



dillithium Press

Sharon Boren • Larry Hovey • Kathleen Hovey

# An ATARI® For Kids

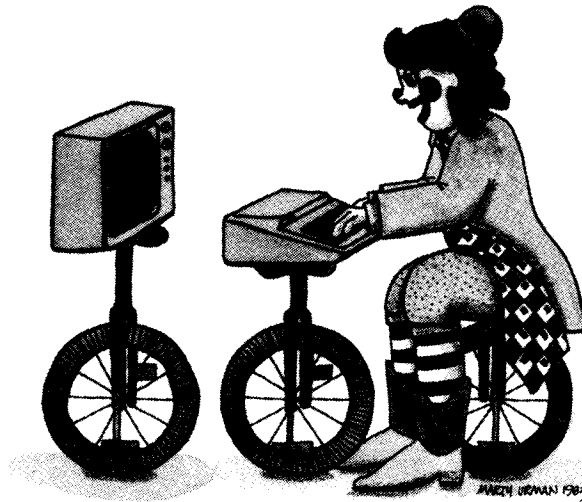




# AN ATARI® FOR KIDS



# **AN ATARI® FOR KIDS**



**Sharon Boren**  
**Larry Hovey**  
**Kathleen Hovey**



**dilithium Press**  
**Beaverton, Oregon**

© 1984 by dillithium Press. All rights reserved.

No part of this book may be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system without permission in writing from the publisher, with the following exceptions: any material may be copied or transcribed for the nonprofit use of the purchaser, and material (not to exceed 300 words and one figure) may be quoted in published reviews of this book.

Where necessary, permission is granted by the copyright owner for libraries and others registered with the Copyright Clearance Center (CCC) to photocopy any material herein for a base fee of \$1.00 and an additional fee of \$0.20 per page. Payments should be sent directly to the Copyright Clearance Center, 21 Congress Street, Salem, Massachusetts 01970.

10 9 8 7 6 5 4 3 2 1

### **Library of Congress Cataloging in Publication Data**

Boren, Sharon, 1956-  
An Atari for kids.

Includes index.

Summary: Teaches beginning programmers how to program the Atari or other microcomputers in the BASIC computer language.

1. Atari computer—Programming—Juvenile literature. 2. Basic (Computer program language)—Juvenile literature. (1. Atari computer—Programming. 2. Microcomputers—Programming. 3. Basic (Computer program language) 4. Programming (Computers) 5. Computers) I. Hovey, Larry. II. Hovey, Kathleen. III. Title.

QA76.8.A82B67 1984 001.64'2 83-25269

ISBN 0-88056-123-8 (pbk.)

Printed in the United States of America

Cover and art by Marty Urman

dillithium Press  
8285 S.W. Nimbus  
Suite 151  
Beaverton, Oregon 97005



# Acknowledgements

We dedicate this book to:

Alan Boren  
Leonard and Bernice Hovey  
Nick and Elefteria Gartelos  
Lenny and Nikki Hovey  
Jerry Willis

And to all the young *computer wizards* and brave teachers for whom *An Atari for Kids* and *An Atari in the Classroom* were written.

# TABLE OF CONTENTS

<b>INTRODUCTION</b>	1
<b>COMPONENT ONE</b>	3
Chapter 1 <b>We Have an ATARI in our Classroom!</b>	4
The microcomputer and its basic parts—keyboard, screen, cassette tape recorder or disk drive, and brain (internal circuitry)	
Chapter 2 <b>ATARI's Keyboard</b>	5
ATARI's keyboard and its functions	
Chapter 3 <b>Turning On the ATARI</b>	10
Turning ATARI on and off; BASIC, ATARI's language; initial information on the screen	
Chapter 4 <b>Using ATARI's Special Keys</b>	12
ATARI's special keys including SHIFT and CTRL (control)	
Chapter 5 <b>Fixing Typing Mistakes</b>	15
Screen editing to correct mistakes	
Chapter 6 <b>Becoming a Programmer</b>	20
Definition of a program; loading programs into ATARI; the NEW program statement	
<b>COMPONENT TWO</b>	21
Chapter 7 <b>Teaching ATARI Simple Tricks</b>	22
Statements preceded by line numbers make programs; PRINT, GOTO, and END program statements; the RUN command	
Chapter 8 <b>Loading and Saving</b>	29
Loading and saving programs; the LIST command	



---

Chapter 9	<b>Teaching ATARI to Do Your Homework</b> BASIC symbols for arithmetic operations	38
Chapter 10	<b>ATARI as a Calculator</b> Solving arithmetic equations in Direct Mode	39
Chapter 11	<b>Arithmetic with Many Numbers</b> The order of arithmetic operations, an abbreviation for PRINT	40
<b>COMPONENT THREE</b>		45
Chapter 12	<b>What Else Can ATARI Do for Me?</b> Other operations besides arithmetic	46
Chapter 13	<b>Flow Diagramming</b> Using algorithms, using flow charts	47
Chapter 14	<b>More About Flow Charts</b> Single alternative decision steps in flow charts	53
Chapter 15	<b>Double Detours</b> Double alternative decision steps in flow charts	55
Chapter 16	<b>Loop de Loop</b> Using loops in programs	61
Chapter 17	<b>Putting It All Together</b> From algorithm to flow chart to program	61
Chapter 18	<b>Printing Whole Equations</b> Programming ATARI to print whole equations	67
Chapter 19	<b>A Different Way</b> Using commas and semicolons, print zones	69
<b>COMPONENT FOUR</b>		73
Chapter 20	<b>ATARI's Memory</b> Memory, numeric variables, the LET program statement	74

---

Chapter 21	<b>Using Variables</b> Variable address and variable contents; printing variable names and contents	76
Chapter 22	<b>Using Variables in Equations</b> Using variables to print and solve arithmetic equations	79
Chapter 23	<b>Important Information</b> Using LET and PRINT statements in proper order	82
Chapter 24	<b>A Shortcut</b> Using commas and colons to shorten programs	86
Chapter 25	<b>What Types of Numbers Does ATARI Like?</b> Using numbers that ATARI can use; E notation numbers	87
<b>COMPONENT FIVE</b>		89
Chapter 26	<b>FOR-NEXT Looping</b> Using FOR-NEXT loops	90
Chapter 27	<b>Stepping</b> The STEP program statement	96
Chapter 28	<b>A Counter</b> Using counters in programs	99
Chapter 29	<b>A Clean Trick</b> Clearing the screen; FOR-NEXT time loops; Using colons with FOR-NEXT statements	102
Chapter 30	<b>Blinkers</b> Programming blinking output	106
Chapter 31	<b>Special Commands</b> BYE and CONT commands; the STOP program statement; Using Memo Pad and Testing Mode	109
Chapter 32	<b>Debugging</b> Types of computer errors; getting rid of errors	112



---

<b>COMPONENT SIX</b>	114
Chapter 33 <b>Strings</b> String variables; the DIM (dimension) program statement	115
Chapter 34 <b>Input</b> Interactive programming; the INPUT program statement	117
Chapter 35 <b>IF-THEN</b> BASIC signs for comparisons; the IF-THEN program statement; the complement of a question	121
Chapter 36 <b>Alphabetizing</b> Programming ATARI to alphabetize	128
Chapter 37 <b>Remarks</b> The REM (remark) program statement for documentation	130
Chapter 38 <b>READ-DATA</b> The READ-DATA program statements	132
Chapter 39 <b>Problem-Solving Programming</b> Programming to solve problems	140
<b>COMPONENT SEVEN</b>	148
Chapter 40 <b>Conversions</b> Programs that convert (change) information	149
Chapter 41 <b>Random Numbers and Integers</b> The RND (random number) and INT (integer) functions	152
Chapter 42 <b>Making Sounds</b> The SOUND program statement	156
Chapter 43 <b>Graphics</b> The GRAPHICS, COLOR, PLOT, and DRAWTO program statements	160
Chapter 44 <b>More Graphics</b> The SETCOLOR program statement; graphics without the text window; combining graphics and sounds	170

---

Chapter 45	<b>Writing Game Programs</b>	174
	Writing user-friendly computer games	
Chapter 46	<b>You Are a Creative Programmer!</b>	178
	Programming as a creative experience	
<b>EPILOG</b>		179
<b>APPENDICES</b>		180
Appendix A	<b>BASIC Commands, Statements, and Functions</b> (with abbreviations)	180
Appendix B	<b>Error Messages</b>	182
Appendix C	<b>Game Controllers</b>	185
<b>GLOSSARY</b>		186
<b>INDEX</b>		193



# INTRODUCTION

*An ATARI for Kids* is part of a three-book set designed to teach children and beginning programmers how to program a microcomputer in the BASIC computer language. Although this book is geared specifically for the ATARI microcomputer, it can be easily adapted for use with other microcomputers as well.

Written at approximately a fourth grade reading level, *An ATARI for Kids* consists of seven components of approximately six chapters each. You become familiar with the keyboard and ATARI operation in the first components, and learn how to write your own BASIC programs as you progress through the book. By the time you have completed the last component you will have the skills needed to write game programs, simple graphics, teaching programs, and programs that solve problems. All programming techniques introduced can be easily understood by the average sixth grade student.

**How to use this book:** Read through the chapters and try the examples on your computer. At the end of some chapters there are notes on worksheets "**to do**." (For example, you'll see "**to do**: Programmer's Pastime #11.") Sometimes these activity worksheets are included at the end of chapters so you may try your hand at writing your own programs. Otherwise, you'll find all these programming worksheets in the second book of the set, *An ATARI in the Classroom: Activity Workbook*. Solutions to the activities can be found in the *Teacher's Guide* (the third book in the set) which also contains lesson plans for each chapter and additional information and ideas for using this material as a computer programming curriculum. *An ATARI for Kids* is the student text in this set. Both the *Activity Workbook* and the *Teacher's Guide* can be ordered from the card at the back of the book.



# COMPONENT 1

## CHAPTER 1

We Have an ATARI in Our Classroom!	4
---------------------------------------	---

## CHAPTER 2

ATARI's Keyboard	5
------------------	---

## CHAPTER 3

Turning on the ATARI	10
----------------------	----

## CHAPTER 4

Using ATARI's Special Keys	12
----------------------------	----

## CHAPTER 5

Fixing Typing Mistakes	15
------------------------	----

## CHAPTER 6

Becoming a Programmer	20
-----------------------	----

# CHAPTER 1

## We Have an ATARI in Our Classroom! . . .

It's not a new student, or an animal. It's not something to eat or sit on. It's bigger than a shoebox and smaller than a car! It's an ATARI!

