  :
  INPUT"Octal number: ";Octal$
  Octal$="&O"+Octal$
  PRINT"Decimal Equivalent: ";VAL(Octal$)
UNTIL (DECIMAL%=0) OR (LEN(Octal$)=2)
```

```
RUN

Decimal number: 8
Octal Equivalent: 000010

Octal number: 100
Decimal Equivalent: 80
```

REMARK Conversions are possible from any base to any other base that ZBasic supports.

See the Chapter "Numeric Conversions" in the front of this manual. See also BIN$, HEX$ and UNS$.

Use DEFSTR LONG if you want to use OCT$,HEX$,BIN$,UNS$,MKI$or CVI with LongIntegers. Use DEFSTR WORD to set back to regular integer.

ON ERROR statement

FORMAT

ON ERROR GOSUB *Line or label*
ON ERROR *Return*
ON ERROR GOSUB *65535*

DEFINITION The ON ERROR allows the user to enable and disable disk error trapping. If ON ERROR is not used ZBasic will display disk errors as they occur and give the user the option of continuing or stopping. Options offered with ON ERROR:

ON ERROR GOSUB *65535*	Enable user disk error trapping. Errors are returned using the ERROR function. You must check for errors---ZBasic will not when this parameter is set.
ON ERROR GOSUB *line*	If a disk error occurs the program does a GOSUB to the line or label specified.
ON ERROR RETURN	Disable user disk error trapping. ZBasic will trap the disk errors and give error messages at runtime.

EXAMPLE

```
ON ERROR GOSUB 65535: REM Enable disk error trapping
"Start"
OPEN "I" ,1, "TEST"
IF ERROR GOSUB"Disk error"
GOTO "Start"
program continues...
:
:
"Disk error"
LONG IF (ERROR AND 255)=3: REM Check for File not found error
  PRINT"Check that correct diskette is in drive: <ENTER>";
  DO
  UNTIL LEN(INKEY$)
  ERROR=0:RETURN
XELSE
  PRINT"A Disk Error has occured:";ERRMSG$(ERROR)
  PRINT"<C>ontinue or <S>top?";
  DO
    temp$=UCASE$(INKEY$)
  UNTIL (temp$="C") OR (temp$="S")
  IF temp$="C" THEN ERROR=0: RETURN
END IF
PRINT"Program aborted!"
ERROR=0
STOP
```

REMARK Also see ERROR and ERRMSG$ and the chapter about "Disk Error Trapping" in the "Files" section of the manual.

See RETURN line for another way of returning from ON ERROR GOSUB line.

Important Note: Always remember to set ERROR=0 after a disk error occurs when you are doing the disk error trapping. Failure to do this will cause ZBasic to continue to return a disk error condition.

FORMAT **ON** *expression* **GOSUB** *line* [, *line*[,*line*…]]

DEFINITION The ON GOSUB statement is used to call one of several subroutines depending on the value of expression.

The ON statement will call the first subroutine if the expression evaluates to one, to the third subroutine if the expression evaluates to three and so on.

The RETURN statement at the end of a subroutine will return the program to the statement immediately following the ON GOSUB.

EXAMPLE

```
"Inventory Menu"
CLS
PRINT "1. Inventory"
PRINT "2. Print Listing"
PRINT "3. Month End"
PRINT "4. EXIT
PRINT
PRINT "Enter item wanted: ";
:
DO
  Item%=VAL(INKEY$)
UNTIL (Item% >0) AND (Item% <5)
:
ON Item% GOSUB "Inventory","Print","EOM","Exit"
GOTO "Inventory Menu"
END
:
"Inventory"
RETURN
:
"Print"
RETURN
:
"EOM"
RETURN
:
"Exit"
END
```

REMARK ZBasic will truncate expression to an integer. For example, if expression equaled 1.9, the ON statement would go to the first line (INT(1.9)=1).

If expression <=0 or > (number of line numbers listed), the program will continue on to the next statement in the program.

ON GOTO statement

FORMAT **ON** *expression* **GOTO** *line* [,*line*[,*line*…]]

DEFINITION The ON GOTO statement is used to branch, or jump, to one of several portions of a program depending on the value of expression.

The ON statement will jump to the first subroutine if the expression evaluates to one, to the third subroutine if the expression evaluates to three, and so on.

EXAMPLE

```
A=RND(4)
ON A GOTO "ONE", "TWO", "THREE", "Last"
END
:
"ONE"
PRINT 1
END
:
"TWO"
PRINT 2
END
:
"THREE"
PRINT 3
END
:
"Last"
PRINT 4
END

RUN

4
```

REMARK ZBasic will truncate expression to an integer. For example, if expression equaled 1.9, the ON statement would go to the first routine (INT(1.9)=1).

If expression <=0 or > (number of line numbers listed), the program will continue on to the next statement in the program.

See "Structure".

FORMAT

OPEN "I", [#] *filenumber*, *filename* [,*record length*]
OPEN "O", [#] *filenumber*, *filename* [,*record length*]
OPEN "R", [#] *filenumber*, *filename* [,*record length*]

DEFINITION

The OPEN statement is used to access a data file. Once a file is opened, information may be read from or written to the file depending on the way the file was opened. The first argument determines access:

"R"	Read/write file: Open file if it exists, create the file if it doesn't.
"I"	Read only file: Open file for input. If file doesn't exist, a disk error occurs (file not found error).
"O"	Write only file: Open file for output. Overwrites the old file.
filenumber	The number you assign to a file which is subsequently used with file commands like READ#,WRITE#,INPUT#,LINEINPUT#, PRINT#,REC,LOC and LOF.
filename	The filename as it appears in a directory. See your DOS manual and the appendix in this manual for information about drive specifiers, pathnames, sub-directories or whatever syntax is used for that computer.
record length	Optional record length to be used with that file (default is 256).

EXAMPLE

```
REM Open a file for READ and WRITE
OPEN "R",1,"INVEN", 180
:
REM Open a file for Input only
OPEN "I", File%, D$+"INVEN", 180
:
REM Open a file for Output only
OPEN "O",2, Filename$
```

REMARK

To configure ZBasic to have more than two files open at a time; see "Configure". Each file buffer will require between 160 and 1024 bytes of memory depending on the Disk Operating System and your version of ZBasic. No more than 99 files may be open at one time.

See your computer appendix for more information about file types, changing directories and more. Also see INPUT#,PRINT#,READ#,WRITE#,LOC and REC.

TO INSURE DATA INTEGRITY, ALWAYS CLOSE OPEN FILES BEFORE EXITING YOUR PROGRAM.

continued...

OPEN statement

OPEN continued

 **Macintosh**: Extra parameters included:

volume%	The number you get from FILES$ that sets the folder or root location of the file. Much easier than pathname specifiers. See appendix for details. Also see FILE$, EJECT,EOF,LOF,"File size",APPEND and pathnames. Example of volume number: `OPEN"type", fnum, "filename", 200, volume%`
Additional types	"R[R]","O[R]","I[R]","A[R]" and "R[D]","O[D]","I[D]","A[D]" The optional "R" or "D" after the file type specifies opening the resource fork (R) or data fork (D). The data fork is the default. See appendix for specifics. The "A" type opens a file for append. Also see APPEND for positioning the file pointer to the end.
Pathnames	Pathnames are supported like: Root:Folder:Fred

 **MSDOS**: The are may ways to specify, create or remove directories and sub-directories. See PATH$, CHDIR,MKDIR and RMDIR in the appendix.

 **Apple ProDOS**: See PATH. Filenames may contain pathname information like: PROFILE/ZBASIC/SOURCE. See appendix for details.

Apple DOS 3.3 uses CP/M type drivespecs like: A: instead of D1, B: instead of D2, etc. Filetype is specified by a leading exclamation mark and a number:

OPEN"-",filenumber, "[[!**type**][**drivespec**] filename", record length

!type=	1= Text file	5= S type file
	2= Integer BASIC	6= Relocatable file type
	3= Applesoft BASIC	7= A type file
	4= Binary file	8= B type file

Example: `OPEN"-", fnum, "!4 A:FRED", 200`

 **CP/M-80**: You may use a drive specifier in the filename:
`OPEN"-",n,"A:Fred.DAT", 200`

TRS-80: You may use a drive specifier in the filename:
`OPEN"-",n,"Fred/DAT.password:1",200`

statement OPEN "C"

FORMAT **OPEN "C"**,-1 or -2[,[*baud rate*][,[*parity*][,[*stopbit*][,*word length*]]]]

DEFINITION This statement is used to set serial communication port parameters. If any of the parameters are omitted the default will be used.

-1	Serial port one	
-2	Serial port two	
baud rate	110, 150, 300(default), 600, 1200, 2400, 4800, 9600	
parity	0 = none<-- default	
	1 = odd	
	2 = even	
stopbit	0 = one <-- default	
	1 = two	
word length	0 = 7 bits	
	1 = 8 bits	<-- default

EXAMPLE

```
REM A Very Cheap Terminal Program
OPEN"C",-1, 300                      <---Change parameters as needed
DO
  READ#-1, A$;0                      <---(;0) Won't "Hang" if nothing at port
 IF LEN(A$) THEN PRINT A$;
  :
  A$=INKEY$
  IF LEN(A$) THEN PRINT#-1,A$;
UNTIL A$="]"                         <--- Set a key to stop
```

REMARK Serial ports may be accessed using the same statements used in disk I/O: PRINT# INPUT#,LINE INPUT#,READ#, and WRITE#. In all of these statements, the port is not read or written to until the status indicates that the port is ready.

The one exception to the paragraph above is when READ# is used to read a string of zero length. In this case, the character will be returned if ready, otherwise a null string will be returned (similar to the INKEY$ function) (Not supported with CP/M).

A port does not have to be opened in order to be accessed. The OPEN "C" statement is used only to set the current port parameter values. Without this statement, the port will simply use the parameters to which it was last set.

All versions have a number of machine specific parameters. See appendix for important details.

continued...

OPEN "C" statement

OPEN "C" continued

The following are examples of sending or receiving files over a modem or serial line. Check appendix and hardware manuals for specifications.

Add your own line numbers, and modify programs as needed. Save with SAVE+ to use later.

SEND FILES TO ANOTHER COMPUTER

```
"SEND FILES"
LINEINPUT"File to send: ";File$
IF LEN(File$)=0 THEN STOP: REM No file? STOP
:
OPEN"I",1,File$
ON ERROR GOSUB 65535: REM Catch errors
:
OPEN"C",-1,300: REM Change parameters as needed
:
DO
  LINEINPUT#1, Line$
  IF LEN(Line$) THEN PRINT#-1, Line$
  DO                  <---- This DO loop is an example of "Handshaking" remove
    READ#-1,A$;0            this loop, and the PRINT# below, if not needed.
  UNTIL ASC(A$)=1
UNTIL ERROR
:
IF ERROR=0
CLOSE#1
PRINT#-1,"*END*": REM Tell receiver "All Done!"
RETURN
```

RECEIVE FILES FROM ANOTHER COMPUTER

```
"RECEIVE FILES"
LINEINPUT"Filename to Receive: ";File$
IF LEN(File$)=0 THEN STOP: REM No File? STOP
:
OPEN"O",1,File$
:
OPEN"C",-1,300: REM Change parameters as needed
:
DO
  LINEINPUT#-1, Line$
  IF Line$<>"*END*" THEN PRINT #1, Line$
  PRINT#-1, CHR$(1);  <--- Goes with "Handshaking" DO Loop above.
UNTIL (Line$="*END*")
:
CLOSE#1
RETURN
```

FORMAT *expression* **OR** *expression*

DEFINITION Performs a logical OR on the two expressions for IF THEN testing and BINARY operations. If either or both conditions are true the statement is true. See truth table below.

In binary/boolean operations if either bit is one than a one is returned.

EXAMPLE

