ADAM™ SmartBASIC™

REFERENCE SECTION

TABLE OF CONTENTS

ABS	A-1
AND	A-2
ARROW KEYS	A-3
ASTERISK	A-4
BACKSPACE	A-5
CATALOG	A-6
CHR$	A-7
CLEAR	A-8
CLOSE	A-9
COLON	A-10
COLOR=	A-11
COMMA	A-12
CONT	A-13
CONTROL KEYS	A-14
DASH	A-15
DATA/READ	A-16
DEF FN	A-17
DEL	A-18
DELETE	A-19
DIM	A-20-22
DOLLAR SIGN	A-23
END	A-24
ERRNUM	A-25
FLASH	A-26
FOR...STEP/NEXT	A-27
FN	A-28
GET	A-29
GOSUB	A-30
GOTO	A-31
GR	A-32
HCOLOR=	A-33-34
HGR	A-35
HGR2	A-36
HLIN	A-37
HOME	A-38
HOME(key)	A-39
HPLOT	A-40
HTAB	A-41
IF...GOTO	A-42
IF...THEN	A-43
INPUT	A-44
INT	A-45
INVERSE	A-46

LEFT$ A-47
LEN A-48
LET A-49
LIST A-50-51
LOAD A-52
LOCK A-53
MID$ A-54
MON A-55
NEW A-56
NOMON A-57
NORMAL A-58
NOT A-59
ONERR GOTO/CLRERR A-60
ON...GOSUB A-61
ON...GOTO A-62
OPEN A-63
OR A-64
PARENS A-65
PDL A-66
PLOT A-67
PLUS SIGN A-68
POS A-69
PRINT A-70
PR# A-71
QUOTATION MARKS A-72
RELATIVE OPERATORS A-73
REM A-74
RENAME A-75
RESTORE A-76
RESUME A-77
RETURN A-78
RETURN(key) A-79
RIGHT$ A-80
RND A-81
RUN A-82
SAVE A-83
SCRN A-84
SEMICOLON A-85
SGN A-86
SLASH A-87
SPC A-88
SPEED= A-89
SQR A-90
STOP A-91
STR$ A-92
TAB A-93
TEXT A-94
TRACE/NOTRACE A-95-96

UNLOCK	A- 97
VAL	A- 98
VLIN	A- 99
VPOS	A-100
VTAB	A-101
WRITE/READ	A-102

For more advanced commands, refer to the ADVANCED REFERENCE SECTION, which follows this section.

WARNING: If you try to use BASIC commands that are not documented on the following pages, you may get unpredictable results.

COMMANDS are used in the immediate mode.

STATEMENTS are used in programs.

OS COMMANDS can only be used in programs by printing them preceded by a CONTROL-D.

ABS

ABS is a special function which strips the minus signs from negative numbers and leaves other numbers unchanged. In other words, it returns the absolute value of the given number. A number's absolute value is its value without a plus or a minus sign.

Test Program:

```
10 X=35
20 PRINT "ABS WORKED IF ";
30 PRINT ABS(-436.38);" ";
40 PRINT ABS(-.63245);" ";
50 PRINT ABS(-X)
60 PRINT "ARE ALL PRINTED AS POSITIVE
NUMBERS."
70 END
```

Sample Run:

```
ABS WORKED IF 436.38 .63245 35
ARE ALL PRINTED AS POSITIVE NUMBERS.
```

AND

AND is a logical operator. This means that it is used to combine and compare two logical values (true/false) to yield a logical result. AND is true (value=1) if both expressions are true. AND is false (value=0) if one or both expressions are false.

Test Program:

```
10 A=6
20 B=4
30 IF A=6 AND B=4 THEN 60
40 PRINT "'AND' BLEW IT"
50 GOTO 70
60 PRINT "'AND' WORKED!"
70 END
```

Sample Run:

```
'AND' WORKED!
```

Function

ARROW KEYS

UP ARROW - will make the cursor move up one line, but will not change the buffer or force a scroll.

DOWN ARROW - will move the cursor down one line, but will not change the buffer or force a scroll.

RIGHT ARROW - will copy the character the cursor is currently under into the input buffer and move the cursor to the right.

LEFT ARROW - will take the last character out of the buffer and move the cursor to the left.

If you hold down any of these keys, the movement of the cursor left, right, up, or down will be repeated.

Test Program:

NONE

Sample Run:

NONE

Operator

ASTERISK
(*)

Asterisks are used in mathematics to indicate multiplication.

Test Program:

```
10 PRINT 10+(6*2)
```

Sample Run:

```
22
```

Function

BACKSPACE
(key)

The BACKSPACE key does the same thing as the left arrow key. See Arrow Keys.

Test Program:

NONE

Sample Program:

NONE

CATALOG

CATALOG prints out an index of all file names in your digital data pack directory to your screen or your printer. This enables you to see what's stored on your digital data pack.

Test Program:

```
]CATALOG,D1
```

The "D1" is optional, and tells ADAM to CATALOG using the first (left-most) digital data pack. If you wish to use the second digital data pack, substitute "D2".

Sample Run:

```
This will print out a listing of the
names of all files stored on your
digital data pack.

VOLUME: HELLO
A   1   FACE
A   1   DAMAGE
a   1   DAMAGE

      250 Blocks Free
```

The capital A's beneath VOLUME tell you that these files are main files. The small a signifies a back up file. The 1's tell you how many blocks of memory your files have taken up. Each block holds 1K or 1024 characters. The final message lets you know how much space you have left (measured in blocks) in memory.

Function

CHR$

This function takes a numeric ASCII decimal code and matches it up with the character or the value of the string variable it represents.

Test Program:

```
10 REM THIS IS A 'CHR$' TEST PROGRAM
20 PRINT CHR$(75)
30 END
```

Sample Run:

```
K
```

75 represents the letter K in ASCII code.

NOTE: CHR$(4) = control D
CHR$(7) = bell
CHR$(10)= linefeed
CHR$(13)= carriage return

See the Compendium (Appendix C) at the back of this manual for a complete ASCII character code table.

CLEAR

CLEAR is used to free up memory space by setting all numeric variables to zero and erasing all string variable data -- so be careful when you use it!

Test Program:

```
10 A=705
20 A$="TEST STRING"
30 PRINT "BEFORE 'CLEAR' A=";A
40 PRINT "STRING VARIABLE A$=";A$
50 CLEAR
60 PRINT "AFTER 'CLEAR' A=";A
70 PRINT "STRING VARIABLE A$=";A$
```

Sample Run:

```
BEFORE 'CLEAR' A=705
STRING VARIABLE A$=TEST STRING
AFTER 'CLEAR' A=0
STRING VARIABLE A$=
```

CLOSE

You must CLOSE a file when you are finished working on it. If you wish to close a file, use the syntax:

CLOSE<filename>,[D#]

If you fail to CLOSE a file, ADAM will allocate the rest of the space on the digital data pack for the file. To fix this problem, type CLOSE<filename> in the immediate mode.

Note: If you DELETE a file that has not been CLOSED ADAM may get confused as to how much space is left on your digital data pack. A remedy is to copy all your files to a new data pack and INIT the old one.

Test Program:

```
(You've stored a file named "FACE")
10 D$=CHR$(4)
20 PRINT D$;"OPEN FACE,D1"
30 PRINT D$;"CLOSE FACE"
```

Sample Run:

```
The file called FACE is now closed.
```

COLON(:)

Colons are used to place several statements on the same line.

