Program

The following three programs show some possible uses of sprites.

```
100 CALL CLEAR
110 CALL CHAR(244,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(246,"183C7EFFFF7E3C18")
130 CALL CHAR(248,"F00FF00FF00FF00F")
140 CALL SPRITE(#1,244,5,92,124,#2,248,7,1,1)
150 CALL SPRITE(#28,33,16,12,48,1,1)
160 CALL SPRITE(#15,246,14,1,1,127,-128)
170 GOTO 170
(Press CLEAR to stop the program.)
```

Line 140 creates a dark blue sprite in the center of the screen and a red striped sprite in the upper-right corner of the screen. Line 150 creates a white sprite near the upper-left corner of the screen and starts it moving slowly at a 45-degree angle down and to the right. The sprite is an exclamation point.

Line 160 creates a dark red sprite at the upper-right corner of the screen and starts it moving very fast at a 45 degree angle down and to the left.

The following program makes a rather spectacular use of sprites.

```
100 CALL CLEAR
110 CALL CHAR(244,"0008081C7F1C0808")
120 RANDOMIZE
130 CALL SCREEN(2)
140 FOR A=1 TO 28
150 CALL SPRITE(#A,244,INT(A/3)+3,92,124,A*INT(RND*4.5)
-2.25+A/2*SGN(RND-.5),A*INT(RND*4.5)-2.25+A/2*SGN(RND-.5))
160 NEXT A
170 GOTO 140
(Press CLEAR to stop the program.)
```

Line 110 defines character 244.

Line 150 defines the sprites, 28 in all. The sprite-number is the current value of A. The character-value is 244. The sprite-color is INT(A/3)+3. The starting dot-row and dot-column are 92 and 124, the center of the screen. The row- and column-velocities are chosen randomly using the value of A*INT(RND*4.5)-2.25+A/2*SGN(RND-.5).

Line 170 causes the sequence to repeat.

The following program uses all the subprograms that relate to sprites except for COLOR. They are CHAR, COINC, DELSPRITE, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION, and SPRITE.

The program creates two double-sized magnified sprites in the shapes of two people walking along a floor. There is a barrier that one of them passes through and the other jumps through. The one that jumps through goes a

little faster after each jump, eventually catching the other one. When this happens, they each become double-sized, unmagnified sprites and continue walking. When they meet for the second time, the one that has been going faster disappears and the other continues walking.

```
100 CALL CLEAR
110 S1$="0103030103030303030303030303030380C
0C080C0C0C0C0C0C0C0C0C0C0C0C0E0"
120 S2$="0103030103070F1B1B030303060C0C0E80C
0C080C0E0F0D8CCC0C0C060303038"
130 COUNT=0
140 CALL CHAR(244,S1$)
150 CALL CHAR(248,S2$)
160 CALL SCREEN(14)
170 CALL COLOR(14,13,13)
180 FOR A=19 TO 24
190 CALL HCHAR(A,1,136,32)
200 NEXT A
210 CALL COLOR(13,15,15)
220 CALL VCHAR(14,22,128,6)
230 CALL VCHAR(14,23,128,6)
240 CALL VCHAR(14,24,128,6)
250 CALL SPRITE(#1,244,5,113,129,#2,244,7,113,9)
260 CALL MAGNIFY(4)
270 XDIR=4
280 PAT=2
290 CALL MOTION(#1,0,XDIR,#2,0,4)
300 CALL PATTERN(#1,246+PAT,#2,246-PAT)
310 PAT=-PAT
320 CALL COINC(ALL,CO)
330 IF CO>0 THEN 370
340 CALL POSITION(#1,YPOS1,XPOS1)
350 IF XPOS1>136 AND XPOS1<192 THEN 470
360 GOTO 300
370 REM COINCIDENCE
380 CALL MOTION(#1,0,0#2,0,0)
390 CALL PATTERN(#1,244,#2,244)
400 IF COUNT>0 THEN 540
410 COUNT=COUNT+1
420 CALL POSITION(#1,YPOS1,XPOS1,#2,YPOS2,XPOS2)
430 CALL MAGNIFY(3)
440 CALL LOCATE(#1,YPOS1+16,XPOS1+8,#2,YPOS2+16,XPOS2)
450 CALL MOTION(#1,0,XDIR,#2,0,4)
460 GOTO 340
470 REM #1 HIT WALL
480 CALL MOTION(#1,0,0)
490 CALL POSITION(#1,YPOS1,XPOS1)
500 CALL LOCATE(#1,YPOS1,193)
510 XDIR=XDIR+1
520 CALL MOTION(#1,0,XDIR)
530 GOTO 300
540 REM SECOND COINCIDENCE
550 FOR DELAY=1 TO 1000 :: NEXT DELAY
```

```
560 CALL MOTION(#2,0,4)
570 CALL DELSPRITE(#1)
580 FOR STEP1=1 TO 20
590 CALL PATTERN(#2,248)
600 FOR DELAY=1 TO 40 :: NEXT DELAY
610 CALL PATTERN(#2,244)
620 FOR DELAY=1 TO 40 :: NEXT DELAY
630 NEXT STEP1
640 CALL CLEAR
```

Lines 110, 120, 140, 150, 250, and 260 define the sprites.

Line 130 sets the meeting counter to zero.

Lines 170 through 200 build the floor.

Lines 210 through 240 build the barrier.

Line 270 sets the starting speed of the sprite that will speed up.

Line 290 sets the sprites in motion.

Line 300 creates the illusion of walking.

Line 320 checks to see if the sprites have met. Line 330 transfers control if the sprites have met.

Lines 340 and 350 check to see if the sprite has reached the barrier and transfer control if it has.

Line 360 loops back to continue the walk.

Lines 370 through 460 handle the sprites running into each other. Lines 380 and 390 stop them.

Line 400 checks to see if it is the first meeting.

Line 410 increments the meeting counter.

Line 420 finds the sprites position.

Line 430 makes them smaller.

Line 440 puts them on the floor and moves the fast one slightly ahead.

Line 450 starts them moving again.

Lines 470 through 530 handle the fast sprite jumping through the barrier. Line 480 stops it.

Line 490 finds where it is.

Line 500 puts it at the new location beyond the barrier.

Lines 510 and 520 start it moving again, a little faster.

Lines 540 through 640 handle the second meeting.

Line 560 starts the slow sprite moving.

Line 570 deletes the fast sprite.

Lines 580 through 630 make the slow sprite walk 20 steps.

SQR --Function--Square Root

Format
SQR(numeric-expression)

Type
REAL

Description
The SQR function returns the positive square root of the value of the numeric-expression.

The value of the numeric-expression cannot be negative.

Examples

```
100 PRINT SQR(4)
```
Prints 2.

```
100 X=SQR(2.57E5)
```
Sets X equal to the square root of 257,000, which is 506.9516742255.

STOP STOP

Format
STOP

Cross Refernce
END

Description
The STOP statement stops the execution of your program.

When your computer encounters a STOP statement, the computer performs the following operations:

It closes all open files.

It restores the default character definitions of all characters.

Restores the default foreground-color (black) and background-color (transparent) to all characters.

Restores the default screen color (cyan).

Deletes all sprites.

Resets the sprite magnification level to 1.

The graphics colors (see DCOLOR) and current position (see DRAWTO) are not affected. If the computer is in Pattern or Text Mode the graphics mode and margin settings remain unchanged.

A STOP statement is not necessary to stop your program; the program automatically stops after the highest-numbered line is executed.

STOP is frequently used before a subprogram that follows the main portion of a program, to ensure that the subprogram is not executed after the execution of the highest-numbered line in the main program.

STOP can be used interchangeably with the END statement, except that you cannot use STOP to end a subprogram.

Program
The following program illustrates a use of the STOP statement. The program adds the numbers from 1 to 100.

```
100 CALL CLEAR
110 TOT=0
120 NUMB=1
130 TOT=TOT+NUMB
140 NUMB=NUMB+1
150 IF NUMB>100 THEN PRINT TOT::STOP
160 GOTO 130
```

STR$** --Function--String-Number **STR$

Format
STR$(numeric-expression)

Type
String

Cross Reference
VAL

Description
The STR$ function returns the string representation of the value of the numeric-expression.

STR$ enables you to use the string representation of the numeric-expression with an instruction that requires a string-expression as a parameter.

STR$ is the inverse of the VAL function.

STR$ removes leading and trailing spaces.