```
A$="HELLO"
IF A$="GOODBYE" OR A$="HELLO" THEN PRINT"YES"
```

RUN

YES

REMARK Truth table for the OR function.

condition OR condition TRUE(-1) if either or both is TRUE, else FALSE(0)

OR

```
1 OR 1 = 1
0 OR 1 = 1
1 OR 0 = 1
0 OR 0 = 0
```

BOOLEAN "16 BIT" LOGIC

```
      00000001          10000101
OR    00001111    OR    10000111
=     00001111    =     10000111
```

Also see AND,XOR and NOT.

Functions with 32 bit LongInteger as well.

OUT statement

FORMAT **OUT** *port*,*data*

DEFINITION The OUT statement sends data to the specified port number.

EXAMPLE

```
OUT 1,12
:
A=6:B=9
OUT A,B
:
OUT A/2,B/3
END
```

REMARK This statement is microprocessor dependent and works only with Z80 and 8086 type processors.

Also see INP for a way of reading data in from the port.

Not supported with these versions.

FORMAT **PAGE**

DEFINITION Returns the current line position of the printer. The first line is line zero.

EXAMPLE

```
PAGE                    <---Also see PAGE statement
PRINT PAGE
LPRINT
LPRINT
LPRINT
PRINT PAGE

RUN

0
3
```

REMARK This function is similar to POS except the line position is returned instead of the character position.

Important Note: If your operating system uses forms control and checks lines per page, you must disable the operating systems forms control or ZBasic's PAGE.

See CSRLN in the MSDOS appendix for getting the line position of the screen cursor.

PAGE function

FORMATS **PAGE** [[*expression*1][,[*expression*2][,[*expression*3]]]]

DEFINITION **PAGE** is used to format output to the printer and to control the number of actual lines per page, printed lines per page and top margin. Following is a description of the parameters:

PAGE	Without parameters will send a page feed to the printer. this forces the print head to move to the defined position of the top of the next page.
*expression*1	The number of printed lines per PAGE
*expression*2	The number of actual lines per PAGE. Also resets the count to zero (normally 66 lines per page).
*expression*3	Lines for the top margin. This number is a subset of expression1. If the line count is zero, this many linefeeds will be output immediately.

EXAMPLE

```
PAGE 60,66,3          <---   Sets Listings to 60 lines per page
                             with 3 lines as top margin. Skips perforations nicely.
```

REMARK WIDTH LPRINT should be set to your printer's character width for proper PAGE operation when doing LLIST.

See PAGE function.

To disable PAGE use PAGE 0

Important Note: If your operating systems uses forms control and checks lines per page, you must disable the operating systems forms control or ZBasic's PAGE.

PATH

FORMATS **PATH**

DEFINITION PATH or PATH type commands are available on many versions of ZBasic that support multi-level directories. Rather than give the exact syntax for each machine this page gives a general overview. See your computer appendix for specifics.

MSDOS

See PATH$ function in the appendix. This allows you to get the current path name so that you can return to that sub-directory. Syntax is PATH$(drive number). Note: Drive A=1, B=2, ...

Pathname syntax example: C:\ZBasic\TEMP

Apple ProDOS

See PATH command in the appendix. Also see the example function on the master disk called: PREFIX.SAMPLE for ways of getting ProDOS pathnames during runtime.

Pathname syntax example: /PROFILE/ZBASIC/OBJECT

Pathnames not supported with DOS 3.3 version.

Macintosh

The most appropriate way of specifying where a file is located is using the volume number. This is recommended in "Inside Macintosh". Volume numbers are obtained easily using the FILES$ function. See Macintosh appendix.

Nevertheless, pathnames are supported and may be used.

Pathname syntax example: Fred:Tom:Harry

Z80

Pathnames are not supported since the operating systems for this CPU do not currently implement sub directories.

EXAMPLE See your appendix for examples.

REMARK This command varies significantly by computer type.

See DIR,OPEN and also be sure to see your appendix for specifics.

Pathnames are not supported with Apple DOS 3.3 or Z80 versions of ZBasic.

PEEK function

FORMAT **PEEK [WORD]** (*expression*)
PEEK LONG (*expression*)*

DEFINITION Returns the contents of the memory location(s) specified by expression:

PEEK	Returns a one byte number (0-255)
PEEK WORD	Returns a two byte number (-32768 to 32767)
PEEK LONG*	Returns a four byte number (*32 bit versions)

EXAMPLE

```
X=VARPTR(A$)            <---Get a safe place in memory to play with
:
POKE X, 10
POKE WORD X+1, 12000
:
PRINT PEEK(X)
PRINT PEEK WORD(X+1)

RUN

10
12000
```

REMARK See POKE,POKE WORD and POKE LONG,USR,MACHLG,CALL,LINE,HEX$, OCT$,UNS$ and the section in the front of this manual; "Machine Language".

Important Note: This function is for people experienced with machine language and the hardware of their computer.

*Macintosh: Always use LongIntegers for expressions to pass an address or to retrieve a four byte LongInteger. See appendix.

MSDOS: An extra parameter is available to determine the segment of the variable: PEEK[WORD] (address,segment). Also see MEM and DEF SEG in the appendix.

FORMAT

PLOT *expr*1,*expr*2 [**TO** *expr*3,*expr*4…]
PLOT [TO] *expr*1,*expr*2 [**TO** *expr*3,*expr*4…]

DEFINITION The PLOT statement is used to draw either one graphic point, or a line between two or more points, in the current COLOR. Examples:

```
PLOT 10,12                    <-- PLOT one point at position 10,12
PLOT 10,12 TO 100,100         <-- PLOT a line from 10,12 to 100,100
PLOT 10,12 TO 10,90 TO 1,1    <-- PLOT two lines: 10,12 to 10,90, to 1,1
PLOT TO 10,12                 <-- PLOT a line from last position to 10,12
```

EXAMPLE

```
CLS
MODE 5                <---Set graphics mode
PLOT 209, 304         <--- Plots one pixel
:
COLOR -1              <--- Sets COLOR to foreground
REM PLOT and angle
PLOT 209,304 TO 987, 643 TO 322,742
END
```

RUN

See illustrations on the following page.

REMARK As with all other ZBasic graphic commands, Device Independent Graphic coordinates of 1024 by 768 are the default. Expressions are truncated to an integer. Character type graphics will be substituted on computers, or modes, without graphic capabilities.

Also see CIRCLE,BOX,FILL,POINT,COLOR.

Macintosh: Use COORDINATE WINDOW to set the pixel graphics. Use COORDINATE to set your own relative coordinates or to set back to 1024x768. The upper left-hand corner of a WINDOW is coordinate 0,0.

MSDOS: Use COORDINATE WINDOW to set pixel coordinates. See COORDINATE to set relative coordinates or to set back to ZBasic coordinates.

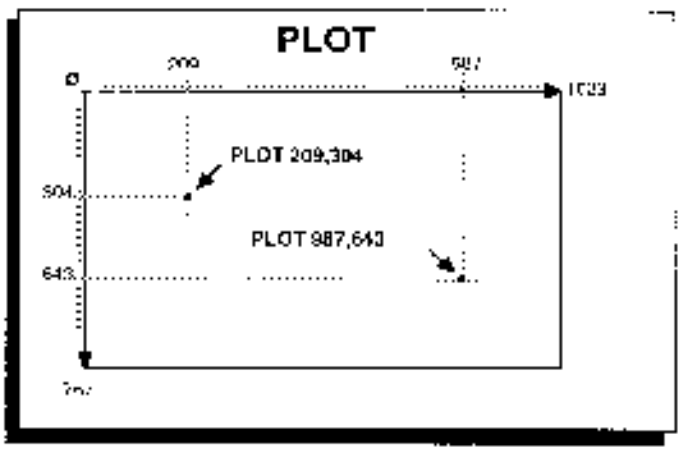
Z80: POKE $xx3F, &C9 for pixel coordinates. POKE $523F, &C3 to set back to ZBasic coordinates. xx= CP/M=01, TRS-80 model 1,3=52, TRS-80 model 4=30.

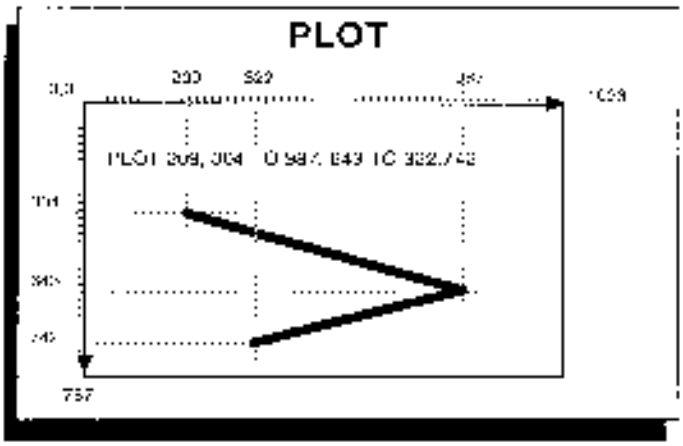
Apple // ProDOS: POKEWORD &85, 0 for pixel coordinates. Use MODE to set back to ZBasic coordinates.

Apple // DOS 3.3: POKE &F388,&60 for pixel coordinates. POKE &F388, &A9 to set back to ZBasic coordinates.

PLOT statement

PLOT continued

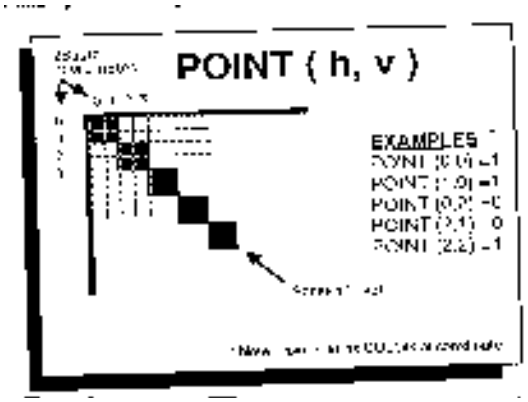




FORMAT **POINT** (*expression*$_1$, *expression*$_2$)

DEFINITION Point is available on many computers to inquire about the COLOR of a specific screen graphic position. As with other commands, ZBasic Device Independent Graphic coordinates may overlap pixels.

In the example: 0=Background (white here), 1 =Forground (black here)



As with all other ZBasic graphic commands, the device independent coordinate system of 1024 X 768 is the default.

EXAMPLE

```
COLOR 1
PLOT 0,0 to 900,767
PRINT POINT(0,0)
```

RUN

```
1
```

REMARK If the coordinate is outside screen coordinates, a -1 will be returned.

See COLOR,BOX,CIRCLE and the section; "Graphics".

See COORDINATE or PLOT for ways of converting some versions of ZBasic to pixel coordinates that can used with POINT.

POINT is not available for CP/M versions (including Kaypro graphic versions).

POKE statement

FORMAT **POKE [WORD]** *expression*%,*expression2*
POKE LONG *expression*&,*expression2*&*

DEFINITION POKE writes the value of expression2 into a memory location. The first expression is the address to POKE. The expression2 is the data to POKE.

TYPE	expression2
POKE	One byte
POKE WORD	Two bytes
POKE LONG*	Four bytes (*32 bit machines only)

EXAMPLE

