ADAM™

SmartBASIC™

PROGRAMMING WITH ADAM™

A Simple Guide to SmartBASIC™

ADAM™ SmartBASIC™

PROGRAMMING MANUAL
REVISED EDITION

WARNING ON CLASS B PRODUCTS

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

Reorient the receiving antenna;
Relocate the computer with respect to the receiver;
Move the computer away from the receiver;
Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:

"How to Identify and Resolve Radio/TV Interference Problems".

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402. Stock No. 004-000-00345-4

Warning: This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

Use of cables other than the Coleco cables (or equivalent) specified in this manual to connect peripheral equipment like printers, modems or video monitors will invalidate the Federal Communications Commission Certification of your computer and may cause interference levels exceeding the limits established by the FCC for this equipment.

When using any expansion ports for accessory equipment, shielded connecting cable may be required. Contact the accessory manufacturer for further details.

LIMITED NINETY DAY WARRANTY

Coleco warrants to the original consumer purchaser of its ADAM Family Computer System in the United States of America and Canada that the product will be free of defects in material or workmanship for 90 days from the date of purchase under normal in-house use.

Coleco's sole and exclusive liability for defects in material and workmanship shall be limited to repair or replacement at its authorized Coleco Service Station. This warranty does not obligate Coleco to bear the cost of transportation charges in connection with the repair or replacement of defective parts.

This warranty is invalid if the damage or defect is caused by accident, act of God, consumer abuse, unauthorized alteration or repair, vandalism, or misuse.

Any implied warranties arising out of the sale of the ADAM Family Computer System including the implied warranties of merchantability and fitness for a particular purpose are limited to the above 90 day period. Coleco shall in no event be liable for incidental, consequential, contingent, or other damages.

This warranty gives you specific legal rights and you may have other rights which vary from State to State. Some states do not allow the exclusion or limitation of incidental or consequential damages or limitations on how long an implied warranty lasts, so the above limitations or exclusions may not apply to you.

SERVICE POLICY

Please read the Set-Up Manual carefully before using the product. If your ADAM Family Computer Systen fails to operate properly, please refer to the troubleshooting checklist in the Set-Up Manual. If you cannot correct the malfunction after consulting the troubleshooting checklist, please call Customer Service on Coleco's toll-free hotline: 1-800-842-1225 nationwide. In Canada, 1-800-361-2122. This service is in operation from 8:00 a.m. to 5:00 p.m. Eastern Standard Time, Monday through Friday.

If Customer Service advises you to return your ADAM Family Computer System, please carry it in or return the entire system postage prepaid and insured, in the original box (if possible), with your name, address, proof of the date of purchase, and a brief description of the problem to the Service Station you have been directed to return it to by the toll-free service information. If your unit is found to be factory defective during the Limited Warranty period, it will be repaired or replaced at no cost to you. If the unit is found to have been consumer damaged or abused and therefore not covered by the warranty, then you will be advised, in advance, of repair costs.

If your computer requires service after expiration of the Limited Warranty period, please call Coleco's toll-free service hotline for instructions on how to proceed: 1-800-842-1225 nationwide. In Canada, 1-800-361-2122.

IMPORTANT: SAVE YOUR RECEIPTS SHOWING DATE OF PURCHASE

ACKNOWLEDGEMENTS

We wish to thank the following Coïeco employees who were instrumental in assembling this manual:

EDITOR:

Barbara Spear

TECHNICAL CONSULTANTS:

Mark Callahan

Michael DeManche

Portions of this manual are an adaptation of The Secret Guide To Computers written by Russ Walter.

Portions of this manual were written by Barbara Spear and Michael DeManche.

Table of Contents

THE FOUR LESSONS

This book has been divided into four lessons. Each lesson is about 20 pages long, and can easily be finished in a brief sitting. Reference information for those who already know BASIC can be found in appendices A,B, and C.

LESSON 1: CHATTING WITH ADAM

This first lesson explains which buttons to press, how to make it "go", and trains you to write many kinds of programs. When you finish this lesson, you'll know how to make ADAM replace your typewriter, your photocopier , and your calculator. How can one computer replace all those three other machines? You'll know. And you'll be programming the computer with confidence.

LESSON 2: ADAM THINKS

This lesson explains how ADAM can think like a human. The lesson climaxes when you learn how to make the computer imitate your own personality. You'll also learn how to make the computer write clever stories, and print letters admitting you to Harvard.

Some programmers believe that every concept in the universe (such as love, hate, politicians, Santa Claus, and non-dairy coffee creamer) can be explained by using just five words. The words are: PRINT, INPUT, GO, END, and IF. When you finish this lesson, you'll understand those five magic words.

LESSON 3: MASTER YOUR COMPUTER

After the mind-blowing stuff in Lesson 2, Lesson 3 comes back down to earth. You learn how to make the computer solve practical problems about debts, diets, translating French, and other real-world hang-ups. You get helpful inside hints on how to remove errors from programs, so that the programs work perfectly. You even learn how to make the computer print boring tables (in case you have a boss you'd like to bore).

But don't think Lesson 3 remains totally on earth. No way! In fact, this lesson contains the biggest mind-blower of all: **ADAM can make its own decisions.** This lesson explains how to tell the computer to make its own decisions. . . by using "random numbers".

You'll learn how to make ADAM play games with you, so that neither you nor the computer can predict who'll win!

LESSON 4: TACKLE THE TOUGH STUFF

This lesson will challenge you, so you can give your own brain a whirl. It digs into the toughest parts of programming: "Loop techniques", "subscripts", and graphics. It also shows you how incredibly powerful your ADAM computer really is. Have fun!

MODEL PROGRAMS

Each lesson contains six model programs. Here they are. You don't have to look at them yet. Later, when you're writing your own programs, you may want to refer back to these pages for help.

LESSON 1: CHATTING WITH ADAM

Printing

```
1 PRINT "I LOVE YOU"
2 PRINT "YOU TURN ME ON"
```

The computer prints:

```
I LOVE YOU
YOU TURN ME ON
```

Semicolon

```
1 PRINT "FAT";
2 PRINT "HER"
```

The computer prints FAT and HER on the same line:

```
FATHER
```

Embedded semicolon

```
1 PRINT "THIN";"KING"
```

The computer prints THIN and KING on the same line:

```
THINKING
```

Goto

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

The computer prints CAT and DOG repeatedly.

Arithmetic

```
10 PRINT 1+2
```

The computer computes 1+2 and prints the answer:

```
3
```

Order of operations

```
10 PRINT 2+3*4
```

The computer starts with 2, then adds three fours. It prints:

```
14
```

LESSON 2: ADAM THINKS

Variable

```
10 X=47
20 PRINT X+2
```

Since X is 47, line 20 says to print 47+2. The computer prints:

```
49
```

Input

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

Line 10 makes ADAM ask-- WHAT IS YOUR NAME? then wait for you to answer; your answer is called N$. If you answer ALBERT, line 20 prints:

```
YOU HAVE A NICE NAME, ALBERT
```

Colon

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

The computer prints:

```
12
```

String variable

```
10 G$="DOWN"
20 PRINT G$
```

Since G$ is "DOWN", line 20 prints:

```
DOWN
```

Stop

```
10 PRINT "BUBBLE"
20 END
30 PRINT "FOX"
```

The computer prints just:

```
BUBBLE
```

IF

```
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
40 PRINT "PLEASE SAY 'MALE' OR 'FEMALE;": GOTO 10
```

Line 10 asks if you're male or female. After you answer, the computer prints an appropriate retort.

Data

```
10 DATA MEAT,POTATOES,

20 READ A$
30 PRINT A$
40 GOTO 20
```

The computer prints:

```
MEAT
POTATOES
LETTUCE
OUT OF DATA
```

Restore

```
10 DATA MEAT,          LETTUCE
POTATOES,LETTUCE,DONE
20 READ A$: IF A$="DONE"
THEN PRINT "THOSE ARE
MY FAVORITE FOODS":
RESTORE: GOTO 20
30 PRINT A$
40 GOTO 20
```

The computer prints MEAT, POTATOES,LETTUCE and THOSE ARE MY FAVORITE FOODS repeatedly.

For

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

The computer prints every number from 1 to 100.

Step

```
10 FOR I = 10 TO 1 STEP -1
20 PRINT I;", ";
30 NEXT I
40 PRINT "BLAST OFF!"
```

The computer prints a rocket countdown:

```
10, 9, 8, 7, 6, 5, 4, 3,
2, 1,    BLAST OFF!
```

Def Fn

```
10 DEF FN D(N)=INT(1+N*RND(1))
20 PRINT FN D(5)
```

The computer prints 1 or 2 or 3 or 4 or 5.

Comma

```
10 PRINT
"THIN","KING"
```

The computer prints THIN in the first zone, and KING in the second:

```
THIN          KING
```

LESSON 4: TACKLE THE TOUGH STUFF

Incrementing

```
10 A=5
20 A=3+A
30 PRINT A
```

Line 20 means: the new A is 3 plus the old A. So the new A is 3+5. Line 30 prints:

8

Summing

```
10 S=0
20 PRINT "NOW THE SUM IS";S
30 INPUT "WHAT NUMBER DO YOU
   WANT ADDED INTO THE SUM? ";X
40 S=S+X
50 GOTO 20
```

The computer imitates an adding machine

Read

```
10 DIM X(3)
20 DATA 57.2, -8.3, 476
30 FOR I = 1 TO 3
40 READ X(I)
50 PRINT X(I)
60 NEXT I
```

Line 10 says X is a list of 3 numbers. Line 30 reads them from the data. Line 50 prints

57.2
-8.3
476

Counting

```
10 C=3
20 PRINT C
30 C=C+1
40 GOTO 20
```

The computer counts, starting at 3. It prints: 3, 4, 5, 6, etc.

Subscripts

```
10 DIM X(3)
20 X(1)=57.2
30 X(2)=-8.3
40 X(3)=476
50 PRINT X(1)
60 PRINT X(2)
70 PRINT X(3)
```

Line 10 says X is a list of. 3 numbers. Lines 20-40 define them. Lines 50-70 prints them:

57.2
-8.3
476

Double subscripts

```
10 DIM X(3,2)
20 X(1,1)=57
30 X(1,2)=8.4
40 X(2,1)=-6
50 X(2,2)=1000
60 X(3,1)=0
70 X(3,2)=7.77
80 PRINT X(1,1)
90 PRINT X(1,2)
```

etc.

Line 10 says X is a table with 3 rows and 2 columns. Lines 20-70 tell which numbers are in the table. Line 80 etc. prints the table.

LESSON 1

CHATTING WITH ADAM

Be Bold

In science fiction, computers blow up. In real life, they never do. No matter what buttons you press, no matter what commands you give, you won't hurt the computer. ADAM is invincible! So go ahead and experiment. If it doesn't like what you say, it will gripe at you, but so what? You wanted a computer that talks to you, right?

Furthermore, anyone is bound to make a mistake when they first use their computer. Fortunately, ADAM tells you before you've gone too far, and often tells you exactly what you've done wrong. So, whenever you have trouble, laugh about it, and say, "Oh boy! Here we go again!" Then get some help.

If you have friends that work with computers or know of a user group, ask them for help. They'll gladly answer your questions, because they like to show off, and because the way they got to know the answers was by asking. Computer folks like to explain computers, so play along with the computer people, boost their egos, and they'll help you, too. Above all, assert yourself, and ask questions. "Shy guys finish last."

When dealing with the computer and the people who surround it, be friendly but also aggressive. If you're taking a computer course, be sure you get your money's worth and ask your teacher and coworkers questions, questions, questions!

If you're using ADAM on a personal basis and find you need help, you can always call us at the number listed in and the SERVICE POLICY SECTION.

Your town probably has a "computer club". (To find out, ask the local schools and computer stores.) Join the club, and make an announcement that you'd like help with your computer. Probably some computer hobbyist will help you.

How To Use The Keyboard

This section outlines the uses of the keys.

Numbers

The top row of keys contains the numbers. Don't confuse the number one with the letter l; the number 1 is in the top row. Don't confuse zero with the letter O; the number zero is in the top row, and will appear on your keyboard and screen with a slash through it.

The SHIFT key

One of your keys has the number four with a dollar sign above it. If you press this key, you'll type a 4. But, if you hold down the SHIFT key, and then press the key with the 4 and the dollar sign, you'll find that you're typing the dollar sign this time.

Here's the general rule: if a key has two symbols on it, and you want to type the top symbol, hold down the SHIFT key. On a single symbol key, hold the shift key down to get a capital letter.

About the SHIFT key. . . beginners often make two boo-boos. The first boo-boo is to forget to press it when you want an upper character on a key. The other boo-boo is to try to hit the SHIFT key and another key at exactly the same time. You can't do it; it's impossible; you'll wind up hitting one key before the other. The trick is to hold down the SHIFT key first; and while you keep holding it down, give the other key a light tap. ADAM also has a LOCK key. Each time the LOCK key is pressed, it switches between upper and lower case characters.

The RETURN key

The most important key is the RETURN key. Press the RETURN key at the end of every line you type. The RETURN key makes the computer read what you typed. The computer ignores what you type until you press the RETURN key. If you forget to press the RETURN key at the end of the line, the computer doesn't read what you typed, and so the computer doesn't do anything, and you wonder why the computer seems broken.

Loading SmartBASIC

After you've stared at the keyboard, you really should get into SmartBASIC. Here's how:

- Turn the computer on.
- Place SmartBASIC digital data pack into Drive.
- Hit the reset key.
- Wait for several seconds.

  The computer will print a bracket on the screen "]", as well as "Coleco SmartBASIC VI.0 at the top of the screen. You are now ready to begin talking to your computer.

NOTE: Do not turn the computer on or off with a digital data pack in the drive.

Programming

Now that you've gotten into SmartBASIC, you're ready to program the computer!

The bracket on the left side of your screen means that ADAM is waiting for a command. Don't type a command until you see the bracket and the blinking cursor. Programming the computer consists of three steps.

Step 1: type the word NEW. That tells the computer you're going to invent a new program and to forget your old program. After you type the word NEW, press the RETURN key. The computer will print a bracket on the screen.

Step 2: type your program A program is a list of numbered commands. You can program ADAM in all upper case, upper and lower, or all lower case; it's your choice. For example, suppose you want the computer to say:

```
I love you
You turn me on
Let's get married
```

Type this program:

```
1 PRINT "I love you"
2 PRINT "You turn me on"
3 PRINT "Let's get married"
```

Every line of the program must be numbered; you must type those numbers. Every line of the above program must say PRINT; you must type the word PRINT. Every line of the above program must have quotation marks; you must type the quotation marks. At the end of each line, press the RETURN KEY.

Step 3: type the word LIST. That tells the computer to print your program on the screen so you can check to see if you've typed the program the way you wanted to.

If you type your program out in lower case, and LIST it, you'll find that ADAM converted it to upper case automatically. The only time ADAM will not convert to upper case automatically is if your lower case material is the name of a variable or within quotes in a PRINT statement. Upper case and lower case variable names many look different to you, but ADAM sees them as the same.

Step 4: type the word RUN. That tells the computer to look at the program you've written, and run through it. After you type the word RUN,

press the RETURN key. The computer will print everything you said. It will print:

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

Then the computer will print a bracket on the screen.

Afterwards, you can feed the computer another program, by going through the three steps again: type the word NEW, type the program, and type the word RUN. Remember to press the RETURN key at the end of each line.

Those three steps are all you have to know about programming! Go ahead, and write some programs! Try it; you'll have lots of fun! A person who writes programs is called a programmer. Congratulations: you're a programmer!

Please note that, as they stand, the sample programs in this book, will not print out on your printer. Unless you type in the instruction to do so, they will appear on your screen only. To print them out, use the command PR#1. When you are all through printing, use the command PR#0 to send your output back to the screen. For more information on PR#, see the Reference Section at the back of this book (Appendix A).

Multiple Copies

Suppose you want to make multiple copies of this poem:

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

First, turn the poem into a program, by typing:

```
NEW
1 PRINT "SPRING HAS SPRUNG."
2 PRINT "THE GRASS HAS RIZ."
3 PRINT "I WONDER WHERE"
4 PRINT "THE FLOWERS IS."
```

Then type the word RUN (and press the RETURN key). The computer will print the poem. If you type the word RUN again, the computer will print the poem again. Each time you type the word RUN, the computer will print the poem. So to print many copies of the poem, first, turn on the printer with PR#1. Then type the word RUN many times, like this:

You type:	**RUN**
The computer types:	SPRING HAS SPRUNG. THE GRASS HAS RIZ. I WONDER WHERE THE FLOWERS IS.
You type:	**RUN**
The computer types:	SPRING HAS SPRUNG. THE GRASS HAS RIZ. I WONDER WHERE THE FLOWERS IS.
You type:	**RUN**
The computer types:	SPRING HAS SPRUNG. etc.

If you despise poetry and all other forms of art, don't despair; by using the same technique, you can make the computer type many copies of your favorite flubbed-up business letter.

EDITING
How To Correct a Mistake

If you make a mistake, you can correct it in many ways:

If you notice the mistake before you press the RETURN key, use the BACKSPACE key or left arrow to move the cursor under your mistake. Retype the line from that point.

If you mess up the whole program, rewrite it, by typing the word NEW and then starting from the beginning again.

If you mess up a line, retype it underneath. For example, suppose you type--

```
1 PRIMT "I LOVE YOU"
```

and after you hit the RETURN key, you notice that you misspelled the word PRINT. Just retype the line, so your screen or paper looks like this:

```
1 PRINT "I LOVE YOU"
```

ADAM will tell you about the error by saying ILLEGAL COMMAND. Here's another example. . . Suppose you type:

```
1 PRINT "I LIKE COFFEE"
2 PRINT "I LIKE TEE"
3 PRINT "I LIKE THE GIRLS"
4 PRINT "AND THE GIRLS LIKE ME"
```

and then you notice you misspelled the word TEA in line 2. Underneath the program, retype that line, so your screen or paper looks like this:

```
1 PRINT "I LIKE COFFEE"
2 PRINT "I LIKE TEE"
3 PRINT "I LIKE THE GIRLS"
4 PRINT "AND THE GIRLS LIKE ME"
2 PRINT "I LIKE TEA"
```

Another way to make corrections is to use the onscreen editing capabilities. (Note: the SmartWRITER editing commands do not work in SmartBASIC.) To use onscreen editing, type "LIST" followed by the number of the line you want to change, then press RETURN. Use the arrow keys to move to the beginning of the line. Use the right arrow key to move the cursor to your mistake. Type your correction. When the line is the way you want it, use the right arrow key to move past the last character. Press RETURN, then check to see if the computer accepted the changes.

To make you life easier, there are control characters that can help you make changes and corrections to your program. While holding the CONTROL key down, press the appropriate key.

CONTROL X "Forgets" the line of characters being typed

CONTROL L Clears the screen

CONTROL O Deletes the character directly above the cursor

CONTROL H Backspaces over characters without deleting them

CONTROL N Inserts spaces to the left of the cursor

CONTROL P Sends all printable characters from the screen to the printer and prints them

CONTROL ← Passes over letters without "forgetting" them

CONTROL → Passes over letters without "remembering" them

Using the CONTROL key in combination with the arrow keys can make inserting easier when the line is full.

How to Delete a Line:

To delete a single line, type the line number followed by RETURN.

To delete more than one line, see the DEL command in the Reference Section.

GRIPES

If the computer gripes, don't worry: just correct your mistake. For example, suppose you type:

```
1 PRIMT "ROSES ARE RED"
```

ADAM will gripe at you:

```
ILLEGAL COMMAND
```

and a caret (^) will appear to point out where in the line your error lies.

Your next job is to correct your error. To do that, just retype line 1 correctly.

You type this (error):

```
1 PRIMT "ROSES
ARE RED"
```

The computer gripes at you:

```
ILLEGAL COMMAND
```

You type this correction

```
1 PRINT "ROSES
ARE RED"
```

along with the rest of the poem:

```
2 PRINT "CABBAGE
IS GREEN"
3 PRINT "MY FACE
IS FUNNY"
4 PRINT "BUT
YOURS IS A
SCREAM"
RUN
```

The computer recites:

```
ROSES ARE RED
CABBAGE IS GREEN
MY FACE IS FUNNY
BUT YOURS IS A
SCREAM
```

Common Bloopers

If you're like most beginners, you'll make these mistakes:

You type the letter O instead of zero, or type zero instead of the letter O. Your typing looks correct to you, but the computer gripes.

