ADAM™ SmartBASIC™

COMPENDIUM OF USEFUL PROGRAMMING INFORMATION

(APPENDIX C)

TABLE OF CONTENTS

ERROR MESSAGES

Run Time Error Messages	ONERR GOTO Codes

BAD SUBSCRIPT (107)

You've tried to reference an array element that's outside the array's dimensions.

BREAK (255)

This message appears when you use CONTROL-C to interrupt a program, or you have a STOP in your program.

CAN'T CONTINUE

You tried to continue a program that doesn't exist, after an error occurred, or after you removed or inserted a line in a program.

DIVIDE BY ZERO (133)

Division by zero is not acceptable.

FATAL SYSTEM ERROR

Your program is corrupted. Type NEW or reboot SmartBASIC.

ILLEGAL FUNCTION ASSIGNMENT (16)

You tried to use a function in an INPUT or READ statement.

ILLEGAL MODE

You can't use DATA, GET, DEF FN, or INPUT in immediate execution mode.

ILLEGAL QUANTITY (53)

This error can be caused by: using LOG with a negative or zero argument; using SQR with a negative argument; or using LEFT$, RIGHT$, MID$, WAIT, POKE, PEEK, TAB, SPC, ON/GOTO, or any graphics function with an inappropriate argument.

NEXT WITHOUT FOR (0)

You typed in NEXT on your program and have omitted the corresponding FOR. Always pair FOR with NEXT.

OUT OF DATA (42)

ADAM is trying to execute a READ statement when all the data has been read. You haven't provided enough data, or your program tried to read too much data.

OUT OF MEMORY (77)

This may be caused by: a program that is too large; excessive variables; more than 14 nested FOR loops; more than 30 nested GOSUB levels; too complicated an expression; setting LOMEM: too high or too low; or setting HIMEM: too high.

OVERFLOW (69)

This results when a calculation answer is too large for ADAM to handle. An underflow will result if the calculation answer is too small for ADAM to handle. In this case, a zero will be substituted for the correct result, and no error message will appear.

REDIMENSIONED ARRAY (120)

After you dimensioned an array, ADAM encountered another dimension statement for the same array.

REENTER (254)

You made an inappropriate response to an INPUT.

RETURN WITHOUT GOSUB (22)

ADAM encountered a RETURN without a corresponding GOSUB statement.

STACK OVERFLOW (77)

You've taken up too much room in your stack by using too many FOR/NEXT statements or GOSUB statements. Too many subroutines will fill your stack to the extent that you need to POP information from the top before you can push any more in from the bottom.

STRING TOO LONG (176)

You've put together a string that has more than 255 characters.

SYNTAX (16)

Check to see if you are missing parentheses, have an illegal character in a line, or incorrect punctuation, etc.

TYPE MISMATCH (163)

You've given a function or variable which expected a numeric argument, a string argument, or vice versa.

UNDEFINED FUNCTION (224)

You tried to use FN for a function that you have not yet defined. See DEF FN.

UNDEFINED STATEMENT (90)

You tried to send a GOTO, GOSUB, or THEN to a line number which doesn't exist.

File Error Messages

CONTROL BUFFER OVERFLOW (12)

You have exceeded the fixed size limit of your input buffer. You have probably used too many characters following a CONTROL-D.

END OF DATA (5)

You've tried to read or write past the end of your data file.

FILE LOCKED (10)

The file to which you are trying to write is locked. Use CATALOG to see which files are locked. Look for filenames with asterisks in front of them. To release a file, see UNLOCK and RECOVER.

FILE NOT FOUND (7)

ADAM can't find the file using the name you've input. Check your spelling (especially the way you used upper and lower case letters). Type CATALOG to be sure the file is on your digital data pack.

FILE TYPE MISMATCH (13)

You tried to run a binary file.

I/O ERROR (8)

This is an input/output error. Be sure your digital data pack is firmly in place.

NO BUFFERS AVAILABLE (12)

You've run out of buffers because you have too many data files open.

NO MORE ROOM (9)

There is no more file space left on your digital data pack. The directory will only hold 35 files. You may have an unclosed data file. See CLOSE.

RANGE ERROR (2)

You've exceeded your available range by making a command parameter too large for ADAM to deal with.

SYNTAX ERROR (11)

You've used a bad file name, wrong parameter, or wrong punctuation in an OS command.