```
X = 12345: XA = VARPTR(X)
PRINT"Byte at ";UNS$(XA);" =";PEEK(XA)
:
POKE XA,99
PRINT"Byte at ";UNS$(XA);" =";PEEK(XA)
:
POKE WORD XA,44444
PRINT"WORD at ";UNS$(XA);" =";UNS$(PEEK WORD(XA))
END
```

```
RUN

Byte at 59009 = 57
Byte at 59009 = 99
Byte at 59009 = 44444
```

REMARK Also see PEEK,PEEK WORD,PEEK LONG,MACHLG,CALL,LINE and the chapter "Machine Language" at the beginning of this manual.

Important Note: Indiscriminate use of this command may cause unpredictable computer operation and loss of data or program. This statement is for experienced machine language programmers only. Porting of programs with POKE is not recommended.

***Macintosh:** Always use LongIntegers for addresses and when using POKE LONG or PEEK LONG.

MSDOS: There is an optional parameter for segment:
POKE[WORD] *address*, *data*, ***segment***. See MEM and DEF SEG in the appendix.

FORMAT **POS** (*byte expression*)

DEFINITION Returns the current horizontal cursor position, from zero to 255, for a screen printer or disk file.

The expression specifies a device as follows:

POS(0) Default device (normally the video monitor)
POS(1) Printer
POS(2) Disk file (limited to one file using carriage returns)

EXAMPLE

```
CLS
PRINT "READ and DISPLAY SCREEN POS"
FOR I = 0 TO 30 STEP 10
  PRINT TAB(I); POS(0)
NEXT
:
PRINT "READ and DISPLAY PRINT POS"
DEFTAB 5
FOR I = 0 TO 6
  LPRINT,
  PRINT POS(1)M
NEXT
END
```

RUN

```
READ and DISPLAY SCREEN POS
0             10            20            30

READ and DISPLAY PRINTER POS
6      12     18     24     30     36
```

REMARK A carriage return will set the POS value to zero. PAGE will return the current line position for the printer.

Also see WIDTH,PAGE and WIDTH LPRINT.

While this command will work the same on all systems, it is dependent on screen and printer widths.

PRINT# statement

FORMAT **PRINT** # *expression*,*list of things to print*.....

DEFINITION Used to PRINT information to a disk file or other device in text format. Numbers or strings will appear in the file or device similar to how they would look on the screen or printer.

The expression is the file number assigned to a disk file or other device in an OPEN statement.

INPUT# or LINEINPUT# are normally used to read back data created with PRINT# (although READ# may also be used).

EXAMPLE

```
A$="TEST":B$="TEST2":C=900
:
OPEN "0" ,1, "TEST.DAT"
PRINT#1,"HELLO"","A$","B$","C <--- Quoted comma delimiters for INPUT#
CLOSE#1
:
OPEN"I",1,"TEST.DAT"
INPUT#1, X$, Y$, Z$, A%       <--- INPUT in same order and same type
:
PRINT X$, Y$, Z$, A%
:
CLOSE#1
END
```

RUN

```
HELLO          TEST           TEST2          900
```

REMARK While this command will work the same on all systems, it is dependent on disk input/output capabilities. Use INPUT# or LINEINPUT# to read back data written with PRINT#.

Be sure to see the entry on INPUT# in this reference section for more information about using PRINT# and INPUT# together and also information about MSBASIC syntax differences.

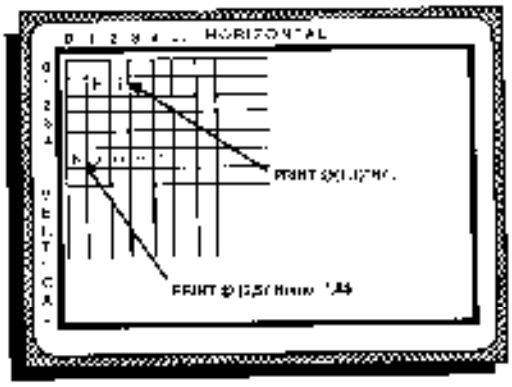
See ROUTE, OPEN, OPEN"C", INPUT#, LINEINPUT#, READ#, WRITE#, LPRINT and the section in the front of this manual called "Files" for more information.

FORMAT **PRINT** [{@|%} (*expr*1, *expr*2)] [*list of things to print*….]

DEFINITION The PRINT statement is used to output information to the current device, normally the video.

@ (*expr1*,*expr*2)	Specifies text coordinates.
% (*expr*1,*expr*2)	Specifies graphic coordinates.
	Note: Expr1=Horizontal. Expr2=Vertical.

EXAMPLE



```
PRINT@(1,1)"Hi";
PRINT@(0,5)"Name:";A$
END
```

REMARK PRINT followed with a semi-colon will disable the carriage return.

A PRINT item followed by a comma will cause the next element to be printed at the next tab stop defined by DEF TAB.

While this command will work the same on all systems, it is dependent on hardware.

See ROUTE for ways of sending PRINT data to another device like a printer, disk file or serial port.

See "Screen and Printer Text Control" in the front section of this manual for other ways of formatting text.

As with all other ZBasic graphics commands, PRINT %(x,y) defaults to printing at the position specified by the Device Independent Graphic coordinates of 1024 x 767. See PLOT or COORDINATE for ways of changing some versions of ZBasic to using other coordinates.

PRINT USING function

FORMAT **PRINT**[# *filenumber*,] **USING** *formatstring* ;*numeric expression*;[**USING**…]

DEFINITION This function permits formatting numeric data in PRINT or PRINT# statements.

The last numeric digit displayed will be rounded up by adding 5 to the first digit on the right that is not displayed.

The formatstring may be a quoted or string variable using the following symbols:

Symbol	Definition
#	Holds place for a digit. More than one may be used. An example of using this symbol to hold dollars and cents: `PRINT USING "$###.##";A#` `$123.45`
,	Insert a comma in that place. An example of using it to format numbers with dollars and cents would be: `PRINT USING"$##,###.##";A#` `$12,235.67`
.	Determines placement of decimal point within the format field: `PRINT USING"$##,###,###.##";A#` `$12,345,678.90`
$	Prints a dollar sign on the left of the format. See examples above.

+	Prints a floating plus or minus sign on the side of the number where the plus sign holds the place.
	`PRINT USING"+####.##";A#` `+1234.56`
	`PRINT USING"+####.##";-1234.56` `-1234.56`

-	Prints a minus sign only if the expression is negative.
	`PRINT USING"+####.##";A#` ` 1234.56`
	`PRINT USING"+####.##";-1234.56` `-1234.56`

*	Fill the spaces before a number with asterisks. One example would be formatting output when printing checks. `PRINT USING"$##,###,##.##";12.34` `$********78.90`

EXAMPLE See examples on next page...

REMARK When ***error*** is printed in the format field, this indicates the occurrence of an overflow condition and replaces the number that would have been printed. An overflow condition is when the value of the expression used would have exceeded the boundaries of the format.

USING not available for string formatting. See LEFT$,RIGHT$,STRING$ and MID$.

This version allows USING without PRINT. A$=USING"####.##";232 is acceptable. See appendix for additions to exponential formatting with this version.

PRINT USING continued

FORMAT EXAMPLES

In all the examples A=12345.678. Note that .678 rounds up to .68.

PRINT USING FORMAT	RESULT
"*$###,###,###,###.##";A	**********$12,345.68
"%###.#";A/1000	%12.3
"+###,###.##";A	+12,345.68
"-###,###.##";-A	-12,345.68
"##/##/##";A	1/23/45
"##:##:##";A	1:23:45
".###,###,###,###";1.345E-8	.000,000,013,450
".############";1.345E-8	.000000013450
"###,###,###,###,###";9.123E15	9,123,000,000,000,000
"###.##E16";123E15*1E-16	12.30E16

PROGRAM EXAMPLE

```
A$="##.##"
:
PRINT USING A$;10.2,USING A$;9.237, USING A$; 4.555
PRINT 10,12,13, USING A$;12.399
:
PRINT@(0,10);USING A$;23.12321
:
PRINT%(0,295);USING "@#####.##";12.33
:
OPEN"O",1,"TESTFILE"
PRINT#1, USING A$;9.999
CLOSE#1
```

RUN

```
10.20         9.24          4.56
10            12            13            12.40

23.12         <--- at text position 0.10
@12.33 <--- at graphic position 0,295
10.00         <--- To disk file "TESTFILE"
```

PSTR$ function/statements

FORMATS

function
PSTR$(*var*%)

statements
READ PSTR$(*var*%)
PSTR$(*var*%) = "*quoted string constant*"

DEFINITION

The statements load the address of a string constant into var%.

The function returns the string pointed to by var%.

EXAMPLE

```
DATA Andy, Dave, Scott, Mike
:
DIM D(4)
:
FOR X=1 TO 4          <---Set Pointer String to DATA items above
  READ PSTR$(D(X))
NEXT
:
"Print PSTR$ of D(n)"
FOR X=1TO4
  PRINT PSTR$(D(X))
NEXT
END
:
PSTR$(g%)="Hello"             <--- Set Pointer String to a constant
PRINT PSTR$(g%)
```

RUN

```
Andy
Dave
Scott
Mike
Hello
```

REMARK

This is a handy way to save string memory. Examples:

`A$="Hi There!"`
A$ will take at least 10 bytes (256 bytes if not defined). The quoted string takes another 10 bytes. Total memory used: 20 bytes

`PSTR$(A)="Hi There!"`
The quoted string "Hi There!" takes 10 bytes. The integer variable "A" takes two bytes. Total memory used: 12 bytes

Macintosh: Use var& instead of var%.

FORMAT **PUT**(x1,y1) *variable* [(*array index*[,*array index*[,…]) [,*mode*]

DEFINITION This statement places the graphic bit image stored in a array with the GET statement, to the screen position at coordinates specified by x1,y1.

If an array has been used then you MUST specify the index number of the array (some versions of BASIC always assume an integer array. ZBasic will allow you to store bit images in any variable type as long as enough memory is available to do so.

Memory required for pixel images id calculated using this formula (based on GET(x1,y1)-(x2,y2) where x1 and y1 designate the upper right-hand-corner of the image and x2 and y2 are the pixel positions designating the lower-left-hand-corner of the image):

6+((y2-y1)+1) * ((x2-x1+1) * bpp+7)/8)

The number of bits per pixel (bpp) depends on system colors or grey levels. See next page for specifics. Also see GET in this reference section, for detailed information about storing the pixel image in an array.

mode	XOR	XORs the pixels over the background pixels. This is the most useful for animation purposes and is also the default.
	OR	ORs the pixels over the existing pixels. This one way to cover the background graphics (overlays the existing graphics).
	AND	ANDs the picture with background.
	PRESET	Similar to PSET except the reverse image is shown (negative).
	PSET	Draws the image over the background exactly as created.

It is recommended that COORDINATE WINDOW be used when using GET.

EXAMPLE