Examples

```
100 NUM$=STR$(78.6)
```
Sets NUM$ equal to "78.6".

```
100 LL$=STR$(3E15)
```
Sets LL$ equal to "3.E+15".

```
100 X$=STR$(A*4)
```
Sets X$ equal to a string representation of whatever value is obtained when A is multiplied by 4. For instance, if A is equal to -8, X$ is set equal to "-32".

SUB --Subprogram **SUB**

Format
SUB subprogram-name[(parameter[,...])]

Cross Reference
CALL, SUBEND, SUBEXIT

Description
The SUB statement is the first statement in a subprogram.

You can use a subprogram to separate a group of statements from the main program. Subprograms are generally used to perform a specific operation several times in the same program or in different programs, or to isolate variables that are specific to the subprogram.

Subprograms are accessed from your main program with a CALL statement. The subprogram-name in the SUB statement is the same name that you use in the CALL statement that transfers control to the subprogram.

The maximum length of a subprogram-name is 15 characters.

A user-written subprogram may have the same subprogram-name as a built-in subprogram. In such a case, a CALL statement will access the user-written subprogram instead of the built-in one.

You can use parameters to pass values to a subprogram. Parameters must be valid names of variables or arrays.

SUBEND must be the last statement executed in a subprogram. When the computer encounters a SUBEND or a SUBEXIT statement in a subprogram, program control returns to the statement immediately following the CALL statement that called the subprogram.

It is recommended that you do not use any statement other than SUBEND or SUBEXIT to leave a subprogram. If you use another statement to leave a subprogram you may still be using variables local to the subprogram, which may cause unexpected results.

Subprograms must have higher line numbers than any part of your main program. A SUB statement cannot be part of an IF THEN statement.

Subprogram Variables

The variables used in a subprogram (other than those used as parameters) are local to the subprogram; that is, even if a variable in your main program has the same name as a variable in a subprogram, the value of that variable outside the subprogram is not affected by changes to its value in the subprogram. If a subprogram is called more than once, any local variables used in the subprogram retain their values from one call to the next.

Parameters

When your program executes a subprogram beginning with a SUB statement with parameters, the parameter values (constants or variables) are passed from the parameter-list of the CALL statement to the subprogram. The parameter-list in the CALL statement must contain the same number of parameters as the SUB statement. Values are passed in the order in which they are listed.

A numeric parameter must be passed a numeric value. A string parameter must be passed a string value.

An array parameter must be passed an array. A string-array parameter must be passed a string array.

To pass an entire array as one parameter, follow the array name with left and right parentheses. If the array has more than one dimension, place one comma between the parentheses for each additional dimension.

Passing Parameters by Reference and Value

When a subprogram manipulates the value of a parameter passed to it, the new parameter value may or may not be passed back to the main program. When a parameter is passed to a subprogram "by reference", the new value is passed back to the main program after the subprogram has executed.

When a parameter is passed to a subprogram "by value", the new value is not passed back to the main program.

> Variables, array elements, and arrays are normally passed by reference. However, if a numeric variable or array element is of a different data-type in the main program than it is in the subprogram, the parameter is passed by value.
>
> To specify that a variable or array element is to be passed by value rather than by reference, enclose it in parentheses in the CALL statement's parameter-list. Note that this option is not available for arrays.
>
> If you use an expression as a parameter, it is evaluated and passed by value.

Examples

100 SUB MENU
Marks the beginning of a subprogram. No parameters are passed or returned.

100 SUB MENU(COUNT,CHOICE)
Marks the beginning of a subprogram. The variables COUNT and CHOICE may be used and/or have their values changed in the subprogram and returned to the variables in the same position in the calling statement.

```
100 SUB PAYCHECK(DATE,Q,SSN,PAYRATE,TABLE(,))
```

Marks the beginning of a subprogram. The variables DATE, Q, SSN, PAYRATE, and the array TABLE with two dimensions may be used and/or have their values changed in the subprogram and returned to the variables in the same position in the calling statement.

Program

The following program illustrates a use of SUB. The subprogram MENU had been previously saved with the MERGE option. It prints a menu and requests a choice. The main program tells the subprogram how many choices there are and what the choices are. It then uses the choice made in the subprogram to determine what program to run.

```
100 CALL MENU(5,R)
110 ON R GOTO 120,130,140,150,160
120 RUN "DSK1.PAYABLES"
130 RUN "DSK1.RECEIVE"
140 RUN "DSK1.PAYROLL"
150 RUN "DSK1.INVENTORY"
160 RUN "DSK1.LEDGER"
170  DATA  ACCOUNTS   PAYABLE,ACCOUNTS   RECEIVABLE,PAYROLL,INVENTORY,GENERAL
LEDGER
```

Beginning of subprogram MENU.

Note that this R is not the same as the R used in lines 100 and 110 in the main program.

```
10000 SUB MENU(COUNT,CHOICE)
10010 CALL CLEAR
10020 IF COUNT>22 THEN PRINT "TOO MANY ITEMS" :: CHOICE=0 :: SUBEXIT
10030 RESTORE
10040 FOR R=1 TO COUNT
10050 READ TEMP$
10060 TEMP$=SEG$(TEMP$,1,25)
10070 DISPLAY AT(R,1):R;TEMP$
10080 NEXT R
10090 DISPLAY AT(R+1,1):"YOUR CHOICE: 1"
10100 ACCEPT AT(R+1,14)BEEP VALIDATE(DIGIT)SIZE(-2):CHOICE
10110 IF CHOICE>COUNT OR CHOICE<1 THEN 10100
10120 SUBEND
```

SUBEND --Subprogram End **SUBEND**

Format
SUBEND

Cross Reference
SUB, SUBEXIT

Description
The SUBEND statement marks the end of a subprogram.

SUBEND must be the last statement executed in a subprogram. When the computer encounters a SUBEND statement in a subprogram, program control returns to the statement immediately following the CALL statement that called the subprogram.

It is recommended that you do not use any statement other than SUBEND or SUBEXIT to leave a subprogram. If you use another statement to leave a subprogram you may still be using variables local to the subprogram, which may cause unexpected results.

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

The only statements that can immediately follow a SUBEND statement are REM, END, or the SUB statement for the next subprogram.

SUBEXIT --Subprogram Exit

Format
SUBEXIT

Cross Reference
SUB, SUBEND

Description
The SUBEXIT statement enables you to leave a subprogram before the computer executes the SUBEND statement that ends the subprogram.

SUBEXIT enables you to have more than one exit from a subprogram.

When the computer encounters a SUBEXIT statement in a subprogram, program control returns to the statement immediately following the CALL statement that called the subprogram.

It is recommended that you do not use any statement other than SUBEND or SUBEXIT to leave a subprogram. If you use another statement to leave a subprogram you may still be using variables local to the subprogram, which may cause unexpected results.

SWAP **SWAP**

Format
CALL SWAP(var1,var2)

Description
The SWAP statement is used to exchange the values of two variables, provided they are of the same type and same precision. If they are not of the same type a "TYPE MISMATCH" error will occur.

The SWAP statement cannot be used to "SWAP" the contents of two arrays, except as individual elements.

```
100 FOR I=1 to 100
110 CALL SWAP(A$(I),B$(I))
120 NEXT I
```

The SWAP statement can also be used to alphabetize two strings.

```
100 INPUT "STRING #1 >":A$
110 INPUT "STRING #2 >":B$
120 IF A$>B$ THEN CALL SWAP(A$,B$)
130 PRINT A$,B$
```

TAB --Function--Tabulate TAB

Format
TAB(numeric-expression)

Cross Reference
DISPLAY, PRINT

Description
The TAB function specifies the starting position of the next item to be printed by a PRINT or DISPLAY instruction.

The numeric-expression specifies the starting position of the next print item in a print-list of a PRINT or DISPLAY instruction.

If the value of the numeric-expression is not an integer, it is rounded to the nearest integer. If the value of the numeric-expression is less than 1, it is replaced by 1.

If the value of the numeric-expression is greater than the record length of the screen or device, it is repeatedly reduced by the record length until it is less than or equal to the record length. The record length of the screen is the width of the screen window defined by the margins. For more information about the record length of a particular device, refer to the owner's manual that comes with that device.

Because the TAB function itself is treated as a separate print item, it must be preceded and/or followed by a print separator (usually a semicolon), unless it is the only item in the print-list.

If the number of characters already printed in the current record is greater than or equal to the position indicated by the value of the numeric-expression, the print item following the TAB is printed in the next record, beginning in the position specified by the value of the numeric-expression.