FOR USERS WITH TWO DIGITAL DATA PACK DRIVES

You can use either drive for SmartBASIC. When you reset ADAM, the computer first looks for a digital data pack in the left drive (D1). If one is found, ADAM expects it to contain SmartBASIC. If there is no tape in the first drive, or the door is open, ADAM tries to boot the drive on the right (D2). The left drive becomes the default drive. If a drive isn't specified, every OS command automatically goes to the default drive. Every time you use an OS command with the drive specified, that drive becomes the default drive.

DIGITAL DATA PACKS

1. Don't store your programs or data files on the SmartBASIC digital data pack. Take it out and put it away as soon as SmartBASIC is loaded. Use a blank pre-formatted digital data pack to store your programs and files.

2. Do yourself a favor. If you're working on a long program, SAVE it every 15 or 20 minutes. Use a new version number each time. This way, if your power goes off unexpectedly, you won't lose everything you've input. You'll only lose what's been entered since the last time you SAVEd. If your digital data pack starts to get full, then DELETE the earliest SAVEd versions.

3. Make extra copies of important programs and data files on a separate digital data pack. Keep one "working" copy, then store the other in a safe place -- away from your "working" copy so you won't get confused.

4. Digital Data Packs are specially designed for your ADAM computer. Although they may look like audio cassette tapes, digital data packs are very different. Audio cassette tapes cannot be used in place of digital data packs. If the tape in a digital data pack breaks, do not splice the ends

together and try to re-use the data pack. To erase information on a digital data pack, delete unwanted files. Never use a bulk tape eraser--if you do, you'll erase the special format that makes the data pack unique for ADAM. Digital data packs cannot be write-protected.

Keep your digital data packs away from magnets. Don't put them on top of the printer; there's a magnet inside. Store your digital data pack in a safe place, away from dust, temperature extremes, electrical currents, and water. Don't open the drive door while the tape is in motion. Never press the RESET button while ADAM is storing a file. Do not store a digital data pack on or near a television or monitor. Keep your data packs away from heat and sunlight. Keep spare digital data packs in their original plastic cases when they are not in use.

SEQUENTIAL TEXT FILES

Sequential text files are information storage files where data records follow one right after another, in sequence.

To create a sequential text file, always begin with OPEN, then follow it with WRITE. All PRINTs will now go to the file until the WRITE is cancelled. To cancel a WRITE command, PRINT CONTROL-D (PRINT CHR$(4)).

The sample program which follows will create a sequential text file called SESAME. The first 13 records contain three strings and the numbers 1-10.

```
10 D$=CHR$(4)
20 PRINT D$; "OPEN SESAME"
50 PRINT D$; "WRITE SESAME"
60 PRINT "HEY CHIP": PRINT "LET'S STEP OUT"
70 PRINT "FOR A BYTE"
80 FOR J=1 TO 10
90 PRINT J: NEXT J
100 PRINT D$; "CLOSE SESAME"
```

BEWARE:: If you OPEN a file already existing and WRITE to it, you'll overwrite part of your original file. Use APPEND to add to files.

To retrieve the file, SESAME, one record at a time, here's what to do:
(and to see what's going on, type MON I)

```
10 D$=CHR$(4)
20 PRINT D$;"OPEN SESAME"
30 PRINT D$;"READ SESAME"
```

```
40 INPUT A$,B$,C$
50 FOR I=1 TO 10
60 INPUT W(I)
70 NEXT I
80 PRINT D$; "CLOSE SESAME"
```

OPEN must come before READ. After the READ, all INPUT comes from the file. You can cancel a READ command by PRINT CHR$(4). Don't forget to CLOSE your file when you're done.

To add data to a sequential text file, try this type of program:

```
10 D$=CHR$(4)
20 PRINT D$; "APPEND SESAME"
40 PRINT "NO, THANKS"
50 PRINT "I'VE A BIT OF A HEADACHE."
60 PRINT D$; "CLOSE SESAME"
```

Each string is an additional record of the file.

RANDOM ACCESS (Fixed Length) TEXT FILES

Think of random access text files as a series of equal-sized pigeon holes in a desk. Each pigeon hole is called a "record".

Random access text files differ from sequential text files in the fact that random access text records must be of a fixed length, where sequential text records may be of any length. The drawback is that when you WRITE to a random access text file, enough space is set aside for a complete record, whether you fill that entire space or not. From this, you see that, while random access text files may not represent the best usage of available space, the files are arranged in such an orderly fashion that it's a quick and easy procedure to recall and edit information from any part of your file!

Use random access text files when you want:

1. fast access to different parts of your files

2. to change pieces of information in your files

fairly frequently (mailing lists, name, address, phone number files, etc.).

The procedure to create or retrieve random access text files is similar to that used for sequential text files. Here are the slight differences you should know about:

OPEN needs a length parameter specified.(maximum length is 255).