```
DIM A(10000)
MODE 7                  <---- Not needed on the Macintosh version
COORDINATE WINDOW       <--- Pixel coordinates
:
CIRCLE 100,100,80
GET (0,0)-(100,100), A(1)
:
FOR x= 1 TO 200 STEP 3
  PUT (x, 90), A(1)   <---Do it twice to XOR the pixels and move the image across
  PUT (x, 90), A(1)       the screen without disturbing the background
NEXT x
:
END
```

This routine moves a section of a circle across the screen. It is XORed to the screen twice so the item doesn't repeat and it will appear to move across the screen without disturbing the background (default PUT mode is XOR).

continued...

PUT statement

REMARKS It is important to see entry under GET for more information.

Macintosh: With this version of ZBasic, PUT has another, optional, parameter: `PUT (x1,y1) [-(x2,y2)], var`. The second parameter allows you to scale the image, making it either larger or smaller by giving the rectangle size in which it is to appear. The x2, y2 parameter is the lower-right corner of the image.

Bits-per-pixel (bpp) will vary by the type of Macintosh you have. The standard black and white Macintoshes have one bit-per-pixel.

The Macintosh II may have up to 16 bits-per-pixel (with up to 256 colors or grey-levels per pixel). Check addendum of Macintosh II for specifics.

MSDOS: Bits-per-pixel (bpp) will vary with the graphics adaptor board being used:

GRAPHIC TYPE	MODE(s)	COLORS	BITS PER PIXEL (bpp)
CGA	5	4	2
CGA	7	2	1
EGA	16-19	3-16	2 (64K or less on EGA card)
EGA	16-19	16	4 (More than 64K on card)
HERCULES	20	1	1

Z80: GET and PUT are not supported with these versions of ZBasic.

Apple // ProDOS and DOS 3.3: GET and PUT are not supported with this version. See DRAW example on ProDOS disk and the BLOAD and BSAVE functions for possible alternatives.

FORMAT **QUIT**

DEFINITION QUIT is used to exit the ZBasic Standard Line editor and return control to the operating system.

EXAMPLE **QUIT**

```
DOS Ready        <----DOS prompt of your System.
```

REMARK We highly recommend saving your program prior to using **QUIT**.

Macintosh: You may also quit from the menu.

MSDOS: SYSTEM functions the same as QUIT.

RANDOM statement

FORMAT **RANDOM [IZE]** [*expression*]

DEFINITION Seeds the random number generator so that ZBasic produces a new sequence of random numbers.

If *expression* is used, the RND function will return a repeatable series of numbers.

EXAMPLE

```
DEFTAB 5
RANDOM 12345
FOR I = 1 TO 5
  PRINT RND(10),
NEXT I
:
RANDOM 12345   <--- Let's see if it repeats as above.
FOR I = 1 TO 5
  PRINT RND(10),
NEXT I: PRINT
```

RUN

```
8      1      10     4      7
8      1      10     4      7
```

```
PRINT"Press any key to set random seed"     <--- Paranoid seed routine
DO
  R=R+1
UNTIL LEN(INKEY$)
RANDOM R
:
FOR I = 1 TO 5
  PRINT RND(10),
NEXT I
END
```

```
RUN
```

```
Press any key to set random seed
1      8      8      5      9
```

REMARK The results of the first two passes were the same because the seed of 12345 was the same. When a different number is used, or no number, the result will be RANDOM.

If expression is the same, the same random pattern will be repeated with all versions of ZBasic.

The [IZE] part of RANDOM is not supported on the Apple // and Z80 versions.

FORMAT **RATIO** *byte expression*1, *byte expression*2

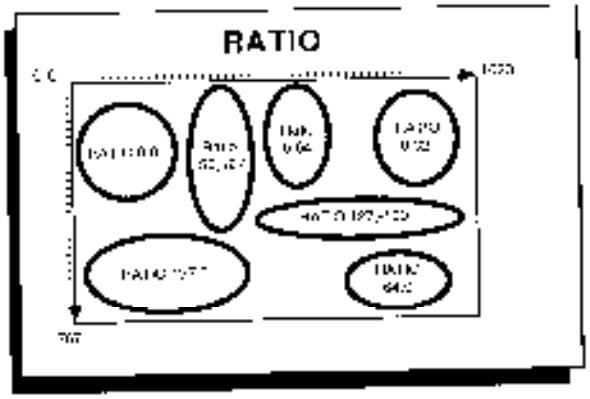
DEFINITION This statement will change the aspect ratio of graphics created with CIRCLE.

byte expression1Horizontal ratio. A number between -128 and +127 that gives the relationship of the width of the circle to normal (zero).

byte expression2Vertical ratio. A number between -128 and +127 that gives the relationship of the height of the circle to normal (zero).

Value	Relationship to normal		
+127	=	2.0	times normal
+64	=	1.5	times normal
+32	=	1.25	times normal
0	=	0	Normal proportion
-32	=	0.75	times normal
-64	=	0.5	times normal
-96	=	0.25	times normal
-128	=	0	times normal (no width or height)

EXAMPLE




```
RATIO -50, 127
CIRCLE h,v,r
```

REMARK RATIO settings are executed immediately and all CIRCLE commands, including CIRCLE TO and CIRCLE PLOT will be adjusted to the last RATIO.

Also see ROUNDRECT toolbox routines for other options to creating circles with various rations.

READ# statement

FORMAT **READ** # *filenumber*, {*var* |*var*$;*stringlength* } [, ...]

DEFINITION Reads strings or numbers saved in compressed format with WRITE# and stores them into corresponding variables. The list may consist of any type string or numeric variables or array variables.

filenumber	The filenumber to work from
var	Any numeric type variable
var$	String variable
;*stringlength*	The number of characters to load into the string variable

Important Note: A string variable must be followed by ;stringlength to specify the number of characters to be read into that string.

EXAMPLE

```
REM The four variables below will require 18 bytes for storage
REM A$=4 bytes, A!= 4 bytes, A#=8 bytes, A%=2 bytes
:
A$="TEST": A!="12345.6":A#="12345.67898":A%=20000
:
OPEN"0",1, "DATAFILE", 18     <--- Write a file with a record length of 18
WRITE #1, A$;4, A!, A#, A%
CLOSE#1
:
OPEN"I" ,1,"DATAFILE", 18
READ#1, Z$;4, Z!, Z#, Z%      <--- Read in same order and type (see notes)
CLOSE# 1
:
PRINT Z$, Z!, Z#, Z%
END

RUN

TEST           12345.612345.67898           20000
```

REMARK Note: Do not mix variable types when using READ# and WRITE#. Reading string data into numeric variables, and visa-versa, will create variables with incoherent data.

READ# and WRITE# store and retrieve numeric data in a compressed format. This saves disk space and speeds program execution.

While you may load numeric data into strings and convert using CVB or CVI, it is best to refrain from this since it requires more time and is less efficient.

See the chapter "Files" for more detailed information using random and sequential files. Also see RECORD, LOC,REC,LOF and "Disk Error Trapping".

FORMAT **READ** [*variable* {-or- PSTR$(*var*%) }[,...]]

DEFINITION The READ statement reads strings or numbers from a DATA statement into corresponding variables.

The variable list can consist of any combination of variable types (string or numeric, including arrays).

If no variable is given the READ statement will skip one DATA item.

EXAMPLE

```
DIM P%(3)
:
DATA Joe, Smith, Harry, "@ Cost"
DATA 1234.5, 567.8, 91011.12, 1314.15
:
READ A$, B$, C$, D$   <--- Regular old fashioned READ
READ A!, B!, C!, D!
PRINT A$, B$, C$, D$
PRINT A!, B!, C!, D!
:
RESTORE               <--- Set pointer back to start of DATA to READ again
FOR X=0 TO 3
  READ PSTR$(P%(X))   <---Use pointer string to point at DATA string constants
NEXT:PRINT
PRINT "PSTR$>"
FOR X= 0 TO 3
  PRINT PSTR$(P%(X)),
NEXT
:
RESTORE 6             <--- Set DATA pointer to the sixth item
READ A#
PRINT A#
END
```

RUN

```
Joe                   Smith         Harry         @ Cost
1234.5 567.8          91011.12      1314.15

PSTR$> Joe            Smith         Harry         @ Cost

567.8
```

REMARK Leading spaces in string data statements will be ignored unless contained in quotes.

Do not read numeric data into string variables and vice versa (no error is generated). Don't read past the end of a data list.

See RESTORE,PSTR$ and DATA.

RECORD statement

FORMAT **RECORD** [#] *filenumber*, *recordnumber* [,*location in record*]

DEFINITION The RECORD statement is used to position the file pointer anywhere in a file. Once the file pointer has been positioned you may read or write data from that position.

RECORD can position both the RECORD pointer and the location within a record.

filenumber	Filenumber from 1 to 99
recordnumber	RECORD number to point to. Default is zero.
location in record	Optional location in RECORD. Default is zero.

EXAMPLE

```
OPEN"R",1,"TESTFILE",30
:
FOR Position = 0 to 29
  RECORD #1, 6, Position
  READ#1, A$;1                <--- Reads one character at a time from record 6.
  PRINT A$;
NEXT
:
CLOSE#1
END

RUN

Fred          Stein
```

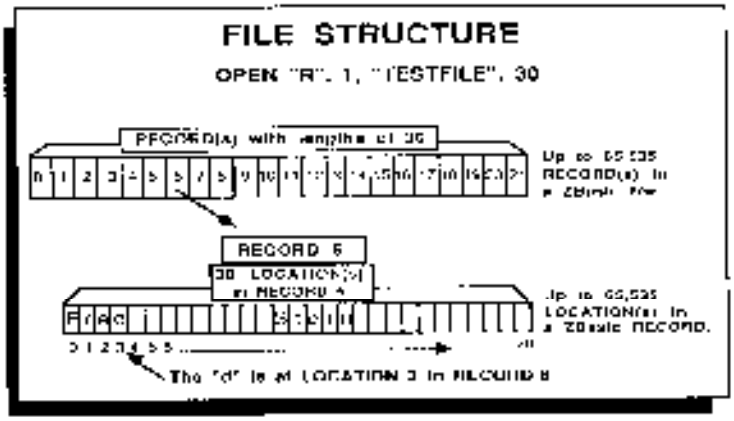
See illustration next page...

REMARK The default RECORD length is 256 bytes. The maximum record length is 65,535. The maximum number of records in a file is 65,535.

See OPEN,READ#,WRITE#,PRINT#,INPUT#,LINEINPUT#,LOC,LOF, REC, CLOSE, and the chapter entitled "Files".

The maximum record length and number of records in a file is 2,147,483,647.

RECORD continued



In the illustration, the name "Fred Stein" was stored in RECORD six of "TESTFILE".

To point to FILE #1, RECORD 6, LOCATION 3 use the syntax:

```
RECORD# 1, 6, 3
```

The location within a record is optional (zero is assumed if no location is given).

If RECORD 1, 6 had been used (without the 3), the pointer would have been positioned at the "F" in "Fred".

If RECORD is not used, reading or writing starts from the current pointer position. If a file has just been opened, the pointer is positioned at the beginning.

After each read or write, the file pointer is moved to the next position in the file.

The maximum record length and number of records in a file for this versions is 2,147,483,647.

REC function

FORMAT **REC** (*filenumber*)

DEFINITION Returns the current position of the record pointer for the file specified by expression. The first record in a file is record zero (0).

Also often used with REC is LOC which returns the position within the record.

EXAMPLE

```
OPEN "O",1,"THISPROG",10      <--- Record length of ten
:
A$="012345"                   <--- String length of six
:
FOR I = 0 TO 3
  PRINT#1, A$;
  PRINT "On pass";I;" file position was ";
  PRINT "Rec="REC(1);" and LOC=";LOC(1)
:
CLOSE#1
END

RUN

On Pass 0 file position was REC=0 and Loc=6
On Pass 1 file position was REC=1 and Loc=2
On Pass 2 file position was REC=1 and Loc=8
On Pass 3 file position was REC=2 and Loc=4

Right after the middle RECORD statement; REC=0 and LOC=4
```

REMARK The default record length is 256 bytes. LOC returns the position within a RECORD.

See OPEN,CLOSE,LOC,LOF,RECORD,READ#,WRITE# and the chapter entitled "Files".

FORMAT **REM** *followed by programming remarks*

DEFINITION The REM statement is used for inserting comments or remarks into a program. ZBasic ignores everything following a REM statement.

To save time, you can type an apostrophe (') at the beginning of a line and it will be converted into a REM statement.

EXAMPLE

```
REM           This is a comment or remark
REM           ZBasic ignores everything following a REM
REM           Including any commands embedded in the remark
:
REM           Colons are often used to make blank lines.
:
:
:
REM           Thoughtful use of REM makes a program easier to read.