You type the letter l instead of the number one.

You type something (such as RUN), but you forget to press the RETURN key afterwards. So the computer keeps waiting for you to press it. Since the computer isn't replying, and since you don't realize the computer is waiting for you, you think the computer is broken.

You start typing a new program, but forget to type the word NEW. So when you type RUN, the computer runs a mish-mash of the new program with the previous program, and reprints some messages from the previous program.

Quotation marks

You must put quotation marks around a data string, if the string contains a comma or a colon.

As a general rule of thumb--when in doubt, or whenever a data statement doesn't seem to be working, insert quotation marks. They can't hurt.

Spaces

```
1 PRINT "JOY"
2 PRINT
3 PRINT "SORROW"
```

Line 1 makes the computer print JOY. Line 2 makes the computer print a blank line, underneath JOY. Altogether, the computer will print:

```
JOY

SORROW
```

You have the ability to compose and print out letters in both SmartWRITER and SmartBASIC modes. Here's how to do it in SmartBASIC:

```
DEAR JOAN,

THANK YOU FOR THE BEAUTIFUL
NECKTIE.  JUST ONE PROBLEM--
I HATE NECKTIES!
        LOVE,
             TURTLENECK-FRED
```

This program prints it:

```
1 PRINT "DEAR JOAN,"
2 PRINT
3 PRINT "THANK YOU FOR THE BEAUTIFUL"
4 PRINT "NECKTIE.  JUST ONE PROBLEM--"
5 PRINT "I HATE NECKTIES!"
6 PRINT "      LOVE,"
7 PRINT "        TURTLENECK-FRED"
```

In the program, each line (except line 2) must contain two quotation marks. To make the computer indent a line, put blank spaces after the first quotation mark.

Semicolons

```
1 PRINT "FAT";
2 PRINT "HER"
```

Line 1 makes the computer print FAT; and line 1 ends with a semicolon. The semicolon makes the computer print the next item on the same line with no spaces between the items; so the computer will print HER next to FAT, like this:

```
FATHER
```

This program shows how an underweight philosophical king spends his time:

```
1 PRINT "THIN";"KING"
```

The computer will print THIN, and will print KING on the same line, like this:

```
THINKING
```

Rearranging Your Program

You don't have to type your program in order. For example, suppose you type:

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

In its mind, the computer automatically rearranges the program, so the numbers are in increasing order, like this:

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

To find out what's in the computer's mind, type the word LIST. Example:

You type this program:

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

You type this command:

```
LIST
```

The computer types the program in increasing order:

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

If you type RUN, the computer will print:

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

Whenever you're confused, type the word LIST. Then the computer will tell you what's in its mind; and what's in its mind might surprise you!

Sore Feet

Type this, and see what happens:

```
NEW
10 PRINT "MY"
90 PRINT "TRUCK"
32 PRINT "SNEEZE"
50 PRINT "TOE"
32 PRINT "NEED A"
70
80 PRINT "HAT"
80
20 PRINT "SORE FEET"
```

Since you typed two versions of line 32, the computer assumes the second is a correction. So the computer assumes that line 32 says PRINT "NEED A".

Since you typed nothing in line 70, the computer ignores line 70.

Since you typed two versions of line 80, the computer assumes the second is a correction. Since the second says nothing, the computer assumes you want line 80 erased.

In its mind, the computer rearranges the program, so the numbers are in increasing order. If you type LIST, the computer will type:

```
10 PRINT "MY"
20 PRINT "SORE FEET"
32 PRINT "NEED A"
50 PRINT "TOE"
90 PRINT "TRUCK"
```

If you type RUN, the computer will print:

```
MY
SORE FEET
NEED A
TOE
TRUCK
```

If you think about that example, you'll notice two things:

To revise a line, just retype it.

To erase line 80, just type--

```
80
```

with nothing else on the line.

The word NEW erases your program from the computer's mind, to make way for a new one. But the word RUN does not erase the program; after the run, you can continue inserting, deleting, and revising lines.

Use tens

If your first line is numbered 1, and your second line is numbered 2, you can't squeeze a line between them, since decimals aren't allowed. So expert programmers number their lines 10, 20, 30, 40, . . . instead of 1, 2, 3,...

GOTO

This program makes the computer go crazy:

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

Line 10 makes the computer print CAT. Line 20 makes it print DOG. Line 30 makes it go back to line 10 again, so it prints DOG again, and comes back to line 30 again, which makes it go back to line 10 again, print CAT and DOG again, then jump back again. . . the computer will print the words CAT and DOG again and again, like this:

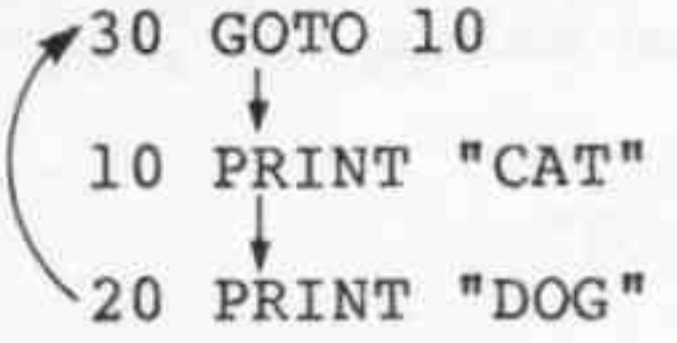
```
CAT
DOG
CAT
DOG
CAT
DOG
CAT
DOG
etc.
```

If you use the PR#1 command, the computer will try to print on your printer the words CAT and DOG again and again, forever. Try running that program; you'll have fun watching the computer go crazy. On your screen, you'll see that the computer is printing the words CAT and DOG at the same time the printer is.

To stop this madness, you must give the computer some sort of a jolt that will put it out of its misery. This is called aborting your program. Here's how:

Hold down the CONTROL key; and tap the C key. Or, if ADAM is waiting for input from you when you decide to abort, hold down the CONTROL key, tap the C, and press RETURN.

Here's a picture of that program:



The computer follows the arrows, which make it go round and round in a loop. Since the computer will try to go round and round the loop forever, the loop is called **infinite.** The only way to stop an infinite loop is to abort the program.

In that program, you typed just three lines; but since the third line said to GOTO the beginning, the computer does an infinite loop. By saying GOTO, you can make the computer do an infinite amount of work. Moral: the computer can turn a finite amount of human energy into an infinite amount of good. Putting it another way: the computer can multiply your abilities by infinity.

For example, suppose you want to send this poem to all your friends:

```
I'M HAVING TROUBLE
WITH MY NOSE.
THE ONLY THING IT DOES IS:
BLOWS!
```

Just run this program:

```
10 PRINT "I'M HAVING TROUBLE"
20 PRINT "WITH MY NOSE."
30 PRINT "THE ONLY THING IT DOES IS:"
40 PRINT "BLOWS!"
50 PRINT
60 GOTO 10
```

Lines 10-40 print the poem. Line 50 prints a blank line at the end of the poem. Line 60 makes the computer do all that repeatedly; the computer will print:

```
I'M HAVING TROUBLE
WITH MY NOSE.
THE ONLY THING IT DOES IS:
BLOWS!

I'M HAVING TROUBLE
WITH MY NOSE.
THE ONLY THING IT DOES IS:
BLOWS!

I'M HAVING TROUBLE
WITH MY NOSE.
THE ONLY THING IT DOES IS:
BLOWS!

etc.
```

The computer will print infinitely many copies--unless you abort your program.

Love

This program's for lovers:

```
10 PRINT "LOVE"
20 GOTO 10
```

The computer will print LOVE repeatedly, like this:

```
LOVE
LOVE
LOVE
LOVE
LOVE
LOVE
LOVE
LOVE
LOVE
etc.
```

To have more lovely fun, put a semicolon at the end of line 10, like this:

```
10 PRINT "LOVE";
20 GOTO 10
```

The semicolon makes the computer print LOVE next to LOVE:

```
LOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELO
ELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVEL
VELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVELOVE
OVELOVELOVELOVELOVEetc.
```

Instead of making the computer print LOVE, try making it print an advertisement (such as BUY JOE'S VEGETABLES), or a political slogan (such as STOP POLLUTION), or your name, or something else that makes you feel emotional. ADAM, like a dog, imitates the personality of its master.

Computer torture

This program tortures the computer:

```
10 GOTO 10
```

Line 10 tells the computer to go to line 10, which tells the computer to go to line 10, which tells the computer to go to line 10. . . The computer will do line 10 again and again, forever. In other words, the computer will spend its entire life just mumbling to itself. That program has turned your beautiful computer into a bumbling idiot. If you're mean enough to type that program into your computer, please have the mercy to abort it after awhile. Otherwise, a member of the Association for Prevention of Cruelty to ADAM Family Computers will come and repossess your fingers.

Human torture

A program that makes the computer do the same thing again and again forever is an infinite loop. Some humans act just like computers. These humans do the same thing again and again, every day. Their lives are sheer drudgery. They are caught in an infinite loop.

Kiss and tell

Here's a poor, innocent program:

```
10 PRINT "YOUR SISTER"
20 PRINT "DISLIKES ANYONE WHO"
30 PRINT "KISSES AND TELLS"
```

It makes the computer print:

```
YOUR SISTER
DISLIKES ANYONE WHO
KISSES AND TELLS
```

To make the program more diabolical, insert line 15, by typing it at the bottom of your program and LISTing the program out to see your line re-shuffled into the correct position.

```
10 PRINT "YOUR SISTER"
15 GOTO 30
20 PRINT "DISLIKES ANYONE WHO"
30 PRINT "KISSES AND TELLS"
```

The computer does line 10, which prints YOUR SISTER. Then it does line 15, which makes it skip ahead to line 30, which prints KISSES AND TELLS. So altogether, the computer prints:

```
YOUR SISTER
KISSES AND TELLS
```

In that program, line 15 made the computer skip ahead to line 30; in other words, it made the computer skip over line 20.

Conveniences

Here's a short-cut you should know about. You can get your computer to PRINT without placing a line number in front of the words:

```
PRINT "I LOVE YOU"
```

When you press the RETURN key at the end of that line, the computer will immediately print I LOVE YOU. You don't have to type the word RUN. This short-cut is called **immediate mode.**

Leaving the Computer

When you're finished using your computer and want to walk away, here's what to do. . .**OPEN THE DOOR TO YOUR DITIGAL DATA PACK DRIVE AND TURN EVERYTHING OFF.** Now wasn't that easy?

Arithmetic

Let's try to make the computer print the answer to 2+2. (and let's hope that the computer says 4.) Run this program:

```
10 PRINT 2+2
```

The computer prints:

```
4
```

If you want to subtract 2 from 9, run this program:

```
10 PRINT 9-2
```

The computer will print:

```
7
```

You can use decimal points and negative numbers. For example, if you run this program:

```
10 PRINT -26.3+1
```

the computer will print:

```
-25.3
```

To multiply, use an asterisk. To multiply 2 by 6, run this program:

```
10 PRINT 2*6
```

The computer will print:

```
12
```

To divide, use the slash beneath the question mark. So to divide 8 by 4, run this program:

```
10 PRINT 8/4
```

The computer will print:

```
2
```

Do not put commas in large numbers. To write four million, do not write 4,000,000; instead write 4000000.

E Notation

If the computer types a number with an E, the E means: move the decimal point. For example, suppose the computer says the answer to a problem is:

```
8.51673E9
```

The E means: move the decimal point. In fact, the E9 means: move the decimal point 9 places to the right. So look at 8.51673, and move the decimal point 9 places to the right. You get 8516730000. So when the computer says the answer is 8.51673E9, the computer really means the answer

is 8516730000 approximately. (The exact answer might be 8516730000.2, or 8516730001.79, or some similar number; what the computer tells you is just an approximation.)

Suppose your computer says the answer to a problem is:

9.26E-4

After the E, the minus sign means: move the decimal point to the left. So, the E-4 means: move the decimal point 4 places to the left. So look at 9.26, and move the decimal point 4 places to the left. You'll get:

.000926

You won't have to worry much about E notation; the computer uses it only if an answer is very, very large or very small. (ADAM uses E notation only if the answer is bigger than 999999999 or tinier than .01.) But when the computer does use E notation, remember to move the decimal point!

Range

The highest number ADAM can handle is 1E38 (which is 1 followed by 38 zeros, which is 100000000000000000000000000000000000. If you try to go much higher, the computer will say OVERFLOW ERROR.

The tiniest decimal the computer can handle is 2E-39, which is a decimal point followed by 39 digits, 38 of which are zeros, like this: .000000000000000000000000000000000002. If you try to go much tinier, the computer will "fake" the answer: it will give zero instead of the correct answer.

Order of Operations

What does "2 plus 3 times 4" mean? The answer depends on who you ask.

To a businessman, it might mean "start with 2 plus 3, then multiply by 4"; that makes 5 times 4, which is 20. But to ADAM, "2 plus 3 times 4" means something different: it means "2 plus three fours", which is 2+4+4+4, which is 14.

If you run this program:

```
10 PRINT 2+3*4
```

the computer will think you mean "2 plus three fours", so it will do 2+4+4+4, and will print this answer:

```
14
```

The computer will not print the businessman's answer, which is 20.

ADAM follows this rule: **do multiplication and division before addition and subtraction.** So when your computer sees 2+3*4, it begins by hunting for multiplication and division. It finds the multiplication sign between the 3 and the 4, so it multiplies the 3 by the 4 and gets 12, like this:

```
2+3*4
   12
```

So the problem becomes 2+12, which is 14, which is the answer the computer prints.

For another example, suppose you run this program:

```
10 PRINT 10-2*3+72/9*5
```

The computer begins by doing all the multiplications and divisions. So it does 2*3 (which is 6), and does 72/9*5 (which is 8*5, which is 40), like this:

```
10-2*3+72/9*5
    6     40
```

So the problem becomes 10-6+40, which is 44, which is the answer the computer prints.

Parentheses

You can use parentheses to change the order by which ADAM does arithmetic. For example, 5*(1+1) means 5*2, which is 10. You can put parentheses inside parentheses: 10-(5*(1+1)) means 10-(5*2), which is 10-10, which is 0.

LESSON 2

ADAM THINKS

Numeric Variables

A letter can stand for a number. For example, X can stand for the number 47, as in this program:

```
10 X=47
20 PRINT X+2
```

Line 10 says X stands for the number 47. In other words, X is a name for the number 47. Line 20 says to print X+2; since X is 47, X+2 is 49; so the computer will print 49. That's the only number the computer will print; it will not print 47.

A letter that stands for a number is called a **numeric variable.** A variable name can be up to 121 letters or numbers, but ADAM only pays attention to the first two. Therefore, APPLE and APE would be understood by ADAM to be the same variable. A variable name must begin with a letter, not a number. Commands and statements cannot be used as variable names. In that program, X is a numeric variable; it stands for the number 47. The value of X is 47. Line 10 is called an assignment statement, because it assigns 47 to X.

You may use the statement LET if you like, or just leave it off. With ADAM, the use of LET is optional.

```
LET X=47
or
X=47
```

Further examples:

```
10 Y=38
20 PRINT Y-2
```

Line 10 says Y is a numeric variable that stands

for the number 38. Line 20 says to print Y-2; since Y is 38, Y-2 is 36; so the computer will print 36.

```
10 B=8
20 PRINT B*3
```

Line 10 says B is 8. Line 20 says to print B*3, which is
8*3, which is 24; so the computer will print 24.

One variable can define another:

```
10 M=6
20 P=M+1
30 PRINT M*P
```

Line 10 says M is 6. Line 20 says P is M+1, which is 6+1, which is 7; so P is 7. Line 30 says to print M*P, which is 42; so the computer will print 42.

A value can change:

```
10 F=4
20 F=9
30 PRINT F*2
```

Line 10 says F's value is 4. Line 20 changes F's value to 9, so line 30 prints 18.

On the left side of the equal sign, you are only allowed to have **one** variable:

P=M+2 (CORRECT) P-M=2 (WRONG) 2=P-M (WRONG)

The variable on the left side of the equation is the only one that changes. For example: the statement P=M+1 changes the value of P but not of M. The statement A=B changes the value of A but not B:

```
10 A=1
20 B=7
30 A=B
40 PRINT A+B
```

Line 30 changes A, to make it equal to B; so A becomes 7. Since both A and B are now 7, line 40 prints 14.

"A=B" has a different effect than "B=A". That's because "A=B changes the value of A but not of B; "B=A" changes the value of B but not of A. Compare these programs:

```
10 A=1                    10 A=1
20 B=7                    20 B=7
30 A=B                    30 B=A
40 PRINT A+B              40 PRINT A+B
```

In the left program (which you saw before), line 30 changes A to 7, so both A and B are 7; line 40 prints 14. In the right program, line 30 changes B to 1, so both A and B are 1; line 40 prints 2.

If you don't asign a value to a numeric variable, the computer assumes it's zero:

```
10 PRINT R
```

Since R hasn't had a value assigned, the computer prints zero.

The computer doesn't look ahead:

```
10 PRINT J
20 J=5
```

When the computer encounters line 10, it doesn't look ahead to find out what J is. As of line 10, J is still unassigned, so the computer prints zero.

When To Use Variables

When should you use variables? Here's a practical example....

Suppose you're selling something that costs $1297.43, and you want to do these calculations:

```
multiply $1297.43 by 2
multiply $1297.43 by .05
add $1297.43 to $483.19
divide $1297.43 by 37
subtract $1297.43 from $8598.61
multiply $1297.43 by 28.7
```

To do these six calculations, you could run this program:

```
10 PRINT 1297.43*2, 1297.43*.05, 1297.43+483.19,
20 PRINT 1297.43/37, 8598.61-1297.43, 1297.43*28.7
```

But that program's silly, since it contains the number 1297.43 six times. This program's briefer, because it uses a variable:

```
10 C=1297.43
20 PRINT C*2, C*.05, C+483.19, C/37, 8598.61-C,
C*28.7
```

So **whenever you need to use a number several times, turn the number into a variable;** it will make your program briefer.

String Variables

A string is any collection of characters, such as "I LOVE YOU" or "76 TROMBONES" or "GO AWAY" or "XYPW EXR ///746".
Each string must be in quotation marks.

A letter can stand for a string--if you put a dollar sign after the letter. Example:

```
10 G$="DOWN"
15 PRINT
20 PRINT G$
```

Line 10 says G$ stands for the string "DOWN".
Line 20 prints:

```
DOWN
```

In that program, G$ is a variable. Since it stands for a string, it's called a string variable. **Every string variable must end with a dollar sign;** the dollar sign is supposed to remind you of a fancy S which stands for String.

If you're paranoid, you'll love this program:

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

Line 10 says L$ stands for the string THEY'RE LAUGHING AT YOU. Lines 20, 30, and 40 make the computer print:

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

The computer can recite nursery rhymes:

```
10 H$="HICKORY, DICKORY, DOCK!"
20 M$="THE MOUSE"
30 C$="THE CLOCK"
40 PRINT H$
50 PRINT M$;" RAN UP ";C$
60 PRINT C$;" STRUCK ONE"
70 PRINT M$;" RAN DOWN"
80 PRINT H$
```

Lines 10-30 assign characters to H$,M$, and C$.
Lines 40-80 make the computer print:

```
HICKORY, DICKORY, DOCK!
THE MOUSE RAN UP THE CLOCK
THE CLOCK STRUCK ONE
THE MOUSE RAN DOWN
HICKORY, DICKORY, DOCK!
```

If you don't assign characters to a string variable, the computer assumes it's blank.

```
10 PRINT F$
```

Since nothing has been assigned to F$, line 10 makes the computer print a line that says nothing; the line printed will be blank.

String Input

Humans ask questions. So if you want to make the computer work like a human, you must make it ask questions, too. **To make the computer ask a question, use the word INPUT.** This program makes the computer ask for your name:

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

Line 10 makes the computer ask WHAT IS YOUR NAME? and then wait for you to answer. Your answer will be called N$. For example, if you answer ALBERT, then N$ is ALBERT. Line 20 makes the computer print: (Be sure to hit the RETURN key to let ADAM know you have finished your input.)

```
YOU HAVE A NICE NAME, ALBERT
```

Here's the whole conversation:

(In these lessons, words to be typed in by you will be in bold print.)