TAB can be used to print to a device or file only if the device or file has been opened in DISPLAY format.

TAB cannot be used with PRINT USING or DISPLAY USING.

Examples
```
100 PRINT TAB(12);35
```
Prints the number 35 at the twelfth position.

```
100 PRINT 356;TAB(18);"NAME"
```
Prints 356 at the beginning of the line and NAME at the eighteenth position of the line.

```
100 PRINT "ABCDEFGHIJKLM";TAB(5);"NOP"
```
Prints ABCDEFGHIJKLM at the beginning of the line and NOP at the fifth position of the next line.

TAN --Function--Tangent **TAN**

Format
TAN(numeric-expression)

Type
REAL

Cross Reference
ATN, COS, SIN

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

The numeric-expression cannot be less than -1.5707963269514E10 or greater than 1.5707963266374E10.

To convert the measure to radians, multiply by pi/180.

Program

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

```
100 A=.7853981633973
110 B=26.565051177
120 C=45*PI/180
130 PRINT TAN(A);TAN(B)
140 PRINT TAN(B*PI/180)
150 PRINT TAN(C)
RUN
1.  7.17470553
.5
1
```

TERMCHAR --Function--Termination Character **TERMCHAR**

Format
TERMCHAR

Type
DEFINT

Cross Reference
ACCEPT, INPUT, LINPUT

Description
The TERMCHAR function returns the character code of the key pressed to exit from the previously executed INPUT, ACCEPT, or LINPUT statement.

In a program, the value returned by TERMCHAR depends on the key pressed to exit from the last instruction that accepted input from the keyboard.

VALUE RETURNED	KEY
1	AID
2	CLEAR
10	DOWN ARROW
11	UP ARROW
12	PROC'D
13	ENTER
14	BEGIN
15	BACK

If you use TERMCHAR as part of a command (unless it is preceded by ACCEPT, INPUT, or LINPUT), the value returned depends on the key pressed to enter the command (ENTER, UP ARROW, or DOWN ARROW).

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

Program

The following program illustrates a use of TERMCHAR. The program displays name, address, and city, state, and zip code information entered from the keyboard. Line 160 enables you to correct errors in previously entered lines by pressing UP ARROW. This returns the cursor to the beginning of the line that immediately precedes the one from which UP ARROW was entered.

```
100 CALL CLEAR
110 R=5::C=12
120 DISPLAY AT(R,C-10):"NAME :"
130 DISPLAY AT(R+1,C-10):"ADDRESS:"
140 DISPLAY AT(R+2,C-10):"C,S,Z:"
```

```
150 ACCEPT AT(R,C)SIZE(-20):A$(R)
160 IF TERMCHAR=11 THEN R=R-1 ELSE R=R+1
170 IF R=7 THEN 150
180 DISPLAY AT(20,1):A$(5):A$(6):A$(7)
190 GOTO 110
(Press CLEAR to stop the program.)
```

TIME/TIME$** **TIME/TIME$

Description
The computer has an internal clock that can be accessed from BASIC.

TIME$ can be used to read the clock and TIME to set the clock.

TO SET CLOCK

Format
CALL TIME("hh:mm:ss")

The string length is always 8 characters. Therefore an hour less than 10 must be preceded by a 0.

The clock works on 24 hour time so all times after 12 noon must have 12 hours added to them.

Example

```
CALL TIME$("06:15:00")      6:15 A.M.
CALL TIME$("18:15:00")      6:15 P.M.
```

TO READ CLOCK

Format
PRINT TIME$
OR
T$=TIME$

Example

```
10 CALL CLEAR
20 DISPLAY AT(24,1):TIME$
30 GOTO 20
```

```
10 CALL TIME("00:00:00")
20 FOR I=1 TO 10000
30 NEXT I
40 PRINT TIME$
```

This is an easy way to see how long a program takes to execute.

TRACE **TRACE**

Format
TRACE ON
TRACE OFF

Description
The TRACE ON instruction causes the computer to display the line number of each line in your program before it is executed.

TRACE ON enables you to see the order in which the computer performs statements as it runs your program. It is valuable as a debugging aid to help you find errors (such as unwanted infinite loops) in your program.

TRACE OFF removes the effect of the TRACE ON command.

You can use TRACE ON or TRACE OFF either as a program statement or a command.

Programs

The following programs display a trace of the order of execution of the program lines.

```
100 FOR J=1 TO 3
110 PRINT "WORD"
120 NEXT J
130 TRACE ON
RUN
```

```
100 FOR J=1 TO 3
110 PRINT "WORD"
120 NEXT J
TRACE ON
RUN
```

UNBREAK **UNBREAK**

Format
UNBREAK [line-number-list]

Cross Reference
BREAK

Description
The UNBREAK instruction removes a breakpoint from each program statement you specify.

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

The line-number-list consists of one or more line numbers separated by commas. When an UNBREAK instruction is executed, breakpoints are removed from the specified program lines.

If you do not include a line-number-list, UNBREAK removes all breakpoints, except for a breakpoint that occurs when a BREAK statement with no line-number-list is encountered in a program.

If the line-number-list includes an invalid line number (0 or a value greater than 32767), the message Bad line number is displayed. If the line-number-list includes a fractional or negative line number, the message Syntax error is displayed. In both cases, the UNBREAK instruction is ignored; that is, breakpoints are not removed even at valid line numbers in the line-number-list. If you were entering UNBREAK as a program statement, it is not entered into your program.

If the line-number-list includes a line number that is valid (1-32767) but is not the number of a line in your program, or a fractional number greater than 1, the message

```
* WARNING
  LINE NOT FOUND
```

is displayed. (If you were entering UNBREAK as a program statement, the line-number is included in the warning message). A breakpoint is, however, removed from any valid line in the line-number-list that precedes the line number that caused the warning.

Examples

```
UNBREAK
450 UNBREAK
```
Removes all breakpoints (except those resulting from a BREAK statement with no line-number-list).

```
UNBREAK 100,130
350 UNBREAK 100,130
```
Removes the breakpoints from lines 100 and 130.

VAL --Function--Value **VAL**

Format
VAL(string-expression)

Type
REAL

Cross Reference
STR$

Description
The VAL function returns the numeric value of the string-expression.

VAL enables you to use the numeric value of the string-expression with an instruction that requires a numeric-expression as a parameter.

VAL is the inverse of the STR$ function.

The string-expression must be a valid representation of a number. The length of the string-expression must be greater than 0 and less than 255. If you use a string constant, it must be enclosed in quotation marks.

Examples

```
100 NUMB=VAL("78.6")
110 PRINT NUMB
```
Prints 78.6.

```
100 LL=VAL("3E15")
```
Sets LL equal to 3E+15, or 315.

VALHEX --Function--Value of Hexadecimal Number **VALHEX**

Format
VALHEX(string-expression)

Type
DEFINT

Description
VALHEX returns the numeric value of the hexadecimal number represented by the string-expression.

The string-expression specifies the hexadecimal (base 16) number to be converted to a decimal (base 10) number. If you use a string constant, it must be enclosed in quotation marks.

The string-expression must contain only valid hexadecimal digits (0-9,A-F). Alphabetic hexadecimal digits must be upper-case letters. VALHEX can convert a hexadecimal number from one to four digits long. If the length of the string-expression is greater than four, VALHEX uses only the last four characters.

VALHEX returns an integer greater than or equal to -32768 (hexadecimal 8000) and less than or equal to 32767 (hexidecimal 7FFF).

Examples

```
100 A=VALHEX("400A")
```
Sets A equal to 16394.

```
100 PRINT VALHEX("8200")
```
Prints -32256.

VCHAR --Subprogram--Vertical Character **VCHAR**

Format
CALL VCHAR(row,column,character-code[,number-of-repetitions])

Cross Reference
DCOLOR, GCHAR, GRAPHICS, HCHAR

Description
The VCHAR subprogram enables you to place a character on the screen and repeat it horizontally.

Row and column are numeric-expressions whose values specify the position on the screen where the character is displayed.

The value of row must be greater than or equal to 1 and not exceed the total number of rows in the present graphics mode.

The value of column must be greater than or equal to 1 and must not exceed the total number of columns in the present graphics mode.

VCHAR is not affected by margin settings.

Character-code is a numeric-expression with a value from 0-255, specifying the number of the character. See Appendix B for a list of ASCII character codes.

The optional number-of-repetitions is a numeric-expression whose value specifies the number of times the character is repeated horizontally. If the repetitions extend past the end of a column, they continue from the first character of the next column. If the repetitions extend past the end of the last column, they continue from the first character of the first column.