READ needs a record parameter specified.

WRITE needs a record parameter specified.

Here's a sample program for you:

```
10 D$=CHR$(4)
20 INPUT "NAME:      ";N$
30 INPUT "PHONE:     ";P$
40 PRINT D$;"OPEN PHONEIND,L200"
50 PRINT D$;"WRITE PHONEIND,R1"
60 PRINT N$: PRINT P$
70 PRINT D$;"CLOSE PHONEIND"
MON C,I,O
```

You'll see this onscreen: (You type the underlined words.

```
NAME: EARLE W. MUNSON
PHONE: (203) 263-3292
OPEN PHONEIND,L200
WRITE PHONEIND,R1
EARLE W. MUNSON
(203) 263-3292
CLOSE PHONEIND
```

Now, if you want to get the first record of PHONEIND, use this:

```
10 D$=CHR$(4)
20 PRINT D$;"OPEN PHONEIND,L200"
30 PRINT D$;"READ PHONEIND,R1"
40 INPUT N1$,P1$
50 PRINT D$;"CLOSE PHONEIND"
]MON C,I,O
```

You'll see this onscreen:

```
OPEN PHONEIND,L200
READ PHONEIND,R1
EARLE W. MUNSON
(203) 263-3292
CLOSE PHONEIND
```

TURNKEY SYSTEM

You can create a turnkey system on ADAM. This is a system that runs the same initial program every time a digital data pack is booted for SmartBASIC. You may use any sort of program that does any task; but most people use what's called a "greeting program". A greeting program will make ADAM seem more human by doing pretty much what the program category implies. . . by greeting you. A greeting program may be as long or as short as you choose to make it; from "HELLO" to an involved conversation which requires input from you to answer questions ADAM poses. Save your program using the name HELLO.

```
10 PRINT,"HELLO, HUMAN"
20 GOTO 20
SAVE HELLO
```

Hit RESET to put you back into SmartBASIC and rewind your tape. HELLO is in your directory on your SmartBASIC tape. SmartBASIC READs your directory and looks for the HELLO file. When it is found, it is LOADed and executed every time you boot SmartBASIC. Only the program named "HELLO" is run when the SmartBASIC tape is booted. If no "HELLO" file is found, ADAM turns control over to you.

ASCII CHARACTER CODES

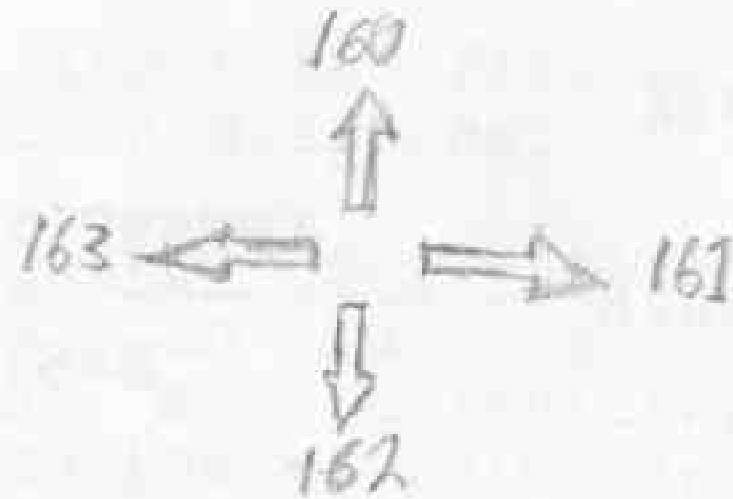
10 GET A$
20 IF ASC(A$) = 3 THEN END
30 Print A$; Goto 10

DEC	HEX	CHARACTER TYPED	MEANING
0	00	CONTROL-@	
1	01	CONTROL-A	
2	02	CONTROL-B	
3	03	CONTROL-C	
4	04	CONTROL-D	
5	05	CONTROL-E	
6	06	CONTROL-F	
7	07	CONTROL-G	bell
8	08	CONTROL-H	backspace
9	09	CONTROL-I TAB	horiz. tab
10	0A	CONTROL-J	line feed
11	0B	CONTROL-K	
12	0C	CONTROL-L	clear screen
13	0D	CONTROL-M Return	
14	0E	CONTROL-N	
15	0F	CONTROL-O	
16	10	CONTROL-P	dumps screen to printer
17	11	CONTROL-Q	
18	12	CONTROL-R	
19	13	CONTROL-S	pause
20	14	CONTROL-T	
21	15	CONTROL-U	
22	16	CONTROL-V	
23	17	CONTROL-W	
24	18	CONTROL-X	
25	19	CONTROL-Y	
26	1A	CONTROL-Z	
27	1B	CONTROL-[ESCAPE/WP	
28	1C	CONTROL-\	
29	1D	CONTROL-]	
30	1E	CONTROL-^	
31	1F	CONTROL-_	
32	20	SPACE	