RUN

ZBasic Ready_
```

REMARK REM statements are not compiled and do not take up any memory in the object code.

Note: Some versions of ZBasic will not convert the apostrophe to REM.

RENAME statement

FORMAT **RENAME** *string*1 {,|TO} *string*2

DEFINITION This statement is used to rename the file string1 to the new name string2.

EXAMPLE

```
DIR

GOOGOO          ZBASIC.COM
FRED.BAS                OLDFILE.BAS

INPUT "FILE NAME TO CHANGE: ";File1$
INPUT "NEW NAME FOR FILE: ";File2$
RENAME File1$ TO File2$

RUN

FILE NAME TO CHANGE: GOOGOO
NEW NAME FOR FILE: GOONIE

DIR

GOONIE          ZBASIC.COM
FRED.BAS                OLDFILE.BAS
```

REMARK This command is also available in command mode. Remember that filename formats are different from system to system and may not be available for some machines.

TRS-80 model 1,3: RENAME not supported with these versions.

Macintosh: Pathnames or volume number may be used.
Macintosh: RENAME file1$ {TO|,} file2$ [, *volume number*%]. Also see NAME.

MSDOS: See CHDIR, PATH$, RMDIR and MKDIR in the MSDOS appendix for controlling pathnames and directories. Also see NAME.

Apple // ProDOS: Pathnames supported.

FORMAT **RENUM** [*new*][,[*old*]][, *increment*]

DEFINITION Used for renumbering program lines.

new	The first new assigned line number desired after renumbering is complete. default = 10
old	The first old line where you want renumbering to begin. default = 0
increment	The increment between line numbers. default = 10 (256 maximum)

If an argument is omitted the default will be used.

This command will automatically update line references (GOTO,GOSUB, etc). If a line reference is to a non-existent line, it will use the next existing line number.

EXAMPLE

```
7  IF I = 200 THEN 567
74  PRINT I
197  I = I + 1: GOTO  74
567 END
```

```
RENUM
LIST

10  IF I = 200 THAN 40
20  PRINT I
30  I  =  I + L: GOTO 10
40  END
```

REMARK Line increments are limited to 256. If you issue a RENUM command that exceeds the number of allowable lines (65,534) , an error will occur and your text will be unaltered.

If you are unsure of what the results may be, SAVE your program BEFORE renumbering!

Some versions offer options for using, or not using, line numbers with full screen editors. Check your appendix for specifics.

See RENUM*,UNNUM,INDENT and FIX in the MSDOS appendix for other options.

RESET statement

FORMAT **RESET**

DEFINITION Closes all open files and devices. Functionally identical to CLOSE without parameters.

EXAMPLE

```
OPEN"O",1,"FRED"
OPEN"I",2,"HARRY"

IF ERROR THEN RESET

END
```

REMARK See CLOSE

Not supported on Apple // or Z80 versions of ZBasic. Simply use CLOSE without a filenumber to close all open files.

FORMAT **RESTORE** [*expression*]

DEFINITION This statement resets the DATA pointer to the first DATA statement or optionally to the DATA item specified by expression.

If the expression is omitted, the first DATA item is assumed. ZBasic automatically sets the pointer to the next item after each variable is READ.

EXAMPLE

```
DATA ZERO, ONE, TWO, THREE, FOUR, FIVE
DATA SIX, SEVEN, EIGHT, NINE, TEN
:
"Start"
DO
  INPUT"What item do you want""Item%
  IF (item%<0) OR (item%>10) THEN "Start"
  RESTORE Item%
  READ A$
  PRINT "Item number:;Item%;" is: ";A$
UNTIL Item%=0
:
RESTORE                <--- Set to beginning of DATA
READ A$: PRINT A$
:
END
```

RUN

```
What item do you want: 4
Item number 4 is: FOUR

What item do you want: 9
Item number 4 is: NINE

What item do you want: 0
Item number 0 is: ZERO

ZERO
```

REMARK If an attempt is made to READ or RESTORE past the last DATA item, the result will be zeros or NULL strings. No error will be returned.

Also see READ, PSTR$ and DATA.

RETURN statement

FORMAT **RETURN** [*line*]

DEFINITION The RETURN statement is used to continue execution at the statement immediately following the last executed GOSUB or ON GOSUB statement.

If optional line is used, the last GOSUB is POPPED off the stack and a GOTO line is performed.

EXAMPLES

```
GOSUB "First"
:
"Second"
PRINT "RETURN comes here."
END
:
"First"
PRINT "This is a subroutine"
RETURN
```

RUN

```
This is a subroutine
Return comes here
```

```
GOSUB "Routine"
END
:
"Weird"
PRINT"Ended Here!"
STOP
:
"Routine"
PRINT"At 'Routine'"
RETURN "Weird"
```

RUN

```
At 'Routine'
Ended Here!
```

REMARK When ZBasic encounters a RETURN statement which was not called by a GOSUB, it will return to the program that executed it (either DOS or the ZBasic editor).

Using RETURN line WITHOUT A GOSUB or from the middle of a LONG FN will cause unpredictable (probably disastrous) system errors.

Use caution when using RETURN line to exit event trapping routines like DIALOG ON,MENU ON, TRON,BREAK ON...

FORMAT **RIGHT$**(*string, expression*)

DEFINITION Returns the right-most expression characters of string.

EXAMPLE

```
A$="HELLO"
:
FOR I = 0 TO 6
  PRINT I, RIGHT$(A$,I)
NEXT I
:
A$ = "JOHN DOE"
:
SP = INSTR(1,A$,"  ")
PRINT"LAST NAME:",
PRINT RIGHT$(A$,LEN(A$)-SP)
:
END
```

```
RUN

0
1             0
2             LO
3             LLO
4             ELLO
5             HELLO
6             HELLO
LAST NAME:    DOE
```

REMARK If expression is more than the characters available, all the characters will be returned.

See LEFT$,VAL,STR$,STRING$,SPACE$,SPC, MID$ and the chapter entitled "String Variables" in the front section of this manual.

RND function

FORMAT **RND** (*expression*)

DEFINITION The RND function returns a random integer number from 1 to expression.

EXAMPLE

```
RANDOM
A=9
:
FOR I=1 TO 5
  PRINT RND(A),
  PRINT RND(10000)*.0001
NEXT I
:
END

RUN

3             .9201
7             .8211
1             .0912
2             .7821
9             .0108
```

REMARK Some versions of BASIC return a floating point random number between 0 and 1; use RND(10000)*.0001 to emulate this (it will slow down execution).

Also see MAYBE and RANDOM.

If the same speed number is used for RANDOM, the random numbers generated by RND will be predictable on the all versions of ZBasic.

The largest number you may use for a RND *expression* is 32,767.

FORMAT **ROUTE** [#] *expression*

DEFINITION This statement is used to route PRINT statements to a specified device. The following are the values to be used as expression.

Device number	**Routes PRINT statements to**
negative numbers	I/O devices; See your appendix for specifics.
0	Screen (default)
1-99	Disk files specified by number
128	Printer

EXAMPLE

```
ROUTE 128
PRINT "HELLO"          <--- This HELLO goes to the printer
:
OPEN"O",1,"Test"
ROUTE 1
PRINT "HELLO"          <--- This HELLO goes to file "Test"
CLOSE#1
:
OPEN"C",-1,300
ROUTE -1
PRINT "HELLO"          <--- This HELLO goes to a serial device
CLOSE#-1
:
ROUTE 0
PRINT"HELLO"           <--- This HELLO goes to the screen
END

RUN

HELLO
```

REMARK You should eventually route the output back to a screen device (ROUTE 0).

See PRINT,OPEN"C" and the chapter "Files" for more information.

Also see ROUTE 128, CLEAR LPRINT, DEF LPRINT and DEF PAGE for more information about routing text and graphic output to the Imagewriter and Laserwriter. Be sure to use CLEAR LPRINT with ROUTE 128 to tell the Macintosh printer driver to print the page.

RUN statement

FORMAT **RUN** [*filenumber*]

DEFINITION The RUN statement does one of two things.

RUN *filenumber* Loads a compiled chain program specified by filenumber and executes it:

```
OPEN"I", 1, "Prog.CHN"
RUN 1
```

RUN Clears all variables and pointers and restarts the current program from the first line.

EXAMPLE

```
OPEN"I",2,"MENU"
RUN 2                   <---Loads and RUNS CHAIN program "MENU"
```

```
TRONB
FOR X=1 TO 100
  PRINT X
NEXT
RUN                     <--- RUNS this program over and over...
```

REMARK Also see the RUN command and the chapters "Running ZBasic Programs" and "Chaining" for more information.

Also see RUN filename$, volumenumber% in the appendix.

FORMAT **RUN** [[{+|*}]['] *filename* ['']]

DEFINITION This command is used from the Standard Line Editor to compile a program:

RUN	Compiles source code in memory and executes.
RUN *filename*	Compiles source code called filename from disk and executes. Source code must have been saved in tokenized format with SAVE (not as a text file).
RUN*	Compiles source code in memory and saves as a stand-alone application on disk. Asks for filename after compiling.
RUN* *filename*	Compiles source code called filename from disk and saves as a stand-alone application on disk. Source code must have been saved tokenized (not as a text file). Asks for filename after compiling.
RUN+	Compiles source code in memory and saves as a chain file to disk (no runtime included). Asks for filename after compiling.
RUN+*filename*	Compiles source code called filename from disk and saves as a chain file to disk (no runtime included). Asks for filename after compiling.

EXAMPLE

```
PRINT "THE PROGRAM RUNS!"