If you use VCHAR to display a character on the screen, and then later use CHAR, COLOR, or DCOLOR to change the appearance of that character, the result depends on the graphics mode.

In Pattern and Text Modes, the displayed character changes to the newly specified pattern and/or color(s).

In High-Resolution Mode, the displayed character remains unchanged.

Examples

```
100 CALL VCHAR(12,16,33)
```
Places character 33 (an exclamation point) in row 12, column 16.

```
100 CALL VCHAR(1,1,ASC("!"),768)
```
Places an exclamation point in row 1, column 1, and repeats it 768 times, which fills the screen in Pattern Mode.

```
100 CALL VCHAR(R,C,K,T)
```
Places the character with an ASCII code specified by the value of K in row R, column C, and repeats it T times.

VERSION

VERSION

Format
CALL VERSION(numeric-variable)

Description
The VERSION subprogram returns a value indicating the version of BASIC being used.

> In MYARC Advanced BASIC, VERSION returns a value of 300 to the numeric-variable you specify.

Example

```
100 CALL VERSION(V)
```
Sets V equal to 300.

WEND

WEND

The WEND statement terminates the loop that begins with WHILE.

Statements between WHILE and WEND are executed repeatedly until the condition stated in the WHILE statement is no longer true.

Unlike FOR-NEXT statements, WHILE-WEND loops may NOT be nested. WEND always continues the most recent WHILE loop until the WHILE statement's condition becomes false.

See WHILE for detailed description of the WHILE-WEND loop.

WHILE **WHILE**

Format
WHILE condition :: ...program .... :: WEND

Cross Reference
WEND, FOR-NEXT, IF-THEN-ELSE

Description
The WHILE statement starts a loop which is executed repeatedly while the WHILE 'condition' is true. The loop is terminated with a WEND statement.

'Condition' is a logical expression, numeric or variable that WHILE evaluates. If the 'condition' is TRUE (or a non/zero value, ie., condition<>0), the program then loops between the WHILE and the WEND statements. When the condition is no longer TRUE (false or condition=0) WHILE passes execution to the statement after WEND.

Unlike FOR-NEXT statements WHILE-WEND loops may NOT be nested.

Example

```
100 A$=INKEY$
110 WHILE A$<>""THEN 100
120 WEND
130 ......  program lines
140 END
RUN
```

This short routine checks the entries into the keyboard buffer until it is empty then proceedes to the rest of the program. The keyboard is said to be "Flushed".

```
100 REM WHILE TEST
110 WHILE NAME$<>"LAST"
120 READ NAME$,PHONE$
130 COUNT=COUNT +1
140 PRINT NAME$;TAB(20);PHONE$
150 WEND
160 PRINT "NUMBER OF NAMES=";COUNT
170 PRINT "WHILE HAS BEEN EVALUATED TO FALSE "
180 DATA MYARC,201-766-1700
190 DATA BILL REISS,201-000-0000
200 DATA LAST,LAST
```

I/O DEFAULTS

MYARC Advanced BASIC includes several features to simplify the direction of input and output to certain devices. These devices are your main mass storage device (typically DSK1), your main printer (typically PIO), your main communications port (typically RS232.BA=4800), and your main modem port (typically RS232.BA=1200). These defaults are set from the operating system defaults when BASIC is initially started.

The following names are used for reassigning a particular default device:

NAME	DEVICE
LPT	Default printing device
CHDIR	Default disk drive or directory
COM	Default communications port
MDM	Default modem port

To change a default, type the command and follow it by the desired device name. For example, the LLIST command prints the program to the main printer port. If your printer is connected to the RS232 port and not the PIO port you would need to redirect the output from LLIST. To do this, type LPT "RS232.BA=4800".

You may also check to se what a particular default is set for at any time. To do this, type any one of the following commands and BASIC will list what that device is set for:

COMMAND	DEVICE
PPT	Lists the default printer
PWD	Lists the default working directory
PCM	Lists the default communications port
PMD	Lists the default modem port

APPENDICES

APPENDIX A:

COMMANDS, STATEMENTS, AND FUNCTIONS

The following is a list of all MYARC Advanced BASIC commands, statements, and functions. Commands are listed first; if a command can also be used as a statement, the letter "S" is listed to the right of the command. Commands that can be abbreviated have the acceptable abbreviations underlined. Next is a list of all MYARC Advanced BASIC statements; those that can also be used as commands have a "C" after them. Finally, there is a list of all MYARC Advanced BASIC functions.

MYARC Advanced BASIC Commands

BREAK	S	LIST		PCM PMD) I/O	Default
BYE		LLIST		PPT PWD)	Commands
CHDIR		LPT	S	RESEQUENCE	
CLOSE		LTRACE	S	RUN	S
CLS	S	MERGE		SAVE	
CONTINUE		NEW		CALL SPEED	
DELETE	S	NUMBER		TRACE ON/OFF	S
KEY	S	OLD		UNBREAK	S

MYARC Advanced BASIC Statements

ACCEPT	C	END		CALL MEMSET	
BEEP	C	CALL ERR	C	CALL MOTION	C
CALL	C	CALL FILES	C	MOUSE	C
CALL BCOLOR	C	CALL FILL		NEXT	C
CALL BTIME	C	FOR TO	C	ON BREAK	
CALL CHAR	C	CALL GCHAR	C	ON ERROR	
CALL CHARPAT	C	GOSUB		ON GOSUB	
CALL CHARSET	C	GOTO		ON GOTO	
CALL CIRCLE	C	CALL GRAPHICS	C	ON WARNING	
CALL CLEAR	C	CALL HCHAR	C	OPEN	C
CLOSE	C	IF THEN ELSE		OPTION BASE	
CALL COINC	C	IMAGE		CALL OUT	C
CALL COLOR	C	CALL INIT	C	CALL PATTERN	C
DATA		INPUT		CALL PEEK	C
CALL DATE	C	INPUT		CALL PEEKV	C
CALL DCOLOR	C	CALL JOYST	C	CALL POINT	C
DEF		CALL KEY	C	CALL POKEV	C
DEFvartype		KILL	C	CALL POSITION	C
CALL DESPRITE	C	LET	C	PRINT	C
DIM	C	CALL LINK	C	PRINT USING	C
DISPLAY	C	LINPUT		RANDOMIZE	C
DISPLAY USING	C	CALL LOAD	C	READ	C
CALL DISTANCE	C	CALL LOCATE	C	CALL RECTANGLE	
CALL DRAW		CALL MAGNIFY	C	REM	C
CALL DRAWTO		CALL MARGIN	C	RESTORE	C

RETURN		STOP	C	CALL VCHAR	C
CALL SAY	C	SUB		CALL VERSION	C
CALL SCREEN	C	SUBEND		WEND	
CALL SOUND	C	SUBEXIT		WHILE	
CALL SPGET	C	CALL SWAP			
CALL SPRITE	C	CALL TIME	C		

MYARC Advanced BASIC Functions

ABS	HEX$	SEG$
ASC	LEFT$	SGN
ATN	LEN	SIN
CDBL	LOG	SQR
CHR$	MAX	STR$
CINT	MIN	TAB
COS	MOD	TAN
CREAL	PI	TERMCHAR
CSING	POS	TIME$
DATE$	REC	VAL
EOF	RND	VALHEX
EXP	RIGHT$	
FREESPACE	RPT$	
INT		

APPENDIX B:

ASCII CODES

The following predefined characters may be printed or displayed on the screen.

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
30	(cursor)	63	? (question mark)
31	(space)	64	@ (at sign)
32	(space)	65	A
33	! (exclamation point)	66	B
34	" (quote)	67	C
35	# (number or pound sign)	68	D
36	$ (dollar)	69	E
37	% (percent)	70	F
38	& (ampersand)	71	G
39	' (apostrophe)	72	H
40	((open parenthesis)	73	I
41	) (close parenthesis)	74	J
42	* (asterisk)	75	K
43	+ (plus)	76	L
44	, (comma)	77	M
45	- (minus)	78	N
46	. (period)	79	O
47	/ (slash)	80	P
48	0	81	Q
49	1	82	R
50	2	83	S
51	3	84	T
52	4	85	U
53	5	86	V
54	6	87	W
55	7	88	X
56	8	89	Y
57	9	90	Z
58	: (colon)	91	[(open bracket)
59	; (semicolon)	92	\ (reverse slant)
60	< (less than)	93	] (close bracket)
61	= (equals)	94	^ (exponentiation)
62	> (greater than)	95	_ (underline)

ASCII Codes (continued)

96	` (accent grave)	112	p
97	a	113	q
98	b	114	r
99	c	115	s
100	d	116	t
101	e	117	u
102	f	118	v
103	g	119	w
104	h	120	x
105	i	121	y
106	j	122	z
107	k	123	{ (left brace)
108	l	124	| (vertical bar)
109	m	125	} (right brace)
110	n	126	~ (tilde)
111	o	127	DEL (appears as a blank)

When key unit = 3 or = 5, the following key presses may also be detected by CALL KEY.