DEC	HEX	CHARACTER TYPED	MEANING
73	49	I	
74	4A	J	
75	4B	K	
76	4C	L	
77	4D	M	
78	4E	N	
79	4F	O	
80	50	P	
81	51	Q	
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	
92	5C	\	
93	5D	]	
94	5E	^	
95	5F	_	
96	60	`	
97	61	a	
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	
103	67	g	
104	68	h	
105	69	i	
106	6A	j	
107	6B	k	
108	6C	l	
109	6D	m	
110	6E	n	
111	6F	o	
112	70	p	

DEC	HEX	CHARACTER TYPED	MEANING
33	21	!	
34	22	"	
35	23	#	
36	24	$	
37	25	%	
38	26	&	
39	27	'	
40	28	(	
41	29	)	
42	2A	*	
43	2B	+	
44	2C	,	
45	2D	-	
46	2E	.	
47	2F	/	
48	30	0	
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	
57	39	9	
58	3A	:	
59	3B	;	
60	3C	<	
61	3D	=	
62	3E	>	
63	3F	?	
64	40	@	
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	
72	48	H	

DEC	HEX	CHARACTER TYPED	MEANING
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	
119	77	w	
120	78	x	
121	79	y	
122	7A	z	
123	7B	bracket (left)	
124	7C	broken vertical line	
125	7D	bracket (right)	
126	7E	tilde	
127	7F	DELETE	
128		HOME	
129		Function I	
130		Function II	
131		Function III	
132		Function IV	
133		Function IV	
134		Function VI	



SHIFT with Funct. keys
137 - 142

144		WILDCARD	
145		UNDO	
146		COPY	
147		GET	
148		INSERT	
149		PRINT	
150		CLEAR	
151		DELETE	
152		WILDCARD(SHIFT)	
153		UNDO(SHIFT)	
154		MOVE(SHIFT)	
155		STORE(SHIFT)	
156		INSERT(SHIFT)	
157		PRINT(SHIFT)	
158		DELETE(SHIFT)	

BASIC MEMORY MAP

```
     HEX                                    DEC

              INTERUPT VECTORS
     100H-----------------------------256
                  BASIC INTERPRETER


     6B0FH----------------------------27407
     *                              (LOMEM:)
     *            SYMBOL TABLE
     *            NUMERIC VARIABLES
     *            ARRAY DESCRIPTIONS
*user RAM
     *
     *            STRING VARIABLES
     *   ---------------------------
     *            TOKENIZED USER
     *                PROGRAM       (HIMEM:)
     *
     *
     *
     *
     D180H----------------------------53632
                  STACK
     D390H----------------------------54160


                  OPERATING SYSTEM
```

PLEASE NOTE THAT POKE DOES NOT ALLOW POKING ABOVE D390H (IN DECIMAL VALUES D390H=54160).

A shape table is used to input and file shapes for use later in high resolution graphics.

Now we're getting complicated. Now you're going to need a little extra help in the form of a hexadecimal calculator, or "hex" calculator; or you can program ADAM to do the calculation. You will not be able, for all practical purposes, to figure out hex values for various shapes without one. Hex values are an easier way to interpret and represent binary numbers.

Plotting Vectors

It is necessary to plot vectors in order to define your shape. Each byte (composed of 8 bits) in a shape definition, is divided into 3 sections. Each section designates a plotting vector...whether to move up, down, left, or right, or whether to plot a point at all. ADAM knows your shape is finished when it reaches a vector of eight zeros.

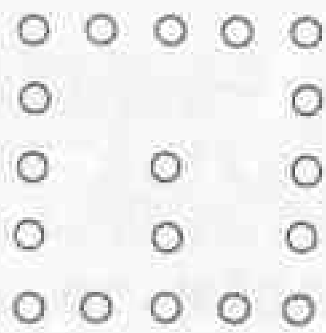
	Sec. C		Sec. B			Sec. A		
Bit #	7	6	5	4	3	2	1	0
	D	D	P	D	D	P	D	D

DD=direction
P=point
IF DD=00 then move up
IF DD=01 then move right
IF DD=11 then move left
IF DD=10 then move down

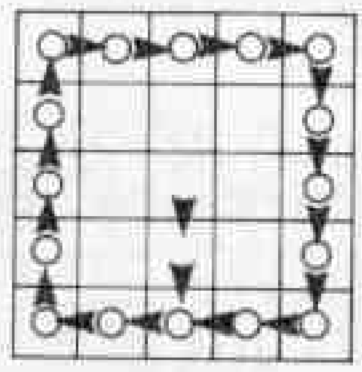
IF P=0 then don't plot a point
IF P=1 then plot a point

Note that Section C has no P in it; therefore P=0 is assumed. Section C can only specify a direction.