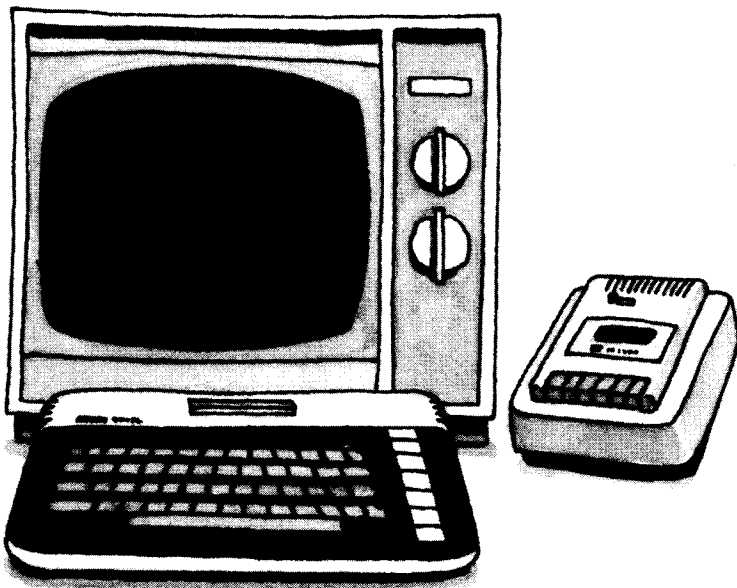
What 's an ATARI? It's a microcomputer made by Atari, a Warner Communications company. We will be seeing more and more microcomputers in homes and classrooms in the future, and now we have an ATARI in our classroom.

What is a microcomputer? It is a small portable computer that anyone can learn to use. Microcomputers can teach us lessons in school, help us with hard assignments, or even be our partner in playing a game. It is a fun and valuable machine to have in a classroom.

Our ATARI microcomputer has four basic parts:

1. a keyboard (push keys with letters, numbers, and signs)
2. a screen
3. a cassette tape or disk drive unit
4. a brain (its insides)







Let's learn about the parts of ATARI so we can use it in our work and play!



# CHAPTER 2

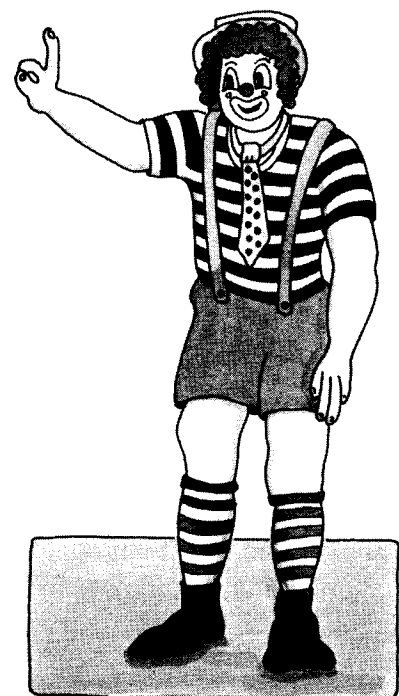
## ATARI'S Keyboard

### DIFFERENT KEYBOARDS

ATARI has sold various models with keyboards that differ slightly. On the older models, the control key looks like this: . On the newer XL models it looks like this: . If you have a newer model, push  when this book tells you to push . Also, on the older models, the reverse field key looks like this . On the newer XL models, and in this book, it looks like this . Whichever keyboard you have, you are ready to learn more about using the ATARI in your classroom.

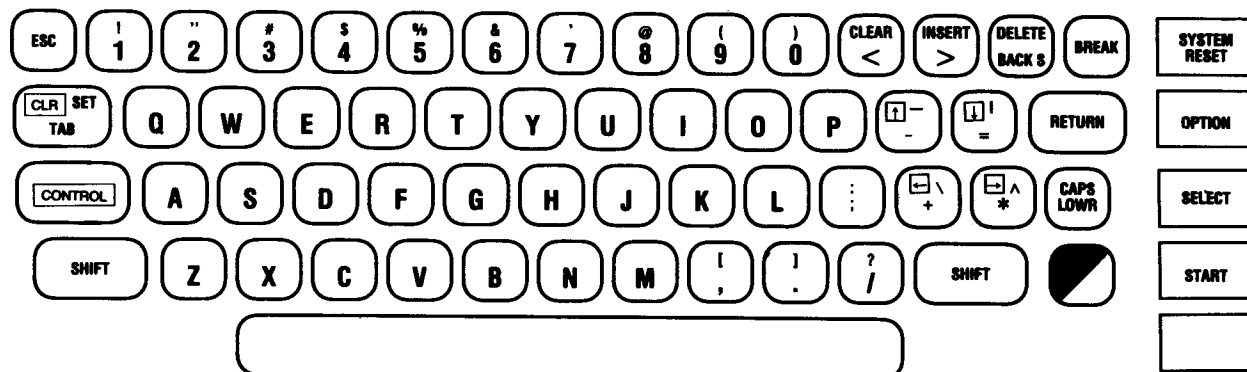
### CAUTION

It's a good idea to protect your ATARI from dust by covering it with a piece of plastic when it is not being used. Also, NEVER bring drinks or food around ATARI. Anything spilled inside the computer can cause real problems!



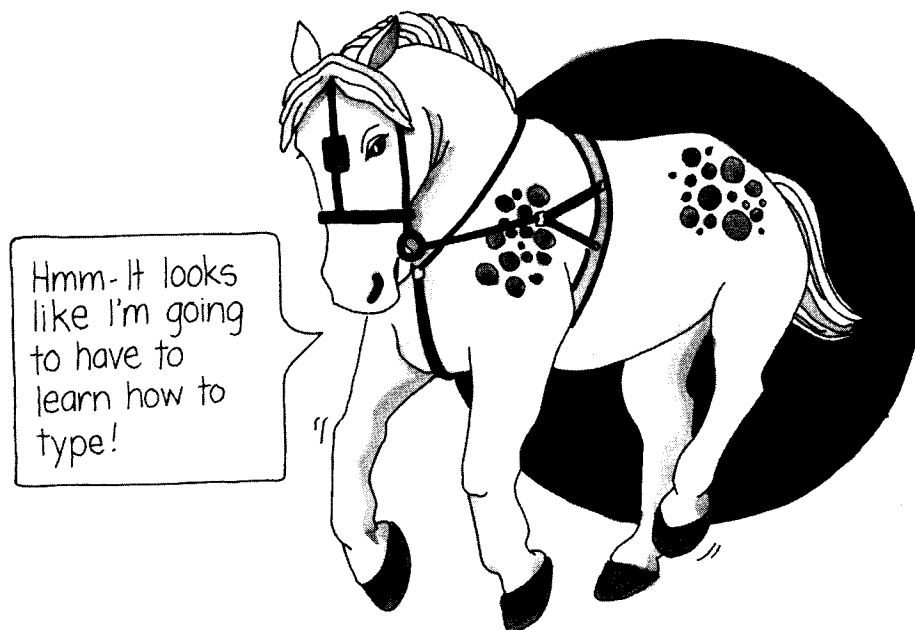


Here is the ATARI XL computer keyboard:

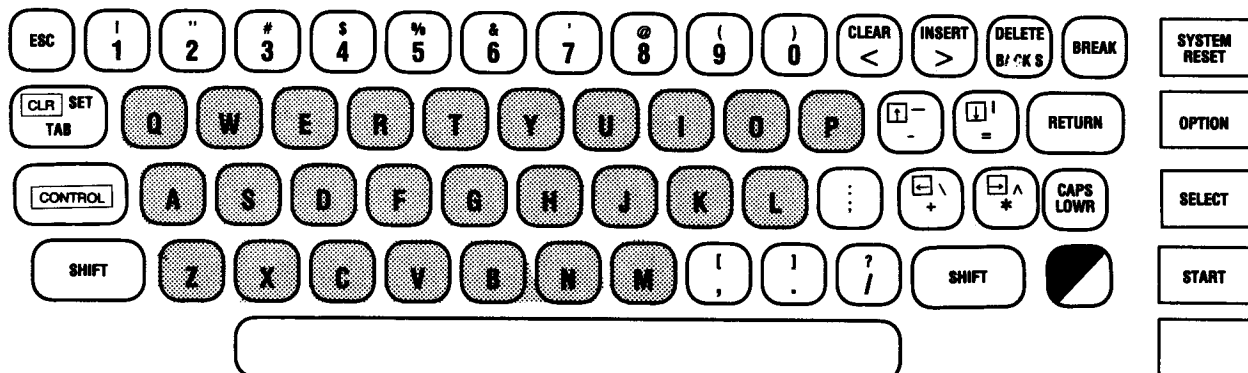


ATARI has a keyboard much like a typewriter.  
You can punch:

1. letters
2. numbers
3. function keys  
( SHIFT , RETURN , BREAK , and more)
4. graphics  
( ♥ , + , ⊥ , and more)
5. special symbol keys  
( + , - , \* , \$ , = , and more)
6. cursor control keys  
( ⬅ , ➡ , ⬆ , ⬇ )

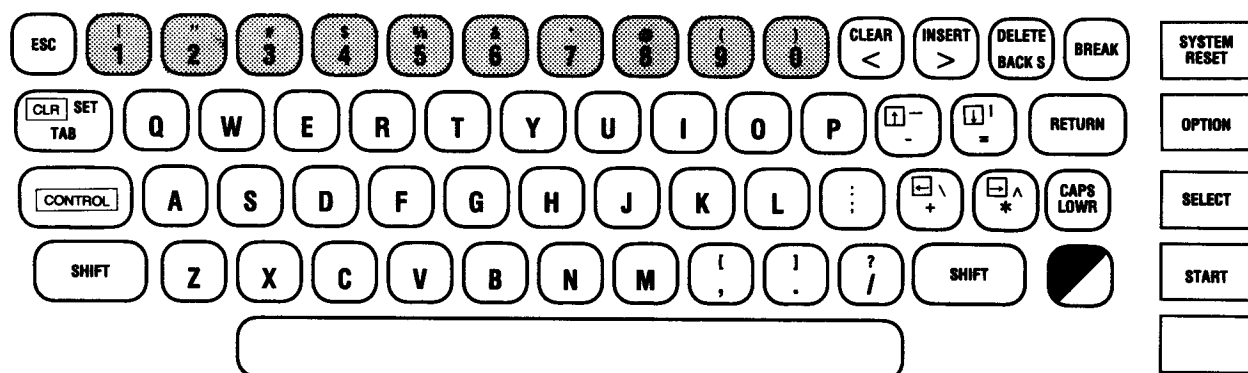


## LETTERS

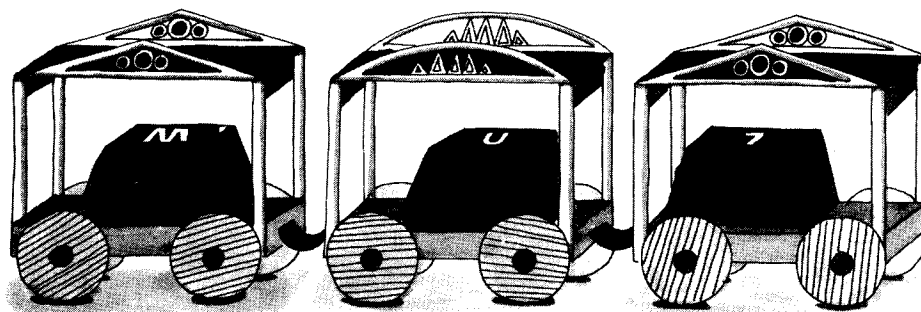


Letter keys on ATARI's keyboard are in the same places as letter keys on a typewriter. To type a letter, press the key.