Test Program:

```
10 A=5: B=7: PRINT A+B
```

Sample Run:

```
12
```

Command
Statement

COLOR=

COLOR= sets the display plotting color for low resolution graphics mode. The color code may also be a numeric variable. See the list of codes and colors below:

CODE	COLOR
0	black
1	magenta (purple)
2	dark blue
3	dark red
4	dark green
5	grey
6	medium green
7	light blue
8	dark yellow (orange)
9	medium red
10	grey
11	light red
12	light green
13	light yellow
14	cyan (aqua)
15	white

Test Program:

```
10 GR
20 COLOR=14
30 HLIN 10,35 AT 20
```

Sample Run:

A line of cyan (aqua) from column 10 to column 35 has been drawn in row 20.

COMMA
(,)

A comma is usually used in a PRINT statement. The comma serves to make the information which follows it print in the next zone of your output device (screen or printer). Commas are used to separate variables in an INPUT statement.

Test Program:

```
10 PRINT 1,2
20 PRINT 3,4
```

Sample Run:

```
1             2
3             4
```

The numbers are grouped into two columns, one column per zone.

Test Program:

```
10 INPUT " NAME 2 COLORS ";A$,B$
20 PRINT A$,B$
```

Sample Run:

```
NAME 2 COLORS  RED,BLUE
RED            BLUE
```

CONT

This command is useful in restarting a program which has been halted prematurely by a CONTROL-C, STOP, or END statement. The difference between CONT and RUN is that RUN will start the whole program over again from the beginning, whereas CONT will continue the program from where it was stopped, as long as you don't make a change to the program before typing in CONT.

Test Program:

```
10 PRINT "I HAVE A MICROCHIP ON MY
SHOULDER"
20 PRINT "ENTER THE 'CONT' COMMAND"
30 STOP
40 PRINT "AND A BYTE ON MY LEG"
50 END
```

Sample Run:

```
I HAVE A MICROCHIP ON MY SHOULDER
ENTER THE 'CONT' COMMAND
?Break In 30
CONT
AND A BYTE ON MY LEG
```

CONTROL-C - Breaks into a program which is in the process of running onscreen. Allows you to stop or abort a running program. Pick up the run where you broke it by typing in CONT. Start it over again, at the beginning, by typing RUN. If your program is waiting for INPUT, the CONTROL-C will not take effect until you press RETURN.

CONTROL-D - When PRINTed allows the use of OS commands from within a program.

CONTROL-H - Does the same as BACKSPACE key.

CONTROL-L - Clears the screen. No data in the buffer is changed.

CONTROL-M - Does the same as RETURN key.

CONTROL-N - Allows you to insert a letter, word, phrase, or paragraph at whatever point you designate.

CONTROL-O - Acts as a delete key. Allows you to erase any information you wish to dispose of.

CONTROL-P - Will output to printer whatever is on your screen. Be sure you have paper in your printer before pressing this! The printer cannot be stopped until it has printed out the whole screen!

CONTROL-S - Freezes your screen without breaking into your program. That is, it temporarily stops the output on your screen until you press CONTROL-S to restart it.

CONTROL-X - Will "undo" an input line. But instead of erasing it, a backslash will appear at the end of the line to be disregarded.

CONTROL-← Moves the cursor left and has no other effect.

CONTROL-→ Moves the cursor right and has no other effect.

Operator

DASH (-)

The dash indicates subtraction.

Test Program:

```
10 PRINT 5-3
```

Sample Run:

```
 2
```

DATA/READ

The DATA statement contains a list of items or facts which will be needed in your program. It's a sort of "reference section" or information pool. Don't use DATA without READ. Every item in your DATA statement must be separated with a comma. If an item must contain a comma, enclose the entire item in quotes.

READ reads from DATA and assigns each item to a variable. DATA statements can be located anywhere in the program.

Test Program:

```
10 DATA MEAT,POTATOES,LETTUCE,TOMATOES,
BUTTER,CHEESE,ONIONS,PEAS
20 READ A$
30 PRINT A$
40 GOTO 20
```

Sample Run:

```
MEAT
POTATOES
LETTUCE
TOMATOES
BUTTER
CHEESE
ONIONS
PEAS
?Out Of Data Error In 20
```

DEF FN

The DEF FN (define function) statement lets you create your own formulas; which can really save you time. Instead of writing the same formula over and over again, just define it once as a function, name it, then just use that name to call it up when you need it. But take care not to begin the names of two different functions with the same two letters (e.g. DULL and DUMMY). This would end up referring to the same function, since both names begin with the same two characters. The variable used in DEF FN is a dummy. It must be a real (floating point) variable. String or interger variables are not allowed. You may change the name of the variable when you use it later in FN.

Test Program:

```
10 PRINT "THIS CONVERTS FAHRENHEIT TO
CELSIUS"
20 T=212 : A = 32
30 DEF FN FTC(T)=(T-32)*5/9
40 PRINT FN FTC(T)
50 PRINT FN FTC(A)
```

Sample Run:

```
THIS CONVERTS FAHRENHEIT TO CELSIUS
100
```

DEL

The DEL command may be used to erase a single line, a sequence of consecutive lines, or an entire program.

Test Program:

```
10 PRINT "TEN"
20 PRINT "TWENTY"
30 PRINT "THIRTY"
40 PRINT "FORTY"

DEL 15,40
or  DEL 20,40
or  DEL 20,999
```

Sample Run:

```
TEN
```

DELETE

The DELETE command may be used to erase unlocked files.

NOTE: Do not DELETE OPEN files. (see CLOSE.)

Test Program:

```
DELETE Joe
```

Sample Run:

```
The data file "Joe" is now erased from
the digital data pack unless the file
was locked.
```

DIM

This statement establishes the number of elements in a numeric or string array. You may have a one- two - three or more dimensional array. The arrays are numbered beginning at 0 and the computer automatically dimensions for 10 (eg. A(0)-A(10)). DIM is useful in composing charts and tables. The maximum number of elements depends on the amount of available memory. The dimension number may be a variable.

Test Program #1--One-Dimension Array

```
10 DIM A(5)
20 PRINT "THIS IS A SINGLE-DIMENSION"
30 PRINT "NUMERIC ARRAY."
40 FOR X=1 TO 5
50 A(X)=X
60 PRINT A(X);" ";
70 NEXT X
80 END
```

Sample Run:

```
THIS IS A SINGLE-DIMENSION
NUMERIC ARRAY.
1 2 3 4 5
```

Test Program #2--Two-Dimension Array

```
10 DIM A(2,4)
20 PRINT "THIS IS A TWO-DIMENSION"
30 PRINT "NUMERIC ARRAY."
40 FOR I=1 TO 2
50 FOR J=1 TO 4
60 A(I,J)=I
70 NEXT J
80 NEXT I
```

```
90 FOR I=1 TO 2
100 FOR J=1 TO 4
110 PRINT A(I,J);" ";
120 NEXT J
130 PRINT
140 NEXT I
150 END
```

Sample Run:

```
THIS IS A TWO-DIMENSION
NUMERIC ARRAY

1    1    1    1
2    2    2    2
```

Test Program #3-- Three-Dimension Array

```
10 DIM A(4,2,2)
20 PRINT "THIS IS A THREE-DIMENSION"
30 PRINT "NUMERIC ARRAY."
40 FOR K=1 TO 2
50 FOR I=1 TO 4
60 FOR J=1 TO 2
70 A(I,J,K)=I
80 NEXT J
90 NEXT I
100 NEXT K
110 FOR K=1 TO 2
120 FOR I=1 TO 4
130 FOR J=1 TO 2
140 PRINT A(I,J,K),
```

DIM

```
150 NEXT J
155 PRINT
160 NEXT I
170 PRINT
180 NEXT K
190 END
```

Sample Run:

```
 1    1

 2    2

 3    3

 4    4


 1    1

 2    2

 3    3

 4    4
```

Operator

DOLLAR
SIGN ($)

This is the symbol which should follow a letter or a letter-number combination that you want to be designated as a string variable.