Here's a sample shape:



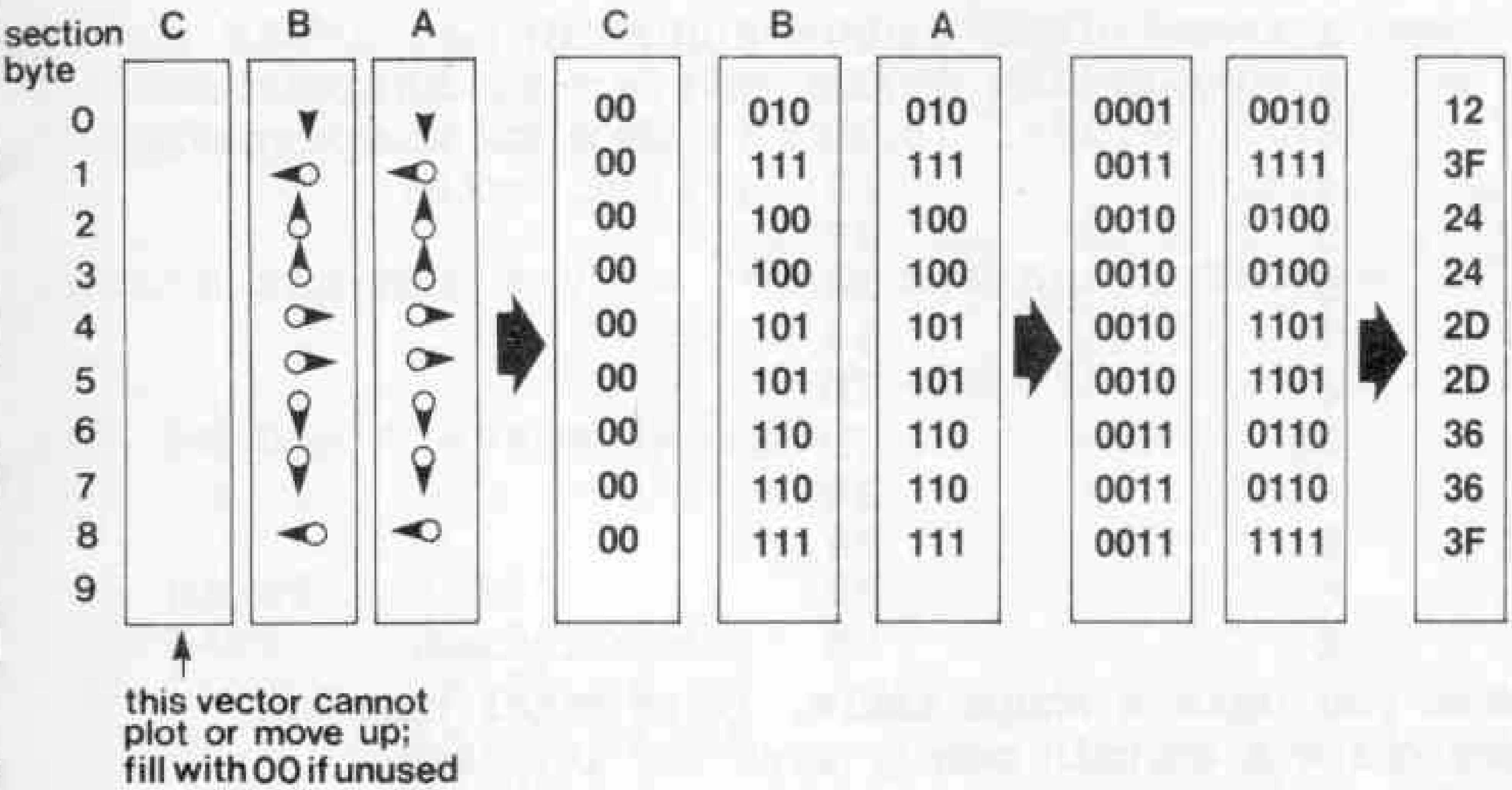
Draw it on a piece of graph paper. Keep one dot per square. Decide on where to begin your shape. We chose the center. Draw a path around your shape with arrows. These arrows are called plotting vectors.