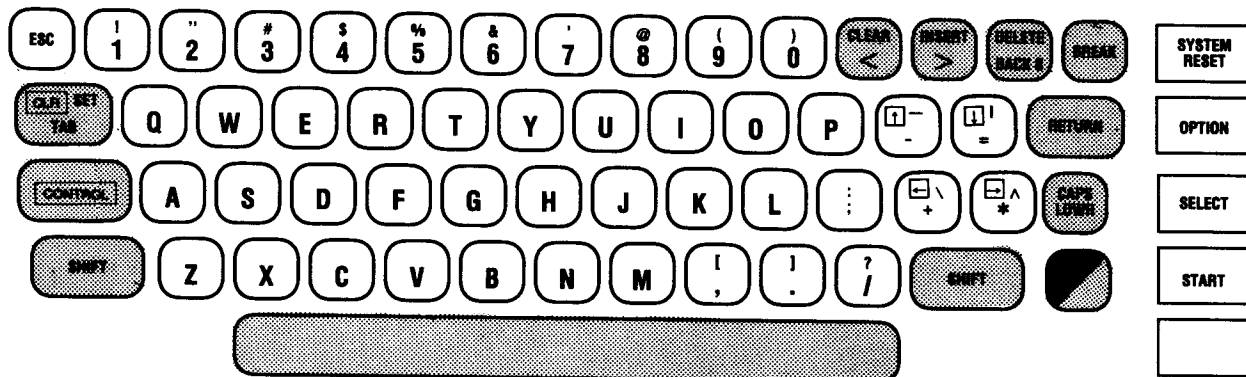
## NUMBERS



Number keys are found on the top row of the keyboard. To type a number, press the key.

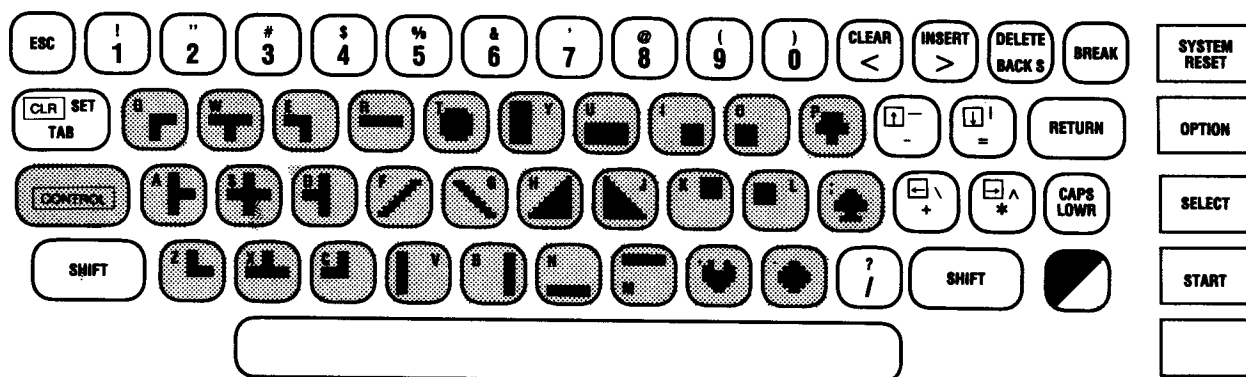


## FUNCTION KEYS



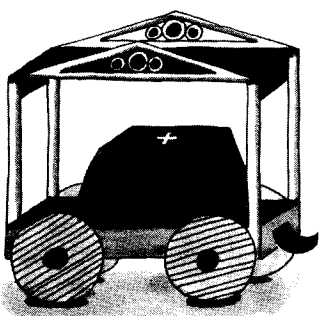
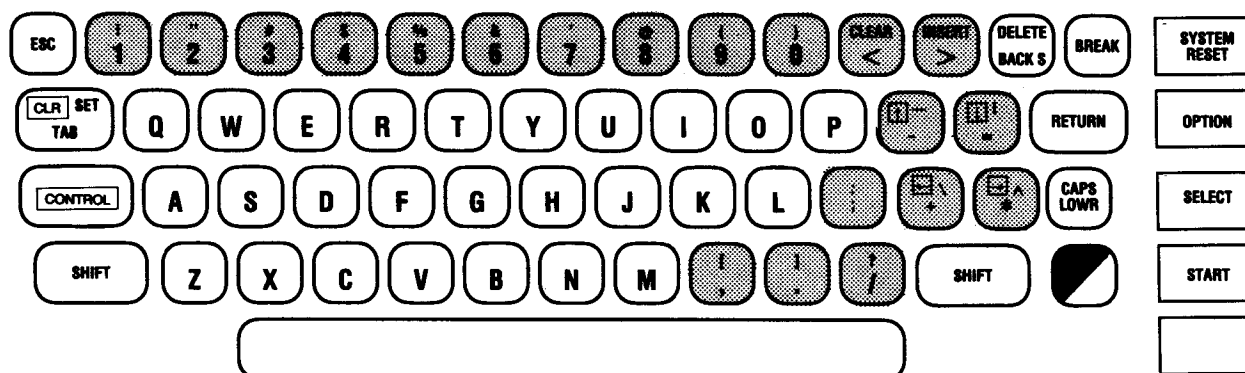
Each **FUNCTION** key does something special. They are very important keys. You will learn more about these keys later.

## GRAPHICS



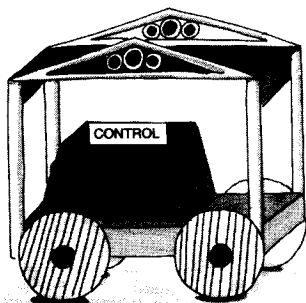
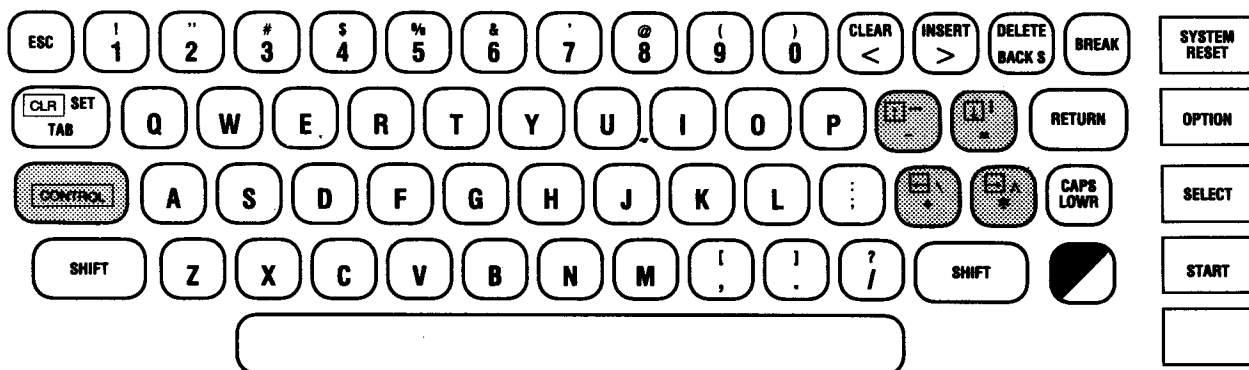
The **GRAPHICS** keys are one way you can make pictures and borders on ATARI's screen. (Later you will learn another way to make graphics.) The graphics symbols are *not* shown on the keyboard. To find the key locations you will have to refer to the above picture when you use the computer. There are 29 different graphic symbols. To type a graphic, press the **CTRL** key first and hold it down. Then type the graphics key.

## SPECIAL SYMBOL KEYS



ATARI has many SPECIAL SYMBOL keys. They are used for doing math as well as punctuating sentences that we write. Some special keys are used as shortcuts in operating ATARI. If the special symbol is on the top of a key, you must press the SHIFT key and symbol key at the same time. If the special symbol is on the bottom of a key, just press the key.

## CURSOR CONTROL KEYS



**CURSOR CONTROL** keys let you move around ATARI's screen and type wherever you like. To move about the screen, you must hold down the **CTRL** key and press the key with the arrow in the direction you want to move, or they can be held down to move more rapidly.

# CHAPTER 3

## Turning on the ATARI

1. Flip the **ON-OFF** switch on ATARI's right side to ON. You can tell when ATARI is on, because the red light on the right hand side of the keyboard comes on.
2. Turn on the television.
3. The T.V. screen should say:


READY

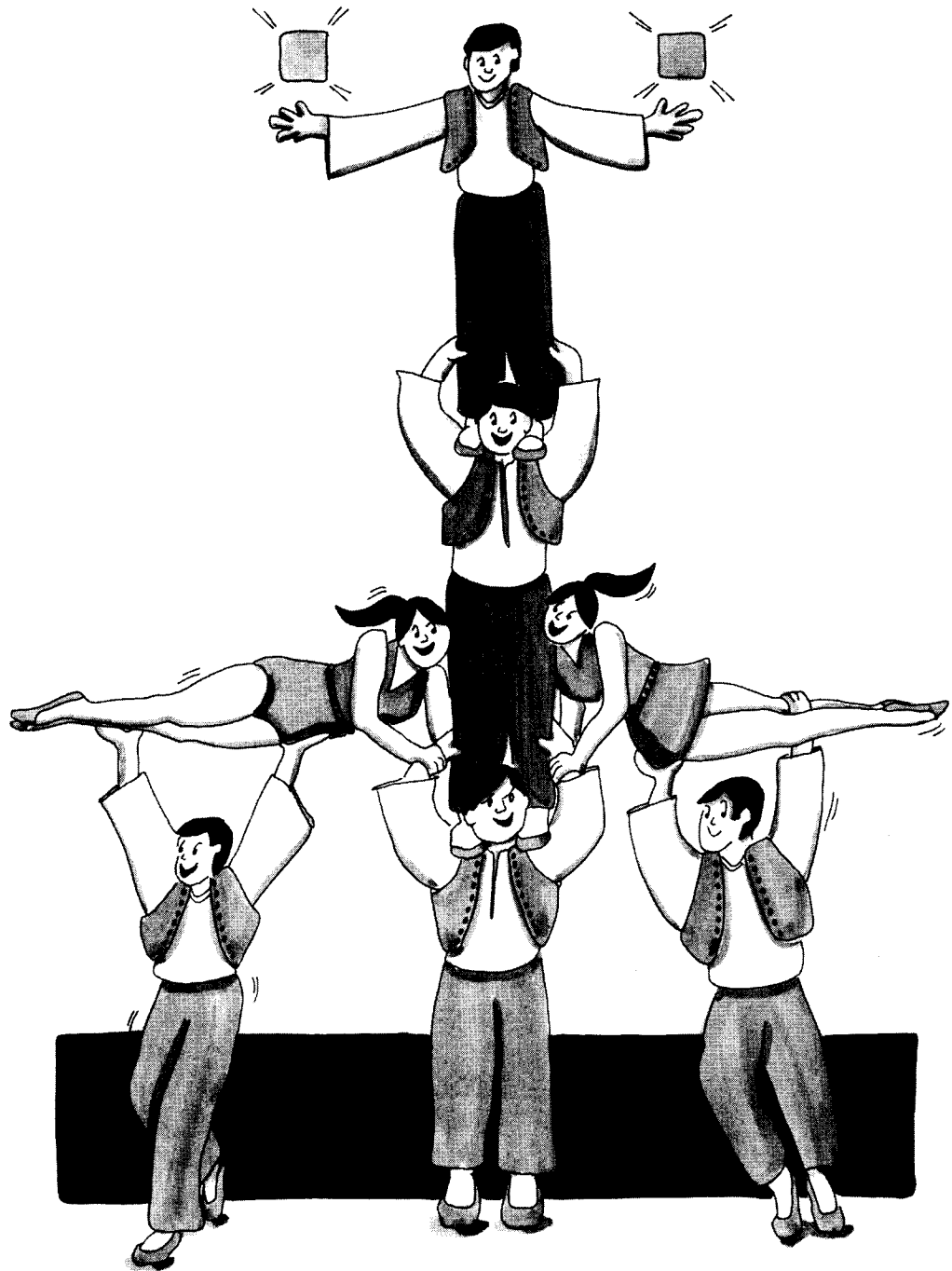


4. **READY** means that ATARI is ready for you to tell it what to do. But before we can tell it what to do, we must learn how to speak to ATARI in a language that it can understand.
5. ATARI doesn't understand English, but it does understand several different **COMPUTER LANGUAGES**. The language that we will speak to ATARI in is called **BASIC**. BASIC is short for Beginner's All-purpose Symbolic Instruction Code.



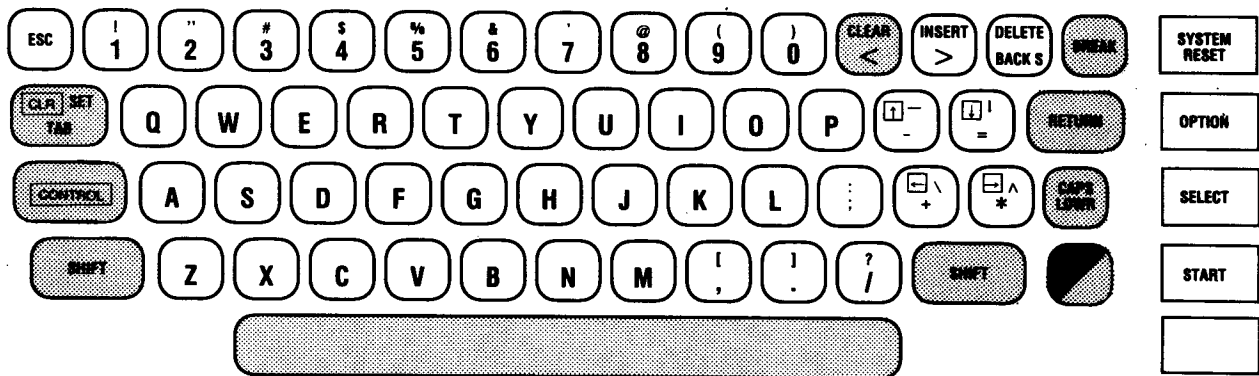


6. The white square  below the READY is called the **CURSOR**. The cursor shows you where the letters and the other symbols will appear on the screen when you type on the keyboard.
7. If you turned ATARI on and something went wrong, just turn off the switch and start over again.



# CHAPTER 4

## Using the ATARI's Special Keys



1. RETURN

**RETURN** tells ATARI that you are finished typing a line, and puts what you have typed into ATARI's memory. It also tells ATARI to put the cursor on the next line.

ALWAYS press RETURN when you are done typing a line.

2. (SPACE BAR)

Pressing the **SPACE BAR** leaves a blank space. This key is not labeled. It is the long bar at the bottom of the keyboard. You must press this key after you type a word or a number.

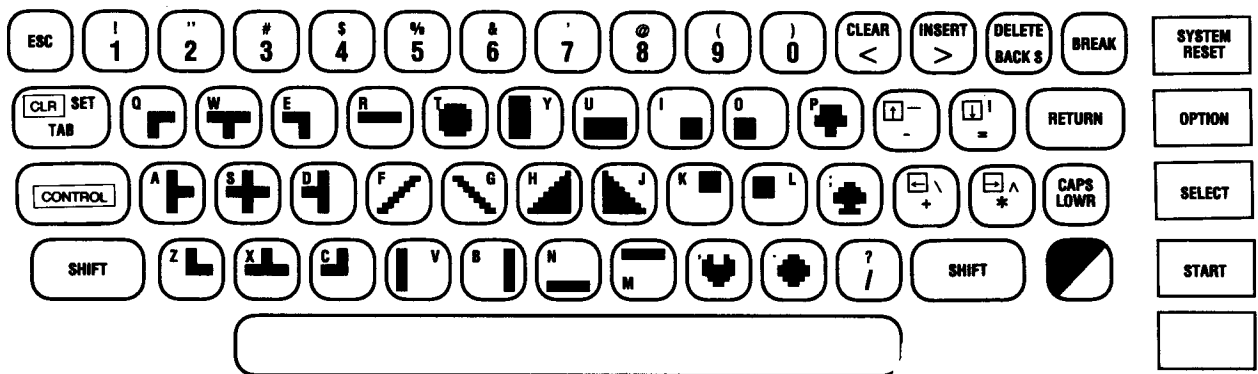
Otherwise your typing will look like this!


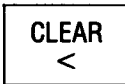
3. SHIFT


HOLD THE **SHIFT KEY** DOWN as you press another key and ATARI will print the symbol on the top or top right of the key, or it will carry out the function labeled at the top of certain keys.

4. CONTROL or CTRL

Hold this key down while pressing the keys with hidden graphic symbols (see following chart). Graphics, not letters, will appear on the screen. The CONTROL key is also used with the **CURSOR CONTROL** (arrow) keys to move the cursor around the screen. (On some of the older models, and in this book, the control key looks like this CTRL.)



5.  + 


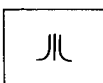
Press SHIFT and hold it down. Then press . This will erase the screen, and send the cursor to the **HOME** position in the upper left corner of the screen.



6. 

Use this key when you want ATARI to STOP what it is doing. ATARI tells you where the program stopped by printing a **BREAK** message like this:

```
390
*** READY ***
```




When you use the BREAK key, ATARI will show you the line where the program has stopped (in this example, at line 390).

7.  or 

Press this key before you type, and ATARI will print in **REVERSE FIELD**. The words will appear in dark characters on a white background instead of white characters on a dark background. Touch  again to end the reverse field. (On some of the older models, the REVERSE FIELD key may look like this .)




8.

CAPS  
LOWR

Pressing the  key will cause ATARI to print with only lowercase (small) letters. To get back to capital letters, hold  and press . During programming, ATARI uses only capital letters. The computer cannot understand lowercase letters within a program, and will give an Error Message if it finds one.


9.

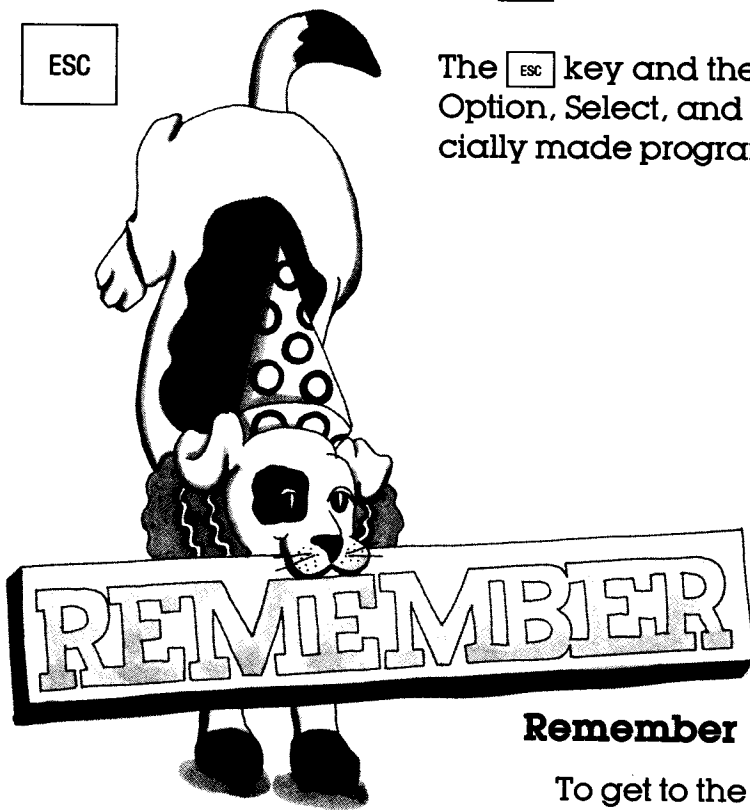
CLR SET  
TAB

The  key allows the cursor to be moved several spaces at one time. Push this key and the cursor will jump several spaces. The jump positions can be set by holding  and pressing this key. The jump positions can be erased by holding  and pressing this key.


10.




ESC




The  key and the **SYSTEM KEYS** (System Reset, Option, Select, and Start), are used with commercially made programs and for other purposes.



### Remember

To get to the bottom symbol on a key, just press the key. For example, press  for the < symbol.

To get the top symbol or function on a key, press  and the key. For example, press  and  for the CLEAR function (clears the screen).

To get graphics, press  and the hidden graphics key. For example, press  and  to get -.