1	Alt 7 (AID)	3	Alt 1 (DEL)
4	Alt 2 (INS)	6	Alt 8 (REDO)
7	Alt 3 (ERASE)	8	Alt S (LEFT ARROW)
9	Alt D (RIGHT ARROW)	10	Alt X (DOWN ARROW)
11	Alt E (UP ARROW)	12	Alt 6 (CMD)
13	ENTER	14	Alt 5 (BEGIN)
15	Alt 9 (BACK)		

APPENDIX C:

MUSICAL TONE FREQUENCIES

The following table gives the frequencies (rounded to integers) of four octaves of the tempered scale (one half step between notes). While this list does not represent the entire range of notes that the computer can produce, it can be helpful for programming music.

FREQUENCY	NOTE	FREQUENCY	NOTE
110	A	440	A (above middle C)
117	A#,Bb	466	A#,Bb
123	B	494	B
131	C (low C)	523	C (high C)
139	C#,Db	554	C#,Db
147	D	587	D
156	D#,Eb	622	D#,Eb
165	E	659	E
175	F	698	F
185	F#,Gb	740	F#,Gb
196	G	784	G
208	G#,Ab	831	G#,Ab
220	A (below middle C)	880	A (above high C)
220	A (above middle C)	880	A (above high C)
223	A#,Bb	932	A#,Bb
247	B	988	B
262	C (middle C)	1047	C
277	C#,Db	1109	C#,Db
294	D	1175	D
311	D#,Eb	1245	D#,Eb
330	E	1319	E
349	F	1397	F
370	F#,Gb	1480	F#,Gb
392	G	1568	G
415	G#,Ab	1661	G#,Ab
440	A (above middle C)	1760	A

APPENDIX D:

CHARACTER SETS

SET	ASCII CODES	SET	ASCII CODES
29	0-7	13	128-135
30	8-15	14	136-143
31	16-23	15	144-151
0	24-31	16	152-159
1	32-39	17	160-167
2	40-47	18	168-175
3	48-55	19	176-183
4	56-63	20	184-191
5	64-71	21	192-199
6	72-79	22	200-207
7	80-87	23	208-215
8	88-95	24	216-223
9	96-103	25	224-231
10	104-111	26	232-239
11	112-119	27	240-247
12	120-127	28	248-255

APPENDIX E:

PATTERN-IDENTIFIER CONVERSION TABLE

BLOCK	BINARY CODE (0=OFF; 1=ON)	HEXADECIMAL NOTATION
____	0000	0
___X	0001	1
__X_	0010	2
__XX	0011	3
_X__	0100	4
_X_X	0101	5
_XX_	0110	6
_XXX	0111	7
X___	1000	8
X__X	1001	9
X_X_	1010	A
X_XX	1011	B
XX__	1100	C
XX_X	1101	D
XXX_	1110	E
XXXX	1111	F

APPENDIX F:

COLOR CODES

COLOR	CODE	COLOR	CODE
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

APPENDIX G:

MATHEMATICAL FUNCTIONS

The following mathematical functions may be defined with DEF as shown.

Function	MYARC Extended BASIC II statement
Secant	DEF SEC(X)=1/COS(X)
Cosecant	DEF CSC(X)=1/SIN(X)
Cotangent	DEF COT(X)=1/TAN(X)
Inverse Sine	DEF ARCSIN(X)=ATN(S/SQR(1/X*X))
Inverse Cosine	DEF ARCCOS(X)=ATN(X/SQR(1/X*X))+PI/2
Inverse Secant	DEF ARCSEC(X)=ATN(SQR(X*X/1))+(SGN(X)/1)*PI/2
Inverse Cosecant	DEF ARCCSC(X)=ATN(1/SQR(X*X/1))+(SGN(X)-1)*PI/2
Inverse Cotangent	DEF ARCCOT(X)=PI/2-ATN(X) or =PI/2+ATN(-X)
Hyberbolic Sine	DEF SINH(X)=(EXP(X)-EXP(-X))/2
Hyberbolic Cosine	DEF COSH(X)=(EXP(X)+EXP(-X))/2
Hyperbolic Tangent	DEF TANH(X)=2*EXP(-X)/(EXP(X)+EXP(-X))+1
Hyperbolic Secant	DEF SECH=2/(EXP(X)+EXP(-X))
Hyperbolic Cosecant	DEF CSCH=2/(EXP(X)-EXP(-X))
Hyperbolic Cotangent	DEF COTH(X)=2*EXP(-X)/(EXP(X)-EXP(-X))+1
Inverse Hyperbolic Sine	DEF ARCSINH(X)=LOG(X+SQR(X*X+1))
Inverse Hyperbolic Cosine	DEF ARCCOSH(X)=LOG(X+SQR(X*X-1))
Inverse Hyperbolic Tangent	DEF ARCTANH(X)=LOG((1+X)/(1-X))/2
Inverse Hyperbolic Secant	DEF ARCSECH(X)=LOG((1+SQR(1-X*X))/X)
Inverse Hyperbolic Cosecant	DEF ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
Inverse Hyperbolic Cotangent	DEF ARCCOTH(X)=LOG((X+1)/(X-1))/2

APPENDIX H:

LIST OF SPEECH WORDS

The following is a list of all the letters, numbers, words, and phrases that can be accessed with CALL SAY and CALL SPGET. See Appendix M for instructions on adding suffixes to anything in this list.

/ (NEGATIVE)
+ (POSITIVE)
0
1
2
3
4
5
6
7
8
9
A (a)
A1 ( )
ABOUT
AFTER
AGAIN
ALL
AM
AN
AND
ANSWER
ANY
ARE
AS
ASSUME
AT
B
BACK
BASE
BE
BETWEEN
BLACK
BLUE
BOTH
BOTTOM
BUT
BUY
BY
BYE
C
CAN
CASSETTE
CENTER
CHECK
CLEAR
COLOR
COME
COMES
COMMA
COMMAND
COMPLETE
COMPLETED
COMPUTER
CONNECTED
CONSOLE
CORRECT
COURSE
CYAN
D
DATA
DECIDE
DEVICE
DID
DIFFERENT
DISKETTE
DO
DOES
DOING
DONE
DOUBLE
DOWN
DRAW
DRAWING
E
EACH
EIGHT
EIGHTY
ELEVEN
ELSE
END
ENDS
ENTER
ERROR
EXACTLY
EYE
F
FIFTEEN
FIGURE
FIND
FINE
FINISH
FINISHED
FIRST
FIT
FIVE
FOR
FORTY
FOUR
FOURTEEN
FOURTH
FROM
FRONT
G
GAMES
GET
GETTING
GIVE
GIVES
GO
GOES
GOING
GOOD
GOOD WORK
GOODBYE
GOT
GRAY
GREEN
GUESS
H
HAD
HAND
HANDHELD UNIT
HAS
HAVE
HEAD
HEAR
HELLO
HELP

List of Speech Words (continued)

HERE
HIGHER
HIT
HOME
HOW
HUNDRED
HURRY
I
I WIN
IF
IN
INCH
INCHES
INSTRUCTION
INSTRUCTIONS
IS
IT
J
JOYSTICK
JUST
K
KEY
KEYBOARD
KNOW
L
LARGE
LARGER
LARGEST
LAST
LEARN
LEFT
LESS
LET
LIKE
LIKES
LINE
LOAD
LONG
LOOK
LOOKS
LOWER
M
MADE
MAGENTA
MAKE
ME
MEAN
MEMORY
MESSAGE
MESSAGES
MIDDLE
MIGHT
MODULE
MORE
MOST
MOVE
MUST
N
NAME
NEAR
NEED
NEGATIVE
NEXT
NICE TRY
NINE
NINETY
NO
NOT
NOW
NUMBER
O
OF
OFF
OH
ON
ONE
ONLY
OR
ORDER
OTHER
OUT
OVER
P
PART
PARTNER
PARTS
PERIOD
PLAY
PLAYS
PLEASE
POINT
POSITION
POSITIVE
PRESS
PRINT
PRINTER
PROBLEM
PROBLEMS
PROGRAM
PUT
PUTTING
Q
R
RANDOMLY
READ (read)
READ1 (red)
READY TO START
RECORDER
RED
REFER
REMEMBER
RETURN
REWIND
RIGHT
ROUND
S
SAID
SAVE
SAY
SAYS
SCREEN
SECOND
SEE
SEES
SET
SEVEN
SEVENTY
SHAPE
SHAPES
SHIFT
SHORT
SHORTER
SHOULD
SIDE
SIDES
SIX
SIXTY
SMALL
SMALLER
SMALLEST
SO
SOME
SORRY