NOTE: L$ is usually pronounced "L String."

Test Program:

```
10 L$="THEY'RE LAUGHING AT YOU"
20 PRINT L$
30 PRINT L$
40 PRINT L$
```

Sample Run:

```
THEY'RE LAUGHING AT YOU
THEY'RE LAUGHING AT YOU
THEY'RE LAUGHING AT YOU
```

END

END terminates program execution. It's use is optional if it is placed at the highest line number.

Test Program:

```
10 PRINT "THIS IS THE FIRST VERSE."
20 END
30 PRINT "THIS IS THE SECOND VERSE."
40 END
```

Sample Run:

```
THIS IS THE FIRST VERSE.
```

Function

ERRNUM

ERRNUM gives you the error code when used with ONERR GOTO.

Test Program:

```
10 ONERR GOTO 60
20 PRINT 10/0
60 PRINT ERRNUM(0)
```

Sample Run:

```
133
```

FLASH

After FLASH is activated, any character printed on your screen will flash rapidly between INVERSE and NORMAL.

Test Program:

```
10 TEXT:HOME
20 FLASH
30 PRINT "WARNING"
40 NORMAL
50 PRINT "HA,HA,ONLY FOOLING"
```

Sample Run:

```
WARNING flashes.
HA,HA,ONLY FOOLING prints normally.
```

FOR and NEXT are used to create finite loops. Whenever you write a program containing FOR, you must use NEXT at the end of the loop. The FOR...NEXT statement acts as a counter. The STEP statement is only necessary if you want to count by increments other than one. For backward steps use the minus sign. The variable index is optional with the NEXT, even with nested loops.

Test Program:

```
10 FOR X=1 TO 100
20 PRINT X
30 NEXT X
```

Sample Run:

```
1
2
3
4
5
etc.
100
```

Test Program:

```
10 FOR X=100 TO 0 STEP -5
20 PRINT X
30 NEXT X
```

Sample Run:

```
100
 95
 90
etc.
 10
  5
  0
```

FN is used to calculate formulas you use many times. First you must use DEF FN to tell ADAM what the formula is and give it a name. In your programs, substitute FN<name><variable> for your formula.

Test Program:

See DEF FN

Sample Run:

See DEF FN

GET

GET is used to get information from the operator of the computer. The program will pause until a key is pressed. It differs from INPUT in that there is no option to prompt, only one keypress is read, the results of the keypress are not displayed on the screen, the cursor is not moved, and it is not necessary to press RETURN. GET can be used with either numeric or string variables.

Test Program:

```
10 PRINT "PRESS THE 'L' KEY TO
CONTINUE:";
20 GET A$
30 IF A$<>"L" THEN 20
40 PRINT
50 PRINT "THANK YOU"
```

Sample Run:

```
PRESS THE 'L' KEY TO CONTINUE
---computer waits for user to press
key---
---computer keeps cycling until correct
key is pressed---
THANK YOU
```

GOSUB

GOSUB is used to extend or "branch" out of the main part of a program into a subroutine. GOSUB must be followed by a line number telling the computer where the first line of the subroutine is located. RETURN must be used at the end of the subroutine in order to return you to the main part of your program.

Test Program:

```
10 PRINT "ONE ";
20 GOSUB 100
30 PRINT "FIVE";
90 END
100 PRINT "TWO ";
110 GOSUB 200
120 PRINT "FOUR ";
130 RETURN
190 END
200 PRINT "THREE ";
210 RETURN
```

Sample Run:

```
ONE TWO THREE FOUR FIVE
```

GOTO

GOTO makes ADAM skip to a line number that you designate.

Test Program:

```
10 PRINT "CAT"
20 PRINT "DOG"
30 GOTO 10
```

Sample Run:

```
CAT
DOG
CAT
DOG
CAT
DOG
etc.

Use CONTROL-C to break out of this loop.
```

GR

GR sends your screen from the TEXT mode into the low resolution graphics mode. The graphics mode provides you with 40 columns and 40 rows for graphics with space for 4 lines of text at the bottom of the screen. For low resolution graphics the screen is numbered 0 to 39 across and 0 to 39 down starting in the upper left corner. GR always sets the low resolution color to black.

Test Program:

```
GR
```

Sample Run:

```
The screen has gone black.
The text cursor is at the bottom left of
the screen.
```

HCOLOR=

HCOLOR= selects the drawing color used only for plotting high resolution graphics. The HCOLOR= code number may be a numeric variable. You have 16 colors from which to choose:

CODE	COLOR
0	black
1	green
2	dark red
3	white
4	black
5	medium red
6	medium blue
7	white
8	dark yellow (orange)
9	dark blue
10	grey
11	light red
12	dark green
13	light yellow
14	cyan (agua)
15	magenta (purple)

Test Program:

```
10 HGR
20 HCOLOR=3
30 HPLOT 140,10 TO 250,140 TO 30,140 TO
140,10
40 HCOLOR=2
45 HPLOT 140,30 TO 250,158 TO 30,158 TO
140,30
47 GET q$
50 HPLOT 140,30 TO 250,158
51 GOSUB 100
52 HPLOT 250,158 to 30,158
53 GOSUB 100
54 HPLOT 30,158 TO 140,30
55 GOSUB 100
99 END
100 FOR t = 1 TO 500:NEXT t
101 RETURN
```

HCOLOR=

Sample Run:

This program draws two triangles, one in white and one in dark red.

Test Program #2--Colors sometimes "bleed" when differently colored lines are plotted close together. This test program illustrates video "bleeding".

```
10 HGR
20 HCOLOR=2
30 HPLOT 40,10 TO 40,150
40 HCOLOR=7
50 HPLOT 30,10 TO 50,150
```

NOTE: If an image is drawn in one color, redrawn in the background color, then drawn again at a nearby location, it will appear to move. This is the basic method of animation.

NOTE: Due to the way color video works, to get true colors you must plot at least two points side by side, ie HPLOT 20, 10 TO 40, 50: HPLOT 21, 10 TO 41, 50.

HGR

The HGR statement sends you into the high resolution graphics screen. This screen provides you with a 256 wide x 159 high grid of points on which graphics images can be seen, with space for four lines of text at the bottom area of the screen.

The screen is numbered starting at the upper left hand corner. Columns 0 to 255 go across and rows 0 to 191 run down. Row values between 159 and 191 will not show. On some screens, you may not be able to see columns 0 to 3.

Test Program:

```
10 HGR
20 HCOLOR=3
30 HPLOT 10,10 TO 100,10
40 HPLOT TO 100,100
50 HPLOT TO 10,10
```

Sample Run:

This program draws a white triangle.

HGR2

The only difference between the HGR and HGR2 statements is that with HGR2, you have full screen graphics, with no space at the bottom of the screen for text. Therefore, you have a 0-255 x 0-191 grid of points available to plot with. On some screens you may not be able to see columns 0 to 3.

Test Program:

```
10 HGR2:HCOLOR=6
20 HPLOT 100,100 TO 190,100 TO 190,190
30 HPLOT 190,190 TO 100,190 TO 100,100
```

Sample Run:

This draws a medium blue square at the bottom of the screen.

HLIN draws a horizontal line, in the designated display color, at the points indicated. This is a low resolution graphics statement.

Test Program:

```
10 GR
20 COLOR=14
30 HLIN 10,35 AT 20
```

Sample Run:

A horizontal line in row 20 has been drawn from column 10 to column 35. The color is cyan (aqua).