**to do:** Exploring ATARI's Keyboard #1

# CHAPTER 5

## Fixing Typing Mistakes

If we type something wrong, ATARI won't understand. That's why it's important to correct typing mistakes.

ATARI lets us know when it doesn't understand. If we misspell a command, or forget to speak in BASIC, ATARI will let us know by printing an error message on the screen:

ERROR



This is ATARI's way of saying, "I don't understand you. Please try again."

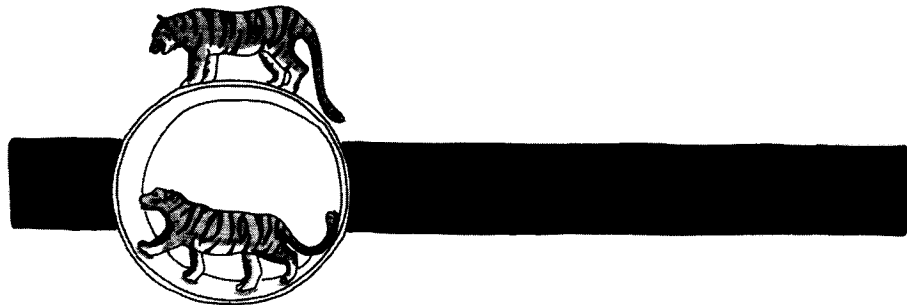
Sometimes the error message is followed by a number:

ERROR—

12 AT LINE 10

This gives us more information about the error—what kind it is, and where it is located. You can look in Appendix B to see what the error numbers mean.

Lucky for us, ATARI has a special feature called *screen editing* which allows us to fix typing mistakes. This helps to keep us from getting so many error messages.





Here's how screen editing works. First we must move the cursor to where the error is located. This can be done by holding the **CTRL** key while pressing the Cursor Control (arrow) key for the direction we want. By doing this we can move the cursor anywhere on the screen without erasing any of our writing.

<b>CTRL</b>	+	<b>↓</b>
<b>CTRL</b>	+	<b>↑</b>
<b>CTRL</b>	+	<b>→</b>
<b>CTRL</b>	+	<b>←</b>

moves the cursor down.

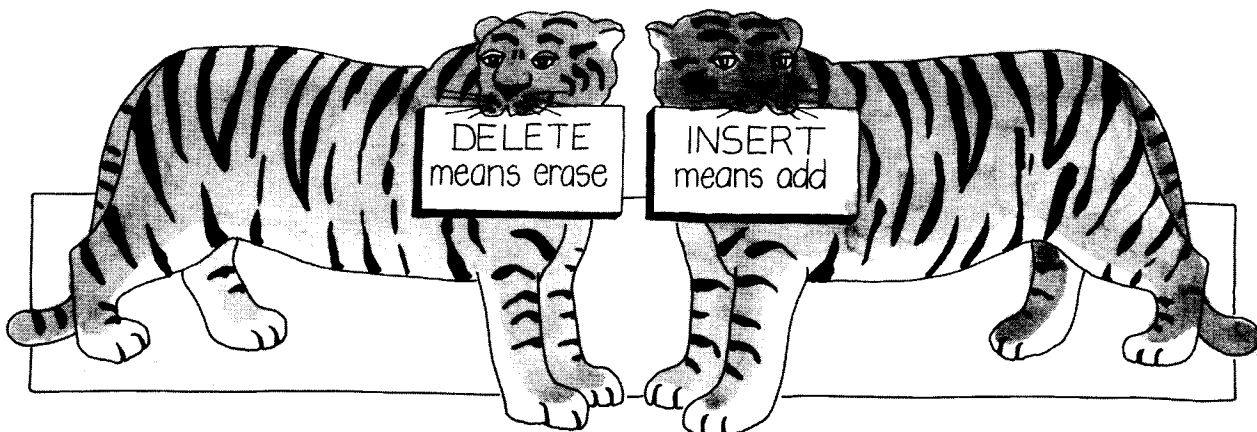
moves the cursor up.

moves the cursor to the right.

moves the cursor to the left.

After the cursor is moved to the location of the mistake, there are several ways to correct it:

1. Press the Space Bar to erase the mistake.
2. Type over the mistake.
3. Use the **DELETE BACKS** key. **DELETE** means to erase. Press this key and ATARI will erase the letter on the left side of the cursor. Now type in the correct letter. Sometimes you may want to erase an entire line. This can be done by holding the **SHIFT** key and pressing **DELETE BACKS**.
4. Use the **INSERT >** key. **INSERT** means to put in. It is handy to use if you forgot to type something. Press **CTRL** and **INSERT >** and ATARI will add a space to the right of the cursor. Now you can type whatever you forgot the first time. Press **SHIFT** and **INSERT >** and ATARI will add a new line.



---

After you have made the needed changes, and the entire line is as you want it, you *must* press the  key. If you forget  , ATARI may fool you. Sometimes it is possible to change only what is printed on the screen through screen editing, and not change the information that has gone inside the computer. If the inside information is not changed, it could cause an error in your program.

Now that you know how easy screen editing is, you don't have to be afraid to make mistakes! With a little practice, you can easily correct them.

**to do:** Exploring ATARI's Keyboard #2, #3, #4, #5, #6, #7



# EXPLORING ATARI's Keyboard #2

1. Turn ATARI on.
2. Press RETURN .  
What did the cursor do?  

---
3. Press SPACE (It's the long unlabeled bar at the bottom.) What did the cursor do this time?  

---
4. Press SHIFT and CLEAR together.  
What did the cursor do?  


---
5. Type your name (first, middle, and last).
6. Hold CTRL and press ← five times.  
Which way did the cursor move?  
Did anything happen to the writing on the screen?  
Now you know how to move the cursor around the screen without erasing any of the writing.
7. Hold CTRL and press the various cursor control keys (those with arrows ↑ ↓ ← → ).  
Notice how the cursor moves around the screen without changing the writing.
8. Use the CTRL and cursor control keys (with arrows) to move the cursor to the first letter of your middle name.
9. Press the space bar two times.  
What happened?  



---
10. Press the DELETE BACK S key five times.  
What happened?  



---
11. Hold SHIFT while pressing CLEAR .  
What happened?  



---





# EXPLORING ATARI's Keyboard #4



1. Turn ATARI on.
2. Type I LIKE YOU ATARI
3. Press  five times.  
What happened? 

---
4. Hold  and press  five times.  
What happened? 

---
5. Hold  and press  five times.  
What happened? 

---
6. Hold  and press  .  
What happened? 

---
7. Type I LOVE YOU ATARI
8. Hold  and press  until the cursor is on the Y of YOU.
9. Hold  and press  four times.  
What happened? 

---
10. Hold  and press  four times.  
What happened? 

---
11. Type YOU back in the new space.
12. If you have time, try typing some lines of your own, and use the various keys to do some screen editing.

# CHAPTER 6

## Becoming A Programmer

To learn how to play a new game you must follow a set of directions. The same is true for ATARI. ATARI cannot play with you, or even talk with you unless it has directions to follow.

We call the set of directions ATARI uses a **PROGRAM**. Computer programs are written by people called **PROGRAMMERS**. As you work through this book, you will learn how to program ATARI and become a computer programmer.

ATARI learns programs in two ways:

1. The programmer types the program on the keyboard. ATARI then copies the program in its memory in order to understand and remember it.
2. ATARI listens to either a cassette tape recorder or a disk drive on which the program is recorded. ATARI copies the program from the tape or diskette (which looks like a 45 rpm record) to its memory where it will remember and understand the program.

For now, ATARI will be working with only one program at a time. Every time you want to do a different program on ATARI, you must first erase the old program from ATARI's memory. Then you can **LOAD** the new program. You erase ATARI's memory by typing **NEW** and pressing .

NEW



**to do:** Component 1 Fun Page  
Evaluate yourself



# COMPONENT 2

## CHAPTER 7

Teaching ATARI Simple Tricks 22

## CHAPTER 8

Loading and Saving 29

## CHAPTER 9

Teaching ATARI to Do Your Homework 38

## CHAPTER 10

ATARI as a Calculator 39

## CHAPTER 11

Arithmetic with Many Numbers 40

# CHAPTER 7

## Teaching ATARI Simple Tricks

If you can teach your pet dog or pet alligator to do tricks, you can certainly teach ATARI tricks. When you teach ATARI a trick, you are actually writing a simple computer program in the BASIC computer language. Let's learn how!

1. Type NEW. Typing NEW is done to make sure ATARI's memory is cleared or erased. You must clear ATARI's memory before each new trick, or program. (Don't forget to press RETURN after you finish typing a line.)

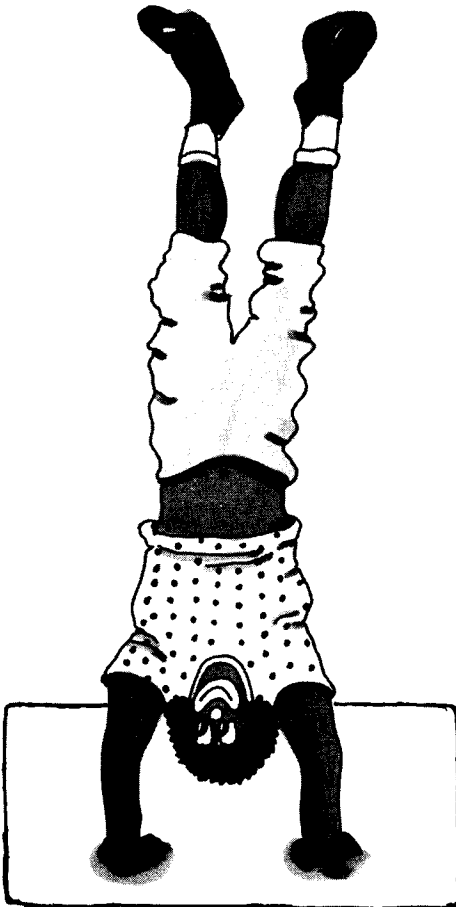
2. ATARI's screen should say:

```
READY  
NEW  
READY  
□
```

3. There are usually many steps in a program. ATARI knows in what order to do each step because at the beginning of each line you place a number called a **LINE NUMBER**. Be sure to number each step in the order you want them to go. If the steps are not in the correct order, ATARI may not be able to run the program and will give an error message.

4. Lines are usually numbered in steps of 10 like this:

```
10  
20  
30
```



- 
5. If we forget to put in a step, there are nine numbers in between each line that we can use to add the missing step—like this:

10  
20  
30  
15

We use 15 for the missing step if the step needs to come second in the program. It's OK to put step 15 last because ATARI sorts through all the line numbers later and puts them in order—like this:

10  
15  
20  
30



## Trick #1: Saying Hello

This trick teaches ATARI to say hello to you! Type the program below:

```
10 PRINT "HELLO ____" (Type your name in  
20 END                the blank.)  
RUN
```

REMEMBER: Press  after typing each line.