List of Speech Words (continued)

SPACE
SPACES
SPELL
SQUARE
START
STEP
STOP
SUM
SUPPOSED
SUPPOSED TO
SURE
T
TAKE
TEEN
TELL
TEN
TEXAS INSTRUMENTS
THAN
THAT
THAT IS INCORRECT
THAT IS RIGHT
THE (the)
THE1(th)
THEIR
THEN
THERE
THESE
THEY
THING
THINGS
THINK
THIRD
THIRTEEN
THIRY
THIS
THREE
THREW
THROUGH
TIME
TO
TOGETHER
TONE
TOO
TOP
TRY
TRY AGAIN
TURN
TWELVE
TWENTY
TWO
TYPE
U
UHOH
UNDER
UNDERSTAND
UNTIL
UP
UPPER
USE
V
VARY
VERY
W
WAIT
WANT
WANTS
WAY
WE
WEIGH
WEIGHT
WELL
WERE
WHAT
WHAT WAS THAT
WHEN
WHERE
WHICH
WHITE
WHO
WHY
WILL
WITH
WON
WORD
WORDS
WORK
WORKING
WRITE
X
Y
YELLOW
YES
YET
YOU
YOU WIN
YOUR
Z
ZERO

APPENDIX I:

ADDING SUFFIXES TO SPEECH WORDS

This appendix describes how to add ING, S, and ED to any word available in the Solid State SpeechTM resident vocabulary.

The code for a word is first read using SPGET. The code consists of a number of characters, one of which tells the speech unit the length of the word. Then, by means of the subprograms listed here, additional codes can be added to give the sound of a suffix.

Words often have trailing-off data that make the word sound more natural but prevent the easy addition of suffixes. In order to add suffixes this trailing-off data must be removed.

The following program allows you to input a word and, by trying different truncation values, make the suffix sound like a natural part of the word. The subprograms DEFING (lines 1000 through 1130), DEFS1 (lines 2000 through 2100), DEFS2 (lines 3000 through 3090), DEFS3 (lines 4000 through 4120), DEFED1 (lines 5000 through 5070), DEFED2 (lines 6000 through 6110), DEFED3 (lines 7000 through 7130), and MENU (lines 10000 through 10120) should be input separately and saved with the MERGE option. (The subprogram MENU is the same one used in th illustrative program with SUB.) You may wish to use different line numbers. Each of these subprograms (except MENU) defines a suffix.

DEFING defines the ING sound. DEFS1 defines the S sound as it occurs at the end of "cats." DEFS2 defines the S sound as it occurs at the end of "cads." DEFS3 defines the S sound as it occurs at the end of "wishes." DEFED1 defines the ED sound as it occurs at the end of "passed." DEFED2 defines the ED sound as it occurs at the end of "caused." DEFED3 defines the ED sound as it occurs a the end of "heated."

In running the program, enter a 0 for the truncation value in order to leave the truncation sequence.

```
100 REM ******************
110 REM REQUIRES MERGE OF:
120 REM MENU (LINES 10000 THROUGH 10120)
130 REM DEFING (LINES 1000 THROUGH 1130)
140 REM DEFS1 (LINES 2000 THROUGH 2100)
150 REM DEFS2 (LINES 3000 THROUGH 3090)
160 REM DEFS3 (LINES 4000 THROUGH 4120)
170 REM DEFED1 (LINES 5000 THROUGH 5070)
180 REM DEFED2 (LINES 6000 THROUGH 6110)
190 REM DEFED3 (LINES 7000 THROUGH 7130)
```

Adding Suffixes to Speech Words (continued)

```
200 REM ********************
210 CALL CLEAR
220 PRINT "THIS PROGRAM IS USED TO"
230 PRINT "FIND THE PROPER TRUNCATION"
240 PRINT "VALUE FOR ADDING SUFFIXES"
250 PRINT "TO SPEECH WORDS.": :
260 FOR DELAY=1 TO 300::NEXT DELAY
270 PRINT "CHOOSE WHICH SUFFIX YOU"
280 PRINT "WISH TO ADD.": :
290 FOR DELAY=1 TO 800::NEXT DELAY
300 CALL MENU(8,CHOICE)
310 DATA 'ING','S' AS IN CATS,'S' AS IN CADS,'S' AS IN WISHES,
    'ED' AS IN PASSED,'ED' AS IN CAUSED,'ED' AS IN HEATED, END
320 IF CHOICE=0 OR CHOICE=8 THEN STOP
330 INPUT "WHAT IS THE WORD? ":WORD$
340 ON CHOICE GOTO 350,379,390,410,430,450,470
350 CALL DEFING(D$)
360 GOTO 480
370 CALL DEFS1(D$)!CATS
380 GOTO 480
390 CALL DEFS2(D$)!CADS
400 GOTO 480
410 CALL DEFS3(D$)!WISHES
420 GOTO 480
430 CALL DEFED1(D$)!PASSED
440 GOTO 480
450 CALL DEFED2(D$)!CAUSED
460 GOTO 480
470 CALL DEFED3(D$)!HEATED
480 REM TRY VALUES
490 CALL CLEAR
500 INPUT "TRUNCATE HOW MANY BYTES?":L
510 IF L=0 THEN 300
520 CALL SPGET(WORDS$,B$)
530 L=LEN(B$)-L-3
540 C$=SEG$(B$1,2)&CHR$(L)&SEG$(B$,4,L)
550 CALL SAY(,C$&D$)
560 GOTO 500
```

Adding Suffixes to Speech Words (continued)

The data has been given in short DATA statements to make it as easy as possible to input. The data statements may be consolidated to make the program shorter.

```
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,184,227,222,249,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR I=1 TO 55::READ A::A$=A$&CHR$(A)::NEXT I
1130 SUBEND

2000 SUB DEFS1(A$)!CATS
2010 DATA 96,0,26
2020 DATA 14,56,130,204,0
2030 DATA 223,177,26,224,103
2040 DATA 85,3,252,106,106
2050 DATA 128,95,44,4,240
2060 DATA 35,11,2,126,16,121
2070 RESTORE 2010
2080 A$=""
2090 FOR I=1 TO 29::READ A::A$&CHR$(A)::NEXT I
2100 SUBEND

3000 SUB DEFS2(A$)!CADS
3010 DATA 96,0,17
3020 DATA 161,253,158,217
3030 DATA 168,213,198,86,0
3040 DATA 223,153,75,128,0
3050 DATA 95,139,62
3060 RESTORE 3010
3070 A$=""
3080 FOR I=1 TO 20::READ A::A$=A$&CHR$(A)::NEXT I
3090 SUBEND
```

Adding Suffixes to Speech Words (continued)

```
4000 SUB DEFS3(A$)!WISHES
4010 DATA 96,0,34
4020 DATA 173,233,33,84,12
4030 DATA 242,205,166,55,173
4040 DATA 93,222,68,197,188
4050 DATA 134,238,123,102
4060 DATA 163,86,27,59,1,124
4070 DATA 103,46,1,2,124,45
4080 DATA 138,129,7
4090 RESTORE 4010
4100 A$=""
4110 FOR I=1 TO 37::READ A::A$=A$&CHR$(A)::NEXT I
4120 SUBEND

5000 SUB DEFED1(A$)!PASSED
5010 DATA 96,0,10
5020 DATA 0,224,128,37
5030 DATA 204,37,240,0,0
5040 RESTORE 5010
5050 A$=""
5060 FOR I=1 TO 13::READ A::A$=A$&CHR$(A)::NEXT I
5070 SUBEND

6000 SUB DEFED2(A$)!CAUSED
6010 DATA 96,0,26
6020 DATA 172,163,214,59,35
6030 DATA 109,170,174,68,21
6040 DATA 22,201,220,250,24
6050 DATA 69,148,162,166,234
6060 DATA 75,84,97,145,204
6070 DATA 15
6080 RESTORE 6010
6090 A$=""
6100 FOR I=1 TO 29::READ A::A$=A$&CHR$(A)::NEXT I
6110 SUBEND
```