You tell ADAM to run:	**RUN**
ADAM asks for your name:	WHAT IS YOUR NAME? **ALBERT**
ADAM comments:	YOU HAVE A NICE NAME, ALBERT

If you try that example, be careful! When you type in line 10, which says INPUT, make sure you type the two quotation marks and the semicolon and the question mark. If you have a string of variables following INPUT, then the question mark

will automatically be inserted for you. But if you have a string within quotes, it will not.

If you wish, run that program again and pretend you're somebody else:

You tell ADAM to run:	RUN
ADAM asks for your name:	WHAT IS YOUR NAME? MARK
ADAM comments:	YOU HAVE A NICE NAME, MARK

College admissions

This program prints a letter admitting you to the college of your choice.

```
10 INPUT "WHICH COLLEGE WOULD YOU LIKE TO ATTEND?
";C$
16 PRINT
17 PRINT
18 PRINT
20 PRINT "CONGRATULATIONS!"
25 PRINT
30 PRINT "YOU HAVE JUST BEEN ADMITTED TO ";C$
40 PRINT "IT FITS YOUR PERSONALITY"
50 PRINT "I HOPE YOU ATTEND ";C$
55 PRINT
56 PRINT
60 PRINT "              RESPECTFULLY YOURS"
61 PRINT
62 PRINT
63 PRINT
70 PRINT "                 THE DEAN OF ADMISSIONS"
```

Line 10 makes the computer ask WHICH COLLEGE WOULD YOU LIKE TO ATTEND?" and then wait for you to answer; your answer will be called C$. If you'd like to be admitted to HARVARD, you'll be pleased:

WHICH COLLEGE WOULD YOU LIKE TO ATTEND? **HARVARD**

CONGRATULATIONS!

YOU HAVE JUST BEEN ADMITTED TO HARVARD
IT FITS YOUR PERSONALITY
I HOPE YOU ATTEND HARVARD

RESPECTFULLY YOURS

THE DEAN OF ADMISSIONS

This program consists of three parts:

1. The computer begins by asking you a question (WHICH COLLEGE WOULD YOU LIKE TO ATTEND?) The computer's question is called the **prompt**, because it prompts you to answer.

2 Your answer (the college's name) is called **your input**, because it's data that you're **putting into** the computer.

3. The computer's reply (the admission letter) is called the computer's **output**, because it's the final answer that the computer puts out.

Opposites

The word INPUT is the opposite of the word PRINT. The word PRINT makes the computer print information out; the word INPUT makes the computer take information in. The word PRINT makes the computer output; the word INPUT makes the computer accept your input. Input and Output are collectively called I/O, so the INPUT and PRINT statements are called I/O statements.

Stories

Let's make the computer write a story, by filling in the blanks:

ONCE UPON A TIME, THERE WAS A PERSON NAMED

_____
your name

WHO HAD A FRIEND NAMED _____
friend's name

_____ wanted
your name

to _____ _____
verb friend's name

BUT _____ DIDN'T WANT
friend's name

TO _____ _____
verb your name

WILL _____ _____ _____
your name verb friend's name

WILL _____ _____ _____
friend's name verb your name

TO FIND OUT, COME BACK AND SEE THE NEXT EXCITING EPISODE

OF _____ AND _____
your name friend's name

To write the story, the computer must ask for your name, your friend's name, and a verb. To make the computer ask, your program must say INPUT. Here's the program:

```
10 INPUT "WHAT'S YOUR NAME? ";Y$
15 PRINT
20 INPUT "WHAT'S YOUR FRIEND'S NAME? ";F$
25 PRINT
30 INPUT "IN 1 WORD, SAY SOMETHING YOU CAN DO TO
YOUR FRIEND ";V$
35 PRINT
40 PRINT "HERE'S MY STORY...."
45 PRINT
50 PRINT "ONCE UPON A TIME, THERE WAS A PERSON
NAMED ";Y$
55 PRINT
60 PRINT "WHO HAD A FRIEND NAMED ";F$
65 PRINT
70 PRINT Y$;" WANTED TO ";V$;" ";F$
75 PRINT
80 PRINT "BUT ";F$;" DIDN'T WANT TO ";V$;" ";Y$
85 PRINT
90 PRINT "WILL ";Y$;" ";V$;" ";F$
95 PRINT
100 PRINT "WILL ";F$;" ";V$;" ";Y$
105 PRINT
110 PRINT "TO FIND OUT, COME BACK AND SEE THE NEXT
EXCITING EPISODE "
115 PRINT
120 PRINT "OF ";Y$;" AND ";F$
```

Here's a sample run:

```
WHAT'S YOUR NAME? DRACULA
WHAT'S YOUR FRIEND'S NAME? MARILYN MONROE
IN 1 WORD, SAY SOMETHING YOU CAN DO TO YOUR
FRIEND? BITE
HERE'S MY STORY....
ONCE UPON A TIME, THERE WAS A PERSON NAMED DRACULA
WHO HAD A FRIEND NAMED MARILYN MONROE
DRACULA WANTED TO BITE MARILYN MONROE
BUT MARILYN MONROE DIDN'T WANT TO BITE DRACULA
WILL DRACULA BITE MARILYN MONROE
WILL MARILYN MONROE BITE DRACULA
TO FIND OUT, COME BACK AND SEE THE NEXT EXCITING
EPISODE
OF DRACULA AND MARILYN MONROE
```

Here's another run:

```
WHAT'S YOUR NAME? SNOW WHITE
WHAT'S YOUR FRIEND'S NAME? GRUMPY
IN 1 WORD, SAY SOMETHING YOU CAN DO TO YOUR
FRIEND? TICKLE
HERE'S MY STORY....
ONCE UPON A TIME, THERE WAS A PERSON NAMED SNOW
WHITE
WHO HAD A FRIEND NAMED GRUMPY
SNOW WHITE WANTED TO TICKLE GRUMPY
BUT GRUMPY DIDN'T WANT TO TICKLE SNOW WHITE
WILL SNOW WHITE TICKLE GRUMPY
WILL GRUMPY TICKLE SNOW WHITE
TO FIND OUT, COME BACK AND SEE THE NEXT EXCITING
EPISODE
OF SNOW WHITE AND GRUMPY
```

Try it: put in your own name, the name of a friend, and an appropriate verb.

This program creates a story that's fancier:

```
10 INPUT "WHAT'S YOUR NAME? ";Y$
20 INPUT "WHAT'S YOUR FRIEND'S NAME? ";F$
30 INPUT "WHAT'S THE NAME OF ANOTHER FRIEND? ";A$
40 INPUT "NAME A COLOR ";C$
50 INPUT "NAME A CITY ";P$
60 INPUT "NAME A FOOD ";D$
70 INPUT "NAME AN OBJECT ";J$
80 INPUT "NAME A PART OF THE BODY ";B$
90 INPUT "NAME A STYLE OF COOKING
(BAKED,FRIED,ETC) ";S$
100 PRINT
110 PRINT "CONGRATULATIONS, ";Y$
115 PRINT
120 PRINT "YOU'VE WON THE BEAUTY CONTEST, BECAUSE
OF YOUR EXTREMELY ATTRACTIVE ";B$
125 PRINT
130 PRINT "YOUR PRIZE IS A ";C$;" ";J$
135 PRINT
140 PRINT "PLUS A TRIP TO ";P$;" WITH YOUR FRIEND
";F$
145 PRINT
150 PRINT "PLUS...AND THIS IS THE BEST PART OF
ALL..."
155 PRINT
```

```
160 PRINT "DINNER FOR THE TWO OF YOU AT ";A$;"'S
RESTAURANT"
165 PRINT
170 PRINT "WHERE ";A$;" WILL GIVE YOU ALL THE
";S$;" ";D$;" YOU CAN EAT"
175 PRINT
180 PRINT "CONGRATULATIONS ";Y$;"...TODAY'S YOUR
LUCKY DAY..."
185 PRINT
190 PRINT "NOW EVERYBODY WANTS TO KISS YOUR
AWARD-WINNING ";B$
```

Want to see a sample run? OK:

```
WHAT'S YOUR NAME? ARLO
WHAT'S YOUR FRIEND'S NAME? OBIE
WHAT'S THE NAME OF ANOTHER FRIEND? ALICE
NAME A COLOR? RED
NAME A CITY? WEST STOCKBRIDGE
NAME A FOOD? TURKEY
NAME AN OBJECT? GARBAGE BAG
NAME A PART OF THE BODY? TOENAIL
NAME A STYLE OF COOKING (BAKED,FRIED,ETC)? ROAST

CONGRATULATIONS, ARLO
YOU'VE WON THE BEAUTY CONTEST, BECAUSE OF YOUR
EXTREMELY ATTRACTIVE TOENAIL
YOUR PRIZE IS A RED GARBAGE BAG
PLUS A TRIP TO WEST STOCKBRIDGE WITH YOUR FRIEND
OBIE
PLUS...AND THIS IS THE BEST PART OF ALL...
DINNER FOR THE TWO OF YOU AT ALICE'S RESTAURANT
WHERE ALICE WILL GIVE YOU ALL THE ROAST TURKEY YOU
CAN EAT
CONGRATULATIONS ARLO...TODAY'S YOUR LUCKY DAY...
NOW EVERYBODY WANTS TO KISS YOUR AWARD-WINNING
TOENAIL
```

Bills

If you're a nasty bill collector, you'll love this program:

```
10 INPUT "WHAT IS THE CUSTOMER'S FIRST NAME? ";F$
20 INPUT "LAST NAME? ";L$
30 INPUT "STREET ADDRESS? ";A$
40 INPUT "CITY? ";C$
50 INPUT "STATE? ";S$
60 INPUT "ZIP CODE? ";Z$
70 PRINT
80 PRINT F$;" ";L$
90 PRINT A$
100 PRINT C$;", ";S$;" ";Z$
110 PRINT
120 PRINT "DEAR ";F$;","
125 PRINT
130 PRINT "  YOU STILL HAVEN'T PAID THE BILL."
140 PRINT "IF YOU DON'T PAY IT SOON, ";F$;","
150 PRINT "I'LL COME AND VISIT YOU IN ";C$
160 PRINT "AND PERSONALLY PITCH A TENT "
170 PRINT "ON YOUR DOORSTEP UNTIL YOU DO."
175 PRINT
180 PRINT "                YOURS TRULY,"
185 PRINT
190 PRINT "                YOUR CREDITOR"
```

Can you figure out what this program does?

Numeric Input

This program makes the computer predict your future:

```
10 PRINT "I WILL PREDICT WHAT WILL HAPPEN TO YOU
IN THE YEAR 2000!"
20 INPUT "IN WHAT YEAR WERE YOU BORN? ";Y
30 PRINT "IN THE YEAR 2000, YOU WILL BE
";2000-Y;"."
```

Here's a sample run:

```
I WILL PREDICT WHAT WILL HAPPEN TO YOU IN THE YEAR
2000!

IN WHAT YEAR WERE YOU BORN? 1954
IN THE YEAR 2000, YOU WILL BE 46.
```

Suppose you're selling tickets to a play. Each ticket costs $2.79. (You decided $2.79 would be a nifty price, because the cast has 279 people.) This program finds the price of multiple tickets:

```
10 INPUT "HOW MANY TICKETS? ";T
15 PRINT
20 PRINT "THE TOTAL PRICE IS $";T*2.79
```

This program tells you how much the "energy crisis" costs you, when you drive your car:

```
10 INPUT "HOW MANY MILES DO YOU WANT TO DRIVE? ";M
20 INPUT "HOW MANY PENNIES DOES A GALLON OF GAS
COST? ";P
30 INPUT "HOW MANY MILES-PER-GALLON DOES YOUR CAR
GET? ";R
35 PRINT
40 PRINT "THE GAS FOR YOUR TRIP WILL COST YOU $";
M*P/(R*100)
```

Here's a sample run:

```
HOW MANY MILES DO YOU WANT TO DRIVE? 300
HOW MANY PENNIES DOES A GALLON OF GAS COST? 97.9
HOW MANY MILES-PER-GALLON DOES YOUR CAR GET? 27

THE GAS FOR YOUR TRIP WILL COST YOU $10.8778
```

This program converts feet to inches:

```
10 INPUT "HOW MANY FEET? ";F
20 PRINT F;" FEET = ";F*12;" INCHES"
```

Here's a sample run:

```
HOW MANY FEET? 3
3 FEET = 36 INCHES
```

Trying to convert to the metric system? This program converts inches to centimeters.

```
10 INPUT "HOW MANY INCHES? ";I
20 PRINT I;" INCHES = ";I*2.54;" CENTIMETERS"
```

Nice day today, isn't it? This program converts the temperature from Celsius to Fahrenheit:

```
10 INPUT "HOW MANY DEGREES CELSIUS? ";C
20 PRINT C;" DEGREES CELSIUS = ";C*1.8+32;"
DEGREES FAHRENHEIT"
```

Here's a sample run:

```
HOW MANY DEGREES CELSIUS? 20
20 DEGREES CELSIUS = 68 DEGREES FAHRENHEIT
```

Digital Data Packs

While you're working on a program, the computer keeps it in the main RAM memory. It is erased from main memory when you type NEW (to begin a new program) or when somebody turns off the computer's power.

To store the program longer, copy it onto the computer's digital data pack. Once you've copied the program onto a digital data pack, it will remain on the digital data pack, even if you type NEW or turn off the power. It will stay on the digital data pack permanently, unless you rename, delete, save over it, or initialize the data packs.

To copy a program onto the digital data pack, type the program and then type the word SAVE. You must also invent a name for the program.

For example, suppose you want to copy this program onto a digital data pack:

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

To name that program "JOE", and copy it onto the digital data pack, type this:

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

Instead of JOE, you can invent a different name, as long as it doesn't exceed 10 characters. In file names, ADAM knows the difference between capital and small letters. For example, "joe","JOE",and "Joe" would be treated as three different files.

To prove that your saved program is on the digital data pack, type this:

```
CATALOG
```

That makes the computer print a catalog of everything you've stored on your digital data pack. The catalog will list the names of all the programs you've stored. And one of the names you'll see will be JOE. See the Reference Section at the back of this book for a full explanation of CATALOG.

Suppose you come back to the computer sometime in the future, and want to use JOE, which is on the digital data pack. Type this:

```
LOAD JOE
```

That makes the computer copy JOE from the digital data pack to the main memory. Then you can type LIST or RUN, and the computer will list or run JOE.

If you ever want to erase JOE from the digital data pack, type this:

```
DELETE JOE
```

If you ever want to revise the version of JOE that's on the digital data pack, copy JOE from the digital data pack to the main memory (by typing LOAD JOE). Then type your revisions, by retyping some of the old program's lines, or by adding new ones. Finally, type this:

```
SAVE JOE
```

That makes the computer replace JOE with your new version.

Be aware that SmartBASIC files and ADAM Word Processing files are compatible. In other words, you can take a SmartBASIC file that you've SAVEd and READ it into the SmartWRITER word processor as a regular document for editing; but all your PRINTs will show up as "?". You can use either PRINT or ?. Because your work is not checked for syntax errors when you code or edit using word processing this option is not recommended. When you LOAD a program that was created or edited using SmartWRITER, any lines having syntax errors will be rejected.

To keep your files from being deleted accidentally, here's what to do:

```
LOCK JOE
```

Stop

The computer understands the word STOP:

```
10 PRINT "BUBBLE GUM"
20 STOP
30 PRINT "FOX"
```

The computer will print BUBBLE GUM and then stop, without printing FOX. After the computer stops, press RETURN and it'll print a bracket. Then you can type any command, such as LIST (to see the program again) or RUN (to make the computer print BUBBLE GUM again) or NEW (to create a new program).

Colons

Colons are useful for putting several statements on the same line, like this:

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

That line says A is 5, and B is 7, and to print A+B. So the computer will print 12.

If/Then

Let's make ADAM interrogate a human. Let's make it begin the interrogation by asking whether the human is male or female. If the human answers MALE, let's make the computer say:

SO IS FRANKENSTEIN

If the human answers FEMALE, let's make the computer say:

SO IS MARY POPPINS

If the human gives a different answer (such as SUPER-MALE or NOT SURE or BOTH or YES, let's make the computer say:

PLEASE SAY 'MALE' OR 'FEMALE'
ARE YOU MALE OR FEMALE?

This will force the human to answer the question correctly. This program does it:

```
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
40 PRINT "PLEASE SAY 'MALE' OR 'FEMALE'": GOTO 10
```

Line 10 makes the computer ask ARE YOU MALE OR FEMALE? and wait for the human's answer, which is called A$. If the human's answer is MALE, line 20 makes the computer print SO IS FRANKENSTEIN and then stop. If the human's answer is FEMALE, line 30 makes the computer print SO IS MARY POPPINS and then stop. If the human's answer is neither MALE nor FEMALE, the computer skips over lines 20 and 30, so it comes to line 40, which makes it print PLEASE SAY 'MALE' OR 'FEMALE' and then go back to line 10, which forces the human to answer the question again. This is one instance where the difference between upper and lower case letters matters.

Here's a sample run:

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

Here's another:

```
RUN
ARE YOU MALE OR FEMALE? FEMALE
SO IS MARY POPPINS
```

Here's another:

```
RUN
ARE YOU MALE OR FEMALE? SUPER-MALE
PLEASE SAY 'MALE' OR 'FEMALE'
ARE YOU MALE OR FEMALE? MALE
SO IS FRANKENSTEIN
```

Let's extend the conversation. If the human says FEMALE, let's make the computer say SO IS MARY POPPINS and then ask DO YOU LIKE HER? If the human says YES, let's make the computer say:

```
I LIKE HER, TOO--SHE IS MY MOTHER
```

If the human says NO, let's make the computer say:

```
NEITHER DO I--SHE STILL OWES ME A DIME
```

If the human says neither YES nor NO, let's make the computer say:

```
PLEASE SAY 'YES' OR 'NO'
DO YOU LIKE HER?
```

Here's the 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":
GOTO 100
40 PRINT "PLEASE SAY 'MALE' OR 'FEMALE'": GOTO 10
100 INPUT "DO YOU LIKE HER? ";B$
110 IF B$="YES" THEN PRINT "I LIKE HER TOO--SHE IS
MY MOTHER": END
120 IF B$="NO" THEN PRINT "NEITHER DO I--SHE STILL
OWES ME A DIME": END
130 PRINT "PLEASE SAY 'YES' OR 'NO'": GOTO 100
```

Line 30 says: if the human's answer is FEMALE, print SO IS MARY POPPINS and then go to line 100, which asks, DO YOU LIKE HER?

Strange programs

The computer is like a human; it would like to make new friends. This program makes the computer show its true feelings:

```
10 INPUT "ARE YOU MY FRIEND? ";A$
20 IF A$="YES" THEN PRINT "THAT'S SWELL": END
30 IF A$="NO" THEN PRINT "GO JUMP IN A LAKE": END
40 PRINT "PLEASE SAY 'YES' OR 'NO'": GOTO 10
```

When you type RUN, the computer asks ARE YOU MY FRIEND? If you say YES, the computer says THAT'S SWELL. If you say NO, the computer says GO JUMP IN A LAKE.

Here's a program that was written by a girl in the sixth grade:

```
10 INPUT "CAN I COME OVER TO YOUR HOUSE TO WATCH
T.V.? ";A$
20 IF A$="YES" THEN PRINT "THANKS.  I'LL BE THERE
AT 5 P.M.": END
30 IF A$="NO" THEN PRINT "HUMPH!  I HAVE BETTER
THINGS TO DO, ANYWAY.": END
40 PRINT "PLEASE SAY 'YES' OR 'NO'": GOTO 10
```

When you type RUN, the computer asks to watch your television. If you say YES, the computer promises to come to your house at 5. If you refuse, the computer insults you.

Another sixth grade girl wrote a program to test your honesty:

```
10 PRINT "KDJFUEOEE*^&^%^KLO*&@!@>]_"
20 PRINT "LLLL]__))*&(~~++?:LJSF,,"
30 PRINT "JSUEU2JFW098B<,.,./[???=-["
40 INPUT "DO YOU UNDERSTAND WHAT I SAID? ";A$
50 IF A$="NO" THEN PRINT "NEITHER DO I": END
60 IF A$="YES" THEN GOTO 80
70 PRINT "PLEASE SAY 'YES' OR 'NO' ": GOTO 10
80 PRINT "IT DOESN'T MEAN ANYTHING!!  I'M A
COMPUTER, YOU KNOW!  YOU CAN'T FOOL ME!"
```

When you type RUN, lines 10-30 print nonsense. Then the computer asks whether you understand that stuff. If you're honest and answer NO, the computer will confess that it was beyond understanding. But if you pretend that you understand the nonsense, and answer YES, the computer will scold you for trying to put one over on it.

A Daddy wrote a program for his five-year-old son, John. When John types RUN, the computer asks WHAT'S 2 AND 2? If John answers 4, the computer says NO, 2 AND 2 IS 22. If he runs the program again and answers 22, the computer says NO, 2 AND 2 IS 4. No matter how many times he runs the program and how he answers the question, the computer says he's wrong. But when Daddy runs the program, the computer replies, YES, DADDY IS ALWAYS RIGHT. Here's how Daddy programmed the computer:

```
10 INPUT "WHAT'S YOUR NAME? ";N$
20 INPUT "WHAT'S 2 AND 2? ";A
30 IF N$="DADDY" THEN PRINT "YES, DADDY IS ALWAYS
RIGHT": END
40 IF A=4 THEN PRINT "NO, 2 AND 2 IS 22": END
50 IF A=22 THEN PRINT "NO, 2 AND 2 IS 4"
```

Your personality

This program makes the computer act human:

```
10 PRINT "HELLO"
20 INPUT "WHAT'S YOUR NAME? ";N$
30 PRINT "GLAD TO MEET YOU, ";N$
40 INPUT "HOW ARE YOU FEELING TODAY? ";F$
50 IF F$="FINE" THEN PRINT "THAT'S GOOD": END
60 IF F$="AWFUL" THEN PRINT "TOO BAD": END
70 PRINT "I FEEL THE SAME WAY"
```

When you type RUN, the computer begins the conversation by saying HELLO and asking for your name. Then it says GLAD TO MEET YOU, followed by your name; for example, if you said your name was JOEY, the computer will say:

```
GLAD TO MEET YOU, JOEY
```

Then the computer asks how you're feeling. If you say FINE, line 50 makes the computer say THAT'S GOOD. If you say AWFUL, line 60 makes the computer say TOO BAD. If you say neither FINE nor AWFUL, the computer skips lines 50 and 60 and arrives at line 70, so it says I FEEL THE SAME WAY (which is a safe, general response).

That program makes the computer imitate an "average American". But you're not average! Make the computer imitate <u>your</u> personality!

The first line of the program could say:

```
10 PRINT "HELLO"
```

or--

```
10 PRINT "HI"
```

or, if you're from Texas--

```
10 PRINT "HOWDY, PARDNER!"
```

or, if you're cool--

```
10 PRINT "HEY, BABY!  WHAT'S HAPPENING?"
```

or, if you're a nurse working in a private practice--

```
10 PRINT "GOOD MORNING.  THE DOCTOR WILL BE WITH
YOU SHORTLY"
```

or, if you're nasty--

```
10 PRINT "GO AWAY!  CAN'T YOU SEE I'M SLEEPING?!"
```

After your opening line, you'll need an INPUT statement, that makes the computer ask the human a question. After the INPUT statement, you'll need a series of IF...THEN statements that make the computer react to the various responses the human might give.

Try to make your program long, so the conversation between the computer and human lasts several minutes. Your program will be a mass of PRINT, INPUT, IF...THEN and GOTO statements.

See how human you can make your ADAM computer! See how well you can make the computer imitate you! And remember to SAVE your program, so the computer will adopt your personality permanently.

Fancy clauses

You can make the IF clause very fancy:

IF clause	Meaning
IF A$="MALE"	If A$ is "MALE"
IF A=4	If A is 4
IF A<4	If A is less than 4
IF A<=4	If A is less than or equal to 4
IF A>4	If A is greater than 4
IF A>=4	If A is greater than or equal to 4
IF A<>4	If A is not 4
IF A$<"MALE"	If A$ is a word that comes before "MALE" in the dictionary
IF A$>"MALE"	If A$ is a word that comes after "MALE" in the dictionary

LESSON 3

MASTER YOUR COMPUTER

Reminders

• If you get lost in your program, type this command:

LIST

• You must put quotation marks around each string, and a dollar sign after each string variable:

A$="JERK"	A$=JERK	A="JERK"
(CORRECT)	(WRONG)	(WRONG)

• To make your computer execute your program again and again, automatically, add an extra line at the bottom of your program. That extra line should tell the computer to GOTO the top line.

• To say "not equal to", say "less than or greater than", like this: <>.

• Remember that IF is always paired with THEN or GOTO.

The only way to become a computer expert is to try writing your own programs, so go to it!

Debugging

An error in a program is called a "bug".

Chances are quite good that, when you run your program after typing it out, it will be error-free. Because of the SmartBASIC specific error message/prompt feature, you will have debugged your program as you went along. Most of your programs will be gold-star programs...that is, they will work on the first run. But the proper way to debug a program is a good thing to know if your interest in computers should grow, so we've inserted a brief section on efficient debugging here.

To find the bug, use three techniques:

1. Inspect the program.
2. Trace the computer's thinking
3. Shorten the program

Here are the details....

Inspect the program

Take a good, hard look at the program. If you stare hard enough, maybe you'll see the bug.

Ask the computer to help you. Make the comp ter print the entire program. To do that, type:

```
LIST
```

Usually, the bug will turn out to be a typing error. Maybe you:

- typed the letter O instead of zero? or
- typed zero instead of the letter O?
- typed the letter "l" instead of the number "1" or the number "1" instead of the letter "l"?
- pressed the SHIFT key when you weren't supposed to? or forgot to press it? or had the SHIFT LOCK on?
- typed an extra letter? or omitted a letter?
- typed a line you thought you hadn't? or omitted a line? or wrote over a line?

Trace the computer's thinking

If you've inspected the program thoroughly, and still haven't found the bug, the next step is to trace the computer's thinking. Pretend you are the computer. Do what your program says. Do you find yourself printing the same wrong answers the computer printed? If so, why?

To help your analysis, make the computer print everything it's thinking, while it's running your program. For example, suppose your program contains lines 10, 20, 30 and 40, and uses the variables B, C, and X$. Insert these lines into your program:

```
15 PRINT "I'M AT LINE 15.  THE VALUES ARE ";B;"
";C;" ";X$
25 PRINT "I'M AT LINE 25.  THE VALUES ARE ";B;"
";C;" ";X$
35 PRINT "I'M AT LINE 35.  THE VALUES ARE ";B;"
";C;" ";X$
45 PRINT "I'M AT LINE 45.  THE VALUES ARE ";B;"
";C;" ";X$
```

Then type the word RUN. The computer will run the program again; but lines 15, 25, 35, and 45 make the computer print everything it's thinking.

Check what the computer prints. If the computer prints what you expect in lines 15 and 25, but prints strange values in line 35 (or doesn't even get to line 35), you know the bug occurs before line 35 but after line 25; so the bug must be in line 30.

If your program contains hundreds of lines, you might not have the patience to insert lines 15,25,35,45,55,65,75,etc. Here's a short-cut....

Halfway down your program, insert a line that says to print all the values. Then run your program. If the line you inserted prints the correct values, you know the bug lies underneath that line; but if the line prints wrong values (or if the computer never reaches that line), you know the bug lies above that line. In either case, you know which half of your program contains the bug. In that half of the program, insert more lines until you finally zero in on the line that contains the bug.

Another way to trace program execution is with TRACE and NOTRACE. You can read more about this in the Reference Section at the back of this manual.

Shorten the program

When all else fails, shorten the program.

Hint: Before you shorten your program (or write tiny experimental ones), SAVE the original version, even though it contains a bug. After you've played with the shorter versions, retrieve the original and fix it.

Finding a bug in a program is like finding a needle in a haystack: the job is easier if the haystack is smaller. So make your program shorter: delete the last half of your program. Then run the shortened version. That way, you'll find out whether the first half of your program is working the way it's supposed to. When you've perfected the first half of your program, tack the second half back on.

Does your program contain a statement whose meaning you're not completely sure of? Check the meaning. Use a reference guide, ask a friend, or write a tiny experimental program that contains the statement, and see what happens when you type RUN.

The easiest way to write a long, correct program is to write a short program first, debug it, then add a few more lines, debug them, add a few more lines, debug them, etc. In other words, start with a small program, perfect it, and then add perfected extras, so you gradually build a perfected masterpiece. This is the idea on which ADAM SmartBASIC is based. If you try to compose a long program all at once--instead of building it from perfected pieces--you could end up with nothing more than a mastermess--full of bugs.

Moral: to build a large masterpiece, start with a small masterpiece. Putting it another way: to build a skyscraper, begin by laying a good foundation; and by double-checking the foundation before you start adding the walls and the roof.

Editing

To list your entire program, just type:

LIST

Here's how to list just line 20:

LIST 20

Here's how to list from line 20 to line 50:

LIST 20-50
or
LIST 20,50

Here's how to list from line 20 to the end:

LIST 20-
or
LIST 20,

Here's how to list from the beginning to line 20:

LIST ,20

To delete line 20, just type:

20 (and press RETURN)

To delete lines 20-50, type this:

DEL 20-50
or
DEL 20,50

To delete (scratch out) all the lines in the program, type
this:

NEW

Pause

To freeze your computer screen while your program is listing, hold down the CONTROL key. While you are holding this key down, lightly tap the S key.

After you've frozen the screen, read what's on it. Then you have two choices: either abort your program with a CONTROL-C or make the computer resume printing. Here's how to make the computer resume printing (onscreen only): hold down the CONTROL key and lightly tap the S key. Right. It's the same way you stopped it!

Data...Read

To make the computer handle a list, begin your program with the word DATA. For example, suppose you want the computer to print this list of food:

```
MEAT
POTATOES
LETTUCE
TOMATOES
BUTTER
CHEESE
ONIONS
PEAS
```

Begin your program by saying:

```
10 DATA MEAT,POTATOES,LETTUCE,TOMATOES,
BUTTER,CHEESE,ONIONS,PEAS
```

You must tell the computer to READ the DATA:

```
20 READ A$
```

Line 20 makes the computer read the first datum (which is MEAT), and call it A$. So A$ is MEAT. This line makes the computer print MEAT:

```
30 PRINT A$
```

This line makes the computer handle the rest of the data:

```
40 GOTO 20
```

Line 40 makes the computer go back to line 20, which reads the next datum (POTATOES). Altogether, the program looks like this:

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

Lines 20-40 form a loop. (Like most loops, the loop's bottom line says GOTO. Since the loop's top line says READ, the loop is called a READ loop.) The computer goes round and round the loop. Each time the computer comes to line 20, it reads another datum: the first time it comes to line 20, it reads MEAT; the next time, it reads POTATOES; the next time, it reads LETTUCE; etc. Line 30 makes the computer PRINT what it's read. Altogether, the computer will print:

```
MEAT
POTATOES
LETTUCE
TOMATOES
BUTTER
CHEESE
ONIONS
PEAS
```

After the computer prints PEAS, it comes to line 40 again, which makes it go back to line 20 again, so the computer tries to read more data again; but no more data remains! So the computer says:

```
OUT OF DATA ERROR IN 20
```

Then the computer stops.

So the last three lines the computer prints are:

```
ONIONS
PEAS
OUT OF DATA ERROR IN 20
```

Instead of saying OUT OF DATA/ERROR IN 20, let's make the computer say THOSE ARE MY FAVORITE FOODS, so that the last three lines look like this:

```
ONIONS
PEAS
THOSE ARE MY FAVORITE FOODS
```

Here's how....At the end of the data, say DONE:

```
10 DATA MEAT,POTATOES,LETTUCE,TOMATOES,
BUTTER,CHEESE,ONIONS,PEAS,DONE
```

When the computer reads the DONE, make the computer say THOSE ARE MY FAVORITE FOODS and then stop:

```
20 READ A$:  IF A$="DONE" THEN PRINT "THOSE ARE MY
FAVORITE FOODS": END
```

The DONE at the end of the data is called the **end mark**, because it marks the end of the data. The routine that says:

```
IF A$="DONE" THEN PRINT "THOSE ARE MY FAVORITE
FOODS": END
```

is called the **end routine**, because the computer does that routine at the end.

That program prints one copy of the computer's favorite foods. If you want the computer to print many copies, change line 20 to this:

```
20 READ A$:  IF A$="DONE" THEN PRINT "THOSE ARE MY
FAVORITE FOODS":  RESTORE:  GOTO 10
```

The word RESTORE tells the computer to go back to the beginning of the data. The computer will print:

```
MEAT
POTATOES
LETTUCE
TOMATOES
BUTTER
CHEESE
ONIONS
PEAS
THOSE ARE MY FAVORITE FOODS
MEAT
POTATOES
LETTUCE
TOMATOES
BUTTER
CHEESE
etc.
```

The computer will print copies of its list of foods again and again, forever, unless you abort your program.

Most practical computer programs involve a list of data. Your job, as a programmer, is to find out what the data is, and to begin your program by saying DATA. After typing the data, the next statement in your program should say READ. Farther down in your program, you should say GOTO, so that you create a READ loop. Your program consists of two parts: the DATA, and the READ loop.

Debts

Suppose these people owe you things:

Person	What person owes
Bob	$537.29
Mike	a dime
Frank	2 golf balls
Harry	a steak dinner at Mario's
Mommy	a kiss

Let's remind those people of their debts, by writing them letters, in this form:

DEAR _____,
person's name

I JUST WANT TO REMIND YOU...

THAT YOU STILL OWE ME _____
debt

Begin with the DATA:

```
10 DATA BOB,$537.29,MIKE,A DIME,FRANK,2 GOLF BALLS,
HARRY,A STEAK DINNER AT MARIO'S,MOMMY,A KISS
```

If you decide to split a long DATA statement up into two or more smaller DATA statements, remember

to leave the comma off the ends of the DATA lines. To be safe, you can insert quotation marks, like this:

```
10 DATA "BOB","$537.29","MIKE","A DIME","FRANK","2
GOLF BALLS","HARRY","A STEAK DINNER AT
MARIO'S","MOMMY","A KISS"
```

On ADAM, the quotation marks are unnecessary as long as your data string itself doesn't contain a comma or a colon.

In this program, the data comes in pairs: the first pair consists of BOB and $537.29. Tell the computer to READ each pair of DATA:

```
20 READ P$,D$
```

Line 30 makes the computer read the first pair of data; so P$ is the first person (BOB) and D$ is his debt ($537.29). These lines print the letter:

```
30 PRINT "DEAR ";P$;","
35 PRINT
40 PRINT "          I JUST WANT TO REMIND YOU..."
50 PRINT "THAT YOU STILL OWE ME ";D$;"."
```

At the end of the letter, leave two blank lines, to make room for your signature:

```
60 PRINT
70 PRINT
```

Then complete the READ loop, by making the computer go back to the beginning of the loop to read the next pair:

```
80 GOTO 20
```

The computer will print a letter to each person, like this:

```
DEAR BOB,

          I JUST WANT TO REMIND YOU...
THAT YOU STILL OWE ME $537.29.
```

DEAR MIKE,

I JUST WANT TO REMIND YOU...
THAT YOU STILL OWE ME A DIME.

etc.

After printing all the letters, the computer will say:

OUT OF DATA ERROR IN 20

Instead of saying OUT OF DATA, let's make the computer say:

I'VE FINISHED WRITING THE LETTERS

To do that, put a DONE at the end of the data, twice:

```
"HARRY","A STEAK DINNER AT MARIO'S","MOMMY",
"A KISS","DONE","DONE"
```

and say what to do when the computer reaches the DONEs:

```
30 READ P$,D$:  IF P$="DONE" THEN PRINT "I'VE
FINISHED WRITING THE LETTERS": END
```

You need two DONE's at the end of the data, because the READ statement says to read two strings (P$ and D$).

Diets

Suppose you're running a diet clinic, and get these results:

Person	Weight before	Weight after
Joe	273 pounds	219 pounds
Mary	412 pounds	371 pounds
Bill	241 pounds	173 pounds
Sam	309 pounds	198 pounds

Here's how to make the computer print a nice report....Begin by feeding it the Data:

```
10 DATA JOE,273,219,MARY,412,371,BILL,
241,173,SAM,309,198
```

You can place your DATA information within quotes, if you prefer. Use of quotes on ADAM is optional and up to the user's discretion. Here is what your DATA statement would look like within quotes:

```
10 DATA "JOE",273,219,"MARY",412,371,
"BILL",241,173
20 DATA "SAM",309,198
```

The data comes in triplets: the first triplet consists of JOE, 273, and 219. Tell the computer to READ each triplet of DATA:

```
20 READ N$,B,A
```

That line makes the computer read the first triplet of data; so N$ is the first person's name (JOE), B is his weight before (273), and A is his weight after (219). These lines print the report about him:

```
30 PRINT N$;" WEIGHED ";B;" POUNDS BEFORE
ATTENDING THE DIET CLINIC"
40 PRINT "BUT WEIGHED ONLY ";A;" POUNDS
AFTERWARDS"
50 PRINT "THAT'S A LOSS OF ";B-A;" POUNDS"
```

At the end of the report about him, leave a blank line:

```
60 PRINT
```

Then complete the READ loop, by making the computer go back to the loop's beginning:

```
70 GOTO 20
```

The computer will print:

```
JOE WEIGHED 273 POUNDS BEFORE ATTENDING THE DIET
CLINIC
BUT WEIGHED ONLY 219 POUNDS AFTERWARDS
THAT'S A LOSS OF 54 POUNDS

MARY WEIGHED 412 POUNDS BEFORE ATTENDING THE DIET
CLINIC
BUT WEIGHED ONLY 371 POUNDS AFTERWARDS
THAT'S A LOSS OF 41 POUNDS

etc.
```

At the end, the computer will say OUT OF DATA ERROR IN 20. Instead, let's make it say:

```
COME TO OUR DIET CLINIC!
```

To do that, put a DONE and two zeros at the end of the data:

```
10 DATA
"JOE",273,219,"MARY",412,371,"BILL",241,173,
20 DATA "SAM",309,198,"DONE",0,0
```

and say what to do when the computer reaches the DONE:

```
30 READ N$,B,A:  IF N$="DONE" THEN PRINT "COME TO
OUR DIET CLINIC!": END
```

You need the two zeros after the DONE, because the READ statement says to read two numbers (B and A) after the string N$. If you omit the zeros, the computer will say OUT OF DATA ERROR IN 20. If you hate zeros, you can use other numbers instead; but most programmers prefer zeros.

French colors

Let's make the computer translate colors into French. For example, if the human says RED, we'll make the computer say the French equivalent, which is:

```
ROUGE
```

Altogether, a run will look like this:

```
RUN
WHICH COLOR INTERESTS YOU? RED
IN FRENCH, IT'S ROUGE
```

The program begins simply:

```
10 INPUT "WHICH COLOR INTERESTS YOU? ";C$
```

Next, we must make the computer translate the color into French. To do that, feed the computer this English-French dictionary:

English	French
white	blanc
yellow	jaune
orange	orange
red	rouge
green	vert
blue	bleu
brown	brun
black	noir

That dictionary becomes the data:

```
20 DATA WHITE,BLANC,YELLOW,JAUNE,
ORANGE,ORANGE,RED,ROUGE
30 DATA GREEN,VERT,BLUE,BLEU,BROWN,BRUN,BLACK,NOIR
```

The data comes in pairs: the first pair consists of WHITE and BLANC. Tell the computer to READ each pair of DATA:

```
40 READ E$,F$
```

That line makes the computer read the first pair of data; so E$ is the first English color (WHITE), and F$ is its French equivalent (BLANC).

But that pair of data might be the wrong pair; for example, if the human requested RED, the human does not want the pair of WHITE and BLANC; instead, the human wants the pair of RED and ROUGE. So you must tell the computer: if the

human's input doesn't match the English in the pair, go read another pair. Here's how to to say that:

```
50 IF E$<>C$ THEN GOTO 40
```

That line says: if E$ (which is the English in the data) is not C$ (the color the human requested), go to 40 (which reads another pair).

Lines 40 and 50 form a loop; the computer goes round and round the loop, until it finds the pair of data that matches the human's request. Since the purpose of the loop is to search for data that matches, it's called a search loop.

After the computer has found the correct English-French pair, make the computer print the French:

```
60 PRINT "IN FRENCH, IT'S ";F$
```

Altogether, the program looks like this:

Ask the human:

```
10 INPUT "WHICH COLOR
INTERESTS YOU? ";C$
```

Use this dictionary:

```
20 DATA WHITE,BLANC,
YELLOW,JAUNE,ORANGE
ORANGE,RED,ROUGE

30 DATA GREEN,VERT,
BLUE,BLEU,BROWN,BRUN,
BLACK,NOIR
```

Look at the dictionary:

```
40 READ E$,F$
```

If not found, look again:

```
50 IF E$<>C$
THEN    GOTO 40
```

Print the French:

```
60 PRINT "IN
FRENCH,
IT'S ";F$
```

Here's a sample run:

```
RUN
WHICH COLOR INTERESTS YOU? RED
IN FRENCH, IT'S ROUGE
```

Here's another:

```
RUN
WHICH COLOR INTERESTS YOU? BROWN
IN FRENCH, IT'S BRUN
```

Here's another:

```
RUN
WHICH COLOR INTERESTS YOU? PINK
OUT OF DATA ERROR IN 30
```

The computer says OUT OF DATA ERROR IN 30 because i can't find PINK in the data. Instead of saying OUT OF DATA ERROR IN 30, it would be nicer to say I WASN'T TAUGHT THAT COLOR.

We want the computer to say I WASN'T TAUGHT THAT COLOR, when it reaches the end of the data. To do that, say DONE at the end of the data; and when the computer reaches the DONE, tell it to say I WASN'T TAUGHT THAT COLOR:

```
10 INPUT "WHICH COLOR INTERESTS YOU? ";C$
20 DATA WHITE,BLANC,YELLOW,JAUNE,ORANGE,ORANGE,
RED,ROUGE
30 DATA GREEN,VERT,BLUE,BLEU,BROWN,BRUN,BLACK,
NOIR,DONE,
40 READ E$,F$: IF E$="DONE" THEN PRINT "I WASN'T
TAUGHT THAT COLOR": END
50 IF E$<>C$ THEN GOTO 40
60 PRINT "IN FRENCH, IT'S ";F$
```

After line 60, the program just ends. Instead of letting the computer end, let's make it automatically rerun the program and translate another color. To do that, say GOTO and RESTORE:

```
10 INPUT "WHICH COLOR INTERESTS YOU? ";C$
20 DATA
WHITE,BLANC,YELLOW,JAUNE,ORANGE,ORANGE,RED,ROUGE
30 DATA
GREEN,VERT,BLUE,BLEU,BROWN,BRUN,BLACK,NOIR,DONE,
DONE
40 READ E$,F$: IF E$="DONE" THEN PRINT "I WASN'T
TAUGHT THAT COLOR":  GOTO 70
50 IF E$<>C$ THEN GOTO 40
60 PRINT "IN FRENCH, IT'S ",F$
70 RESTORE
80 GOTO 10
```

To exit this, and any other program containing a loop that you must break, hit CONTROL-C.

For...Next

Let's make the computer print every number from 1 to 100, like this:

```
1
2
3
4
5
6
7
etc.
100
```

To do that, type this line:

```
20 PRINT X
```

and also tell the computer that you want X to be every number from 1 to 100. To tell the computer that, say "FOR X = 1 TO 100", like this:

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

Whenever you write a program that contains the word FOR, you must say NEXT. So your program should look like this:

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

That program works; it makes the computer print every number from 1 to 100.

Here's how it works...The computer begins at line 10, which says that you want X to be every number from 1 to 100. So X starts at 1. Then the computer comes to line 20, which says to print X; so the computer prints: 1

Then the computer comes to line 30, which says to do the same thing for the next X, and for the next X, and for the next X; so the computer prints 2, and 3, and 4, and so on, all the way up to 100.

That program makes the computer print many numbers. That's because the computer does line 20 many times (once for each X). The computer does line 20 many times because that line is between the words FOR and NEXT: it is underneath FOR, and above NEXT. The computer repeats anything that's between the words FOR and NEXT.

Party

For a more advanced example, let's make the computer print these lyrics:

```
I SAW 2 GULLS
MEET 2 BUOYS
TRA-LA-LA!

I SAW 3 GULLS
MEET 3 BUOYS
TRA-LA-LA!

I SAW 4 GULLS
MEET 4 BUOYS
TRA-LA-LA!
```

```
I SAW 5 GULLS
MEET 5 BUOYS
TRA-LA-LA!

THEY ALL HAD A PARTY!
HA-HA-HA!
```

To do that, type these lines:

```
20 PRINT "I SAW ";X;" GULLS"
30 PRINT "MEET ";X;" BUOYS"
40 PRINT "TRA-LA-LA!"
50 PRINT
```

This gives you each line of each verse plus a blank line underneath each verse.

Now you have to tell the computer that you want X to be every number from 2 up to 5. Here's how:

```
10 FOR X = 2 TO 5
20 PRINT "I SAW ";X;" GULLS"
30 PRINT "MEET ";X;" BUOYS"
40 PRINT "TRA-LA-LA!"
50 PRINT
60 NEXT X
```

At the end of the song, print the closing couplet:

```
10 FOR X = 2 TO 5
20 PRINT "I SAW ";X;" GULLS"
30 PRINT "MEET ";X;" BUOYS
40 PRINT "TRA-LA-LA!"
50 PRINT
60 NEXT X
70 PRINT "THEY ALL HAD A PARTY!"
80 PRINT "HA-HA-HA!"
```

That program works; it makes the computer print the entire song.

Since the computer does lines 20-50 repeatedly, those lines form a loop. Here's the general rule: the statements between FOR and NEXT form a loop. The computer goes round and round the loop, for X=2, X=3, X=4, and X=5. Altogether, it goes around the loop 4 times, which is a finite number. Therefore, the loop is <u>finite</u>.

If you don't like the letter X, choose a different letter. For example, you can choose the letter I:

```
10 FOR I = 2 TO 5
20 PRINT "I SAW ";I;" GULLS"
30 PRINT "MEET ";I;" BUOYS"
40 PRINT "TRA-LA-LA!"
50 PRINT
60 NEXT I
70 PRINT "THEY ALL HAD A PARTY!"
80 PRINT "HA-HA-HA!"
```

Most programmers prefer the letter I, when using the word FOR. In other words, most programmers say "FOR I", instead of "FOR X". Saying "FOR I" is an "old tradition". We'll follow the tradition: in the rest of this book, we'll say "FOR I", except in situations where some other letter feels peculiarly more natural.

Squares

To find the square of a number, multiply the number by itself. The square of 3 is "3 times 3", which is 9. The square of 4 is "4 times 4", which is 16.

Let's make the computer print the square of 3, 4, 5 etc., up to 100, like this:

```
THE SQUARE OF 3 IS 9
THE SQUARE OF 4 IS 16
THE SQUARE OF 5 IS 25
THE SQUARE OF 6 IS 36
THE SQUARE OF 7 IS 49
etc.
THE SQUARE OF 100 IS 10000
```

To do that, type this line:

```
20 PRINT "THE SQUARE OF ";I;" IS ";I*I
```

now tell the computer that you want I to be every number from 3 up to 100. Here's the program:

```
10 FOR I = 3 TO 100
20 PRINT "THE SQUARE OF ";I;" IS ";I*I
30 NEXT I
```

Secret meeting

This program prints 12 copies of the same message:

```
10 FOR I = 1 TO 12
20 PRINT "HUSH,HUSH!"
30 PRINT " WE'RE HAVING A SECRET MEETING..."
40 PRINT "    IN THE COMPUTER ROOM..."
50 PRINT "      TONIGHT..."
60 PRINT "       AT 2 A.M."
70 PRINT "         WEAR A FUNNY HAT."
80 PRINT
90 PRINT
100 NEXT I
```

Lines 80 and 90 leave blank lines at the end of each copy, for your signature.

Midnight

This program makes the computer count to midnight:

```
10 FOR I = 1 TO 12
20 PRINT I
30 NEXT I
40 PRINT "MIDNIGHT"
```

The computer will print:

```
1
2
3
4
5
6
7
8
9
10
11
12
MIDNIGHT
```

Let's put a semicolon at the end of line 20:

```
10 FOR I = 1 TO 12
20 PRINT I;" ";
30 NEXT I
40 PRINT "MIDNIGHT"
```

The semicolon makes the computer print each item on the same line, like this:

```
1 2 3 4 5 6 7 8 9 10 11 12 MIDNIGHT
```

If you want the computer to press the RETURN key before MIDNIGHT, insert a PRINT line:

```
10 FOR I = 1 TO 12
20 PRINT I;" ";
30 NEXT I
35 PRINT
40 PRINT "MIDNIGHT"
```

Line 35 makes the computer press the RETURN key just before MIDNIGHT, so the computer will print MIDNIGHT on a separate line, like this:

```
1 2 3 4 5 6 7 8 9 10 11 12
MIDNIGHT
```

In line 20, the semicolon means: do not press the RETURN key after I. Line 35 means: do press the RETURN key. So line 35 undoes line 20, and makes the computer press the RETURN key before MIDNIGHT.

Let's make the computer count to MIDNIGHT three times, like this:

```
1 2 3 4 5 6 7 8 9 10 11 12
MIDNIGHT
1 2 3 4 5 6 7 8 9 10 11 12
MIDNIGHT
1 2 3 4 5 6 7 8 9 10 11 12
MIDNIGHT
```

To do that, put the entire program between the words FOR and NEXT:

```
5 FOR A = 1 TO 3
10 FOR I = 1 TO 12
20 PRINT I;" ";
30 NEXT I
35 PRINT
40 PRINT "MIDNIGHT"
50 NEXT A
```

That version contains a loop inside a loop: the loop that says "FOR I" is inside the loop that says "FOR A". The A loop is called the **outer loop**; the I loop is the **inner loop**. The inner loop's variable must differ from the outer loop's. Since we called the inner loop's variable "I", the outer loop's variable must not be called "I"; so we picked the letter A instead. Notice that each NEXT loops back to a specific FOR. Including the variable after the NEXT makes your programs easier to read; but SmartBASIC really is smart! It will keep track of which NEXT goes with each FOR automatically.

Often, programmers think of the outer loop as a bird's nest, and the inner loop as an egg inside the nest. So programmers say the inner loop is nested in the outer loop. The two loops are nested loops. Be careful if you use variables with your NEXT's or your loops won't be nested and you will get an error.

Favorite color

This program plays a guessing game:

```
10 PRINT "I'LL GIVE YOU FIVE GUESSES...."
20 FOR I = 1 TO 5
30 INPUT "WHAT'S MY FAVORITE COLOR? ";G$
40 IF G$="PINK" THEN GOTO 100
50 PRINT "NO."
60 NEXT I
70 PRINT "SORRY, YOUR FIVE GUESSES ARE UP! YOU
LOSE."
80 END
100 PRINT "CONGRATULATIONS! YOU DISCOVERED MY
FAVORITE COLOR."
110 PRINT "IT TOOK YOU ";I;" GUESSES."
```

Line 10 warns the human that only five guesses are allowed. Line 20 makes the computer count from 1 to 5; to begin, I is 1. Line 30 asks the human to guess the computer's favorite color; the guess is called G$. If the guess is PINK, the computer jumps from line 40 to line 100, prints CONGRATULATIONS, and tells how many guesses the human took. But if the guess is not PINK, the computer proceeds from line 40 to line 50, prints NO, and goes on to the next guess. If the human guesses five times withou success, the computer proceeds from line 60 to line 70 and prints SORRY...YOU LOSE.

For example, if the human's third guess is PINK, the computer prints:

```
CONGRATULATIONS! YOU DISCOVERED MY FAVORITE COLOR.
IT TOOK YOU 3 GUESSES.
```

If the human's very first guess is PINK, the computer prints:

```
CONGRATULATIONS! YOU DISCOVERED MY FAVORITE COLOR.
IT TOOK YOU 1 GUESSES.
```

It's ungrammatical, but understandable.

Lines 20-60 form a loop. Line 20 says the loop wil normally be done five times. The line after the loop, line 70, is the loop's **normal exit.** But if the human happens to input PINK, the computer jumps out of the loop early, to line 100, which is the loop's **abnormal exit.** Be careful when you use abnormal exits. It's very easy for both you and the computer to get confused. Usually its better to use a counter and IF...THEN rather than abnormally exit a FOR...NEXT. (See Chapter Four.)

Short cuts

On ADAM, when you type NEXT, you don't have to type the variable even for nested loops. So, instead of saying:

```
50 NEXT I
```

you can say:

```
50 NEXT
```

Instead of saying:

```
10 FOR I = 1 TO 5
20 PRINT "FAT"
30 PRINT "CAT"
40 PRINT
50 NEXT I
```

You can put the entire loop on a single line, like this:

```
10 FOR I = 1 TO 5: PRINT "FAT": PRINT "CAT": PRINT:
NEXT
```

Step

The FOR statement can be varied:

Statement	Meaning
FOR I=5 TO 17 STEP 3	I will be every third number from 5 to 17. So I will be 5, then 8, the 14, then 17.
FOR I=17 TO 5 STEP -3	I will be every third number from 17 down to 5. So I will be 17, then 14, then 11, then 8, then 5.

To count down, you must use the word STEP. To coun
from 17 down to 5, give this instruction:

```
FOR I = 17 TO 5 STEP -1
```

This program prints a rocket countdown:

```
10 FOR I = 10 TO 1 STEP -1
20 PRINT I
30 NEXT I
40 PRINT "BLAST OFF!"
```

The computer will print:

```
10
9
8
7
6
5
4
3
2
1
BLAST OFF!
```

This statement is tricky:

```
FOR I = 5 TO 16 STEP 5
```

It says to start I at 5, and keep adding 5 until I gets past 16. So I will be 5, then 10, then 15. I won't be 20 inside the loop, since 20 is past 16. The first value of I is 5; the last value printed inside the loop is 15.

```
10 FOR I=5 TO 16 STEP 5
20 PRINT I;" IN THE LOOP"
30 NEXT I
40 PRINT I;" AFTER THE LOOP"
```

Prints out:

```
5 IN THE LOOP
10 IN THE LOOP
15 IN THE LOOP
20 AFTER THE LOOP
```

In the statement FOR I = 5 TO 16 STEP 5, the **first value**, or **initial value** of I is 5, the **limit value** is 16, and the **step size** or **increment** is 5. The I is called the **counter** or **index** or **loop-control variable**. Although the limit value is 16, the **last value** or **terminal value** is 20.

Random Numbers

Usually, the computer is predictable: it does exactly what you say. But sometimes you want the computer to be unpredictable. In games, for instance; you want the computer to be unpredictable, to "surprise" you. Without an element of surprise, the game would be boring. And if you want the computer to act artistic, and create a new, original masterpiece that's a "work of art", you need a way to make the computer get a "flash of inspiration". And flashes of inspiration are not predictable; they are surprises.

This makes the computer print a list of unpredictable numbers from 1 to 6:

```
10 FOR I=1 TO 10
20 PRINT INT(1+6*RND(1))
30 NEXT
```

Unpredictable numbers are called random numbers. Line 20 makes the computer print a random number from 1 to 6. So the computer will print 1 or 2 or 3 or 4 or 5 or 6; you can't predict which of those numbers the computer will print; the computer's choice will be a surprise.

To make the computer print many such random numbers, say GOTO:

```
10 PRINT INT(1+5*RND(1))
20 GOTO 10
```

The computer will print many numbers, like this:

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

Each number will be 1 or 2 or 3 or 4 or 5. The order in which the computer prints the numbers is unpredictable. The program is an infinite loop; to stop it, you must abort your program. Use Control-C to break into the loop and stop the RUN.

To get sequences of random numbers that are the same each time, use a negative number inside the RND's parentheses.

```
10 INPUT N
20 LET N=RND(-N)
30 PRINT INT(1+RND(1))
40 GOTO 30
```

This will print a sreies of 0s and 1s that will be the same each time the program is run. Change the sequence by INPUTing a new value for N.

We can make random work better by using the DEF FN statement. At the same time, we'll use less paper (or less screen) by making the computer print the numbers across instead of down.

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

To do this, put a semicolon in the PRINT statment:

```
10 DEF FN D(N)=INT(1+N*RND(1))
20 PRINT FN D(5);" ";
30 GOTO 20
```

That program prints random numbers up to 5. To see random numbers up to 1000, say FN D(1000).

```
10 DEF FN D(N)=INT(1+N*RND(1))
20 PRINT FN D(1000);" ";
30 GOTO 20
```

The computer will print something like this:

485 729 537 1000 13 1 842 156 1000 972 etc.

This program plays a guessing game:

```
5 INPUT N
6 N=RND(-N)
10 DEF FN D(N)=INT(1+N*RND(1))
20 PRINT "I'M THINKING OF A NUMBER FROM 1 TO 100."
30 C=FN D(100)
40 INPUT "WHAT DO YOU THINK MY NUMBER IS? ";G
50 IF G<C THEN PRINT "YOUR GUESS IS TOO LOW.": GOTO
40
60 IF G>C THEN PRINT "YOUR GUESS IS TOO HIGH.": GOTO
40
70 PRINT "CONGRATULATIONS!  YOU FOUND MY NUMBER!"
```

Input any one, two, or three digit number when asked by the computer.

Line 20 makes the computer say:

I'M THINKING OF A NUMBER FROM 1 TO 100.

Line 30 makes the computer think of a random number from 1 to 100; the computer's number is called "C". Line 40 asks the human to guess the number; the guess is called "G". If the guess is less than the computer's number, line 50 makes the computer say YOUR GUESS IS TOO LOW and then GOTO 40, which lets the human guess again. If the guess is greater than the computer's number, line 60 makes the computer say YOUR GUESS IS TOO HIGH and then GOTO 40. When the human guesses correctly, the computer arrives at line 70, which prints:

CONGRATULATIONS! YOU FOUND MY NUMBER!

Here's a sample run:

```
RUN
I'M THINKING OF A NUMBER FROM 1 TO 100.
WHAT DO YOU THINK MY NUMBER IS? 54
YOUR GUESS IS TOO LOW.
WHAT DO YOU THINK MY NUMBER IS? 73
YOUR GUESS IS TOO HIGH.
WHAT DO YOU THINK MY NUMBER IS? 62
YOUR GUESS IS TOO LOW.
WHAT DO YOU THINK MY NUMBER IS? 68
YOUR GUESS IS TOO LOW.
WHAT DO YOU THINK MY NUMBER IS? 70
YOUR GUESS IS TOO HIGH.
WHAT DO YOU THINK MY NUMBER IS? 69
CONGRATULATIONS!  YOU FOUND MY NUMBER!
```

This program makes the computer talk about your friends:

```
5 INPUT N
6 N=RND(-N)
10 DEF FN D(N)=INT(1+N*RND(1))
20 INPUT "TYPE THE NAME OF SOMEONE YOU LOVE...? ";N$
30 R=FN D(3)
40 IF R=1 THEN PRINT N$;" WEARS LOUD, STRIPED
SOCKS.  ARE YOU SURE YOU WANT TO GET INVOLVED?":
GOTO 20
50 PRINT N$;" LOVES YOU, TOO": GOTO 20
```

Note that lines 5 and 6 area a kind of internal RESET in this program. This is called a "seed". This two-line "seed" is what tells ADAM which random sequence to use.

Type in any one, two, or three digit number when the question mark appears.
Line 20 makes the computer ask:

```
TYPE THE NAME OF SOMEONE YOU LOVE...?
```

Suppose the human says SUZY. The N$ is SUZY. Line 30 says R is a random number from 1 to 3. If R is 1, line 40 makes the computer say:

```
SUZY WEARS LOUD, STRIPED SOCKS.  ARE YOU SURE YOU
WANT TO GET INVOLVED?
```

If R is 2 or 3, the computer arrives at line 50, which prints:

```
SUZY LOVES YOU, TOO
```

In that program, the chance is only 1 out of 3 that the computer will say WEARS LOUD, STRIPED SOCKS etc. The chance is 2 out of 3 that the computer will be nicer, and say LOVES YOU, TOO. Here's a sample run:

```
RUN
? 123
TYPE THE NAME OF SOMEONE YOU LOVE...? SUZY
SUZY LOVES YOU, TOO
TYPE THE NAME OF SOMEONE YOU LOVE...? JOAN
JOAN WEARS LOUD, STRIPED SOCKS.  ARE YOU SURE YOU
WANT TO GET INVOLVED?
TYPE THE NAME OF SOMEONE YOU LOVE...? ALICE
ALICE LOVES YOU, TOO
TYPE THE NAME OF SOMEONE YOU LOVE...? FRED
FRED LOVES YOU, TOO
TYPE THE NAME OF SOMEONE YOU LOVE...? UNCLE CHARLIE
UNCLE CHARLIE WEARS LOUD, STRIPED SOCKS.  ARE YOU
SURE YOU WANT TO GET INVOLVED?
etc.
```

This program predicts what will happen to you today:

```
5 INPUT N
6 N=RND(-N)
10 DEF FN D(N)=INT(1+N*RND(1))
20 PRINT "YOU WILL HAVE A ";
30 R=FN D(5)
40 IF R=1 THEN PRINT "WONDERFUL";
50 IF R=2 THEN PRINT "BETTER-THAN-AVERAGE";
60 IF R=3 THEN PRINT "SO-SO";
70 IF R=4 THEN PRINT "WORSE-THAN-AVERAGE";
80 IF R=5 THEN PRINT "TERRIBLE";
90 PRINT " DAY TODAY"
```

The computer will say:

YOU WILL HAVE A WONDERFUL DAY TODAY

or

YOU WILL HAVE A TERRIBLE DAY TODAY

or some in-between comment. For inspiration, run that program when you get up in the morning.

Print Zones

What are zones and how do they work? The leftmost part of the screen is the first zone; to the right of that zone lies the second zone. Each print zone is 16 characters wide. This allows 2 zones per line on the screen and 5 zones per line on the printer.

A comma makes the computer jump to a new zone; here's an example:

```
10 PRINT "THIN","KING"
```

The computer will print THIN and KING on the same line; but because of the comma before KING, the computer will print KING in the second zone, like this:

```
THIN            KING
```

This program does the same thing:

```
10 PRINT "THIN",
20 PRINT "KING"
```

Line 10 makes the computer print THIN and then jump to the next zone. Line 20 makes the computer print KING. The computer will print:

```
THIN            KING
```

This program makes the computer greet you:

```
10 PRINT "HI","HI"
```

The computer will print HI two times; each time will be in a new zone, like this:

```
HI            HI
```

This program prints a list of words and their opposites:

```
10 PRINT "GOOD","BAD"
20 PRINT "BLACK","WHITE"
30 PRINT "GRANDPARENT","GRANDCHILD"
40 PRINT "HE","SHE"
```

Line 10 makes the computer print GOOD, then jump to the next zone, then print BAD. Altogether, the computer will print:

```
GOOD                BAD
BLACK               WHITE
GRANDPARENT         GRANDCHILD
HE                  SHE
```

The first zone contains a column of words; the second zone contains the opposites. Altogether, the computer's printing looks like a table. So whenever you want to make a table, use zones, by putting commas in your program.
Let's make the computer print this table:

```
NUMBER              SQUARE
3                   9
4                   16
5                   25
6                   36
7                   49
8                   64
9                   81
10                  100
```

Here's the program:

```
10 PRINT "NUMBER","SQUARE"
20 FOR I = 3 TO 10
30 PRINT I,I*I
40 NEXT I
```

Line 10 prints the word NUMBER at the top of the first column, and the word SQUARE at the top of the second. Line 20 says I goes from 3 to 10; to begin I is 3. Line 30 makes the computer print:

```
3             9
```

Line 40 makes the computer do the same thing for the next I, and for the next I, and for the next; so the computer prints the whole table.

Here's another program:

```
10 PRINT "YOU'RE NICE","...LIKE ME!"
```

The computer will print YOU'RE NICE, then jump to a new zone, then print ...LIKE ME! like this:

```
YOU'RE NICE                   ...LIKE ME!
```

LESSON 4

TACKLE THE TOUGH STUFF

Incrementing

Here's a strange program; until you stop and think about it:

```
10 A=5
20 A=3+A
30 PRINT A
```

Line 20 means: the new A is 3 plus the old A. So the new A is 3+5, which is 8. Line 30 prints:

```
8
```

Here's another one. See if you can see the logic behind it:

```
10 B=6
20 B=B+1
30 PRINT B*2
```

Line 20 says the new B is "the old B plus 1". So the new B is 6+1, which is 7. Line 30 prints:

```
14
```

In that program, line 10 says B is 6; but line 20 increases B, by adding 1 to B; so B becomes 7. Programmers say that B has been **increased** or **incremented.** In line 20, the "1" is called the **increase** or **increment.**

Decrementing

The opposite of increment is **decrement:**

```
10 J=500
20 J=J-1
30 PRINT J
```

Line 10 says J starts at 500; but line 20 says the new J is "the old J minus 1"; so the new J is 500-1, which is 499. Line 30 prints:

```
499
```

In that program, J was decreased, or **decremented.** In line 20, the "1" is called the decrease or **decrement.**

Counting

Suppose you want the computer to count, starting at 3, like this:

```
3
4
5
6
7
8
etc.
```

This program does it, by incrementing:

```
10 C=3
20 PRINT C
30 C=C+1
40 GOTO 20
```

In that program, C is called the **counter,** because it helps the computer count. Line 10 says C starts at 3. Line 20 makes the computer print C; so the computer prints:

```
3
```

Line 30 increases C, by adding 1 to it; so C becomes 4. Line 40 sends the computer back to line 20, which prints the new value of C:

```
4
```

Then the computer comes to line 30 again, which increases C again, so C becomes 5. Line 40 sends the computer back to line 20 again, which prints:

5

The program is an infinite loop: the computer will print 3,4,5,6,7,8,9,10,11,12, and so on, forever, unless you abort your program.

Here's the general procedure for making the computer count:

1. Start at some value (such as 3).

2 Use C. (for example, tell the computer to PRINT C.)

3. Increase C (by saying C=C+1).

4. GO back TO step 2.

To read the printing more easily, put a semicolon at the end of the PRINT statement:

```
10 C=3
20 PRINT C;" ";
30 C=C+1
40 GOTO 20
```

The semicolon makes the computer print horizontally:

3 4 5 6 7 8 9 etc.

This program makes the computer count, starting at 1:

```
10 C=1
20 PRINT C;", ";
30 C=C+1
40 GOTO 20
```

The computer will print 1, 2, 3, 4, etc.

This program makes the computer count, starting at 0:

```
10 C=0
20 PRINT C;", ";
30 C=C+1
40 GOTO 20
```

The computer will print 0, 1, 2, 3, 4, etc.

Let's make the computer print a table showing each number and its square, like this:

```
NUMBER                    SQUARE
0                         0
1                         1
2                         4
3                         9
4                         16
5                         25
6                         36
7                         49
8                         64
etc.
```

This program does it:

```
5 PRINT "NUMBER","SQUARE"
10 C=0
20 PRINT C,C*C
30 C=C+1
40 GOTO 20
```

Line 5 prints the headings. Lines 10-40 make the computer count (0,1,2,3,etc.); but instead of printing just C, line 20 also prints C*C.

Let's make the computer print a table that shows the squares of decimals, like this:

NUMBER	SQUARE
0	0
.1	.01
.2	.04
.3	.09
.4	.16
.5	.25
.6	.36
.7	.49
.8	.64
.9	.81
1	1
1.1	1.21
1.21	1.44
1.3	1.69
1.4	1.96
1.5	2.25

etc.

To make the computer do that, just change line 30, so that the computer increases by .1 instead of 1:

```
5 PRINT "NUMBER","SQUARE"
10 C=0
20 PRINT C,C*C
30 C=C+.1
40 GOTO 20
```

Quiz

Let's make the computer give this quiz:

```
What's the capital of Nevada?
What's the chemical symbol for iron?
What word means 'brother or sister'?
What was Beethoven's first name?
How many cups are in a quart?
```

To make the computer score this quiz, we'll have t
tell it the correct answers, which are:

```
CARSON CITY
FE
SIBLING
LUDWIG
4
```

So our program will contain this data:

```
10 DATA "WHAT'S THE CAPITAL OF NEVADA? ","CARSON
CITY"
20 DATA "WHAT'S THE CHEMICAL SYMBOL FOR IRON? ","FE"
30 DATA "WHAT WORD MEANS 'BROTHER OR SISTER'?
","SIBLING"
40 DATA "WHAT WAS BEETHOVEN'S FIRST NAME? ","LUDWIG"
50 DATA "HOW MANY CUPS ARE THERE IN A QUART? ","4"
```

Tell the computer to READ the data:

```
100 READ Q$,A$
```

That line reads a pair of data: it reads a questio
(Q$) and the correct answer (A$). The next step i
to make the computer ask the question, and wait for
the human's response:

```
110 PRINT Q$;
120 INPUT "??";H$
```

Line 120 prints question marks after the question,
and waits for the human to respond; the human's
response is called H$.

To complete the program, evaluate the human's
response. If the human's response (H$) is the
correct answer (A$), make the computer say CORRECT,
and then GOTO the next question:

```
130 IF H$=A$ THEN PRINT "CORRECT": GOTO 100
```

But if the human's response is wrong, make the computer say NO and reveal the correct answer:

```
140 PRINT "NO, THE ANSWER IS: ";A$: GOTO 100
```

Here's a sample run:

```
RUN
WHAT'S THE CAPITAL OF NEVADA??? LAS VEGAS
NO, THE ANSWER IS: CARSON CITY
WHAT'S THE CHEMICAL SYMBOL FOR IRON??? FE
CORRECT
WHAT WORD MEANS 'BROTHER OR SISTER'??? I GIVE UP
NO, THE ANSWER IS: SIBLING
WHAT WAS BEETHOVEN'S FIRST NAME??? LUDVIG
NO, THE ANSWER IS: LUDWIG
HOW MANY CUPS ARE IN A QUART??? 4
CORRECT
OUT OF DATA ERROR IN 100
```

To give a quiz about different topic, change the data in lines 10-50.

Instead of making the computer say OUT OF DATA ERROR IN 100, let's make it say:

```
I HOPE YOU ENJOYED THE QUIZ
```

To do that, write an end mark and an end routine:

```
60 DATA "DONE","DONE"
100 READ Q$,A$: IF Q$="DONE" THEN PRINT "I HOPE YOU
ENJOYED THE QUIZ": END
```

Let's also make the computer count how many questions the human answered correctly. To do that, we need a counter. As usual, we'll call it C:

```
5 C=0
10 DATA "WHAT'S THE CAPITAL OF NEVADA? ","CARSON
CITY"
20 DATA "WHAT'S THE CHEMICAL SYMBOL FOR IRON? ","FE
30 DATA "WHAT WORD MEANS 'BROTHER OR SISTER'?
","SIBLING"
40 DATA "WHAT WAS BEETHOVEN'S FIRST NAME? ","LUDWIG
50 DATA "HOW MANY CUPS ARE IN A QUART? ","4"
60 DATA "DONE","DONE"
100 READ Q$,A$: IF Q$="DONE THEN PRINT "YOU ANSWERE
";C;" OF THE QUESTIONS CORRECTLY": PRINT "I HOPE YOU
ENJOYED THE QUIZ": END
110 PRINT Q$
120 INPUT "??";H$
130 IF H$=A$ THEN PRINT "CORRECT": C=C+1: GOTO 100
140 PRINT "NO, THE ANSWER IS: ";A$: GOTO 100
```

Line 5 begins the counter at 0 (because at the beginning, the human hasn't answered any questions correctly yet). When the program ends, line 100 prints the value of the counter. Line 130 makes sure that each time the human answers a question correctly, the counter increases.

Line 100 prints a message such as:

YOU ANSWERED 2 OF THE QUESTIONS CORRECTLY

Summing

Let's make the computer imitate an adding machine, so a run looks like this:

```
RUN
NOW THE SUM IS 0
WHAT NUMBER DO YOU WANT TO ADD TO THE SUM? 5
NOW THE SUM IS 5
WHAT NUMBER DO YOU WANT TO ADD TO THE SUM? 3
NOW THE SUM IS 8
WHAT NUMBER DO YOU WANT TO ADD TO THE SUM? 6.1
NOW THE SUM IS 14.1
WHAT NUMBER DO YOU WANT TO ADD TO THE SUM -10
NOW THE SUM IS 4.1
etc.
```

Here's the program:

```
10 S=0
20 PRINT "NOW THE SUM IS ";S
30 INPUT "WHAT NUMBER DO YOU WANT TO ADD TO THE SUM?
";X
40 S=S+X
50 GOTO 20
```

To exit from these looping programs, hit CONTROL-C.

Line 10 starts the sum at 0. Line 20 prints the sum. Line 30 asks the human what number to add to the sum; the human's number is called X. Line 40 adds X to the sum, so the sum changes. Line 50 makes the computer go to line 20 which prints the new sum. Lines 20-50 form an infinite loop, which you must eventually abort. Here's the general procedure for making the computer find a sum:

1. Start S at 0

2. Use S. (For example, tell the computer to PRINT S)

3. Find out what number to add to S. (For example, tell the human to input a number X)

4. Increase S (by saying S=S+the number to be added)
5. GO back TO step 2.

Checking account

If your bank is nasty, it treats your checking account something like this: you must pay a 20 cent service charge for each good check; you must pay a $10 penalty for each check that bounces; and the money you've deposited earns no interest.

This program makes the computer imitate such a bank.

```
10 S=0
20 PRINT "YOUR CHECKING ACCOUNT CONTAINS $ ";S
30 INPUT "DEPOSIT OR WITHDRAW? ";A$
40 IF A$="DEPOSIT" THEN GOTO 100
50 IF A$="WITHDRAW" THE GOTO 200
60 PRINT "PLEASE SAY 'DEPOSIT' OR 'WITHDRAW'": GOTO
30
100 INPUT "HOW MUCH TO YOU WANT TO DEPOSIT? ";D
110 S=S+D
120 GOTO 20
200 INPUT "HOW MUCH DO YOU WANT TO WITHDRAW? ";W
210 W=W+.20
220 IF W<=S THEN PRINT "OKAY": S=S-W: GOTO 20
230 PRINT "THAT CHECK BOUNCED": S=S-10: GOTO 20
```

Line 10 starts the sum at 0. Line 20 prints the sum. Line 30 asks whether the human wants to deposit or withdraw.

If the human says DEPOSIT, the computer goes from line 40 to line 100, which asks how much to deposit. Line 110 increases the sum in the account by adding the deposit.

Line 120 sends the computer back to line 20, which tells the human the new sum (i.e., the balance), an gets ready for the next transaction.

If the human says WITHDRAW instead of deposit, the computer goes from line 50 to line 200, which asks how much to withdraw. Line 210 adds the 20 cent charge to the withdrawal amount. Line 220 compares the withdrawal amount (W) against the sum in the account (S). If W<=S, the computer says OKAY, processes the check, decreases the sum in the account (by subtracting W), and gets ready for the next transaction (by going back to line 20). If W>S instead, the computer says THAT CHECK BOUNCED, decreases the sum in the account by $10 (the penalty fee), and goes back to line 20, for the next transaction.

Again, to exit this program, press CONTROL-C and RETURN.

Series

Let's make the computer add together all the numbers from 7 to 100. In other words, let's make the computer find the sum of this series: 7+8+9+...+100. Here's how:

```
Set the sum at zero:                10 S=0
Make I go from 7 to 100:            20 FOR I=7 TO 100
Add each I to sum:          30 S=S+I
                            40 NEXT I
Print final sum:            50 PRINT S
```

Let's make the computer add together the squares of all the numbers from 7 to 100. In other words, let's make the computer find the sum of this series: (7 squared) + (8 squared) + (9 squared) + ...+ (100 squared). Here's how:

```
10 S=0
20 FOR I = 7 TO 100
30 S=S+I*I
40 NEXT I
50 PRINT S
```

It's the same as the previous program, except that line 30 says to add I*I instead of I. Line 50 prints the final sum, which is 338259.

Data

This program adds together the numbers in the data:

```
10 S=0
20 DATA 5, 6.1, etc.
30 DATA etc.
40 DATA etc.
50 DATA 0
60 READ X:  IF X=0 THEN PRINT S: END
70 S=S+X
80 GOTO 60
```

Line 10 starts the sum at zero. Lines 20-40 contain the numbers to be added. The zero in line 50 is an end mark. Line 60 reads an X from the data. If the X it reads is zero, then end of the

data has been reached, so we want the computer to print the sum (S) and stop; but if the X it reads is not zero, the computer proceeds from line 60 to line 70, adds X to the sum, and goes from line 80 to line 60, which reads another X.

Single Subscripts

Instead of being a single number, X can be a whole list of numbers, like this:

```
X=      57.2
        -8.3
        476
        2.008
        -19
        372.402
        0
        215.6
```

Here's how to make X be that list of numbers. . . Begin your program by saying:

```
10 DIM X(8)
```

That says X will be a list of 8 numbers. DIM means **dimension**; the line says the dimension of X is 8. Next, tell the computer what numbers are in X. Type these lines:

```
20 X(1)=57.2
30 X(2)=-8.3
40 X(3)=476
50 X(4)=2.008
60 X(5)=-19
70 X(6)=372.402
80 X(7)=0
90 X(8)=215.6
```

Line 20 says: X's first number is 57.2. Line 30 says X's second value is -8.3. The remaining lines define the other numbers in X.

If you'd like the computer to print all those numbers, type this:

```
100 PRINT X(1),X(2),X(3)...etc.
```

That means: print all the numbers in X. The computer will print:

```
57.2        -8.3
476         2.008
-19         372.402
0           215.6
```

To achieve the same results, in a single column configuration, you can say:

```
100 FOR I = 1 TO 8: PRINT X(I): NEXT
```

In that program, line 20 talks about X(1): Instead of saying X(1), math books say:

X_1

The "1" is called a **subscript.** Similarly, in line 30, which says "X(2)=-8.3", the number 2 is a subscript. Some programmers pronounce line 30 like this: "X subscripted by 2 is -8.3". To be briefer, most programmers say this instead: "X sub 2 is -8.3". Some programmers simply say: "X 2 is -8.3".

In that program, X is called an **array.** Definition: an array is a variable that has subscripts. An array may be multidimensional in that it may have 1 to 88 dimensions.

Data

That program said X(1) is 57.2, and X(2) is -8.3, and so on. This program does the same thing, more briefly:

```
10 DIM X(8)
20 DATA 57.2, -8.3, 476, 2.008, -19, 372.402, 0,
215.6
30 FOR I = 1 TO 8: READ X(I): NEXT
40 FOR I = 1 TO 8: PRINT X(I): NEXT
```

Let's make the program fancier, by adding extra lines. This line makes the computer add the eight numbers together and print the sum:

```
50 PRINT X(1)+X(2)+X(3)+X(4)+X(5)+X(6)+X(7)+X(8)
```

That line makes the computer print:

```
1095.91
```

Let's make the computer print the numbers in reverse order (starting with the eighth number, and ending with the first). In other words, let's make the computer print X(8), the print X(7), then print X(6), etc. To do that, you could say:

```
60 PRINT X(8)
70 PRINT X(7)
80 PRINT X(6)
etc.
```

But this way is shorter:

```
60 FOR I = 8 TO 1 STEP -1
70 PRINT X(I)
80 NEXT I
```

Lines 60-80 print:

```
215.6
0
372.402
-19
2.008
476
-8.3
57.2
```

So, if you run this program, you'll get the initial listing, the total, and the initial listing again, but in reverse order.

Analysis

Suppose you want to analyze 50 numbers. Begin your program by saying:

```
10 DIM X(50)
```

Then type 50 numbers, as data, like this:

```
20 DATA etc.
30 DATA etc.
40 DATA etc.
```

Tell the computer to READ the data:

```
100 FOR I = 1 TO 50: READ X(I): NEXT
```

After line 100, you have many choices, depending on which problem you want to solve. . .

Problem: print all the values of X. Solution:

```
110 FOR I = 1 TO 50
120 PRINT X(I)
130 NEXT I
```

Problem: print all the values of X, in reverse order. Solution:

```
110 FOR I = 50 TO 1 STEP -1
120 PRINT X(I)
130 NEXT I
```

Problem: print the sum of all the values of X. In other words, print X(1)+X(2)+X(3)+...+X(50).
Solution: start the sum at 0:

```
110 S=0
```

and then increase the sum, by adding each X(I) to it:

```
120 FOR I = 1 TO 50
130 S=S+X(I)
140 NEXT I
```

Finally, print the sum:

```
150 PRINT "THE SUM OF ALL THE NUMBERS IS ";S
```

Problem: find the average value of X. In other words, find the average of the 50 numbers. Solution: begin by finding the sum:

```
110 S=0
120 FOR I = 1 TO 50
130 S=S+X(I)
140 NEXT I
```

and then divide the sum by 50:

```
150 PRINT "THE AVERAGE IS ";S/50
```

Problem: find out whether any of the values of X is 79.4. In other words, find out whether 79.4 is a number in the list. Solution: if X(I) is 79.4, print YES:

```
110 FOR I = 1 TO 50
120 IF X(I)=79.4 THEN PRINT "YES, 79.4 IS IN THE
LIST":END
130 NEXT I
```

otherwise, print NO:

```
140 PRINT "NO, 79.4 IS NOT IN THE LIST"
```

Problem: in the list of numbers, count how often th number 79.4 appears. Solution: start the counter at zero:

```
110 C=0
```

and increase the counter each time you see the number is 79.4:

```
120 FOR I = 1 TO 50
130 IF X(I)=79.4 THEN C=C+1
140 NEXT I
```

Finally, print the counter:

```
150 PRINT "THE NUMBER 79.4 APPEARS ";C;" TIMES"
```

Problem: print all the values of X that are negative. In other words, print all the numbers that have minus signs. Solution: begin by announcing your purpose:

```
110 PRINT "HERE ARE THE VALUES THAT ARE NEGATIVE:"
```

and then print the values that are negative; in other words, print each X(I) that's less than 0:

```
120 FOR I = 1 TO 50
130 IF X(I)<0 THEN PRINT X(I)
140 NEXT I
```

Problem: print all the values of X that are "above average". Solution: find the average, and make A the average:

```
110 S=0
120 FOR I = 1 TO 50
130 S=S+X(I)
140 NEXT I
150 A=S/50
```

then announce your purpose:

```
160 PRINT "THE FOLLOWING VALUES ARE ABOVE AVERAGE:"
```

Finally, print the values that are above average; in other words, print each X(I) that's greater than A:

```
170 FOR I = 1 TO 50
180 IF X(I)>A THEN PRINT X(I)
190 NEXT I
```

Problem: find the biggest value of X. In other words, find which of the 50 numbers is the biggest. Solution: let B stand for the biggest number. Begin by tentatively setting B equal to the first number:

```
110 B=X(1)
```

but if another number is bigger than that B, change B:

```
120 FOR I = 2 TO 50
130 IF X(I)>B THEN B=X(I)
140 NEXT I
```

Afterwards, print B:

```
150 PRINT "THE BIGGEST NUMBER IN THE LIST IS ";B
```

Problem: find the smallest value of X. In other words, find which of the 50 numbers is the smallest. Solution: let S stand for the smallest number. Begin by tentatively setting S equal to the first number:

```
110 S=X(1)
```

but if another number is smaller than S, change S:

```
120 FOR I = 2 TO 50
130 IF X(I)<S THEN S=X(I)
140 NEXT I
```

Afterwards, print S:

```
150 PRINT "THE SMALLEST NUMBER IN THE LIST IS ";S
```

Problem: find out whether the values of X are in strictly increasing order. In other words, find out whether the following statement is true: X(1) is a smaller number than X(2), which is a smaller number than X(3), which is a smaller number than X(4), etc. Solution: if X(I) is not smaller than X(I+1), print NO:

```
110 FOR I = 1 TO 49
120 IF X(I)>=X(I+1) THEN PRINT "NO, THE LIST IS NOT
IN STRICTLY INCREASING ORDER": END
130 NEXT I
```

otherwise, print YES:

```
140 PRINT "YES, THE LIST IS IN STRICTLY INCREASING
ORDER"
```

Multiple Arrays

Suppose your program involves two lists of numbers. Suppose the first list is called A and the second list is called B. Suppose A's list contains 18 numbers and B's list contains 57 numbers. To say all that, begin your program with this statement:

```
10 DIM A(18), B(57)
```

Double Subscripts

You can make X be a table of numbers, like this:

```
X=    57      8.4
      -6      1000
      0       7.77
```

Here's how to make X be that table:
Begin by saying:

```
10 DIM X(3,2)
```

That says X will be a table, having 3 rows and 2 columns.

Then tell the computer what numbers are in X. Type these lines:

```
20 X(1,1)=57
30 X(1,2)=8.4
40 X(2,1)=-6
50 X(2,2)=1000
60 X(3,1)=0
70 X(3,2)=7.77
```

Line 20 says: the number in X's first row and first column is 57. Line 30 says the number in X's first row and second column is 8.4. The remaining lines define the other numbers in X.

If you'd like the computer to print all those numbers, type this:

```
80 FOR I = 1 TO 3: FOR J = 1 TO 2: PRINT X(I,J),:
NEXT: PRINT: NEXT
```

In that program, X is called a **table** or a **doubly subscripted array.**

Multiplication table

This program prints a multiplication table:

```
10 DIM X(10,2)
20 FOR I = 1 TO 10
30 FOR J = 1 TO 2
40 X(I,J)=I*J
50 NEXT J
60 NEXT I
70 FOR I = 1 TO 10: FOR J = 1 TO 2: PRINT
X(I,J),:NEXT: PRINT: NEXT
```

Instead of multiplication, you can have addition, subtraction, or division: just change line 40.

Most programmers follow this tradition: the row's number is called I, and the column's number is called J. Line 40 obeys that tradition. Notice I comes before J in the alphabet; I comes before J in X(I,J); and FOR I comes before FOR J in lines 20-30. If you follow the I-before-J tradition, you'll make fewer errors.

Sums

Suppose you want to analyze this table:

```
32.7        19.4
-8          402
5106        -.2
36.0        .04
777         666
1.99        2.99
50          40
12          21
0           1000
```

The table has 9 rows and 2 columns; so begin your program by saying:

```
10 DIM X(9,2)
```

Each row of the table becomes a row of the data:

```
11 DATA 32.7, 19.4
12 DATA -8, 402
13 DATA 5106, -.2
14 DATA 36.0, .04
15 DATA 777, 666
16 DATA 1.99, 2.99
17 DATA 50, 40
18 DATA 12, 21
19 DATA 0, 1000
```

Make the computer read the data:

```
20 FOR I = 1 TO 9: FOR J = 1 TO 2: READ X(I,J):
NEXT: NEXT
```

To make the computer print the table, say this:

```
30 FOR I = 1 TO 9: FOR J = 1 TO 2: PRINT X(I,J),:
NEXT: PRINT: NEXT
```

Problem: find the sum of all the numbers in the table. Solution: start the sum at 0:

```
100 S=0
```

and then increase the sum, by adding each X(I,J) to it:

```
100 FOR I = 1 TO 9
120 FOR J = 1 TO 2
130 S=S+X(I,J)
140 NEXT J
150 NEXT I
```

Finally, print the sum:

```
160 PRINT "THE SUM OF ALL THE NUMBERS IS ";S
```

The computer will print:

```
THE SUM OF ALL THE NUMBERS IS 8158.82
```

Problem: find the sum of each row. In other words, make the computer print the sum of the numbers in the first row; and then print the sum of the numbers in the second row; and then print the sum of the numbers in the third row; etc. Solution: the general idea is:

```
100 FOR I = 1 TO 9
110 print the sum of row I
120 NEXT I
```

Here are the details:

```
100 FOR I = 1 TO 9
110 S=0
111 FOR J = 1 TO 2
112 S=S+X(I,J)
113 NEXT J
114 PRINT "THE SUM OF ROW ";I;" IS ";S
115 NEXT I
```

The computer will print:

```
THE SUM OF ROW 1 IS 52.1
THE SUM OF ROW 2 IS 394
THE SUM OF ROW 3 IS 5105.8
etc.
```

Problem: find the sum of each column. In other words, make the computer print the sum of the numbers in the first column; and then print the sum of the numbers in the second column; and then print the sum of the numbers in the third column; etc. Solution: the general idea is:

```
100 FOR J = 1 TO 2
110 print the sum of column J
120 NEXT J
```

Here are the details:

```
100 FOR J = 1 TO 2
110 S=0
111 FOR I = 1 TO 9
112 S=S+X(I,J)
113 NEXT I
114 PRINT "THE SUM FOR COLUMN ";J;" IS ";S
115 NEXT J
```

The computer will print:

```
THE SUM OF COLUMN 1 IS 6007.59
THE SUM OF COLUMN 2 IS 2151.23
```

In all the other examples we looked at, FOR I comes before FOR J, but in this unusual example, FOR I comes after FOR J.

String Arrays

You've seen that X can be a list of numbers, or a table of numbers. Similarly, on ADAM, X$ can be a list of strings, or a table of strings. For example, you can make X$ be this table:

```
          DOG          CAT
X$=       WOOF         MEOW
          HOTDOG       CATSUP
```

by saying this:

```
10 DIM X$(3,2)
20 X$(1,1)="DOG"
30 X$(1,2)="CAT"
40 X$(2,1)="WOOF"
etc.
```

or by saying this:

```
10 DIM X$(3,3)
20 DATA DOG,CAT,WOOF,MEOW,HOTDOG,CATSUP,
30 FOR I = 1 TO 3: FOR J = 1 TO 2: READ X$(I,J):
NEXT: NEXT
```

LOW RESOLUTION GRAPHICS

UNDERSTANDING CONCEPTS

The only concept you really need to understand in this section is how graph paper works. "It doesn't work, it just sits there". . . I know, I know. But for accuracy, it might be a good idea to take out a piece of graph paper, since the screen doesn't divide itself into an actual grid pattern that you can see. However, if you're into the more ethereal and less concrete, then picture, in your mind, a piece of graph paper. On this graph paper, there are 40 columns and 40 rows, with space for four lines of text at the bottom. Think of your screen as if it was this piece of graph paper. Type GR. You will notice that the screen has gone black. Don't worry, you didn't kill it. You'll also see that the text cursor is now at the bottom of the screen, and your text has been cleared.

Drawing with ADAM

Now, since man cannot live by black-and-white alone you have 16 colors to choose from, merely by typing COLOR= at this time. The chart below will show you which color options you have, and the codes you must enter in order to use them:

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

NOTE: The listed colors will appear somewhat different, depending upon what sort of television set your ADAM is hooked up to. Also--if you want color graphics, ADAM must be hooked up to a color television. You won't see color on a black and white set!

To get ADAM to draw for you, in straight lines, you have to input instructions on which way to draw the lines, vertically or horizontally.

Horizontal Lines

The command for a horizontal line is HLIN. You must type this in, along with a location on the screen grid (remember the graph paper?). Here's an example:

```
HLIN 5,20 AT 35
```

5,20 indicates the column in which the line is to begin and end. This one would, therefore, begin at column 5 and end at column 20. AT 35 designates the row in which the line will be drawn. The 5 and the 20 are known as first end point and second end point, respectively. The first end point may be greater than, equal to, or less than the second end point. Remember that these "points" are locations and not values.

Try running this program:

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

Line 20 sets the color to cyan (a shade of blue). Line 30 tells ADAM to draw a line in row 20 from column 10 to column 35. Another thing to remember is that the first column and the first row are not numbered 1, but 0. Don't worry, you still have 40 columns, but they're numbered 0 to 39, and you still have 40 rows, too, which are also numbered 0 to 39.

Right now you're probably mumbling to yourself abo people who tell you that no math is required to program ADAM. Take heart--this isn't <u>really</u> math, so stop playing with your mental blocks and put th away. What you really have in front of you is a space-age version of an Etch-A-Sketch. Remember those? Red plastic with two white dials--one for vertical and one for horizontal? And you could erase the screen by shaking it up and down? Well, DON'T shake ADAM to erase the screen! This gets expensive. The analogy may be a bit simplified, but for our purposes, the idea of plotting coordinates may frighten you less if you think of your computer graphics as an electronic Etch-A-Sketch.

Vertical Lines

So. We have horizontal lines. On to vertical. It's almost the same set-up as HLIN, except the command is VLIN. The main difference between VLIN and HLIN is this:

HLIN plots column coordinates first and uses a row coordinate for an end point.

VLIN plots row coordinates first and uses a column coordinate for an end point.

Oh, it is not either confusing. Try this program for VLIN:

```
10 GR
20 COLOR=1
30 VLIN 10,21 AT 20
```

Line 20 sets the color to magenta. Line 30 tells your computer to draw a vertical magenta line from row 10 to row 21 in (AT) column 20.

Are you beginning to see the creative possibilities in this? Are you parents thinking, "Hmmmm, this could be a great way to keep the kids busy!"? Are you kids thinking, "Hmmmmm, this could be a great way to keep my parents busy!"?

Block Coloring

If you want to move from line drawing into filling in a whole square, or "block" on your "graph-paper" grid, you'll need the PLOT statement.

Try this one:

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

What happened on your screen? Did you get a block of yellow (13) at column 20, row 2? You can create a pattern on your screen by writing up a long program that changes the colors many times in many different grid locations. It's easy! It's fun! It's computer needlepoint!

When plotting coordinates, keep in mind that coordinate position 0,0 (column 0, row 0) is at the top left corner. 39,0 (column 39, row 0) is at the top right corner. 0,39 (column 0, row 39) is at the bottom left corner. 39,39 (column 39, row 39) is at the bottom right corner of your screen.

Here's a wild program. Run this one:

```
10 GR
20 COLOR=INT(RND(1)*16)
30 X=INT(RND(1)*40)
40 Y=INT(RND(1)*40)
50 PLOT X,Y
60 GOTO 20
```

Didn't know ADAM could dance, did you?

NOTE: Variables X and Y signify columns and rows respectively.

SCRN Function

Our next function is called SCRN. This reads the color displayed at any designated position on your

graphics screen. So, if you had done some compute coloring and needed to know what color was in the block at column 5 row 9, type:

```
SCRN (5,9)
```

and your computer will tell you what color you selected for that coordinate block.

These functions are not intended for use with high-resolution graphics. For information on the use of high-resolution graphics, please consult the reference section at the back of this manual.

When you're finished experimenting with graphics, you can return to a full text display screen simpl by typing:

```
TEXT
```

And that's all there is to low-resolution graphics. Wasn't as hard as you thought it would be, was it?

For an interesting example that shows how you can use the game controllers to draw pictures, see the PDL function in the Reference Section.

Congratulations, ADAM programmer!

School's out, and you've just graduated! In successfully completing the initial learning proces on your ADAM Family Computer, you are now an official SmartBASIC programmer. Should you care to press on and learn a little more, turn to the Reference Section which follows. This section discusses more advanced commands which you may find will be handy to understand for future use.

ADAM™ SmartBASIC™

REFERENCE SECTION

TABLE OF CONTENTS

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.

ADAM™ SmartBASIC™

ADVANCED REFERENCE SECTION

TABLE OF CONTENTS

APPEND

APPEND allows you to append more data to an existing sequential file. This is used only for sequential files. It is also important to bear in mind that a sequential file should be closed after you finish writing. Otherwise ADAM will use all the remaining space on the data pack for your file and will not be able to APPEND. It's syntax is:

APPEND <filename>,[D#]

where D#, or Device Number.

APPEND is similar to OPEN in that both commands open your file; but it differs from OPEN in three important respects:

1. To use APPEND, the file must already exist. You cannot use APPEND to create a file.
2. APPEND will place the pointer at the end of the file, where data will be added. OPEN places the pointer at the beginning of the file.
3. There is no need to do a WRITE after the APPEND.

Test Program:

```
10 D$=CHR$(4)
20 PRINT D$;"APPEND FACE,D1"
30 PRINT "ADDING TO FACE"
40 PRINT D$;"CLOSE FACE"
]MON C
```

Sample Run:

```
APPEND FACE
CLOSE FACE
```

Function

ASC

For every character you use there is, programmed into your computer, a corresponding ASCII decimal number value for it. ASC is the function that converts upper and lower case characters and symbols to their ASCII values.

Test Program:

```
10 PRINT "THE ASCII CODE FOR LETTER H IS
";
20 PRINT ASC("H")
30 IF ASC ("H")=72 THEN 60
40 PRINT "ASC BLEW IT"
50 GOTO 70
60 PRINT "ASC IS CORRECT"
70 END
```

Sample Run:

```
THE ASCII CODE FOR LETTER H IS 72
ASC IS CORRECT
```

ATN, or arctangent, is a trigonometric function. It is defined as the angle in a right triangle required for a particular ratio of the length of the side opposite it to the length of the side adjacent to it. ATN is the complement of TAN, and is expressed not in degrees, but in Radians. Its syntax is:

ATN(exprnm)

where (exprnm)=numerical expression.

NOTE: pi=4*ATN(1)

Test Program:

```
10 PRINT "ENTER A TANGENT VALUE:  ";
20 INPUT N
30 A=ATN(N)
40 PRINT "THE ARCTANGENT OF ";N;" IS
";A;" RADIANS"
50 END
```

Sample Run:

```
ENTER A TANGENT VALUE:  1
THE ARCTANGENT OF 1 IS .785398164
RADIANS
```

BLOAD

The command BLOAD retrieves images stored in binary files on your digital data pack and loads them into memory.
The syntax for BLOAD is:

BLOAD <filename>,A#,D#

A# is the memory location into which the binary file will be LOADed. Use of a memory address is optional for BLOAD, as is drive designation (D#). If these values are not input then ADAM assumes a preset default mode and adjusts for these values itself. BLOAD is useful for LOADing shape tables, among other things.

NOTE: To use BLOAD, you first need to do a BSAVE.

Test Program:

```
BLOAD shape, A51456
```

Sample Run:

Loads a previously BSAVE'd binary file named "shape" into memory starting at memory location 51456.

BRUN

BRUN is used in binary files and is much the same as BLOAD, except that after you load the file, BRUN will execute a machine language "jump" to the starting address automatically. It's syntax is:

BRUN <filename>,A#,D#

The A and D values are optional. If you don't specify an address or drive, then ADAM will go into the preset default for that information, will go directly to the address under which your image was stored, and will assume Drive 1.

Test Program:

NONE

Sample Run:

NONE

BSAVE

BSAVE is used to save binary information on your digital data pack. The syntax of a BSAVE command would be:

BSAVE <filename>, A#,L#,D#

A = specifies the starting address of the memory portion to be stored on the digital drive. Its use is NOT optional. You may input your memory address as either a decimal or a hexadecimal. If hexadecimal is used, the values must be preceded by a dollar sign ($). The range acceptable for decimal values is 0 to 65535. Be careful about using space reserved for SmartBASIC or the operating system. (See the memory map in Appendix C.)

L = the length of the image or information to be saved. Its use is NOT optional. Length specifies the number of bytes to be stored. Again, you may use decimals or hexadecimals to specify this.

D = the drive used. Its use in the program is optional.

Test Program:

```
BSAVE shape, A51456,L14
```

Sample Run:

This will save the shape table created in the SHAPE TABLE example to a binary file named "shape."

WARNING: FOR EXPERTS ONLY! Statement

CALL

CALL lets you execute a machine-language subroutine at a specified memory location, the decimal value of which may be 0 through 65535. Its syntax is:

CALL <location of machine-language sub>

Test Program:

NONE

Sample Run:

NONE

COS

COS is a trigonometric function which will compute the cosine of an angle, expressed in radians rather than degrees. Cosine is defined for a right triangle as the ratio of the length of the side adjacent to the angle to the length of the hypotenuse. Its syntax is:

```
COS(exprnm)
```

where exprnm is a numeric expression.

Test Program:

```
10 PRINT "ENTER ANGLE (IN RADIANS):  ";
20 INPUT T
30 Y=COS(T)
40 PRINT "'COS' OF ";T;" IS ";Y
50 END
```

Sample Run:

```
ENTER ANGLE (IN RADIANS):  ?1
'COS' OF 1 IS .540302307
```

DRAW

DRAW is used in high resolution graphics to draw a pre-defined shape at a specific location. DRAW assumes that a shape table has already been loaded into memory. You must follow the statement DRAW with the shape table index number of the shape you wish to have ADAM take from your shape table and draw on the screen. The screen location at which you want your shape drawn follows the word AT and is a two-expression term, separated by a comma. If the screen location is omitted, the shape will be drawn at the last point plotted; or at 0,0 if this is the first plot.

Test Program:

```
5 HIMEM:51455
10 DATA 01,00,04,00
20 DATA 54,63,36,36
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: HCOLOR=3
120 SCALE=10
130 FOR i=1 TO 64
140 ROT=i
150 DRAW 1 AT 125,85
160 XDRAW 1 AT 125,85
170 NEXT
```

Sample Run:

A white square with a straight line from the midpoint downward.

EXP

EXP raises e to the indicated power, exprnm.
EXP's syntax is:

```
EXP(exprnm)
```

where exprnm is a numeric expression.

Test Program :

```
10 PRINT "ENTER 'EXP' VALUE: ";
20 INPUT N
30 E=EXP(N)
40 PRINT "'EXP' OF ";N;" IS ";E
50 END
```

Sample Run:

```
ENTER 'EXP' VALUE: ?1
'EXP' VALUE OF 1 IS 2.71828183
```

NOTE: You can also raise numbers to various powers by using the CARET (^) key. For example, 10^2 would equal 100.

FRE(expr) allows you to keep a running tally on exactly how many bytes of allocated space you have available to you in memory.

Test Program:

```
PRINT FRE(0)
```

Sample Run:

```
25820
```

Test Program:

```
10 B=FRE(1)
20 A$="HELLO"
30 C=FRE(1)
40 PRINT B-C
```

Sample Run:

```
13
```

It takes 13 bytes to store HELLO.

HIMEM:

HIMEM: or highest program memory location, is automatically preset, but you may change the settings if you wish to. HIMEM: is used to protect the area of memory above a designated location, since HIMEM: will create a boundary to allow you to safely input your data without writing over important data above your boundary. But, since HIMEM: is pre-set to allow you the maximum free memory space, we DO NOT recommend that you reset it unless you are working with assembly language. It is imperative that you investigate reserved memory before going ahead and changing the HIMEM: boundary.The HIMEM: syntax is:

HIMEM:<memory location>

where memory location has a range of 0-65535 as well as -32767 down to -1. Positive or negative values, if equivalent, may be used interchangeably (-32767 equivalent to 65535).

Test Program:

NONE

Sample Run:

NONE

INIT

INIT, or initialize, is a command that should only be used in the immediate mode. Use INIT when you wish to wipe out your old directory in your digital data pack. INIT will reinitialize the directory (delete all file name entries), so be very careful when using it! INIT will not initialize a SmartBASIC tape, so don't worry about that. INIT syntax is:

```
INIT <volumename>,D#
```

The D value is optional, and indicates a particular digital data drive.

> **NOTE:** You don't need to initialize your blank digital data packs before you use them. Also, it is not possible to turn an ordinary cassette into a blank digital data pack using this command.
>
> Be careful not to 'INIT' over a Super Game!

Test Program:

```
NEW
INIT HELLO,D1
---wait while ADAM initializes
]CATALOG
```

Sample Run:

```
VOLUME: HELLO

253 Blocks Free
```

This says that you're operating in Drive 1, that your volume name is HELLO with 253 blocks free.

LOG computes the natural logarithm of any number whose value is larger than 0.

Test Program:

```
10 PRINT "ENTER POSITIVE NUMBER  ";
20 INPUT Q
30 L=LOG(Q)
40 PRINT "'LOG' OF ";Q;" IS ";L
50 END
```

Sample Run:

```
ENTER POSITIVE NUMBER  ?10
'LOG' OF 10 IS 2.30258509
```

WARNING: FOR EXPERTS ONLY! Statement

LOMEM:

The opposite of HIMEM:, LOMEM: sets the lowest memory location available. It's syntax is:

LOMEM:<memory location>

where <memory location> is an arithmetic expression between -32767 and +65535. Note that either positive or negative addresses can be used if equivalent.

Again, it is inadvisable to change the value of LOMEM: unless you are a very experienced programmer. We do not recommend it unless you are involved with assembly language.

Test Program:

NONE

Sample Run:

NONE

WARNING: FOR EXPERTS ONLY! Statement

PEEK

PEEK allows you to look at whatever data is in the memory location you specify. No writing, just looking! You are able to PEEK into RAM (read/write memory). The syntax of PEEK is:

PEEK <memory location>

where <memory location> is the address of the memory location you wish to read. Range for PEEK is 0 - 65535; but PEEK may also be used with POKE to read what has been POKED in at whatever address is within POKE range, as well.

Test Program:

```
5 HIMEM: 51456
10 FOR X=51457 TO 51467
20 READ Y
30 POKE X,Y
40 NEXT X
50 FOR X=51457 TO 51467
60 Y=PEEK(X)
70 PRINT CHR$(Y);
80 NEXT X
90 DATA 80,69,69,75,45,65,45,66,79,79,33
100 END
```

Sample Run:

PEEK-A-BOO!

WARNING: FOR EXPERTS ONLY! Statement

POKE

POKE alters the contents of a memory location, in that it will store a designated value directly into a specific memory location. Syntax for POKE is:

POKE <memory location>,<value 0-255>

where value is the decimal number representing the eight-bit quantity of data to be stored in the memory location specified. Memory location range is -32767 through 65535 inclusive. You can POKE into RAM (read/write memory) only.

NOTE: The SmartBASIC POKE will not function when using values above 54160 or below 0.

Test Program:

```
5 HIMEM:51455
10 DATA 01,00,04,00
20 DATA 54,63,36,36
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: HCOLOR=3
120 SCALE=10
130 FOR i=1 TO 64
140 ROT=i
150 DRAW 1 AT 125,85
160 XDRAW 1 AT 125,85
170 NEXT
```

Sample Run:

This will give you the "default shape", which is a square with a straight line running from the mid point, downward. For a picture of it, see the Shape Table section of the Compendium at the back of this manual.

WARNING: FOR EXPERTS ONLY!

Statement

POP

The POP statement "pops" or discards the line number of the most recent GOSUB. So when a RETURN statement appears, it will return to the statement following the secondmost GOSUB.

Test Program:

```
10 PRINT "ONE ";
20 GOSUB 100
30 PRINT "FOUR"
90 END
100 PRINT "TWO ";
110 GOSUB 200
120 PRINT "ONE TWENTY"
130 RETURN
140 STOP
200 PRINT "THREE ";
210 POP
220 RETURN
```

Sample Run:

```
ONE TWO THREE FOUR

At line 210 you "popped" past line 110
and returned after line 20 to print line
30.
```

POSITION moves the position-in-the-file pointer forward only. It can be used only in sequential files, and only to skip the number of records you select. You cannot write to your file using POSITION. Its syntax is:

```
POSITION <filename>,R#
```

where R#=record number (relative)

Test Program: (Assumes you have a sequential file named UP with at least 4 records).

```
10 D$=CHR$(4)
20 PRINT D$; "OPEN UP"
30 PRINT D$; "POSITION UP,R3"
40 PRINT D$; "READ UP"
50 INPUT A$: PRINT A$
60 PRINT D$; "CLOSE UP"
]MON C
```

Sample Run:

```
OPEN UP
POSITION UP,R3
READ UP
(prints 'record 4'as 4th record
 in file 'UP')
CLOSE UP
```

RANDOM/Fixed Length Files

Random access files will allow you to open a file at a particular record number, READ it and/or WRITE a new record into it. R#=record number and L#=record length. Please keep in mind that the file must already exist, and the READ or WRITE must be within the existing file. And always, **always** remember to CLOSE your file when you're through.

For more information on random files, refer to the Compendium at the back of this book. (Appendix C).

Test Program:

```
10 D$=CHR$(4)
20 PRINT D$;"OPEN DOOR,L20"
30 PRINT D$;"READ DOOR,R5"
40 INPUT A$:PRINT A$
50 PRINT D$;"CLOSE DOOR"
        or
30 FOR I=1 TO 10
40 PRINT D$;"WRITE DOOR,R";I
50 PRINT "RECORD NUMBER";I
60 NEXT
70 PRINT D$;"CLOSE DOOR"

]MON C
NOTE:R#=record number
     L#=record length
```

Sample Run:

```
OPEN DOOR,L20
READ DOOR,R5
CLOSE DOOR

        or

OPEN DOOR,L20
WRITE DOOR,R1
RECORD NUMBER 1
: (write ten records)
CLOSE DOOR
```

The files you access in this manner do not have to be sequential.

The RECOVER statement is used to access a back up file. Its syntax is:

RECOVER <filename>,D#

where D# means "device number" and is optional.

Test Program:

```
]CATALOG
Volume:  BOZO
A   1 FOO
a   1 FOO
253 BLOCKS FREE
]DELETE FOO
]CATALOG
Volume:  BOZO
a   1 FOO
254 BLOCKS FREE
]RECOVER FOO
]CATALOG
Volume:  BOZO
A   1 FOO
254 BLOCKS FREE
]LOAD FOO
```

Sample Run:

```
You will now have access to your back up
file of FOO.  The most recent version of
"FOO" has been deleted.  To keep both
versions, first RENAME "FOO".
```

ROT=

The ROT=, or ROTation statement sets the spatial orientation of a shape in high resolution graphics before your shape is drawn. The number following ROT= dictates the rotation of the shape in 5.625 degrees/unit. Therefore, ROT=0 orients the shape exactly as defined in the shape table; whereas ROT=16 turns the shape 90 degrees in a clockwise direction. ROT=32 turns it 180 degrees. ROT=64 turns it one complete revolution so that you're back where you started from. The number following the = may be a variable.

NOTE: Because the ratio of width to height of most TVs and monitors is not 1, the proportions of a shape may change as the shape is rotated.

Test Program:

```
10 HGR2
20 HCOLOR=6
30 SCALE=10
40 ROT=0
50 DRAW 1 AT 50,100
60 ROT=8
70 DRAW 1 AT 120,100
```

Sample Run:

You will end up with two blue shapes on your screen—one oriented as originally defined, and one rotated 45 degrees.

SCALE=

SCALE= is used in high resolution graphics to set the relative size at which a shape will be drawn. This scaling factor may be as low as 1 or as high as 255; 1 being an exact size reproduction from the one specified; 255 being the shape reproduced at 255 times its originally defined size. The number following = may be a variable.

Test Program:

```
10 HGR
20 HCOLOR=5
30 ROT=0
40 SCALE=1
50 DRAW 1 AT 50,50
60 SCALE=3
70 DRAW 1 AT 100,150
80 SCALE=2
90 DRAW 1 AT 200,100
```

Sample Run:

Color is set to medium red. You will draw one shape at original size and orientation at column 50, row 50. The same shape will be redrawn at column 100, row 150 at 3 times the original size. The same shape will be redrawn a third time at column 200, row 100 at twice its original size.

SIN

SIN, a trigonometric function, is used to compute the sine of an angle (expressed in radians). Sine is the ratio of the length of the side opposite the angle under examination to the length of the hypotenuse in a right triangle. Syntax of SIN is:

```
SIN(x)
```

where "x" is an arithmetic expression.

Test Program:

```
10 PRINT "ENTER ANGLE (IN RADIANS):  ";
20 INPUT M
30 Y=SIN(M)
40 PRINT "'SIN' OF ";M;" IS ";Y
50 END
```

Sample Run:

```
ENTER ANGLE (IN RADIANS):  1
'SIN' OF 1 IS .841470988
```

TAN

In trigonometry, TAN is used to find the tangent of an angle, expressed in radians. Tangent is the ratio of the length of the side opposite the angle in question to the length of the side adjacent to it in a right triangle. Its syntax is:

```
TAN(x)
```

where "x" can be an arithmetic expression.

Test Program:

```
10 PRINT "ENTER ANGLE (IN RADIANS):  ";
20 INPUT Z
30 Y=TAN(Z)
40 PRINT "'TAN' OF ";Z;" IS ";Y
50 END
```

Sample Run:

```
ENTER ANGLE (IN RADIANS):  1
'TAN' OF 1 IS 1.55740772
```

WARNING: FOR EXPERTS ONLY! Function

USR

USR executes a machine-language function routine and requires a full knowledge of machine-language programming. USR stands for User Supplied Routine and is usually used to perform a high-speed computation that can neither be done quickly nor expressed in SmartBASIC. Store a USR function in memory with a POKE statement.

Test Program:

NONE

Sample Run:

NONE

WARNING: FOR EXPERTS ONLY!!

Command Statement

WAIT

WAIT does what its name implies. It halts a program execution until a designated bit pattern appears at a particular port location. If you try to access a port device that is not receiving, a WAIT command will "hang up" your system. Syntax of WAIT is:

WAIT <port #(0-255)>,<value 1>,<value 2>

where value 1 is ANDed with data from <port number (0-255)>. Value 2 is optional, it is XORed with value 1. If value 2 is omitted, zero is assumed. WAIT will loop until the logical operation is zero.

Test Program:

NONE

Sample Run:

NONE

In high resolution graphics, XDRAW does the same thing DRAW does, except for the fact that the color you choose in drawing the shape is the complement of the color already existing at each plotted point. You may also use XDRAW to erase a shape (by plotting a new shape at the same location, SCALE, and ROT), and leave the surrounding graphics untouched.

IF YOUR COLOR IS:	THEN XDRAW COLOR IS:
MED.BLUE	MED.RED
MED.RED	MED.BLUE
DK.RED	GREEN
GREEN	DK.RED
BLACK	WHITE
WHITE	BLACK

Test Program:

```
5 HIMEM:51455
10 DATA 01,00,04,00
20 DATA 54,63,36,36
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: HCOLOR=3
120 SCALE=10
130 FOR i=1 TO 64
140 ROT=i
150 DRAW 1 AT 125,85
160 XDRAW 1 AT 125,85
170 NEXT
```

Sample Run:

A square with a straight line from the midpoint, downward. See the Compendium (APPENDIX C) for a picture of it.

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

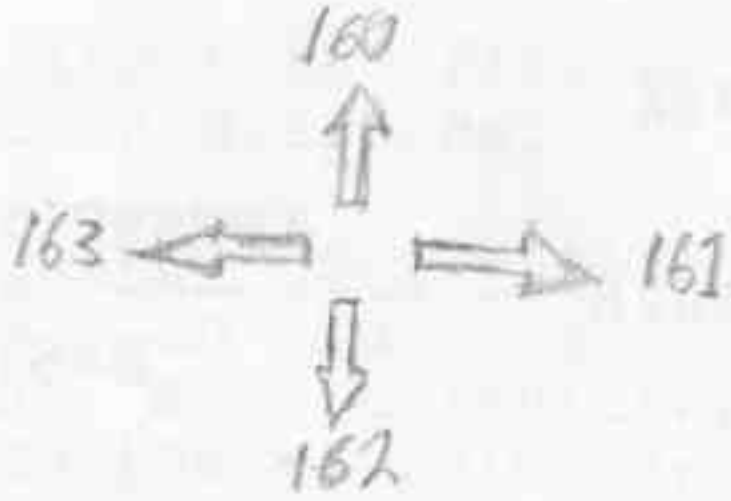
10 GET A$
20 IF ASC(A$) = 3 THEN END
30 PRINT A$: GOTO 10

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 V	
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).

SHAPE TABLE

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.

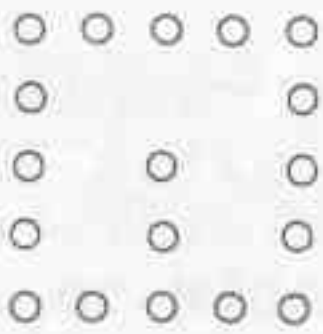
	Sec. C			Sec. B				Sec. A	
Bit #	7	6	5	4	3	2	1		0
	D	D	P	D	D	P	D		D

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

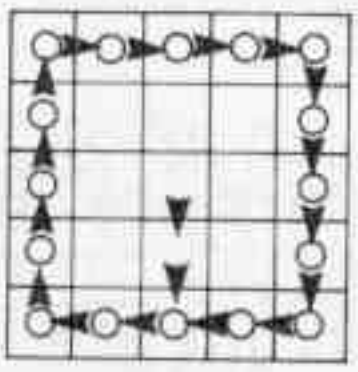
IF P=0 then don't plot a point
IF P=1 then plot a point

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:



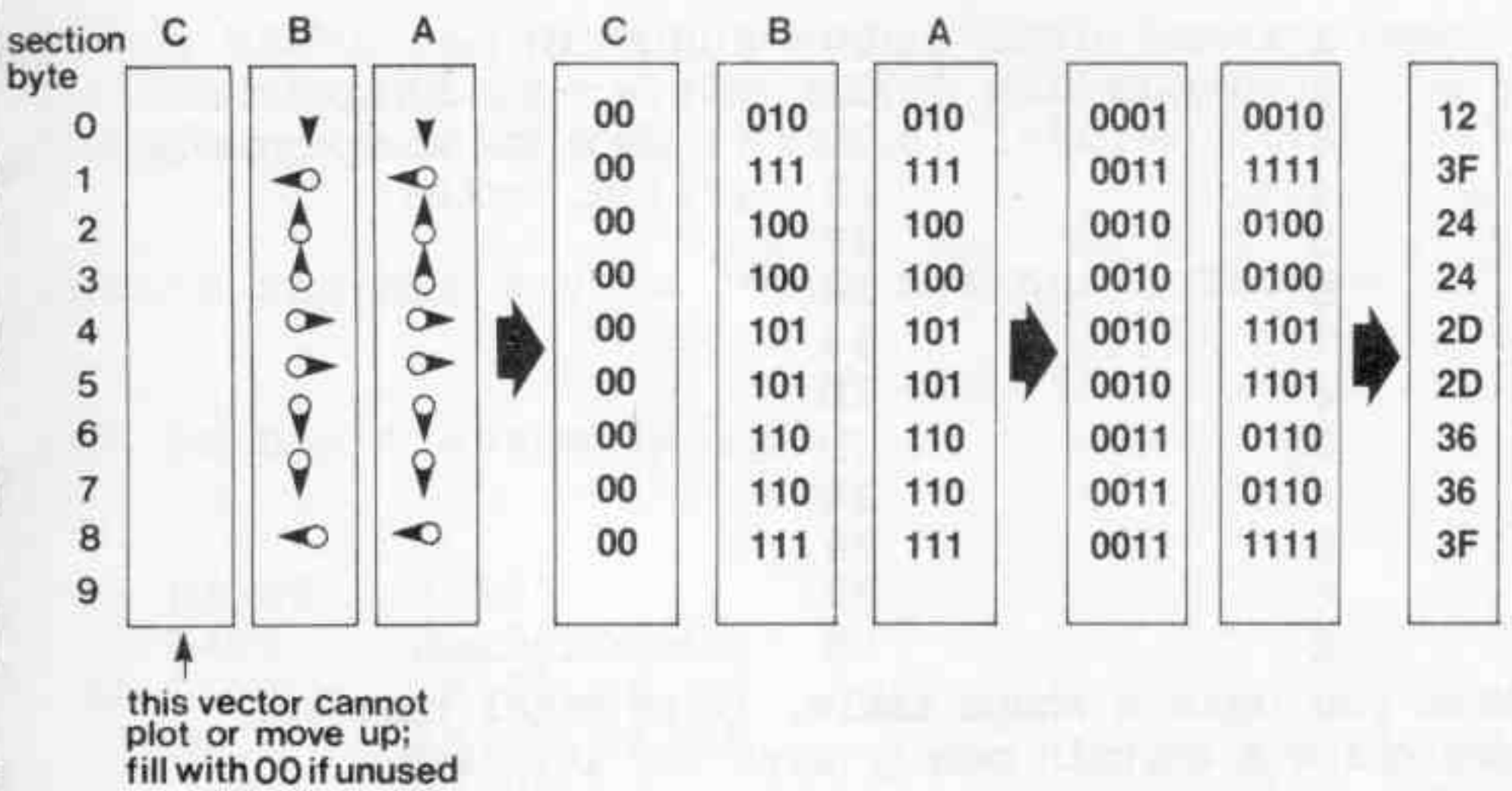
Draw it on a piece of graph paper. Keep one dot per square. Decide on where to begin your shape. We chose the center. Draw a path around your shape with arrows. These arrows are called plotting vectors.



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 read only 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.

INDEX

INDEX

NOTES

41607

COLECO