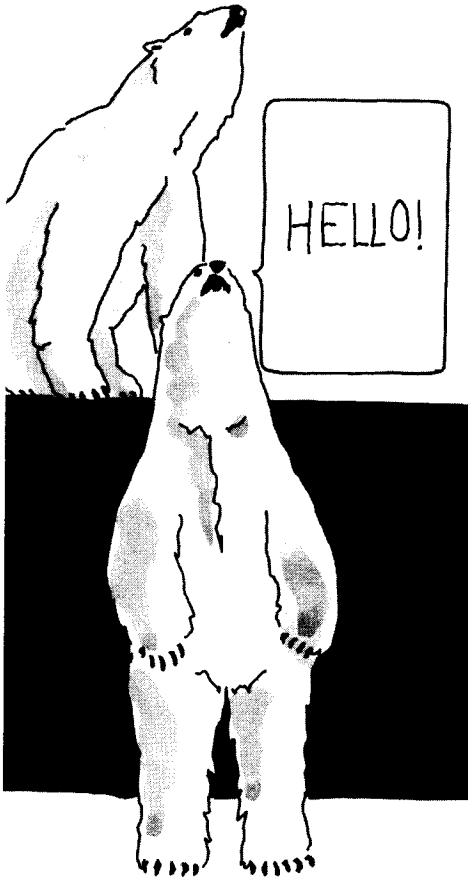
Line 10 tells ATARI to print whatever is inside the quotation marks. Line 20 tells ATARI that the program has ended. (It is not *necessary* to use **END**. ATARI will assume that the program is over if there are no more line numbers and program statements. However, it is good programming technique to use **END** as the last statement of your program, and you should always try to do so.)

**RUN** tells ATARI to execute, or do, the program. ATARI will do the program once each time you type **RUN** and press .

Whatever ATARI prints is called **OUTPUT**. The **OUTPUT** for the above program would be:

HELLO (your name)

As you probably can guess, whatever goes into ATARI is called **INPUT**. Input is fed into ATARI by typing on the keyboard, or by loading information from the cassette tape recorder or disk drive.



---

## Trick #2: Drawing Pictures

This trick (program) uses graphics and teaches ATARI to draw. Type:

```
NEW
10 PRINT "_____"; (Type your favorite
20 GOTO 10             graphics in the
RUN                   blank.)
```

**PRINT** tells ATARI to write on the screen. ATARI will print whatever you put inside quotation marks.

**GOTO** tells ATARI to go to line 10 over and over again. This program never ends.

When you are tired of watching this program run, push  . This will cause the program to stop running. You can now type RUN to start it up again, or retype line 10 with a different graphic, or turn off ATARI for now.





## **Remember!!!**

1. Every step in your program must begin with a LINE NUMBER.
2. Press  when you are done typing a line.
3. Put `` `` around what you want ATARI to say.
4. Type RUN when you want ATARI to do (execute) a program.
5. Press BREAK when you want ATARI to stop running a program.
6. Type NEW before you do a new program. This clears ATARI's memory by erasing the old program.

**to do:** Programming Your ATARI #1, #2, #3, #4, #5

# Programming Your ATARI #2

## Speaking Nonstop

1. Write a program that tells ATARI to print your name over and over again!
2. RUN your program on ATARI.

## Use this format

```
10 PRINT "      " (Type your name inside the quotes.)  
20 GOTO 10
```

## Write your program here

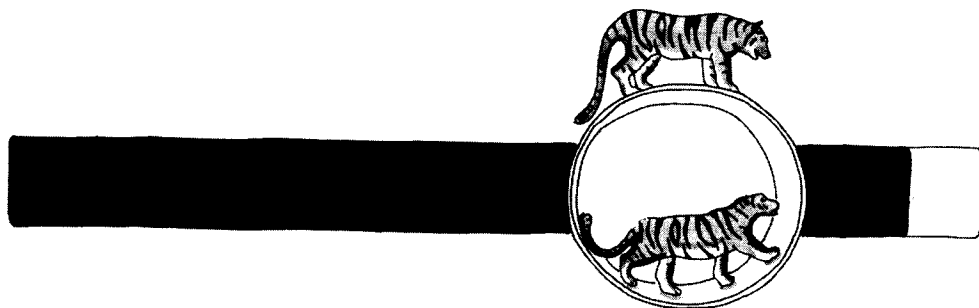
# Programming Your ATARI #3

## Top Secret

1. Your mission is to write a program that tells ATARI to print a top secret message in a secret code. Use the graphic symbols on the keys as your code. For example, if we wanted a word in our message to say SAW, the code would be + - T because these graphic symbols appear on the S, A, and W keys.
2. Give your program to a friend. Have your friend RUN it on ATARI and try to decode the secret message.

Your success as a secret agent depends on this program. Good luck! (This page will self-destruct in two days if your program is not finished.)

**Write your program here**





# CHAPTER 8

## Loading and Saving

Learning to write your own programs can be fun, but sometimes you may want to use a program written by someone else. Or, you may want to save a program you have written to use at a later time. To do either, you will have to learn to use the cassette tape recorder or disk drive that goes with the ATARI computer.

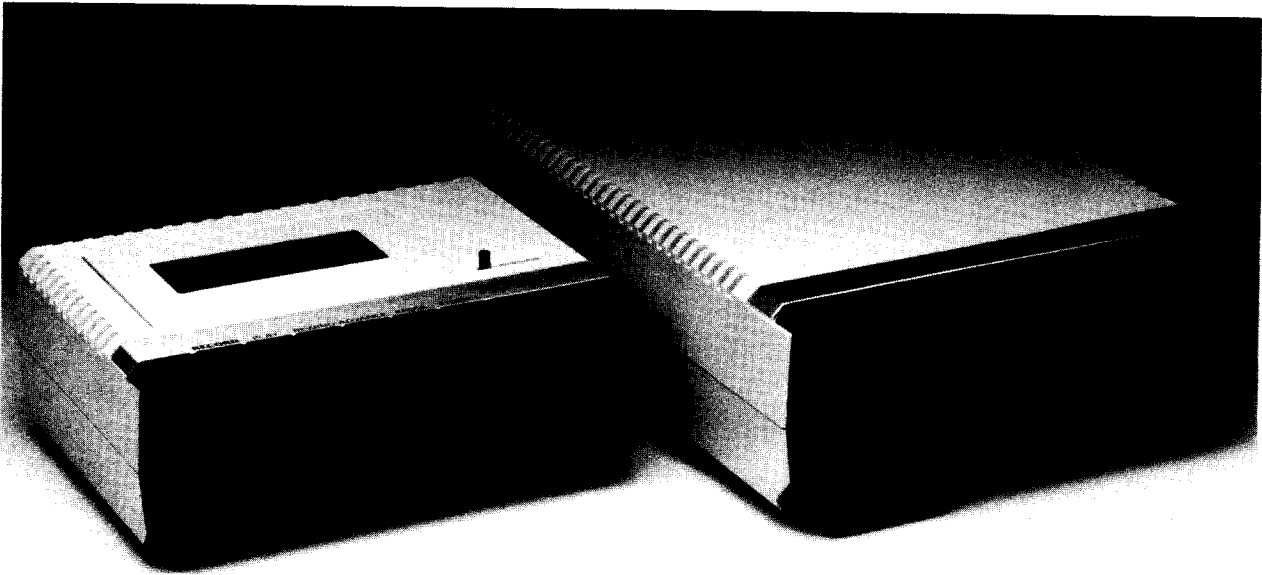
ATARI's cassette tape recorder probably looks like the tape recorder you may have at home or at school.



---

A regular cassette tape is used to store the programs. Any cassette tape can be used, but the short ones are best. Long tapes require a lot of rewinding time when you use both sides of the tape.

Rather than a cassette recorder, your ATARI might have a disk drive unit.



ATARI Cassette Recorder and Disk Drive

The disk drive stores information on something that looks like a 45 rpm record, called a diskette. Either the tape unit or the disk drive will work well with the ATARI computer. The disk drive works quite a bit faster, but it is also a lot more expensive.

Let's see how the tape recorder and disk drive work with ATARI. We'll start with the tape recorder. (If your ATARI has the disk system, you may want to skip this section and go on to the disk drive one.)

# Cassette Tape Recorder

Although cassette recorders for ATARI may vary slightly, most have the following button controls:

**REC** This is the record button. REC and PLAY pushed down together allow programs to be stored on the tape.

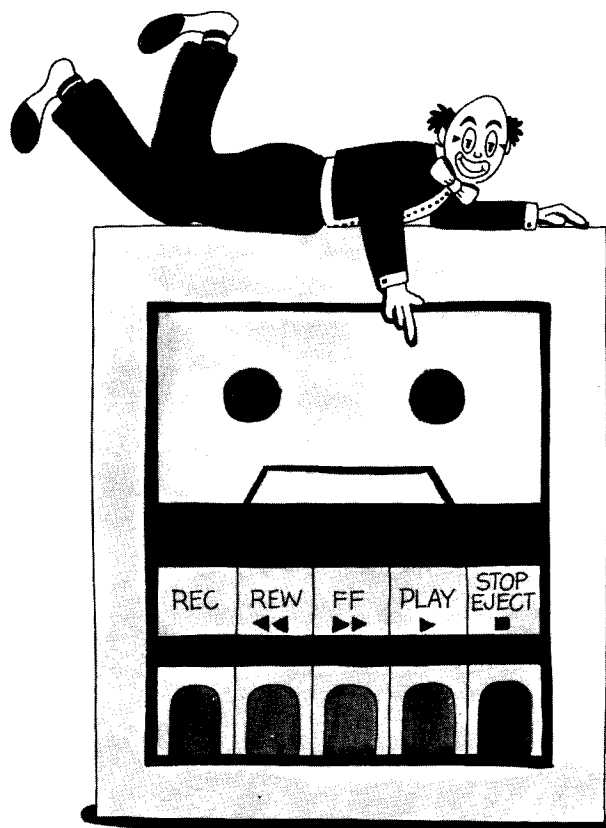
**PLAY** This allows programs to be read from the recorder to the computer.

**REWIND** This winds the tape rapidly backward.

**ADVANCE** This moves the tape rapidly forward.

**STOP/EJ** Pushed once, this button causes the recorder to stop. Pushed a second time the recorder door opens so the cassette can be taken out or put in. On some machines there are two buttons instead of one.

**PAUSE** Some recorders have this button which allows the machine to be stopped for a moment while still in the record or play position.