Adding Suffixes to Speech Words (continued)

```
7000 SUB DEFED3(A$)!HEATED
7010 DATA 96,0,36
7020 DATA 173,233,33,84,12
7030 DATA 242,205,166,183
7040 DATA 172,163,214,59,35
7050 DATA 109,170,174,68,21
7060 DATA 22,201,92,250,24
7070 DATA 69,148,162,38,235
7080 DATA 75,84,97,145,204
7090 DATA 178,127
7100 DATA 7010
7110 A$=""
7120 FOR I=1 TO 39::READ A::A$=A$&CHR$(A)::NEXT I
7130 SUBEND

10000 SUB MENU(COUNT,CHOICE)
10010 CALL CLEAR
10020 IF COUNT>22 THEN PRINT "TOO MANY ITEMS" :: CHOICE=0 :: SUBEXIT
10030 RESTORE
10040 FOR I=1 TO COUNT
10050 READ TEMP$
10060 TEMP$=SEG$(TEMP$,1,25)
10070 DISPLAY AT (I,1):I;TEMP$
10080 NEXT I
10090 DISPLAY AT(I+1,1):"YOUR CHOICE: 1"
10100 ACCEPT AT(I+1,14)BEEP VALIDATE(DIGIT)SIZE(-2):CHOICE
10110 IF CHOICE<1 OR CHOICE>COUNT THEN 10100
10120 SUBEND
```

Adding Suffixes to Speech Words (continued)

You can use the subprograms in any program once you have determined the number of bytes to truncate. The following program uses the subprogram DEFING in lines 1000 through 1130 to have the computer say the word DRAWING using DRAW plus the suffix ING. Note that it was found that DRAW should be truncated by 41 characters to produce the most natural sounding DRAWING. The subprogram DEFING in lines 1000 through 1130 is the program you saved with the MERGE option.

```
100 CALL DEFING(ING$)
110 CALL SPGET("DRAW",DRAWS$)
120 L=LEN(DRAW$)-3-41! 3 BYTES OF SPEECH OVERHEAD, 41 BYTES TRUNCATED
130 DRAW$=SEG$(DRAW$,1,2)&CHR$(L)&SEG$(DRAW$,4,L)
140 CALL SAY("WE ARE",DRAW$&ING$,"A1 SCREEN")
150 GOTO 140
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,184,227,222,249,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR I=1 TO 55::READ A::A$=A$&CHR$(A)::NEXT I
1130 SUBEND
```

(Press SHIFT C to stop the program.)

APPENDIX J:

ERROR MESSAGES

The following lists all the error messages that MYARC Advanced BASIC gives. The first list is alphabetical by the message that is given, and the second list is numeric by the number of the error that is returned by CALL ERR. If the error occurs in the execution of a program, the error message is often followed by IN line-number.

Sorted by Message

#	Message	Descriptions of Possible Errors
74	BAD ARGUMENT	* Bad value given in ASC, ATN, COS, EXP, INT, LOG, SIN, SOUND, SQR, TAN, or VAL. * An array element specified in a SUB statement. * Bad first parameter or too many parameters in LINK.
61	BAD LINE NUMBER	* Line number less than 1 or greater than 32767. * Omitted line number. * Line number outside the range 1 through 32767 produced by RES.
57	BAD SUBSCRIPT	* Use of too large or small subscript in an array. * Incorrect subscript in DIM.
79	BAD VALUE	* Incorrect value given in AND, CHAR, CHR$, CLOSE, EOF, FOR, GOSUB, GOTO, HCHAR, INPUT, MOTION, NOT, OR, POS, PRINT, PRINT USING, REC, RESTORE, RPT$, SEG$, SIZE, VCHAR, or XOR. * Array subscript value greater than 32767. * File number greater than 255 or less than zero. * More than three tones and one noise generator specified in SOUND. * A value passed to a subprogram is not acceptable in the subprogram. For example, a sprite velocity value less than -128 or a character value greater than 143. * Value in ON...GOTO or ON...GOSUB greater than the number of lines given. * Incorrect position given after the AT clause in ACCEPT or DISPLAY.
67	CAN'T CONTINUE	* Program has been edited after being stopped by a breakpoint. * Program was not stopped by a breakpoint.
69	COMMAND ILLEGAL IN PROGRAM	* BYE, CON, LIST, MERGE, NEW, NUM, OLD, RES, or SAVE used in a program.
84	DATA ERROR	* READ or RESTORE with data not present or with a

string where a number value is expected.
* Line number after RESTORE is higher than the highest line number in the program.
* Error in object file in LOAD.

109 FILE ERROR
* Wrong type of data read with a READ statement.
* Attempt to use CLOSE, EOF, INPUT, OPEN, PRINT, PRINT USING, REC, or RESTORE with a file that does not exist or does not have the proper attributes.
* Not enough memory to use a file.

44 FOR-NEXT NESTING
* The FOR and NEXT statements of loops do not align properly.
* Missing NEXT statement.

130 I/O ERROR
* An error was detected in trying to execute CLOSE, DELETE, LOAD, MERGE, OLD, OPEN, RUN, or SAVE.
* Not enough memory to list a program.

16 ILLEGAL AFTER SUBPROGRAM
* Anything but END, REM, or SUB after a SUBEND.

36 IMAGE ERROR
* An error was detected in the use of DISPLAY USING, IMAGE, or PRINT USING.
* More than 10 (E-format) or 14 (numeric format) significant digits in the format string.
* IMAGE string is longer than 254 characters.

28 IMPROPERLY USED NAME
* An illegal variable name was used in CALL, DEF, or DIM.
* Using a MYARC Advanced BASIC reserved word in LET.
* Using a subscripted variable or a string variable in a FOR.
* Using an array with the wrong number of dimensions.
* Using a variable name differently than originally assigned.
  A variable can be only an array, a numeric or string variable, or a user defined function name.
* Dimensioning an array twice.
* Putting a user defined function name on the left of the equals sign in an assignment statement.
* Using the same variable twice in the parameter list of a SUB statement.

81 INCORRECT ARGUMENT LIST
* CALL and SUB mismatch of arguments.

83 INPUT ERROR
* An error was detected in an INPUT.

60 LINE NOT FOUND
* Incorrect line number found in BREAK, GOSUB, GOTO, ON ERROR, RUN, or UNBREAK, or after THEN or ELSE.
* Line to be edited not found.

62 LINE TOO LONG
* Line too long to be entered into a program.

39 MEMORY FULL
* Program too large to execute one of the following: DEF, DELETE, DIM, GOSUB, LET, LOAD, ON...GOSUB, OPEN, or SUB.
* Program too large to add a new line, insert a line, replace a line, or evaluate an expression.

49 MISSING SUBEND
* SUBEND missing in a subprogram.

47 MUST BE IN SUBPROGRAM
* SUBEND or SUBEXIT not in a subprogram.

19 NAME TOO LONG
* More than 15 characters in variable or subprogram name.

43 NEXT WITHOUT FOR
* FOR statement missing, NEXT before FOR, incorrect FOR-NEXT nesting, or branching into a FOR-NEXT loop.

78 NO PROGRAM PRESENT
* No program present when issuing a LIST, RESEQUENCE, RESTORE, RUN, or SAVE command.

10 NUMERIC OVERFLOW
* A number too large or too small resulting from a *,+,-,/ operation or in ACCEPT, ATN, COS, EXP, INPUT, INT, LOG, SIN, SQR, TAN, or VAL.
* A number outside the range -32768 to 32767 in PEEK or LOAD.

70 ONLY LEGAL IN A PROGRAM
* One of the following statements was used as a command: DEF, GOSUB, GOTO, IF, IMAGE, INPUT, ON BREAK, ON ERROR, ON...GOSUB, ON...GOTO, ON WARNING, OPTION BASE, RETURN, SUB, SUBEND,or SUBEXIT.