HOME

HOME clears the screen (text window) and sends the cursor to the upper left corner (beginning of line 1). If you're in the graphics screen, the lower 4 text lines are cleared and the cursor is sent to the beginning of line 21.

Test Program:

```
10 FOR X=1 TO 12
20 PRINT "ERASE THIS"
30 NEXT X
40 HOME
50 PRINT "DO YOU WANT ANYTHING ELSE
ERASED?"
60 END
```

Sample Run:

```
DO YOU WANT ANYTHING ELSE ERASED?
```

Function

HOME
(key)

The HOME key will move the cursor to the upper left corner of the screen. In graphics mode, the cursor moves to the upper left corner of the text window.

NOTE: HOME and HOME key do different things in graphics mode. The HOME key moves the cursor to the upper left corner of the window. HOME drops the cursor down one line. (See HOME.)

Test Program:

NONE

Sample Run:

NONE

HPLOT

HPLOT plots lines and points on the high resolution graphics screen in the current display color. The first expression in the HPLOT pair refers to column (0 to 255) location. This is followed by a comma, which is followed by a row location. Row locations are numbered 0 to 158 for mixed text and graphics and 0 to 191 for full screen graphics. To plot a line, specify two points separated by the word TO.

Test Program:

```
10 HGR
20 HCOLOR=3
30 HPLOT 50,60 TO 80,100
```

Sample Run:

This program draws a line starting at column 50, row 60 and ending at column 100, row 80.

Test Program:

```
10 HGR
20 HCOLOR=1
30 HPLOT 50,60 to 80,100
```

Sample Run:

This program draws a green line starting at column 50, row 60 and ending at column 100, row 80.

HTAB will move the cursor left or right on a horizontal line without moving or affecting any text. You must specify the column number for HTAB. Columns are numbered from 1 to 31, left to right. The number used with HTAB can be a variable.

Test Program:

```
5 HTAB 1:PRINT "X";
10 HTAB 15:PRINT "X";
20 HTAB 20:PRINT "X";
30 HTAB 30:PRINT "X";
40 HTAB 31:PRINT "X";
```

Sample Run:

This will cause your cursor to jump across the screen, stopping to print an X at columns 1,15,20,30, and 31.

IF...GOTO

IF...GOTO indicates a branching statement which jumps you to a different area of your program according to whether or not certain conditions within the statement are met.

Test Program:

```
10 A=20/4
20 IF A=5 GOTO 50
30 PRINT "STAND IN THE CORNER,
'IF-GOTO'."
40 GOTO 60
50 PRINT "'IF-GOTO' WORKS!"
60 END
```

Sample Run:

```
'IF-GOTO' WORKS!
```

IF...THEN

IF...THEN is a decision statement. Everything between the IF and THEN is tested to see if it is "true." If a "true" condition exists, then the program tries to execute the instructions following the word THEN. If the test result is "false" then the program jumps to the next line.

IF...THEN can also be used for branching in the same way as IF...GOTO.

Test Program:

```
10 INPUT "ARE YOU MALE OR FEMALE? ";A$
20 IF A$="MALE" THEN PRINT "SO IS
FRANKENSTEIN": END
30 IF A$="FEMALE" THEN PRINT "SO IS MARY
POPPINS": END
```

Sample Run:

```
ARE YOU MALE OR FEMALE?  MALE
SO IS FRANKENSTEIN
```

INPUT

INPUT is a statement which allows the operator to assign values to variables from keyboard to memory.

Test Program:

```
10 INPUT "WHAT IS YOUR NAME? ";N$
20 PRINT "YOU HAVE A NICE NAME, ";N$
```

Sample Run:

```
WHAT IS YOUR NAME? ALBERT
YOU HAVE A NICE NAME, ALBERT
```

The INTeger function is used to round numbers to their whole number (or integer) value. In other words, anything to the right of the decimal point is discarded, no matter what its value.
Therefore, when the computer "rounds" your number, it rounds it down...except if you're dealing with a negative number. Then it will round to the next smaller integer. For example, -4.65 becomes -5.
Rounding with INT does not necessarily provide you with the nearest integer unless you first add .5 before applying the function. For example, INT(AGE+.5)

Test Program:

```
10 A=45.67: B=-3.1
20 PRINT INT(A);" ";INT(B);" ";
30 PRINT INT(A+B);" ";INT(6*B)
```

Sample Run:

```
 45  -4  42  -19
```

INVERSE

Take a look at your screen. Anything you type on it, under normal circumstances, appears as a pattern of white dots on a black background. But if you activate INVERSE, your screen will switch to a pattern of black dots on a white backgound.

Test Program:

```
10 TEXT:HOME
20 PRINT "NORMAL TEXT"
30 INVERSE
40 PRINT "INVERSE TEXT"
50 NORMAL
60 PRINT "NORMAL AGAIN"
```

Sample Run:

```
NORMAL TEXT              white on black

INVERSE TEXT             black on white

NORMAL AGAIN             white on black
```

The LEFT$ function is used to extract a specified number of digits or characters from strings. Be sure to remember to count spaces.

Test Program:

```
10 A$="THEODORE"
20 B$=LEFT$("TESTING",4)
30 PRINT LEFT$(A$,3);" 'LEFT$' FUNCTION
PASSED THE ";B$
40 END
```

Sample Run:

```
THE 'LEFT$' FUNCTION PASSED THE TEST
```

LEN counts the number of characters in a string. LEN stands for LENgth. Be sure to remember to count spaces. To use LEN to determine the number of digits in a number, if A is a positive, whole number, you would say, A=LEN(STR$(A))

Test Program:

```
10 A$="U.S. GRANT": B=LEN(A$)
20 C=1822: D$=STR$(C): E$="1882": F=-27
30 PRINT LEN(A$);" ";B;" ";LEN(D$);
" ";LEN(E$);" ";LEN(STR$(F))
```

Sample Run:

```
 10   10   4   4   3
```

LET

The statement LET assigns values to variables, both numeric and string. Its use is optional.

Test Program:

```
10 LET X=20
20 PRINT "AFTER 19 COMES ";X
30 END
```

Sample Run:

```
AFTER 19 COMES 20
```

LIST

LIST is used to display your program in its entirety in correct numerical (line number) order. LIST may also be used to display individual lines, as well as particular sections of your program. You can temporarily halt the listing of a program by pressing CONTROL-S. To resume, press CONTROL-S again. To LIST a program to the printer, type PR#1:LIST:PR#0.

Test Program:

```
2 PRINT "LOOKING"
1 PRINT "HERE'S"
4 PRINT "YOU,"
3 PRINT "AT"
5 PRINT "KID"

LIST
1 PRINT "HERE'S"
2 PRINT "LOOKING"
3 PRINT "AT"
4 PRINT "YOU,"
5 PRINT "KID"
```

Sample Run:

```
HERE'S
LOOKING
AT
YOU,
KID
```

Test Program:

```
List 2
```

Sample Run:

```
2 PRINT "LOOKING"
```

Command
Statement

LIST

Test Program:

```
List 2,4
```

Sample Run:

```
2 PRINT "LOOKING"
3 PRINT "AT"
4 PRINT "YOU"
```

LOAD

The LOAD command retrieves a program, it does not execute it. It merely reads a copy into your computer's memory. Any program already in memory is erased and replaced by the new program. Once this is done, the program is available to be executed with a RUN command.

NOTE: Upper and lower case letters are not the same in file names. Be sure you LOAD the program the same way you SAVE it.