Next, unwrap the vectors from your shape, and write them out in a straight line. Be sure to include the points on the backs of the arrows which have them.

Next, transfer these vectors into a table like this one:

vector	code	
▲	000	move only
►	001 or 01	
▼	010 or 10	
◄	011 or 11	
○▲	100	plot and move
○►	101	
○▼	110	
◄○	111	



Now, another table. Now you need to recode your vector information into hexadecimal bytes. Use the hexadecimal codes listed here:

CODES

Binary	Hexadecimal	Decimal
1111	F	15
1110	E	14
1101	D	13
1100	C	12
1011	B	11
1010	A	10
1001	9	9
1000	8	8
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

You're almost done. Convert your data into this form:

BYTE 0 01 (number of shapes)

1	00	(unused)
2	04	(index to shape def)
3	00	(index to shape def)
4	12	(first byte)
5	3F	
6	24	
7	24	
8	2D	
9	2D	
A	36	
B	36	
C	3F	
D	00	(last byte)

When you input a shape table, it is vital that you designate a certain memory area for it. (see HIMEM: and LOMEM:) For example: 10 HIMEM: 51455.

And now, your shape table!

C900,01	20 POKE 51456,01
C901,00	30 POKE 51457,00
C902,04	40 POKE 51458,04
C903,00	50 POKE 51459,00
C904,12	60 POKE 51460,18
C905,3F	70 POKE 51461,63
C906,24	80 POKE 51462,36
C907,24	90 POKE 51463,36
C908,2D	100 POKE 51464,45
C909,2D	110 POKE 51465,45
C90A,36	120 POKE 51466,54
C90B,36	130 POKE 51467,54
C906,3F	140 POKE 51468,63
C90D,00	150 POKE 51469,00

The second column is the one you'll actually be typing in. The first column is the hex values of the second.

Now, we must tell SmartBASIC where the shape table is located in memory. This is done using two POKEs:

```
160 POKE 16766,0
170 POKE 16767,201
175 HGR:HCOLOR=3
180 SCALE=10
190 FOR i=1 to 64
200 ROT=i
210 DRAW 1 AT 125,85
220 XDRAW 1 AT 125,85
230 NEXT
```

Using BSAVE and BLOAD, you can save memory image of the shape table. See BSAVE and BLOAD in the Reference sections for further information.

Here's another way to input the Shape Table:

How to Use a Shape Table

```
  5 HIMEN  :51455
 10 DATA    01,00,04,00
 20 DATA    18,63,36,36
 30 DATA    45,45,54,54
 40 DATA    63,00
 50 FOR i  = 0 TO 13
 60 READ a
 70 POKE 51456+i, a
 80 NEXT
 90 POKE 16766, 0
100 POKE 16767, 201
110 HGR
120 FOR c = 1 TO 15
130 HCOLOR  = c
140 FOR i = 0 TO 32
150 ROT  = i: SCALE  = (i+2)*.9
160 DRAW 1 AT 125, 95
170 NEXT: NEXT
180 GOTO 120
```

Gorgeous!!

To interrupt this floor show, use CONTROL-C.

Lines 90 and 100 contain the location of the shape table in memory in a converted form. The table starts at 51456 decimal. This is C900 hex. The hex value is separated into two bytes. The least significant byte is 0 hex or 0 decimal. The most significant byte is C9 hex or 201 decimal. The decimal bytes are stored least first, most second.

READ ONLY MEMORY (ROM)

ROM contains the programs which enable ADAM to understand and act on commands you type in at the keyboard. Unlike RAM, ROM's contents never change -- even if the power is turned off. It's sort of like a sleeping person's personality...though it is not in evidence while asleep, it still exists, unchanged.

RANDOM ACCESS MEMORY (RAM)

RAM is a read/write memory. Its contents change constantly, depending upon which tasks you're currently using ADAM to perform. RAM works only as long as the power is on. When you turn ADAM off, read/write memory data disappears. The programs in RAM are classified as application programs.

ADAM comes with 80K RAM. After SmartBASIC is loaded, approximately 25950 bytes are available for programs and variables.

If you have the 64K memory expander, see the owner's manual that comes with it.