RUN

THE PROGRAM RUNS!
```

REMARK Compiling from disk will destroy any text currently in memory. If an error is encountered when compiling from disk, ZBasic will load the source code and print an error message.

After a successful compilation, typing MEM will return memory used for the object code and variables.

See "Executing Programs" in the front of this manual for more information about compiling large programs.

Also see COMPILE and LCOMPILE for ways of compiling a program and seeing all the compile time errors at once (instead of one at a time as with RUN).

SAVE command

FORMAT **SAVE** [[{*|+}] ['] *filename* ['']]

DEFINITION SAVE is used from the Standard Line Editor to save the source code in memory. You may save your source code in a number of formats:

SAVE	Saves program in tokenized format. This requires less room on the disk and saving and loading is much faster than with text files. In order to compile a file from disk a program must be saved in this format.
SAVE*	Saves program in TEXT or ASCII format. This allows you to load the program into other word processors or editors. Loads more slowly than SAVE above.
SAVE+	Same as SAVE* but line numbers are removed. Be sure your program doesn't uses label references with GOTO, GOSUB or other commands, since when a program is re-loaded, line numbers are added back in increments of one which will make line number references incorrect.

Note: Source code is the program you type in. Object code is the machine language program created when you compile the source code with RUN. See RUN for more information about compiling and saving compiled programs to disk.

EXAMPLE

```
SAVE* PROGRAM.TXT      <---SAVE program in ASCII (text)
SAVE  AR.BAS           <---SAVE program tokenized (condensed)
SAVE+ FILE.TXT         <---SAVE program in ASCII - with no line numbers
```

REMARK Also see LOAD,APPEND,MERGE and RUN.

FORMAT

SELECT [*expression or simplestring*]
CASE [IS] relational condition [, relational condition][,...]
statements...
CASE [IS] condition [, condition][,...]
statements...
CASE [IS] boolean expression
statements...
CASE ELSE
END SELECT

DEFINITION

Provides a structured and efficient way of doing multiple comparisons with a single expression. While IF-THEN or LONG-IF statements could be used, they are harder to follow when reading program listings.

EXAMPLE

```
X=CARDTYPE:REM MSDOS Cardtype example.
SELECT X
  CASE 0
    PRINT"CGA CARD":MODE 7
  CASE 1
    PRINT"EGA CARD":MODE 19
  CASE 2
    PRINT"EGA with Mono":MODE 18
  CASE 3
    PRINT"HERCULES CARD":MODE 20
  CASE 255
    PRINT "Monochrome Monitor":MODE 2
  CASW ELSE
    PRINT"No Video card installed"
END SELECT
```

REMARK

See CASE and END SELECT for more examples.

Important Note: Exit a SELECT structure only at the END SELECT.

SELECT is not supported with the Apple or Z80 versions of ZBasic. Use IF-THEN or LONG-IF to accomplish the same thing.

SGN function

FORMAT **SGN**(*expression*)

DEFINITION Returns the sign of *expression*.

If *expression* is:

```
Positive             +1 is returned.
Zero                  0 is returned.
Negative             -1 is returned.
```

EXAMPLE

```
DEFDBL A-Z: DEFTAB 8: WIDTH 40
PRINT"       X","ABS(X)", "INT(X)","FRAC(X)",SGN(X) "
:
FOR X = -15.0 TO +15.0 STEP 3.75
  PRINT USING"-##.##";X,
  PRINT USING "##.##";ABS(X),
  PRINT USING"-##.##";INT(X),
  PRINT USING "-#.##";FRAC(X),
  PRINT USING "-#.##";SGN(X)
NEXT X
```

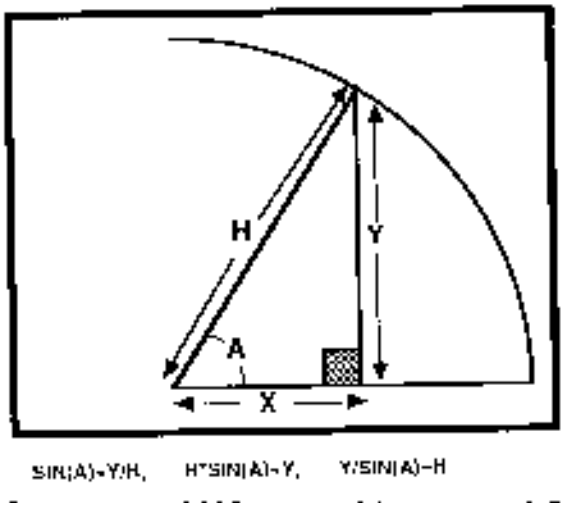
RUN

```
  X           ABS(X)  INT(X)  FRAC(X)SGN(X)
-15.00 15.00          -15.00     .00 -1.00
-11.25 11.25          -11.00   -.25 -1.00
- 7.50  7.50          - 7.00   -.50 -1.00
- 3.75  3.75          - 3.00   -.75 -1.00
   .00   .00             .00    .00   .00
  3.75  3.75            3.00    .75  1.00
  7.50  7.50            7.00    .50  1.00
 11.25 11.25           11.00    .25  1.00
 15.00 15.00           15.00    .00  1.00
```

REMARK Also see UNS$, FRAC, INT, ABS and negation.

FORMAT **SIN** (*expression*)

DEFINITION The SIN function returns the sine of the expression in radians.



EXAMPLE

```
X#=SIN(123)
PRINT SIN(X2#)
```

REMARK SIN is a scientific function. The precision for scientific functions may be configured. See "Configure" in the front of this manual for more information.

See the "Math" and "Expressions" sections of this manual and ATN, TAN, COS, EXP,SQR,^.

INTEGER SINE: ZBasic provides a predefined USR function to do hi-speed integer sines. This speeds up sine speed by up to 30 times:

USR8(angle) returns the integer sine of angle in the range +-255 (corresponding to +-1). The angle must be in brads: See CIRCLE for examples of brads. Example:

```
MODE 7 :CLS
FOR I=0 TO 255
  PLOT I<<2,-USR8(I)+384
NEXT I
```

SOUND statement

FORMAT **SOUND** *frequency, duration*

DEFINITION **SOUND** may be used to create sound effects or music.

frequency Frequency 120 Hz to 10,000 Hz.
duration Duration in 1 millisecond increments.

Note: Hz (Hertz) represents cycles-per-second.

EXAMPLE

```
DO
  INPUT"Tone: ";Tone
  INPUT"Duration: ";Duration
  :
  SOUND Tone, Duration
  :
UNTIL (Tone=0) OR (Duration=0)
```

Example frequencies you may use in your program to create music or sound effects. (Choose the duration as required.) Quality of sound may vary by machine.

			OCTAVES				
NOTES	**1**	**2**	**3**	**4**	**5**	**6**	**7**
C	33	66	132	264	528	1056	2112
C^{b}	35	70	140	281	563	1126	2253
D	37	74	148	297	594	1188	2376
E^{b}	39	79	158	316	633	1267	2534
E	41	82	165	330	660	1320	2640
F	44	88	176	352	704	1408	2816
G^{b}	46	93	187	375	751	1502	3004
G	49	99	198	396	792	1584	3168
A^{b}	52	105	211	422	844	1689	3379
A	55	110	220	440	880	1760	3520
B^{b}	57	115	231	462	924	1848	3696
B	61	123	247	495	990	1980	3960

REMARK Some computers may not have sound. See your computer appendix for more information.

CP/M-80: Sound not supported. CHR$(7) may sound a bell on some systems.
TRS-80 model 1,3: Requires that a speaker be connected to the cassette port.
TRS-80 model 4: Frequency range of internal speaker limitied to 0,0 to 7,31.

See appendix for using four voice sound and utilizing the sound buffer.

FORMAT **SPACE$** (*expression*)

DEFINITION Returns a string of spaces expression characters long (range of 0 to 255).

EXAMPLE

```
PRINT "ZEDCORZEDCORZE"
FOR X=7 TO 0 STEP -1
  PRINT SPACES$(X);"ZEDCOR"
NEXT
PRINT"ZEDCORZEDCORZEDCOR"
END

RUN

ZEDCORZEDCORZE
       ZEDCOR
      ZEDCOR
     ZEDCOR
    ZEDCOR
   ZEDCOR
  ZEDCOR
 ZEDCOR
ZEDCOR
ZEDCORZEDCORZEDCOR
```

REMARK See STRING$,MID$,RIGHT$,LEFT$,INSTR and SPC.

SPC function

FUNCTION **SPC** (*expression*)

DEFINITION SPC prints *expression* spaces from 0 to 255

Prints the number of spaces specified by expression.

EXAMPLE

```
DO
  PRINT"*";SPC(RND(20));"+"
UNTIL LEN(INKEY$)
```

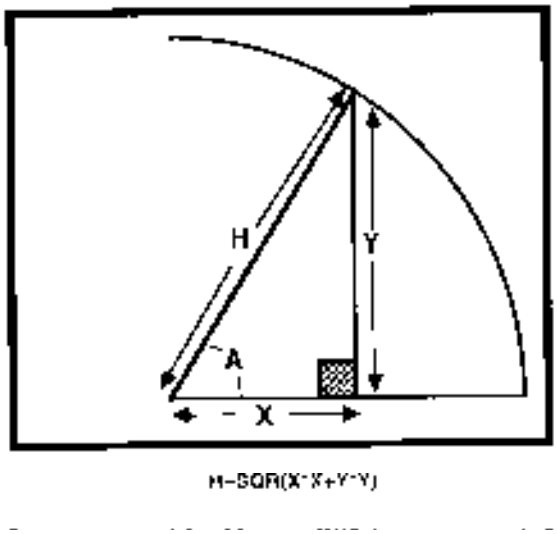
RUN

```
                +
               +
  +
        +
                  +
+
```

REMARK Also see SPACE$,LEFT$,STRING$,RIGHT$,MID$ and INSTR.

FORMAT **SQR** (*expression*)

DEFINITION The SQR function returns the square root of *expression.*



H=SQR(X*X+Y*Y)

EXAMPLE

```
A=9
PRINT SQR(A)
```

RUN

3

REMARK SQR is a scientific function. Scientific functions may be configured to a different precision. See "Configure" in the front of this manual for more information.

For more information on scientific functions see the "Math" and "Expression" sections of this manual and ATN, SIN, COS, TAN, EXP and ^.

STEP statement

FORMAT **FOR** *variable* = *expr*1 TO *expr*2 [**STEP** *expr*3]
"
"
"
NEXT [*variable*][,*variable*…]

DEFINITION This parameter allows you to set the increments used in a FOR-NEXT loop. If STEP is omitted than one is assumed.

EXAMPLE

```
FOR X= 0 TO 10 STEP 2
  PRINT X
NEXT
:
FOR X = 10 TO 0 STEP -1
  PRINT X
NEXT
END

RUN

0 2 4 6 8 10
10 9 8 7 6 5 4 3 2 1 0
```

REMARK Also see FOR, NEXT, DO, UNTIL, WHILE, WEND and the chapter on "Loops".

IF STEP =0 an endless loop will result.

If *expr*1 or *expr*3 change while the loop is executed this change will be in effect when NEXT is encountered.

Avoid long or complex loop expressions for *expr*1 or *expr*3 as they are evaluated every loop and will slow execution.

FORMAT **STOP**

DEFINITION **STOP** halts execution of a ZBasic program and prints the line number where execution stopped (if line numbers weren't used the lines are numbered in increments of one).

STOP when used from ZBasic will return to the Standard Line Editor.

STOP when used from a stand-alone program will return to the operating system.

EXAMPLE

```
PRINT"HELLO"
STOP

RUN

Break in 00002
ZBasic Ready
```

REMARK STOP closes all files.

END may be used when no message is desired.

See TRONB and TRONX for ways of inserting break points in your programs so that <BREAK> may be used to exit a running a program.

STR$ function

FORMAT **STR$**(*expression*)

DEFINITION STR$ returns the string equivalent of the number represented by *expression*. This is used to convert numbers or numeric variables to a string.

This function is the compliment of VAL. VAL returns the numeric value contained in a string.

EXAMPLE

```
Integer% =20000
Single! =232.123
Double# = .12323295342
:
A$=STR$(Integer%)     :PRINT A$
A$=STR$(Single!)      :PRINT A$
A$=STR$(Double#)      :PRINT A$
:
X#=VAL(A$)
PRINT X#

RUN

20000
232.123
.12323295342
.12323295342
```

REMARK Also see BIN$, OCT$, HEX$, MKI$, CVI,MKB$, CVB and VAL.

FORMAT **STRING$** (*expr*$_1$, *string*)
STRING$ (*expr*1, *expr*2)

DEFINITION Returns a string of the length expr1 consisting of the characters specified by either the ASCII equivalent of expr2 or the first character of string.

EXAMPLE