Test Program:

```
10 PRINT "THIS PROGRAM IS NOW LOADED"
20 END
SAVE TEST
NEW
LOAD TEST
RUN
```

Sample Run:

```
THIS PROGRAM IS NOW LOADED
```

LOCK

LOCK will protect your file from being deleted accidently.

Test Program:

```
10 PRINT "MY DAD"
20 PRINT "IS SAD"
SAVE DAD
LOCK DAD
]CATALOG
```

Sample Run:

If you try to delete DAD, the message that the file is locked will appear on the screen. CATALOG will warn you of a locked file by displaying an * by the file name.

```
VOLUME: HELLO

*A       DAD
```

MID$

The MID$ function is used to manipulate strings. It isolates and extracts a substring from any specified location from within a string. For example, PRINT MID$("HI THERE",4,5) prints THERE, because the 4 tells it to begin at the fourth position from the left of the character string which puts you at the "T" in "THERE". The five says to print the next 5 characters. If the last number is omitted, all the characters to the end of the string will be printed.

Test Program:

```
10 A$="YELLOW SUBMARINE"
20 C$=MID$(A$,4)
30 B=4
40 PRINT C$, MID$(A$,B,7)
50 PRINT A$
```

Sample Run:

```
LOW SUBMARINE   LOW SUB
YELLOW SUBMARINE
```

MON, or **monitor**, allows you to monitor information entering and leaving your digital data pack, using four parameters. C, I, O and L.

C = monitors commands to the digital data pack

I = causes input from the digital data pack to be displayed.

O = causes output to the digital data pack to be displayed.

L = monitors input from the digital data pack when LOADing a SmartBASIC file.

For example:
MON L
LOAD <filename>
will show you each line as it comes into memory.

Syntax for MON is:
MON C,I,O,L

C,I,O, and L may be used in any combination or order, but at least one must be present or the command will be ignored. To disable MON, use NOMON with the appropriate parameters.

Test Program:

NONE

Sample Run:

NONE

NEW

The NEW command will delete your current program from memory and clear all values assigned to numeric and string variables.

Test Program:

```
10 PRINT "TEN"
20 A=20: PRINT 20
NEW
30 PRINT "TWENTY"
40 PRINT A
```

Sample Run:

```
TWENTY
0
```

NOMON

NOMON or no monitor, cancels the MON command. NOMON uses the same parameters as MON (C,I,O,L), but NOMON can specify which parameters it wants to stop monitoring.

Test Program:

```
MON I,O,C,L
NOMON O
```

Sample Run:

```
NOMON O cancelled the monitoring of
output to the digital data pack.  The
monitoring of I,C, & L are left
untouched.
```

NORMAL

NORMAL is used to cancel the INVERSE and FLASH text display modes and return to white on black character display. Normal is the usual mode of text display.

Test Program:

```
10 TEXT:HOME
20 PRINT "NORMAL"
30 INVERSE:PRINT
40 PRINT "INVERSE"
50 NORMAL:PRINT
60 PRINT "NORMAL AGAIN"
```

Sample Run:

```
NORMAL              white on black
INVERSE             black on white
NORMAL AGAIN        white on black
```

NOT

NOT is a logical operator which is used in comparison statements to reverse the condition of your premise. In other words, NOT is true (value=1) if the argued expression is completely false.

Test Program:

```
10 X=7
20 IF NOT (X<4) THEN 50
30 PRINT "'NOT' DIDN'T WORK"
40 GOTO 60
50 PRINT "'NOT' WORKED!"
60 END
```

Sample Run:

```
'NOT' WORKED!
```

ONERR GOTO overrides the computer's normal error-handling procedures and instead, when an error is encountered, sends control out to a special error subroutine that you must write. ONERR GOTO must appear in the program before an error in execution is committed. This is referred to as **error-trapping.** Error-trapping may be disabled at any time with the statement CLRERR. You return from the error subroutine with the statement RESUME.

Refer to the Compendium at the back of this book for a complete list of error codes.

Test Program--counts number of items in a DATA statement

```
10 ONERR GOTO 60
20 N=0
30 READ A
40 N=N+1
50 GOTO 30
60 PRINT N
70 END
80 DATA 20,864,218,10,299
```

Sample Run:

```
 5
```

Without line 10, the error message "OUT OF DATA ERROR IN 30 would have been printed and the number 5 would not have been displayed. The error-handling subroutine consists of lines 60 and 70.

ON...GOSUB

The ON...GOSUB statement instructs the computer to branch out into subroutines depending on the value of a variable or numeric expression. The locations of the subroutines are listed by line number following the ON...GOSUB statement. The variable or expression used with ON...GOSUB must be positive. If the variable or expression is zero or greater than the number of branches listed, the ON...GOSUB is ignored.

Test Program:

```
10 PRINT "ENTER THE NUMBER 1,2,3,4,OR 5";
15 INPUT "";X
20 IF X<1 OR X>5 THEN PRINT "OUT OF
RANGE, RETYPE: ";:GOTO 15
30 ON X GOSUB 100,130,140,140,100
40 GOTO 10
100 PRINT "VALUE IS 1 OR 5 "
125 RETURN
130 PRINT "VALUE IS 2 "
135 RETURN
140 PRINT "VALUE IS 3 OR 4 "
145 RETURN
```

Sample Run:

```
ENTER THE NUMBER 1,2,3,4, OR 5 5
VALUE IS 1 OR 5
ENTER THE NUMBER 1,2,3,4, OR 5 2
VALUE IS 2
etc.
```

The ON...GOTO statement is applied the same way as ON...GOSUB.
Refer to ON...GOSUB and GOTO for further information.

Test Program:

```
NONE
```

Sample Run:

```
NONE
```

OPEN

OPEN does what its name implies in order to give you access to a file. When you OPEN a file, certain information is provided to the computer as a result of typing in the command. . . information concerning whether or not the file is on the digital data pack, and if so, where. You must use OPEN within a program. It must be in a PRINT statement and be preceded by CONTROL-D. In SmartBASIC, you use CHR$(4) to print a CONTROL-D. A maximum of 2 files may be OPEN at once. However, under some circumstances, a user may only be able to OPEN one file at a time. An error message will appear if a user tries to OPEN more files than are permitted at the time.
Syntax for OPEN is:

OPEN <filename>,L<length>

the length specification is needed only for use with random files.

NOTE: See the Compendium (Appendix C) for more information on sequential and random text files.

Test Program:

```
        (You've stored a file named "FACE")
        10 D$=CHR$(4)
        20 PRINT D$;"OPEN FACE"
        30 PRINT D$;"CLOSE FACE"
]CATALOG
```

Sample Run:

```
        You have just accessed your file.
```

OR

OR is a logical operator. Like AND, OR has a value of 1 for a true statement and 0 for a false one. Unlike AND, OR is true if either or both of the original expressions are true.

Test Program:

```
10 INPUT "WHAT'S YOUR NAME?";N$
20 IF N$="AL" OR N$="LYNN" THEN
PRINT "HELLO ";N$:END
30 PRINT "UNAUTHORIZED!"
```

Sample Run:

```
WHAT'S YOUR NAME? LYNN
HELLO LYNN
```

Operator

PARENS
( )

Parentheses are used in arithmetic problems to indicate order of operations. Anything in parentheses is calculated first. Parentheses override the "multiplication and division" first rule. Parentheses are also used with all functions and arrays.

Test Program:

```
10 PRINT (10*(5-3))/2
```

Sample Run:

```
 10
```

PDL refers to "paddles" or hand control units. These control units are not only good for game play, but for precise positioning of your cursor in graphics. With ADAM's PDL function it's like having a joystick, paddle, and numeric keypad in a single unit.