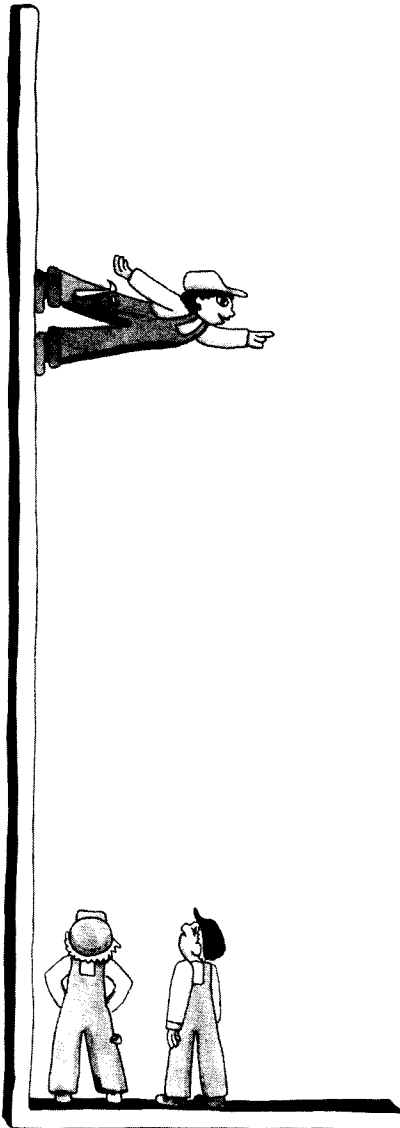
Let's learn to use the cassette recorder with the computer.

1. Write a program on the computer. Let's use the one from the last chapter as an example.

```
10 Print "_____";  
      (Type your favorite graphics in  
      the blank).
```

```
20 GOTO 10
```

2. Push the STOP/EJ button on the recorder and place a blank or unwanted tape in the recorder. (If the tape has previously been used, the new program you save is recorded over the top of the old).
3. Locate the beginning of the tape by pressing the REWIND button. When the tape is completely rewound and stopped, press the STOP/EJ button. (Notice that when you press the button once, the recorder stops. Press twice and the tape ejects.)
4. Type **CSAVE** on the keyboard and press . You will hear two beeps. (The "C" in "CSAVE" stands for cassette.)
5. Push RECORD and PLAY at the same time.
6. Press  on the keyboard. (If the volume on the T.V. is turned up, you should hear various tones as the program is copied.)
7. When the tape stops, the screen will say READY, which means the program has been copied on the cassette tape. (It's a good practice to make a back-up, or second copy of important programs.) Push the STOP button.



You should now have a copy of your program on the tape. Let's load it back into the computer to see. To do that, first get rid of the program that is in memory by typing NEW. Then, clear the screen by holding  and pressing . Now make sure the program is gone by typing RUN. Nothing should happen. Now put the program back in ATARI's memory with the following steps:

1. Put the tape in the recorder.
2. Locate the beginning of the tape by pushing REWIND. After the tape stops, push STOP/EJ.
3. Type **CLOAD** on the keyboard and press . You will hear one beep.
4. Push PLAY on the recorder, and press  on the keyboard.
5. When the recorder stops, the screen will say READY, and the stored program will be in ATARI's memory. Push the STOP/EJ button.

To see if the program has been successfully loaded back into memory, type **LIST**. LIST is a command that tells ATARI to show on the screen any program that is in memory. Now try out the program by typing RUN.

You've now extended your programming capabilities by knowing how to save and load programs with the cassette tape recorder. This skill will become more important as you write longer programs, and as you want to use programs that someone else has made.

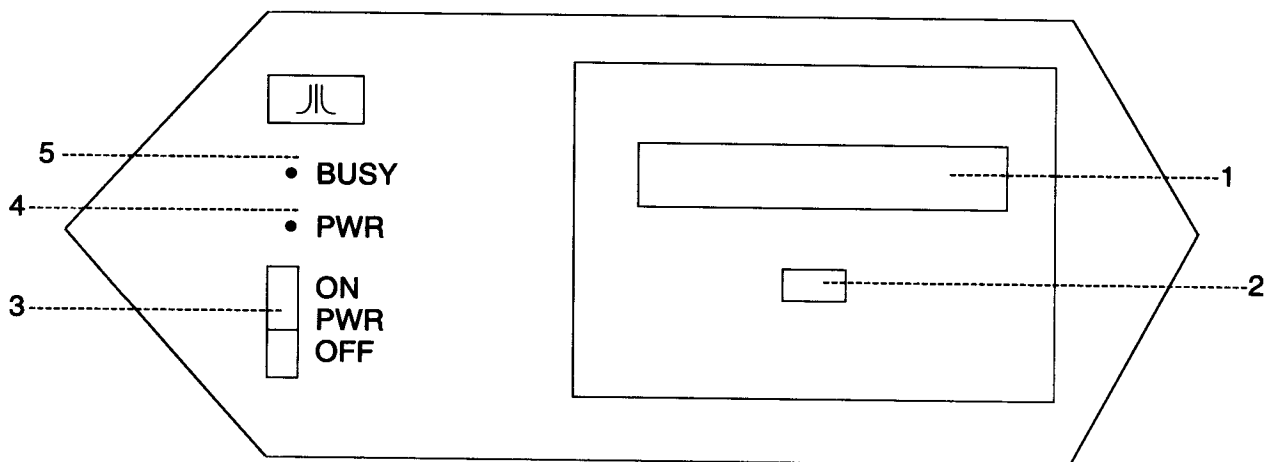


It's easiest to use short tapes and store one program at the beginning of each side. However, you can store several programs on one side of the tape by using the counter on the top of the recorder. Here's how. Start by rewinding the tape completely. Reset the counter to 000, then wind the tape forward to at least 10 counts past the end of the last program on the tape. Always write down the first and last counter numbers of a program so you know where to start the next one. When you want to read back a program, rewind the tape completely, set the counter to 000, then wind it forward to the number of the program you want. Sometimes it's a little difficult to have several programs on one tape, but with a little practice it's workable.

**CAUTION**—the leader (the nonusable beginning of the tape) can be no longer than 9 seconds. This is because ATARI ignores the first 9 seconds of a tape before it begins recording a program. If the leader is too long, ATARI will try to place and read the program on the leader. This will cause an error message.

## Disk Drive

Some ATARI systems have a disk drive rather than a tape recorder. The disk drive is easy to operate and has only a few control buttons and lights as shown below.

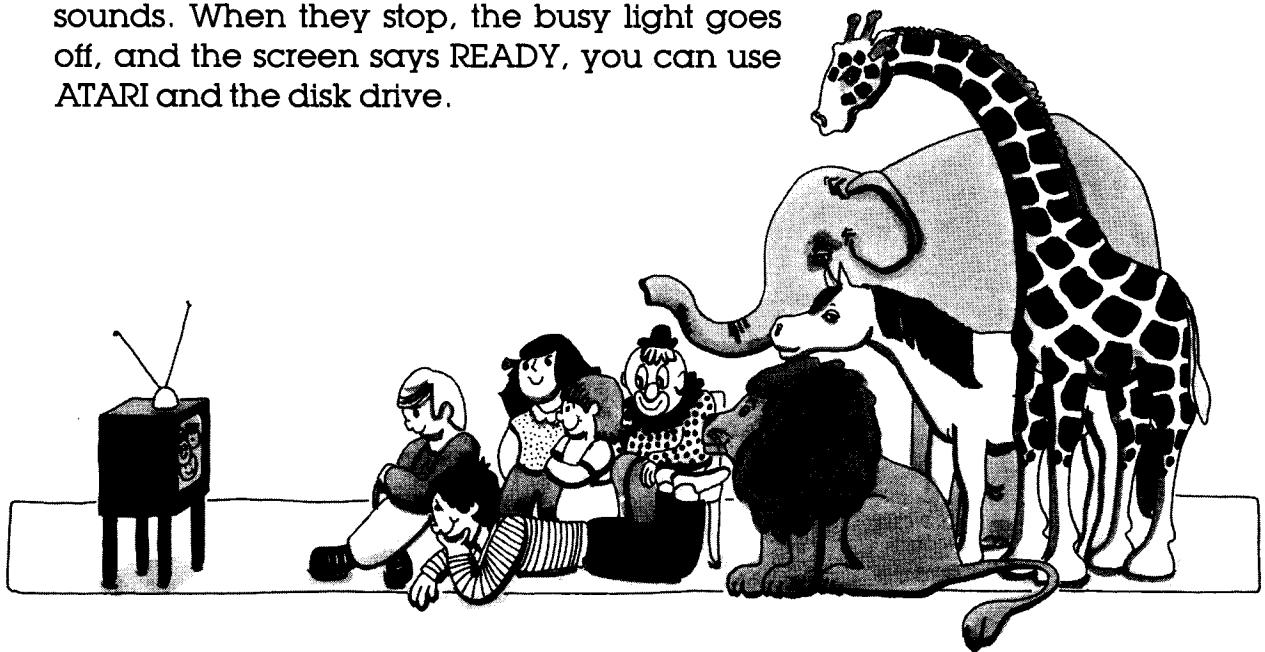


The diskette is loaded through the door (1), which is opened by pushing the rectangular button (2) beneath it. (Some disk drives are opened by pulling on the door rather than pushing a button.) The drive unit is turned on with the power switch (3). When the machine is turned on, the power light (4) will show red. The busy light (5) will show red when the disk drive is doing a job.

NEVER use the disk drive while the busy light shows red. Another important thing—the disk drive must be turned on BEFORE you use the keyboard to write a program.

Here are the steps to follow when using the disk drive:

1. Turn on the television.
2. Turn on the disk drive. Be sure to wait until the busy light goes off. (It will make some sounds at this time.)
3. Open the disk drive door by pressing the rectangular button beneath it.
4. Insert the diskette so that the notched out space is on the left hand side. (Most diskettes have an indicator arrow to show you how to insert them into the drive.) Close the drive door.
5. Turn on the computer. It will make some more sounds. When they stop, the busy light goes off, and the screen says READY, you can use ATARI and the disk drive.



Once ATARI and the disk drive are turned on properly, you can learn the steps for saving a program on a diskette:

1. Using the ATARI keyboard, write your program. Be sure the disk drive is properly turned on before you begin writing the program. Let's use the program from the last chapter as an example:

```
10 Print "_____"; (Type in your  
favorite graphic.)
```

```
20 GOTO 10
```

2. Type **SAVE "D: filename"**. Be sure to use the quotation marks. (The filename can be any name you want, up to eight letters and numbers, but must begin with a letter.)
3. Press . The disk drive will make a sound as the program is moved from ATARI's memory to storage on a disk. Wait until the sound stops, and the red busy light goes off.