```
PRINT STRING$ (5,"#")
PRINT STRING$ (10,65)
PRINT STRING$ (10,CHR$(65))
:
A$ = STRING$(3,"*") + "TEST"+ STRING$(3,"&")
PRINT A$
END
```

```
RUN

#####
AAAAAAAAAA
***TEST&&&
```

REMARK STRING$ is more efficient than using an equivalent string of characters.

See SPACE$,LEFT$,RIGHT$,MID$,INSTR,VAL,STR$,INDEX$ and SPC.

SYSTEM statement

FORMAT **SYSTEM**

DEFINITION Same as END. Provided for compatibility with other versions of BASIC.

EXAMPLE

```
PRINT"HELLO"
SYSTEM

RUN

HELLO
```

REMARK Functionally identical to the ZBasic END statement. See END and STOP.

Not Supported with Apple // or Z80 versions of ZBasic. Use END.

FORMAT **SWAP** *var*1, *var*2

DEFINITION SWAP exchanges the contents of *var*1 and *var*2. The variables can be of any type except INDEX$ variables.

*Var*1 and *var*2 must be of the same type.

EXAMPLE

```
B$="YES"
A$="NO"
PRINT A$, B$
SWAP A$, B$
PRINT A$, B$
PRINT
:
A=1:B=100
PRINT A,B
SWAP A,B
PRINT A,B
END

RUN

YES             NO
NO              YES

1               100
100             1
```

REMARK SWAP will execute faster and take less memory than similar methods using "holding variables".

SWAP *does not* function with INDEX$.

TAB function

FORMAT **TAB** (*expression)*

DEFINITION Tab will move the cursor to the positions; 0 through 255, designated by expression.

Three devices may be used with Tab:

DEVICE	FORM	WILL POSITION
SCREEN	PRINT	CURSOR
PRINTER	LPRINT	PRINT HEAD
DISK	PRINT#	FILE POINTER

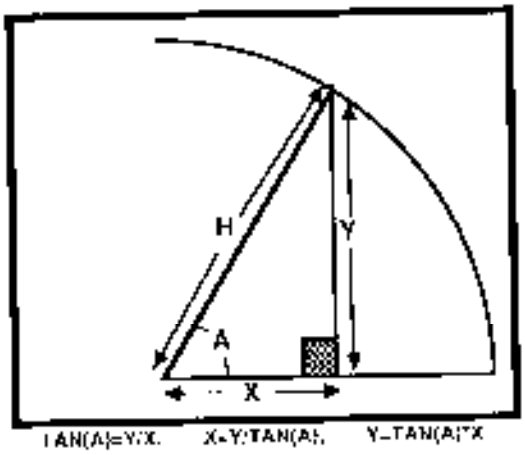
EXAMPLE

```
DATA Fred Smith, 12 E. First, Tucson, AZ, 85712
DATA Dana Andrews, 32 Main, LA, CA, 90231
:
PRINT "Name"TAB(15) "Address"TAB(30) "City"TAB(40) "State  ZIP"
PRINT STRING$(50,"-")
:
FOR Item= 0 TO 1
  RESTORE Item*5
  READ N$, A$, C$, S$, Z$
  PRINT N$ TAB(15) A$ TAB(30) C$ TAB(40) S$"     "Z$
NEXT
END

RUN

Name                 Address       City          State  ZIP
-------------------------------------------------
Fred Smith           12 E. First          Tucson AZ      85712
Dana Andrews   32 Main       LA            CA      90231
```

REMARK Tab will start numbering from the zero position. Also see DEFTAB,PRINT@, PRINT%,POS,PAGE,WIDTH and WIDTH LPRINT.

FORMAT **TAN** (*expression*)

DEFINITION Returns the value of the tangent of the expression in radians.



EXAMPLE

```
X#=TAN(T+Z)/3
```

REMARK TAN is a scientific function. Scientific accuracy may be configured differently than single or double precision. See "Configure" at the beginning of this manual.

Also see ATN,COS,SIN,EXP,SQR and ^.

For more information on scientific functions see "Math" and "Expressions" in the front section of this manual.

TIME$ function

FORMAT **TIME$**

DEFINITION Returns an eight character string which represents the systems clock value in the format HH:MM:SS where HH=1 to 24 hours, MM= 0 to 60 minutes, SS= 0 to 60 seconds.

EXAMPLE

```
PRINT TIME$
DELAY 1000
A$=TIME$
PRINT A$

RUN

10:23:32
10:23:33
```

REMARK See DATE$ and DELAY.

This function will return a 00:00:00 if the system or version has no clock.

Macintosh: Set time from the Control Panel Desk Accessory. Also see TIMER for other ways of getting seconds.

MSDOS: Set time using TIME$= hh, mm,ss. Also see TIMER.

Apple: See appendix for variations of system clocks.

Z80: See appendix for your particular hardware.

FORMAT **TROFF**

DEFINITION TROFF is used to turn off the trace statements: TRON, TRONX, TRON and TRONS.

EXAMPLE

```
TRON
FOR X=1 TO 3
NEXT
:
TROFF
PRINT "Line tracing now off"
FOR X=1 TO 10
NEXT

RUN

00001 00002 00003 00002 00003 00002 00003 00004 Line tracing
now off
```

REMARK See also TRON, TRONS, TRONB, TRONX and the chapter on "Debugging Tools".

TRON statement

FORMAT **TRON** [{B|S|X}]

DEFINITION These statements are used for tracing program execution, single stepping through a program, and setting break points for monitoring the <BREAK> key so that you can break out of a program.

TRACING PROGRAM FLOW

TRON	Prints the line numbers of the program as each line is executed so you can trace program flow and check for errors.
TRON S	Lets you single step through a program. Program execution will pause at the beginning of every line in the program following TRON S (up to the end of the program or when a TROFF is encountered). Press any key to continue or press the <CTRLZ> key to enable/disable single-stepping. <BREAK> also works.

SETTING BREAK POINTS

TRON X	Sets a break point at that line in a program and checks to see if the <BREAK> key has been pressed.
TRON B	Sets a break point at the beginning of every line in the program following it (up to the END or until a TROFF is encountered).

Note: The <BREAK> key is checked at the beginning of a line. IF <BREAK> is encountered in a program compiled with RUN, program exits to the Standard Line Editor. If <BREAK> is encountered in a stand-alone program, exit is to the system.

<CTRL S> will pause execution when encountered during execution of TRONB, TRONX or TRON. Any key will restart. <CTRL Z> will activate/deactivate single-step mode when any TRON is active. Note: INKEY$ may lose keys if TRON is used.

EXAMPLE

```
TRON:TRONS
:
PRINT "HELLO"
:
TROFF
```

```
RUN

00001 <KEY> 00002 <KEY> 00003 <KEY> HELLO 00004 <KEY>
```

REMARK Every line between a TRON and TROFF may use up to eight extra bytes per line. Use TRON sparingly to save memory and increase execution speed. See chapter entitled "Debugging Tools" for more information. INKEY$ may lose keys with TRON.

Macintosh: <BREAK> is <Command Period>. Also see BREAK ON, and TRON WINDOW in appendix for other ways of tracing program flow and variable values.
MSDOS: <BREAK> is <CTRL C>.
CP/M: <BREAK> is <CTRL C>.
Apple // ProDOS or DOS 3.3: <BREAK> is <CTRL C> or <CTRL RESET>.
TRS-80: <BREAK> is <BREAK>.

FORMAT **UCASE$** (*string*)

DEFINITION Returns a string with all characters converted to uppercase (capital letters).

EXAMPLE

```
PRINT UCASE$("hello")
:
A$="HeLLo"
PRINT UCASE(A$)
END

RUN

HELLO
HELLO
```

```
DO
  key$=UCASE$(INKEY$)
UNTIL LEN (key$)
PRINT key$
END

RUN

S               <---always returns an uppercase character
```

```
REM This function converts a string to Lowercase
:
LONG FN lcase$(string$)
  FOR X=1 TO LEN(string$)
    A=PEEK(VARPTR(string$)+X)
    IF (A>64) AND (A<91) THEN A=A+32
    POKEVARPTR(string$)+X,A
  NEXT
END FN=string$
:
PRINT FN lcase$("HELLO")

RUN

hello
```

REMARK This function is very useful when sorting data containing upper and lower case and for checking user input without regard to case.

Also see LEFT$,RIGHT$,MID$,INSTR,STR$,VAL, and the chapter "String Variables" in this manual.

UNS$ function

FORMAT **UNS$** (*expression*)

DEFINITION Returns a sting which equals the integer value of expression in an unsigned decimal format.

EXAMPLE

```
PRINT UNS$(-1)
PRINT UNS$ (4)
:
PRINT
PRINT 65535

RUN

65535
00004

-1
```

REMARK This function is useful for displaying integers in an unsigned format (0 through 65,535 instead of -32,768 through 32,767).

See STR$, DEC$, OCT$, HEX$, VAL and the chapter on "Numeric Conversions".

See DEFSTR LONG for enabling this function to work with LongIntegers.

FORMAT

DO
.
.
UNTIL *expression*

DEFINITION

UNTIL is used to mark the end of a DO loop.

The DO loop repeats until the expression following the UNTIL is true (non-zero).

A DO loop will always execute at least once.

EXAMPLE

```
DO
  X=X+1
UNTIL x=100
PRINT X
:
"Wait for a key"
DO
 I$=INKEY$
UNTIL LEN(I$)
END

RUN

100
<KEY PRESS>
```

REMARK

Notice ZBasic will automatically indent DO loop structures two spaces. See the chapter on "Formatting Program Listings" for other ways of formatting listings.

Also see FOR, NEXT, STEP, WHILE, WEND and the chapter on "Loops" in the technical section of the manual.

WHILE,WEND may be used to exit a loop immediately if a condition is false.

USR function

FORMAT **USR** *digit* (*word expression*)

DEFINITION The USR function calls the user created subroutine, defined with DEFUSR, specified by a *digit* 0 to 9, and returns the value of integer *expression* in the 16 bit accumulator.

EXAMPLE

```
REM EXAMPLE ONLY    DO NOT USE!
:
DEFUSR2 = LINE "Routine two"
X=USR2(938)
PRINT X
END
:
"Routine two"
MACHLG &8B,&C4,&C3:RETURN

RUN

23921
```

REMARK A machine language return is necessary at the end of a USR routine.

ZBasic provides pre-defined USR functions that perform some powerful functions like integer sine and cosine. See next page.

Macintosh: Be sure to use LongIntegers whenever referencing memory addresses. Also see CALL in the Macintosh appendix.

MSDOS: See CALL in your appendix.

Apple ProDOS: See MLI in the ProDOS appendix.

Predefined USR functions.

These pre-defined USR functions are available for all versions of ZBasic. See your Computer Appendix for possible other USR functions.

USR6(*expr*)
Returns the last line number executed that used any of the TRON functions (expr is not used).

```
TRONX
I=USR6(0)
PRINT I
```

USR7(*expr*)
Returns ZBasic's random number seed used in the RND function (expr is not used).

```
FOR I=1 TO 10
  PRINT USR7(0)
NEXT I
```

USR8(*angle*)
Returns the integer sine of angle in the range +-255 (corresponding to +-1). The angle must be in brads.

```
MODE7 :CLS
FOR I=0 to 255
  PLOT I<<2,-USR8(I)+384
NEXT I
```

USR9(*angle*)
Returns the integer cosine of angle in the range +-255 (corresponding to +-1). The angle must be in brads.