For the front controller, #1, the values are as follows. To determine the values for the rear controller, subtract one from the PDL numbers listed.

PDL(X)	FUNCTION	RANGE
1	Up and Down	0-255
3	Left and Right	0-255
5	Direction	Up=1, Down=4 Left=8 Right=2
7	Left Trigger	Off=0, On=1
9	Right Trigger	Off=0 On=1
11	ASCII code for keypad	Nothing pressed =0
13	Keypad # pressed	*=10,#=11, nothing pressed=15
15	(Reserved for future use)	

Test Program:

```
10 GR: COLOR=1
20 LET c=PDL(13)
30 IF c=15 THEN GOTO 50
40 COLOR=c
50 LET x=39*PDL(3)/255:y=39*PDL(1)/255
60 PLOT x,y
70 IF PDL(7)=1 THEN END
80 GOTO 20
```

Sample Run:

```
Experiment with your front game
controller to see what this program
does.
```

PLOT

PLOT is for coloring in entire blocks of your low resolution graphics screen grid. The range is 0 to 39.

Test Program:

```
10 GR
20 COLOR=13
30 PLOT 20,2
```

Sample Run:

A block of light yellow will appear at column 20, row 2.

PLUS SIGN (+)

The plus sign is used in arithmetic to signify addition. It is also used to concatenate strings.

Test Program:

```
10 PRINT 1 + 2
```

Sample Run:

```
 3
```

Test Program:

```
10 LET A$="FAT"
20 LET B$="HER"
30 LET C$=A$+B$
40 PRINT C$
```

Sample Run:

```
FATHER
```

POS

The screen can display 31 characters of text per line. The positions of these characters are numbered from 0 to 30 (beginning at the left of the screen). POS gives you the current horizontal position of the cursor relative to the left edge of the screen.

Test Program:

```
10 PRINT "E";: A=POS(0)
20 PRINT " PLURIBUS";: B=POS(0)
30 PRINT " UNUM ";: C=POS(0)
40 PRINT A;"   ";B;"   ";C
```

Sample Run:

```
E PLURIBUS UNUM   1   10   15
```

PRINT

PRINT can do several different things. PRINT followed by nothing yields a blank line when RUN. PRINT writes to your output device (screen or printer). PRINT displays variable values. To print out your work, type PR#1 then, at the end of your program, type PR#0 to send output back to the monitor. (See PR#.)

You can use ? as an abbreviation for PRINT. When you use SmartWRITER to edit a SmartBASIC program, you will see ? used in place of the word PRINT.

Test Program:

```
10 PRINT "SPRING HAS SPRUNG."
20 PRINT "THE GRASS HAS RIZ."
30 PRINT "I WONDER WHERE"
40 PRINT "THE FLOWERS IS."
```

Sample Run:

```
SPRING HAS SPRUNG.
THE GRASS HAS RIZ.
I WONDER WHERE
THE FLOWERS IS.
```

PR#

PR#(device) is an output related command which transfers output to your printer or your screen. PR#1 transfers output to your printer, while PR#0 returns output to your TV screen or monitor.

Test Program:

```
10 PRINT "ADAM'S THE GREATEST!"
PR#1:LIST:PR#0
```

Sample Run:

```
ADAM'S THE GREATEST!
--prints out on your screen.

10 PRINT "ADAM'S THE GREATEST!"
--prints out on your printer.
```

NOTE: Before you try to print anything on your printer, be sure that there is paper in it.

Test Program:

```
10 PR#1
20 PRINT "HI"
30 PRINT "THERE!"
40 PR#0
```

Sample Run:

```
HI
THERE!
--prints out on your printer.
```

QUOTATION MARKS(" ")

Quotes are necessary around any letters or words that you want to be printed on your output device (screen or printer). Without the quotes, ADAM will recognize any typed material following a PRINT statement as variables. Single quotes or apostrophes cannot substitute for quotation marks.

Test Program:

```
10 PRINT "BO DEREK STARRED IN  ";
20 PRINT "10"
```

Sample Run:

```
BO DEREK STARRED IN 10
```

Relative
Operators

<=
>=
<
>
<>
=

These signs are used in IF...THEN statements to compare two values. They are, top to bottom: less than or equal to, greater than or equal to, less than, greater than, not equal to, equal to.

Test Program:

```
5 INPUT "GUESS MY NUMBER  ";A
10 IF A>4 THEN PRINT "TOO BIG":GOTO 5
20 IF A<4 THEN PRINT "TOO LOW":GOTO 5
30 PRINT "YOU GOT IT!"
40 END
```

Sample Run:

```
GUESS MY NUMBER  2
TOO LOW
GUESS MY NUMBER  6
TOO BIG
GUESS MY NUMBER  4
YOU GOT IT!
```

REM

The REM statement is like writing a note to yourself on a scratch pad. The computer ignores it, so it is not executed, but will be displayed when the program is listed. Most people use REM to jot down the purpose of their program or a program line. REM stands for REMark. REM statements can appear throughout your program--wherever you wish to make a notation for human eyes only.

> **NOTE:** Whenever a REM statement is used as one of several statements on a single line, the REM statement must be last. Otherwise the statements following it will be overlooked by the computer.

Test Program:

```
10 PRINT "THIS IS A 'REM' TEST PROGRAM"
20 REM PRINT "THIS SHOULDN'T PRINT"
30 REM PRINT "REM FLUNKED THE TEST IF
LINE 20 PRINTED OUT"
40 PRINT "REM WORKED":REM TELL THE USER
THAT REM WORKED
50 END
```

Sample Run:

```
THIS IS A 'REM' TEST PROGRAM
REM WORKED
```

RENAME changes the name of a file. The format is RENAME oldname,newname.

Test Program:

```
RENAME HELLO,BYE
```

Sample Run:

```
HELLO is now BYE.
```

RESTORE

RESTORE causes the DATA pointer to return to the first piece of data in the first DATA list, and read it all over again, from the beginning. This allows you to use data stored in DATA statements more than once.

Test Program:

```
10 DATA 1,2,3,4,5
20 DATA 2,4,6,8,10
25 FOR I=1 TO 5
30 READ A,B
35 PRINT A;" ";B
38 NEXT I
40 RESTORE
50 GOTO 25
```

Sample Run:

```
 1   2

 3   4

 5   2

 4   6

 8   10

 1   2

 3   4

 .   .

 .   .

 .   .
```

The DATA list is read and printed again and again. Use CONTROL-C to break out of this loop.

RESUME

RESUME is usually used as the final statement in ONERR-GOTO routines, telling the computer to resume executing the program. In other words, RESUME returns control to the main program from an error routine. Never use RESUME in the immediate mode, but always within a program.

Test Program:

```
10 ONERR GOTO 60
20 INPUT "TYPE A POSITIVE NUMBER:  ";N
30 R=SQR(N)
40 PRINT "THE SQUARE ROOT OF   ";N;" IS
";R
50 END
60 N=-N
70 RESUME
```

Sample Run:

```
TYPE A POSITIVE NUMBER:  64
THE SQUARE ROOT OF 64 IS 8
```

NOTE: If you type in a negative number, the computer will ignore the negative sign, assuming that you didn't intend to put it there.

RETURN

The RETURN statement is always preceded by a GOSUB statement. The purpose of RETURN is to "jog the computer's memory". After branching into a subroutine, when the computer encounters the statement RETURN, it "remembers" where it branched from, and so branches back to the statement after the most recently encountered GOSUB. RETURN cannot be used by itself. It must be paired with GOSUB. The computer keeps track of which RETURN matches which GOSUB. Also use RETURN with ON...GOSUB.