This program will give you the address locations where many useful settings and pointers are stored. Most locations are two byte addresses with the least significant byte first. This information requires advanced knowledge not covered in this manual. Make sure you know what you are doing before you try to use it.

```
 10 PRINT "  Put paper in printer":  PRINT
 20 PRINT "  Hit any key when ready.":  GET q$
 30 PR#1
100 IF PEEK (259) = 205 GOTO 140
120 address = (PEEK(257)+PEEK(258)*256)+54
130 GOTO 150
140 address = PEEK(260)+PEEK(261)*256
150 FOR i = 1 TO 13
160 READ desc$, offset
170 PRINT desc$; " is at "; address+offset
180 NEXT
190 PR#0: END
200 DATA     "Himem setting",0,"Lomem setting",6
210 DATA     "Pointer to start of numeric values",
              10
220 DATA     "Pointer to end of numeric values",20
230 DATA     "Pointer to start of string space",22
240 DATA     "Pointer to end of string space",26
250 DATA     "Line number where ONERR will GOTO,37
260 DATA     "Speed setting",40
270 DATA     "USR function address",41
280 DATA     "Floating point accumulator",73
290 DATA     "Floating point operand",82
300 DATA     "Ampersand routine address",43
310 DATA     "Number of significant digits on
              output",89
```

GLOSSARY OF TERMS

ADDRESS - a number used to identify memory location.

ARGUMENT - the value a function operates on.

ARRAY - a variable collection distinguished through use of numerical subscripts and referred to by the same name.

ASCII - acronym for American Standard Code for Information Interchange. Comprised of numbers ranging from 0 to 127 which stand for various keyboard characters or operations.

ASSEMBLY LANGUAGE - a low-level programming language which is so close to the actual machine language that ADAM uses internally, that programs can be executed almost directly because the computer understands it so well.

BINARY - representing numbers in powers of 2, using digits 0 and 1.

BINARY FILE - file whose information is still in "raw" form - not expressed as text.

BIT - a binary digit (0 or 1). The smallest possible unit of information.

BOOT - starting up ADAM by loading a program into memory from a digital data pack.

BRANCH - to send program execution to a line out of program sequence.

BUFFER - a reserved area of memory for special information manipulation. In a way, it's a "holding area" for information in transit.

BUG - a programming error.

BYTE - a unit of information composed of 8 bits. Its value range may be from 0 to 255.

CHARACTER - any symbol used in displaying or printing information in a form readable by a human being (e.g a letter, digit, punctuation mark, etc.)

CHARACTER CODE - a number used in place of a character to facilitate processing by ADAM.

COMMAND - a word you type in which directs ADAM to perform an immediate action.

CONCATENATE - to chain together strings.

CONDITIONAL BRANCH - a branch which depends on the truth or value of a condition or expression.

DEFAULT - a pre-programmed value, setting, or action which the computer automatically switches to when no other specific information has been given.

DEFERRED EXECUTION - using line numbers when you type out your program. This postpones program execution until you type RUN.

DIMENSION - the maximum size allowed to one of the array subscripts.

DIRECTORY - a listing of all files on your digital data pack.

DIGITAL DATA PACK DRIVE - the device where you put your digital data pack in order to use ADAM. The Drive reads the magnetic tape and writes information onto it, if instructed to do so.

DISPLAY - information exhibited on the screen of a display device.

DISPLAY DEVICE - anything which exhibits information visually (e.g. television screen, monitor, etc.).

EDIT - changes or modifications made to a document (e.g. insert, delete, replace, move, etc.).

ELEMENT - an individual variable in an array.

EMBEDDED - something contained within. (e.g. CELLAR DOOR has an embedded space between the R and the D).

ERROR CODE - a symbol or number representing a specific error.

ERROR MESSAGE - a message from ADAM telling you about a programming error or an execution error.

EXECUTE - to carry out a specified action.

EXPRESSION - a mathematical formula for use in a program calculation.

FILE - a collection of information sorted under a certain name on your digital data pack.

FUNCTION - a calculation that is pre-programmed to be automatically executed, if requested, at any point in the program. All functions consist of a name followed by parentheses enclosing a number. For some functions, the actual number you chose is not important.

GRAPHICS - information presented as pictures or images.

HARD COPY - computer printout on paper.

HARDWARE - the actual physical components which make up ADAM...circuits, transistors, microchips, etc.

HEXADECIMAL - number representation in powers of 16. Use digits 0 to 9 and letters A-F.

IMMEDIATE EXECUTION - the execution of a program line (typed without a line number) as soon as it is typed and RETURN is pressed.

INDEX - a number used to identify a member of a sequential list or table.

INTEGER - a whole number with no fractional part.

K or KILOBYTE - 2 to the 10th power, or 1024. 32K=32*1024=32768.

KEYWORD - a particular word that defines a certain statement or command (e.g. PRINT, RUN, etc.).

LOGICAL OPERATOR - operators such as AND, OR, and NOT that combine logical values to produce logical results.