25 OPTION BASE ERROR
* OPTION BASE executed more than once, or with a value other than 1 or zero.

97 PROTECTION VIOLATION
* Attempt to save, list, or edit a protected program.

48 RECURSIVE SUBPROGRAM CALL
* Subprogram calls itself, directly or indirectly.

51 RETURN WITHOUT GOSUB
* RETURN without GOSUB or an error handled by the previous execution of an ON ERROR statement.

56 SPEECH STRING TOO LONG
* Speech string returned by SPGET is longer than 255 characters.

40 STACK OVERFLOW
* Too many sets of parentheses.
* Not enough memory to evaluate an expression or assign a value.

54 STRING TRUNCATED
* A string created by RPT$, concatenation ("&" operator), or a user defined function is longer than 255 characters.
* The length of a string expression in the VALIDATE clause is greater than 254 characters.

24 STRING-NUMBER MISMATCH
* A string was given where a number was expected or vice versa in a MYARC Advanced BASIC supplied function or subprogram.
* Assigning a string value to a numeric value or vice versa.
* Attempting to concatenate ("&" operator) a number.
* Using a string as a subscript.

135 SUBPROGRAM NOT FOUND
* A subprogram called does not exist or an assembly language subprogram named in LINK has not been loaded.

14 SYNTAX ERROR
* An error such as a missing or extra comma or parenthesis, parameters in the wrong order, missing parameters, missing keyword , misspelled keyword, keyword in the wrong order, or the like was detected in a MYARC Advanced BASIC command, statement, function, or subprogram.
* DATA or IMAGE not first and only statement on a line.
* Items after final ")".
* Misssing "#" in SPRITE.
* Missing ENTER, tail comment symbol (!), or statement separator symbol (::).
* Missing THEN after IF.
* Missing TO after FOR.
* Nothing after CALL, SUB, FOR, THEN, or ELSE.
* Two E's in a numeric constant.
* Wrong parameter list in a MYARC Advanced BASIC supplied subprogram.
* Going into or out of a subprogram with GOTO, GOSUB, ON ERROR, etc.
* Calling INIT without the Memory Expansion peripheral attached.
* Calling LINK or LOAD without first calling INIT.
* Using a constant where a variable is required.
* More than seven dimensions in an array.

17 UNMATCHED QUOTES
* Odd number of quotes in an input line.

20 UNRECOGNIZED CHARACTER
* An unrecognized character such as ? or % is not in a quoted string.
* A bad field in an object file accessed by LOAD.

Error Messages (concluded)

Sorted by #

#	Message
10	NUMERIC OVERFLOW
14	SYNTAX ERROR
16	ILLEGAL AFTER SUBPROGRAM
17	UNMATCHED QUOTES
19	NAME TOO LONG
20	UNRECOGNIZED CHARACTER
24	STRING-NUMBER MISMATCH
25	OPTION BASE ERROR
28	IMPROPERLY USED NAME
36	IMAGE ERROR
39	MEMORY FULL
40	STACK OVERFLOW
43	NEXT WITHOUT FOR
44	FOR-NEXT NESTING
47	MUST BE IN SUBPROGRAM
48	RECURSIVE SUBPROGRAM CALL
49	MISSING SUBEND
51	RETURN WITHOUT GOSUB
54	STRING TRUNCATED
56	SPEECH STRING TOO LONG
57	BAD SUBSCRIPT
60	LINE NOT FOUND
61	BAD LINE NUMBER
62	LINE TOO LONG
67	CAN'T CONTINUE
69	COMMAND ILLEGAL IN PROGRAM
70	ONLY LEGAL IN A PROGRAM
74	BAD ARGUMENT
78	NO PROGRAM PRESENT
79	BAD VALUE
81	INCORRECT ARGUMENT LIST
83	INPUT ERROR
84	DATA ERROR
97	PROTECTION VIOLATION
109	FILE ERROR
130	I/O ERROR
135	SUBPROGRAM NOT FOUND

APPENDIX K

GRAPHICS MODES - Summary

GRAPHICS MODE	SCREEN DIMEN.	SCREEN SIZE	DEFAULT MARGINS	MODE NAME	NO.OF PATNS.	COLORS PER SCREEN	PATTERN SIZE	SPRITE MODE	MEMORY/ SCREEN
1,1	256,192	32x24	3,30 1,24	Pattern Graphic1	256	16	8 x 8	1 4/line	4K/scr 32 pgs
1,2	256,192	32x24	3,30 1,24	Graphic2	768	16	8 x 8	1 4/line	16K/scr 8 pgs
1,3	256,192	32x24	3,30 1,24	Graphic3	768	16	8 x 8	2 8/line	16K/scr 8 pgs
2,1	256,192	40x24	1,40 1,24	Text- 1	256	2	6 x 8	None	4K/scr 3 pgs
2,2	256x212	40x26	1,40 1,24	Bitmap-1	???	16	6 x 8	2 8/line	32K/scr 4 pgs
2,3	256,212	40x26	1,40 1,24	Bitmap-4	???	256	6 x 8	2 8/line	64K/scr 2 pgs
3,1	512,212	80x26	1,80 1,24	Text- 2	256	2+2	6 x 8	None	8K/scr 16 pgs
3,2	512x212	80x26	1,80 1,24	Bitmap-2	???	4	6 x 8	2 8/line	32K/scr 4 pgs
3,3	512x212	80x26	1,80 1,24	Bitmap-3	???	16	6 x 8	2 8/line	64K/scr 2 pgs

APPENDIX L

PROGRAM - ILLUSTRATING MOUSE COMMANDS

The following program illustrates the use of several MOUSE Commands to draw lines on the screen. Press MOUSE button 1 to start drawing a line and hold it down until you are done drawing.

```
100 CALL GRAPHICS(2,3) :: REM 256 COLOR BIT MAPPED MODE
110 CALL SPRITE(#1,33,16,1,1) :: REM DEFINE MOUSE AS !
120 CALL SEEMOUSE :: REM MAKE SURE MOUSE IS VISIBLE ON SCREEN
130 CALL MOUSE(Y,1) :: REM TEST FOR BUTTON PRESS
140 IF Y=0 THEN 130 :: REM WAIT FOR A BUTTON PRESS
150 CALL MOUSEDRAG(ON) :: REM BUTTON PRESSED SO START DRAWING
160 CALL MOUSE(Y,1) :: REM TEST BUTTON STATUS
170 IF Y=1 THEN 160 :: REM DRAW UNTIL RELEASED
180 CALL MOUSEDRAG(OFF) :: REM STOP DRAWING WHEN RELEASED
190 GO TO 130 :: REM GO TO WAIT FOR NEXT BUTTON PRESS
```

APPENDIX M:

ADDITIONAL EXTENDED ASCII CODES FOR KEYBOARD MODE 6

In addition to the normal ASCII codes returned in keyboard mode 5, the following additional Extended Codes are also returned in keyboard mode 6:

EXTENDED CODE(HEX)	FUNCTION
3	NUL Character
F	Back Arrow
10-19	ALT Q,W,E,R,T,Y,U,I,O,P
1E-26	ALT A,S,D,F,G,H,J,K,L
2C-32	ALT Z,X,C,V,B,N,M
3B-44	F1-F10 Function Keys (Base Case)
47	Home
48	Up Arrow
49	Page Up
4B	Left Arrow
4D	Right Arrow
4F	End
50	Down Arrow
51	Page Down
52	INS
53	DEL
54-5D	F11-F20 (Upper Case F1-F10)
5E-67	F21-F30 (CTRL F1-F10)
68-71	F31-F40 (ALT F1-F10)
72	CTRL PRTSC(Start/Stop Echo to Printer)
73	CTRL Right Arrow (Reverse Word)
74	CTRL Left Arrow (Advance Word)
75	CTRL END (Erase to End of Line)
76	CTRL PG DN (Erase to End of Screen)
77	CTRL HOME (Clear Screen and Home)
78-83	ALT 1,2,3,4,5,6,7,8,9,0,-,=
84	CTRL PG UP (Top 25 Lines of Text and Home Cursor)

- continue on next page -

The following combinations of keys produce special effects:

- ALT CTRL DEL makes the keyboard routine initiate the equivalent of a system reset/reboot.
- CTRL BREAK makes the keyboard routine invoke the (Keyboard break) interrupt.
- CTRL NUM-LOCK makes the keyboard routine wait for you to press any key but NUM-LOCK. This gives you a way to suspend an operation temporarily, then resume.
- SHIFT PRTSC makes the keyboard routine invoke the Print Screen Interrupt.
- The keyboard treats the following keys as a group, rather than individually: CTRL, SHIFT, NUM-LOCK, CAPS-LOCK, and INS. The service routine for the Keyboard I/O interrupt returns a "shift status" byte that tells you when one of these keys are pressed.