Test Program:

```
10 GOSUB 40
20 PRINT "WORKED"
30 GOTO 70
40 PRINT "THE RETURN STATEMENT ";
50 RETURN
60 PRINT "DIDN'T WORK"
70 END
```

Sample Run:

```
THE RETURN STATEMENT WORKED
```

RETURN
(key)

The RETURN key will send the current input line to the computer, will clear to the end of the line, and move to the beginning of the next line. ADAM will not look at a program line or command until you press RETURN at the end of it. CONTROL-M does the same thing.

Test Program:

```
NONE
```

Sample Run:

```
NONE
```

RIGHT$

The RIGHT$ function works on the same idea as MID$ and LEFT$, except that RIGHT$ will return a specific number of characters at the right, or end of the string.The syntax of RIGHT$ is:

RIGHT$(<string variable>,number)

where "number" must be from 1-255.

Test Program:

```
10 A$="PICKLE CROCK"
20 B$=RIGHT$(A$,5)
30 PRINT "ALLIGATORS LOVE ";B$;" 'N
ROLL."
40 END
```

Sample Run:

```
ALLIGATORS LOVE CROCK 'N ROLL.
```

Function

RND

RND returns a random real number less than 1 and greater than or equal to zero. Its syntax is:

RND(x)

where "x" is an arithmetic expression. If x is positive, each time RND(x) is used you will get a new random number. The sequence of random numbers will be the same each time Smart BASIC is booted. If x is less than zero, then the same random number will be generated every time that particular x is used. Different random sequences may be brought about by using different negative arguments. This negative "seed" procedure is useful in program debugging. Should your arithmetic expression be zero, then you'll get the most recent previous random number (sequence) generated.

Test Program:

```
5 INPUT N
10 N=RND(-N)
20 DEF FN D(N)=INT(1+N*RND(1))
30 FOR I=1 TO  6
40 PRINT FN D(5)
50 NEXT
```

Sample Run:

```
?
3
3
2
4
4
3
```

ADAM will print a random number between one and five.

RUN

RUN tells the computer to execute or "run through" the program stored in main memory. RUN followed by a filename tells the computer to LOAD and RUN a program stored on the digital data pack. RUN followed by a line number begins running your program at the line number. RUN clears all variables.

Test Program:

```
10 PRINT "I LOVE YOU"
20 PRINT "YOU TURN ME ON"
30 PRINT "LET'S GET MARRIED"
RUN
```

Sample Run:

```
I LOVE YOU
YOU TURN ME ON
LET'S GET MARRIED
```

NOTE: RUN can also be used within a program to have one program automatically load and execute another program on your digital data pack. When this statement is executed in the first program, all string and numeric variables are cleared, and the second program begins execution. For example, suppose you had a program named DUMMY that you wanted to run right after a program called MAIN.

Here's how:

Type the program DUMMY (whatever it may be)
SAVE DUMMY
Type the program MAIN
SAVE MAIN
RUN MAIN

Be sure to put the line (line #) PRINT CHR$(4);"RUN DUMMY" as the last line in the program MAIN.

SAVE

SAVE allows you to store your program on a digital data pack for future use. The name you invent for your program must not exceed 10 characters. The syntax is SAVE<filename>,D#. With only one drive, D1 is optional.

Test Program:

```
10 PRINT "MY DAD"
20 PRINT "IS SAD"
SAVE DAD
```

Sample Run:

> If you type CATALOG at this point, you will be able to prove that your program DAD is now stored on your digital data pack.

NOTE: If you give your program a filename that already exists, the computer will not destroy your old program. It will be saved as a backup copy. Any backup copies that exist with the same name will be destroyed.

The SCRN function is used to identify colors at particular locations on your graphics screen grid. The location of the graphics block is specified by low resolution X,Y coordinates. (See GR) The color number is the same as low resolution color (see COLOR=).

Test Program:

```
10 GR
20 COLOR=8
30 PLOT 30,20
40 IF SCRN(30,20)=8 THEN 70
50 PRINT "SCRN FAILED"
60 GOTO 80
70 PRINT "SCRN WORKED"
80 END
```

Sample Run:

```
SCRN WORKED
```

Operator

SEMICOLON
(;)

Semicolons allow you to join together in an unbroken line, words or letters within quotation marks, or string variables. Like the comma, semicolons are used in PRINT statements. When items are separated by semicolons, no space is printed between them.

Test Program:

```
10 PRINT "ME";"ET";"ING"
```

Sample Run:

```
MEETING
```

The SGN function tells whether a given number is positive, negative, or zero. This is quite useful whenever a course of programming action depends on whether a certain number is less than, equal to, or greater than another number. If SGN(X) is positive, SGN replies with a 1. If zero, a 0. And if negative, a -1. For example, PRINT SGN(-5),SGN(3), SGN(0) prints: -1 1 0

Test Program:

```
10 X=-6
20 Z=SGN(X)
30 IF Z=-1 THEN 60
40 PRINT "SGN DIDN'T WORK"
50 GOTO 70
60 PRINT "SGN FUNCTIONS CORRECTLY"
70 END
```

Sample Run:

```
SGN FUNCTIONS CORRECTLY
```

Operator

SLASH (/)

The slash is used mathematically to indicate division.

Test Program:

```
10 C=1297.43
20 PRINT C/37
```

Sample Run:

```
35.0656757
```

The SPC function is used in conjunction with the PRINT statement to insert spaces among text being displayed. The format is SPC(X) where X is any whole number from 0 to 255. This is the number of spaces you wish to be inserted in your displayed material. It is useful for right-justifying copy, as well as line indentations.

Test Program:

```
10 FOR I=1 TO 4
20 READ A$
30 PRINT SPC(10-LEN(A$));A$
40 NEXT I
50 DATA WASHINGTON,
JEFFERSON,ADAMS,MADISON
```

Sample Run:

```
WASHINGTON
 JEFFERSON
    ADAMS
  MADISON
```

SPEED=

SPEED= allows you to control how fast or slowly the computer sends characters to an output device (your screen, your printer). Your range is any whole number from 0 to 255. Zero is the slowest speed, 255 is the fastest, and is also the normal speed of your output devices. SPEED= only affects text. The number following the = may be a variable.

Test Program:

```
10 PRINT "AT FULL SPEED"
20 SPEED=150
30 PRINT "NOT QUITE SO FAST"
40 SPEED=100
50 PRINT "SLOWER THAN BEFORE"
60 SPEED=50
70 PRINT "THIS IS QUITE SLOW"
80 SPEED=0
90 PRINT "DO YOU REALLY WANT TO GO THIS
SLOW?"
100 SPEED=255
110 PRINT "BACK TO FULL SPEED"
```

Sample Run:

```
AT FULL SPEED
NOT QUITE SO FAST
SLOWER THAN BEFORE
THIS IS QUITE SLOW
DO YOU REALLY WANT TO GO THIS SLOW?
BACK TO FULL SPEED
```

These lines will appear on your output device at a slower and slower rate, until the last line, which is back to normal speed.

SQR (X) is the square root function. It calculates the square root for X when X is any positive number.

Test Program:

```
10 PRINT "THE SQUARE ROOT OF 64 IS ";
20 PRINT SQR(64)
30 PRINT "'SQR' WORKED IF THE ANSWER IS
8"
40 END
```

Sample Run:

```
THE SQUARE ROOT OF 64 IS 8
'SQR' WORKED IF THE ANSWER IS 8
```

STOP

The placement of STOP within your program will stop program execution at that line, and the computer will display BREAK IN X where X is whatever line number your STOP statement is on. The computer will now be in immediate execution mode. To continue your program, type in CONT. If you type in RUN, your program will start all over again, from the beginning. You can also use GOTO at this juncture, if you wish your program's run to resume on a line other than the one immediately following your STOP statement. STOP does not clear variables.