LOW-LEVEL LANGUAGE - a language that's very close to the machine language that ADAM's processor can execute directly.

MACHINE LANGUAGE - the internal language that ADAM speaks and translates everything into before executing programs or storing in memory.

MAIN MEMORY - a component in your computer which stores information for recall later on. See RAM and ROM.

MICROCOMPUTER - ADAM is a microcomputer, along with any other computer whose processor is a microprocessor.

MODE - the state of a computer system which determines its behavior.

OPERATOR - a symbol which directs that an action be performed on one or more values to yield a result.

OS COMMAND - a command which tells ADAM to operate the digital data pack or other peripheral device. Cannot be used directly in a program; must be printed using PRINT and CONTROL-D.

PEEK - allows you to <u>read only</u> from a location in ADAM's memory.

PERIPHERAL - at or outside ADAM's boundaries.

PERIPHERAL DEVICE - a device such as a television screen monitor, printer, or disk drive.

PLOTTING VECTOR - used in shape definition, plotting vectors each represent single steps in plotting the points of a shape and deciding on which direction to move on the screen.

POKE - used to store information in a specified location in ADAM's memory.

POP - wipes out the top entry from a stack.

PROCESSOR - this is where all computations are performed.

PROMPT - a message from ADAM which appears on your screen to remind you that some action on your part is expected before your program can continue.

RADIAN - a measure of angle. There are 2pi radians in a circle of 360 degrees. One radian equals approximately 57.2957795 degrees.

RAM MEMORY - Memory whose contents can be accessed in an arbitrary order.

REAL NUMBER - a number which may include a fractional part.

RELATIONAL OPERATOR - a symbol which compares two entities to arrive at a logical result (e.g. < > <= >= = <>).

RESERVED WORD - a special word which has a single purpose in programming, and therefore cannot be used as a program name. See KEYWORD.

ROM MEMORY - memory whose information can only be read.

ROUTINE - a piece of your program which performs some task directly related to accomplishing the overall task of the main program.

SCROLL - the onscreen shifting of information up or down in order to make room for other information appearing at the other end.

SEED - a value used to start a flow of a repeatable sequence of random numbers.

SHAPE DEFINITION - a coded description of a shape to be drawn. Used in high resolution graphics.

SHAPE TABLE - a group of shape definitions and their index numbers.

SHAPE TABLE INDEX - a table of contents of your shape table which gives you the addresses of where in memory your shapes are located.

SOFTWARE - programs which determine ADAM's behavior.

SPACE CHARACTER - press the space bar, and you'll see one.

STACK - a list where entries are periodically added and removed at one end only (usually the top).

STATEMENT - an instruction in a line of your program which tells ADAM what to do.

STRING - a sequence of text characters which conveys information.

SUBROUTINE - a section of your program which can be executed in an area out of sequence. Control is returned to the program's regular sequence once the branch execution is completed.

SYNTAX - the set of rules governing the structure of programming statements and commands.

SYSTEM - a collection of interrelated parts assembled to perform some function.

TEXT - information presented in an understandable form to human beings.

TEXT FILE - a file with information expressed in text form. Indentified as a file type H or h in the catalog.

UNCONDITIONAL BRANCH - a branch whose execution doesn't depend on the truth of a given condition.

USER - what you are when you operate ADAM.

VALUE - information which can be stored as a variable, string, or number.

SmartBASIC Variables

Smart BASIC has three kinds of variables: Integer, Floating Point, and String. Integer and floating point are both numeric. A variable name can be one or more letters or numbers long, but the first character must be a letter. Upper and lower case letters are all converted to lower case. The variable name can not be exactly the same as a statement or command word (i.e. plot is not legal) but the names can contain letters that are the same as a statement or command word (i.e. plotter is letgal).

Integer variables are indicated by putting % after the variable name (LET b%=2). Integer variables can be as small as -32767 and as large as 32767. Each integer variable takes up 5 bytes of memory. Since they take up less room than floating point variables, they are often used for arrays. But they have limitations. They can only be whole numbers. If a decimal number is assigned to an integer variable, the decimal part is lost (not rounded).

Floating point variables are the normal numeric variables. They can be very large or very small and keep the decimal parts of numbers. Each floating point variable takes 10 bytes of memory.

String variables are used to store ASCII characters, including letters, numbers, punctuation marks, and control characters. String variables are indicated by putting a $ after the variable name. Each string variable takes up 8 bytes + 1 byte per character.

Any of these variables can be used as array variables, and each is recognized separately. That means that the variables x, x%, x$, x(0), x%(0), and x$(0) are all different and can stand for different things.