```
MODE7 :CLS
FOR I=0 to 255
  PLOT I<<2,-USR9*I)+384
NEXT I
```

USR statement

FORMAT **USR** *digit* (*expression*)

DEFINITION This statement will call the USR routine defined by DEFUSR digit and transfer the result of *expression* in the integer accumulator.

EXAMPLE

```
Example only DO NOT USE
:
DEFUSR0=LINE "Machine language"
USR0(0)
END
:
"Machine Language"
MACHLG &39, &C9:  RETURN
```

REMARK The USR routine must be set by the program or be a predefined USR routine. Also see DEFUSR, USR function,LINE,CALL,MACHLG,the chapter about "Machine Language" in this manual, and your computer appendix.

Macintosh: Be sure to use LongIntegers whenever referencing memory addresses. Also see CALL in the Macintosh appendix.

MSDOS: See CALL in your appendix.

Apple ProDOS: See MLI in the ProDOS appendix.

FORMAT **VAL** (*string*)

DEFINITION Returns the numeric value of the first number in a string.

The VAL function will terminate conversion at the first non-numeric character in string.

This function is the compliment of STR$. STR$ will convert a numeric expression to a string.

EXAMPLE

```
A$="HELLO"
B$="1234.56"
C$="99999"
:
PRINT "The value of A$=";VAL(A$)
PRINT "The value of B$=";VAL(B$)
PRINT "The value of C$=";VAL(C$)
:
PRINT
PRINT "The value of 9876.543=";VAL("9876.543")
END
```

```
RUN

The value of A$= 0
The value of B$= 1234.56
The value of C$= 99999

The value of 9876.543= 9876.543
```

REMARK The numeric value returned by VAL will be in floating point format.

See STR$, UNS$, HEX$, OCT$ and BIN$,INT,FRAC,ABS,FIX.

Also see the chapter on "Math" and "Expressions" in the front section of this manual.

VARPTR function

FORMAT **VARPTR**(*variable*)

DEFINITION Returns the address of a variable . Any variable type may be used except INDEX$.

EXAMPLE

```
A$="HELLO"
:
PRINT "Address of A$=";VARPTR(A$)
PRINT "Length of A$ =";PEEK(VARPTR(A$))
:
PRINT "Contents of A4= ";
FOR X=1 TO LEN(A$)
  PRINT CHR$(PEEK(VARPTR(A$)+X));
NEXT
END
```

RUN

```
Address of A$= 23456
Length of A$ = 5
Content of A$= HELLO
```

REMARK The following paragraphs describe which address VARPTR will be pointing to with different variable types.

INTEGER	Points to the 1st byte of an integer variable.
SNG/DBL	Points to the sign/exponent byte
STRING	Points to the length byte
ARRAY	Points to the element specified

See the sections in the front of this manual for the variable type you interested in to see how variables are stored in memory.

Macintosh: Be sure to use LongIntegers to store addresses.

MSDOS: var=VARPTR(var) returns two values: The address of var and the segment of var in a special variable called VARSEG. See appendix for details.

FORMAT WHILE *expression*
.
.
WEND

DEFINITION This statement is used to terminate a WHILE loop. When expression becomes false the loop will exit at the first statement following the WEND.

EXAMPLE

```
"Get a YES Answer and nothing else!"
INPUT"What is your answer <Y/N>:";A$
WHILE A$ <>"Y"
  INPUT"Please reconsider and say <Y>:";A$
WEND
PRINT"Thank you for seeing things my way..."
:
program continues....
```

RUN

```
What is your answer <Y/N>: N
Please reconsider and say <Y>: Y
Thank you for seeing things my way...
```

```
WHILE X*X <23000
  PRINT X*X,
  X=X+1
WEND
END
```

RUN

```
0             1             4             9             16...
```

REMARK ZBasic will automatically indent all lines two spaces between WHILE and WEND when you use LIST. This makes programs much easier to read.

Also see FOR,NEXT,STEP,DO,UNTIL and the chapters on "Loops" and "Structure" in the front of this manual.

A structure error will occur if a WHILE exists without a matching WEND. To find a missing WEND, LIST the program and track back from the last indent.

WHILE statement

FORMAT **WHILE** *expression*
.
.
WEND

DEFINITION In a WHILE statement, expression is tested for true before the loop is executed and will exit to the statement immediately following the matching WEND when expression becomes false.

EXAMPLE

```
"GET A KEY"
WHILE LEN(Key$)=0
  Key$=INKEY$
WEND
PRINT Key$
END

RUN

<key pressed>
```

```
WHILE X<100
  X=X+1
WEND
PRINT X
END

RUN

100
```

REMARK ZBasic will automatically indent all lines two spaces between the WHILE and WEND when you use LIST. This makes programs much easier to read.

Also see FOR,NEXT,STEP,DO,UNTIL and the chapters on "Loops" and "Structure" in the front of this manual.

A structure error will occur if a WHILE exists without a matching WEND. To find a missing WEND,LIST the program and track back from the last indent.

FORMAT **WIDTH** [**LPRINT**] [=] *byte expression*

DEFINITION Sets the allowable number of characters on a line before generating an automatic linefeed.

The optional LPRINT designates printer width.

If byte expression is set to 0, ZBasic will not send an automatic CR/LF. The range of byte expression is 0 to 255.

EXAMPLE

```
10 X=X+1
20 PRINT X
30 GOTO 10

WIDTH 8
LIST

00010 X=
      X+
      1
00020 PR
      IN
      T
      X
00030 GO
      TO
       1
      0
```

REMARK The default setting for the screen width is zero which disables the auto CR/LF after the limit has been reached.

To return WIDTH to normal, type WIDTH 79 (for 80 column screens) or WIDTH 0. When widths are set, listings are wrapped around nicely for easy reading.

To effect a smaller width, set byte expression to the width desired. To assure valid results for the POS statement and to keep the line position count used by tabs correct, be sure WIDTH is set to the actual screen width minus one.

WRITE# statement

FORMAT **WRITE#***expr*1,{*var*%}|*var*!|*var*#|{*var*$;*stringlength*}[,…]

DEFINITION Writes the contents of string or numeric variables in compressed format to a disk file (or other device) specified by *expr*1. The list may consist of any variable type or types, string or numeric, including arrays, in any order. *Constants may not be used!*

A string variable ***must*** be followed by ;*stringlength* which specifies the number of characters of that string to be written.

If the string is longer than *stringlength*, only those characters in range will be written. If the string is shorter than *stringlength*, the extra characters will be spaces.

READ# is the statement normally used to read back data written with WRITE# and will automatically read back the data written in compressed format.

EXAMPLE

```
REM The four variables below will require 18 bytes for storage
REM A$=4 bytes, A!= 4 bytes, A#=8 bytes, A%=2 bytes
:
A$="TEST": A!="12345.6":A#="12345.67898":A%=20000
:
OPEN"0",1, "DATAFILE", 18     <--- Write a file with a record length of 18
WRITE #1, A$;4, A!, A#, A%
CLOSE#1
:
OPEN"I" ,1,"DATAFILE", 18
READ#1, Z$;4, Z!, Z#, Z%      <---Read in same order and type (see notes)
CLOSE# 1
:
PRINT Z$, Z!, Z#, Z%
END
```

RUN

```
TEST          12345.6       12345.67898          20000
```

REMARK Note: Do not mix variable types when using READ# and WRITE#. READ# and WRITE# store and retrieve numeric data in a compressed format. This saves disk space and speeds program execution.

See the chapter "Files" for more detailed information using random and sequential files. Also see RECORD, LOC,REC,LOF and "Disk Error Trapping".

continued...

WRITE# continued

READ# and WRITE# STRINGS WITH VARIABLE LENGTHS

READ# and WRITE# offer some benefits over PRINT# and INPUT# in that they will read and write strings with ANY embedded ASCII or BINARY characters.

This includes quotes, commas, carriage returns, control codes or any ASCII characters in the range of 0-255.

The following programs demonstrate how to save strings in condensed format, using only the amount of storage required for each string variable.

WRITE#

```
OPEN"O",1,"NAMES"
REM LB$=LENGTH BYTE
DO
  INPUT"Name: "; N$
  INPUT"Age:"; AGE
  LB$=CHR$(LEN(NAME$))
  WRITE#1,LB$;1,N$;ASC(LB$),AGE
UNTIL N$="END"
CLOSE#1
END
```

READ#

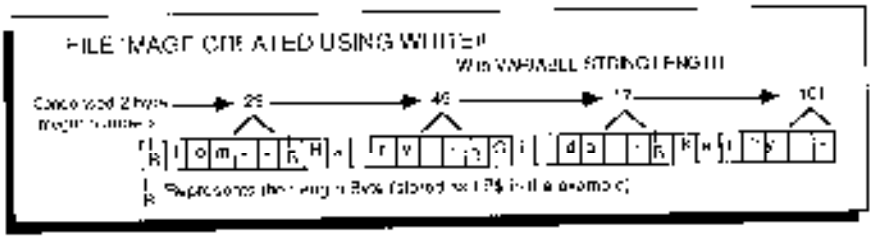
```
OPEN"I",1,"NAMES"
REM LB$=LENGTH BYTE
:
DO
  READ#1, LB$;1, B$;ASC(LB$), AGE
  PRINT N$","AGE
UNTIL N$="END"
CLOSE#1
END
```

The WRITE# program stores a one byte string called LB$ (length byte). The ASCII of this string (a number from 0 to 255) tells us the length of N$.

Notice in line 4 of READ#, that LB$ is read BEFORE N$, thus allowing us to read the length of N$ first. All data in file handling statements is processed IN-ORDER.

This illustration shows how the data is saved to the disk when string data is saved using the variable length method. LB for "Tom" would be 3, LB for "Harry" would be 5, etc.

VARIABLE STRING LENGTH WRITE#



XELSE statement

FORMAT

```
LONGIF expression
.
XELSE
.
ENDIF
```

DEFINITION This statement is used to separate the FALSE from the TRUE section of a LONG IF structure.

The statements following the XELSE will only be executed if the statement following the LONG IF is false.

EXAMPLE

```
LONGIF 10 = 0
  PRINT"TRUE"
XELSE
  PRINT"FALSE"
ENDIF
END

RUN

FALSE
```

REMARK All program lines between the LONG IF and XELSE are indented two characters when using LIST. This makes a program easier to read.

A structure error will occur the XELSE does not have a matching LONG IF.

FORMAT *expression*$_1$ **XOR** *expression*$_2$

DEFINITION Provides a means of doing a logical EXCLUSIVE OR on two expressions for IF-THEN testing and BINARY operations.

This operator will return true if one condition is true and one condition is false. False will be returned if both conditions are true or both conditions are false.

EXAMPLE

```
A$="Hello"
IF A$="Hello" XOR A$="Goodbye" PRINT "YES"
IF A$="Hello" XOR A$="Hello" PRINT "YES"

RUN

YES
```

REMARK

XOR TRUTH TABLES

condition XOR condition TRUE(-1) if only one condition is TRUE, else FALSE(0)

XOR

```
1 XOR 1 = 0
0 XOR 1 = 1
1 XOR 0 = 1
0 XOR 0 = 0
```

BOOLEAN "16 BIT" LOGIC

```
        00000001          10000101
XOR     00001111    XOR   10000111
=       00001110    =     00000010
```

```
FALSE XOR FALSE = FALSE
TRUE   XOR FALSE = TRUE
FALSE XOR TRUE  = TRUE
TRUE  XOR TRUE  = FALSE
```