Test Program:

```
10 PRINT "A GREEN LIGHT MEANS GO, A RED
LIGHT MEANS "
20 STOP
30 PRINT "PULL OVER.  YOU JUST WENT
THROUGH A STOP LIGHT"
40 END
```

Sample Run:

```
A GREEN LIGHT MEANS GO, A RED LIGHT
MEANS
BREAK IN 20

CONT

PULL OVER.  YOU JUST WENT THROUGH A STOP
LIGHT.
```

STR$ will convert numbers or numeric variables into strings. The advantage of this is that string modifiers may be used to manipulate them. (e.g. MID$, LEFT$, ASC) The maximum length of a string is 255 characters.

Test Program:

```
10 INPUT "HOUSE NUMBER, STREET NAME?
";N,S$
20 PRINT "YOUR ADDRESS IS ";STR$(N)+" "+S$
```

Sample Run:

```
HOUSE NUMBER, STREET NAME? 10, DOWNING
STREET
YOUR ADDRESS IS 10 DOWNING STREET
```

TAB is useful for displaying items on the screen in designated positions. This function is essential in the makeup of charts, tables, etc. When using TAB, keep these things in mind:

1. TAB can only be used within a PRINT statement.

2. Your line length available for positioning text is 255 characters long.

3. TAB will not cause the cursor to backspace. If the TAB value is less than or equal to the cursor position nothing will happen.

4. TAB value must be positive.

5. If the line to contain the displayed items already has some characters displayed in it, TAB will convert to spaces everything from the cursor to the TAB value

Test Program:

```
10 PRINT TAB (25);"UPHILL"
20 PRINT TAB (20);"GOING"
30 PRINT TAB (15);"ILL"
40 PRINT TAB (10);"FEEL"
50 PRINT TAB (5);"I"
```

Sample Run:

```
                         UPHILL
                   GOING
              ILL
         FEEL
    I
```

TEXT

TEXT is used to switch your screen out of the graphics mode back to a full TEXT or narrative screen. Be aware that when you type TEXT, your screen will be cleared and the cursor will be positioned at the upper left corner. Similarly, a program can be ended with 999 TEXT:END to leave the screen in a clean form for the next user.

Test Program:

```
10 TEXT
20 PRINT "'TEXT' WORKED"
30 END
GR
RUN
```

Sample Run:

```
'TEXT' WORKED
```

TRACE and NOTRACE are command statements used in program debugging. Use of the TRACE command will cause the computer to print out the line numbers of your program, as each one is executed. This allows you to isolate a bug more easily and quickly. You'll probably have little use for TRACE and NOTRACE, because of ADAM's specific, line-by-line error messages--but it's something which may be useful to you in the future, should your interest in computers continue.

NOTRACE turns off or cancels the TRACE command.

Test Program:

```
10 PRINT "'TRACE' IS A LINE-BY-LINE"
20 TRACE
30 PRINT "ERROR DETECTIVE"
40 GOTO 60
50 NOTRACE
55 GOTO 100
60 PRINT "UNTIL FOILED BY 'NOTRACE'"
80 PRINT "THAT NEFARIOUS"
90 GOTO 50
100 PRINT "CHAMPION OF BUGS"
110 END
```

Continued...

TRACE and NOTRACE

Sample Run:

```
'TRACE' IS A LINE-BY-LINE
#30 ERROR DETECTIVE
#40#60#60 UNTIL FOILED BY 'NOTRACE'
#80 THAT NEFARIOUS
#90#50#50 CHAMPION OF BUGS
```

If you use TRACE with operating system commands, print a CHR$(13); just before the normal CHR$(4).

```
10 TRACE
20 PRINT CHR$(13);CHR$(4);"LOAD JOE"
```

When TRACE encounters a GOTO or GOSUB, the line # jumped to is also printed out.

UNLOCK

UNLOCK removes LOCK and allows a locked file to be deleted.

Test Program:

```
10 PRINT "MY DAD"
20 PRINT "IS SAD"
SAVE DAD
LOCK DAD
CATALOG (to see the asterisk, indicating
that DAD is locked)
UNLOCK DAD
```

Sample Run:

```
You will now be able to delete DAD.
```

The VAL function is the opposite of the STR$ function, in that it takes numbers written as strings and converts them back to numeric notation.

Test Program:

```
10 A$="37.50"
20 PRINT VAL (A$);"  ";A$
30 END
```

Sample Run:

```
37.5   37.50
```

Test Program:

```
10 A$=STR$(14.50)
20 PRINT VAL(A$)
30 END
```

Sample Run:

```
14.5
```

Test Program:

```
10 A$="999 QUAKER LN."
20 B$="SOUTH"
30 PRINT VAL(A$),VAL(B$)
```

Sample Run:

```
999                                    0
```

VLIN draws a vertical line, in the designated display color, at the points indicated. This is a low resolution graphics statement.

Test Program:

```
10 GR
20 COLOR=9
30 VLIN 10,21 AT 20
```

Sample Run:

A vertical line in orange is drawn from row 10 to row 21 at column 20.

The screen can display 24 lines of text. The positions of these lines are numbered 0 to 23. VPOS gives you the current vertical position of the cursor relative to the top line. Note that the VTAB statement is numbered from 1 to 24.

Test Program:

```
10 VTAB 1
20 PRINT "THE TOP LINE IS AT ",
   VPOS(1)
30 FOR T=1 TO 3000:NEXT
40 VTAB 24
50 PRINT "THE BOTTOM LINE IS AT ",
   VPOS(1)
60 FOR T=1 TO 3000:NEXT T
```

Sample Run:

```
THE TOP LINE IS AT 0


THE BOTTOM LINE IS AT 23
```

VTAB is a way to position text on the screen at a specified location. VTAB (or vertical tab) has a value of 1 to 24, because this is the number of rows your screen contains. VTAB directs the computer as to where to print. The number following VTAB can be a variable.

Test Program:

```
10 PRINT "YOU WILL PLEASE ENTER A VTAB
VALUE "
20 INPUT X
30 VTAB X
40 PRINT "SEE, IT WORKED!  THIS PRINTED
ON LINE ";X
50 END
```

Sample Run:

```
YOU WILL PLEASE ENTER A VTAB VALUE 4



SEE, IT WORKED!  THIS PRINTED ON LINE 4
```

WRITE must be used before any PRINT statements may be used to write data to a file. The format is:

WRITE<filename>,[D#]

WRITE must appear within a PRINT command, preceded by a CONTROL-D. READ must be used before any INPUTs from a data file. The syntax is:

READ<filename>,[D#]

The D# refers to the drive number and may be omitted if you only have one drive.

For additional information, see SEQUENTIAL TEXT FILES in Appendix C.

Test Program:

```
10 D$=CHR$(4)
20 PRINT D$;"OPEN WELCOME"
30 PRINT D$;"WRITE WELCOME"
40 PRINT "THIS WILL BE STORED"
50 PRINT D$;"CLOSE WELCOME"
60 PRINT D$;"OPEN WELCOME"
70 PRINT D$;"READ WELCOME"
80 INPUT B$ :PRINT B$
100 PRINT D$;"CLOSE WELCOME"
]MON C
```

Sample Run:

```
OPEN WELCOME
WRITE WELCOME
CLOSE WELCOME
OPEN WELCOME
READ WELCOME
?THIS WILL BE STORED
CLOSE WELCOME
```

The INPUT B$ doesn't require operator input, as the computer is taking its input from the file rather than the screen.