It's just as easy to reload the program from the disk into ATARI's memory. After the disk drive and computer are properly turned on, type **LOAD "D: filename"**. The busy light will show red and the disk drive will make a sound while the program is being loaded back into ATARI. When the screen reads **READY**, and **AFTER** the busy light goes off, you can work with the program. (Be sure to write down the names of your files so that you can use them later. If you forget the name, your teacher can refer to the Disk Drive Manual for information on how to use the Disk Operating System (DOS) and find your file name again.)

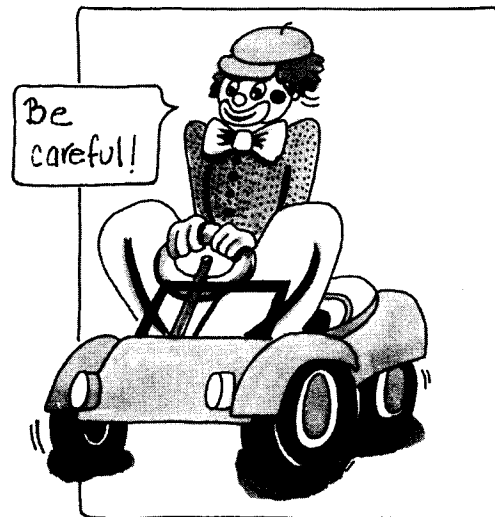




---

**CAUTION!** Be careful when handling the disks. Never touch exposed surfaces, or hold a disk with a finger through the middle hole. Hold it only by its black sealed envelope. Put the disk back in its storage envelope when it is not being used. Also, keep it away from magnetic sources and excessive heat. When writing on a diskette label, do it gently with a felt tip pen.

**to do:** Using a program from your teacher, or a simple one you have written, practice loading and saving with the cassette tape recorder or disk drive system.



# CHAPTER 9

## Teaching ATARI to Do Your Homework

Have you ever dreamed of having a computer that could do your homework for you? It is possible! You can teach ATARI to do your math assignments for you. ATARI can figure out math and give the correct answers about a billion times faster than you can!

Here are six kinds of arithmetic that ATARI can do for you:

	Our Symbol	ATARI's Symbol	Example
1. addition	+	+	2+2
2. subtraction	-	-	4-2
3. multiplication	×	*	2*2
4. division	÷	/	4 / 2
5. powers	10 <sup>2</sup>	^	10^2
6. square root	$\sqrt{25}$	SQR( )	SQR(25)

When ATARI writes zero it uses a 0 with a / through it, like this: 0. This way, ATARI won't get O (oh) and 0 (zero) mixed up.

Computers are not perfect! They can make mistakes! Most computers have certain strong points, and certain weak points. One weakness with older ATARI models, is that when you use powers the computer does not give an exact answer. For example, ATARI prints  $2 \wedge 2 = 3.9999998$ , when the answer really is 4. The new X-L models have been updated to give the correct answer of  $2 \wedge 2 = 4$ .

# CHAPTER 10

## ATARI as a Calculator

You can use ATARI like a calculator to do the six kinds of arithmetic we just learned about. This is easy to do. Instead of typing PRINT, you type a question mark (?) for a shortcut.

1. To add  $678+789$ , you type: ? (the same as PRINT)  $678+789$  .

ATARI will print the answer on the next line.

2. To multiply  $5*50*500$  type: ?  $5*50*500$  .
3. To subtract  $800-5-2$  you type: ?  $800-5-2$  .
4. To find the square root of 49 you type: ?  $SQR(49)$  .

When you use ATARI as a calculator, you are using **DIRECT** or **IMMEDIATE MODE** programming. ATARI gives you the answer *immediately* after you press RETURN.



# CHAPTER 11

## Arithmetic with Many Numbers

ATARI does arithmetic in a certain order. Let's look at an equation with many numbers to see what that order is. (It is OK to put more than one arithmetic operation in the same equation.)

**Equation:**  $5*(6-2)+9/3\wedge 2$

**What ATARI does**

1. PARENTHESES are done first.

$$6-2=4$$

2. POWERS are done next.

$$3\wedge 2=9$$

(As mentioned in Chapter 9, older ATARI models have a limitation when doing powers, and do not give an exact answer.)

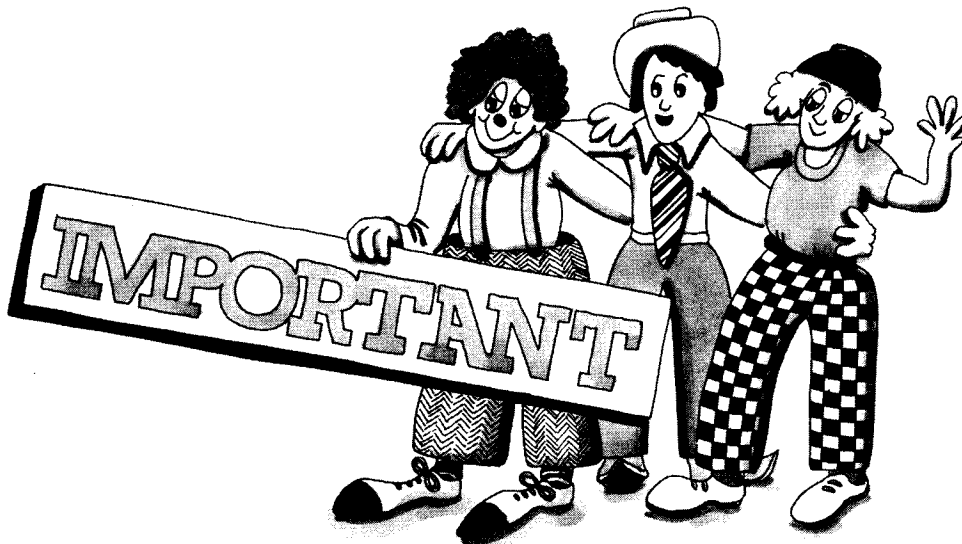
3. MULTIPLICATION and DIVISION are done together. (The numbers at the left are done first, then the numbers to the right.)

$$5*4=20+9/9$$

4. ADDITION and SUBTRACTION are done last. They are also done together. (The numbers at the left are done first, then the numbers to the right.)

$$20+1=21$$

NOTE: Any square roots are done third with multiplication and division.



### Important

If there is more than one thing done at the same time, ATARI always does the thing at the left *first*.

For example, in the equation  $2+3+4$

ATARI adds  $2+3$  first ( $2+3=5$ )

ATARI then adds  $5+4$  ( $5+4=9$ )

Example:

$$5*(6-2)+9/3\wedge 2$$

PARENTHESES are done first.

first

$$5* 4 +9/3\wedge 2$$

POWERS are done next,

second

$$5* 4 +9/9$$

then MULTIPLICATION and DIVISION.  
(left to right)

third

fourth

$$20 + 1$$

ADDITION (and SUBTRACTION) is last.

last

answer = 21

### Another Shortcut . . .

Here is a fast way to get the answers to many short equations. You want answers to:

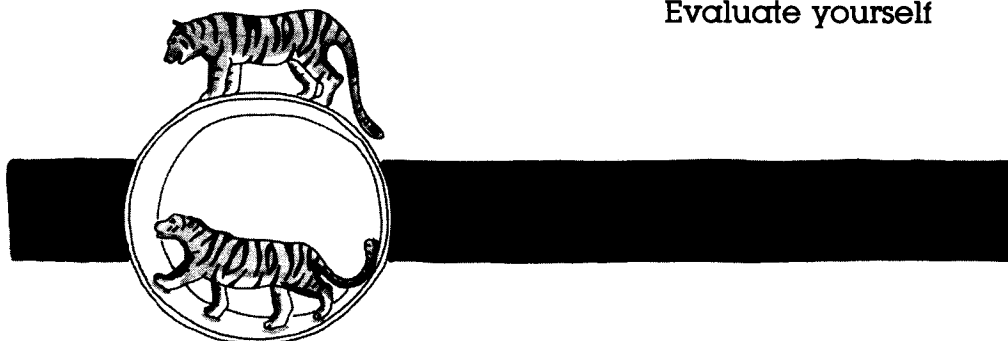
1.  $99*66$
2.  $74+47$
3.  $89-78$
4.  $4\wedge 2$

You type: ? 99\*66 , 74+47 , 89-78 , 4^2

**IMPORTANT!** Use commas to separate each equation.

When using one ? statement, you may type as many equations as will fit on three lines of ATARI's screen.

**to do:** Programmer's Pastime #1, #2, #3, #4  
Component 2 Fun Page  
Evaluate yourself



# PROGRAMMER'S PASTIME #2

What order does ATARI follow in doing arithmetic for equations with many numbers?

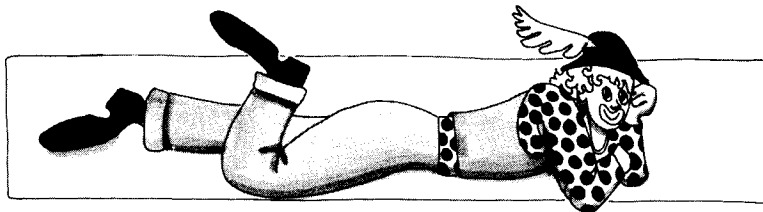
\_\_\_\_\_ are done first.  
\_\_\_\_\_ are done second.  
\_\_\_\_\_ and \_\_\_\_\_ are done third (left to right).  
\_\_\_\_\_ and \_\_\_\_\_ are done last (left to right).

Use your mental powers and write the answer that ATARI would give for these equations. Remember to do the arithmetic in the same order that ATARI would!

1.  $2*3+1$  \_\_\_\_\_
2.  $2+8*2*1$  \_\_\_\_\_
3.  $3*3+9+20$  \_\_\_\_\_
4.  $11+4*3$  \_\_\_\_\_
5.  $22+8+12*1$  \_\_\_\_\_

Try some more:

1.  $8/2-3$  \_\_\_\_\_
2.  $20/4+6-5$  \_\_\_\_\_
3.  $30-10/2$  \_\_\_\_\_
4.  $50-20/10+3$  \_\_\_\_\_
5.  $16/2-8/2$  \_\_\_\_\_
6.  $14/2+4/2-6$  \_\_\_\_\_



# PROGRAMMER'S PASTIME #4

Cowboy Clyde typed in the following equations, but ATARI wouldn't give him answers. Do you know why?

Find out what's wrong with the way his equations are typed. Write the correct way in the blanks.

1. ?  $62+4\times 20$

\_\_\_\_\_

2.  $23/4\times 6$

\_\_\_\_\_

3. ?  $\text{SQR } 16$

\_\_\_\_\_

4.  $80/4+2\times 3*\text{SQR } 25$

\_\_\_\_\_



## CHALLENGE

If there's some arithmetic in a long equation that you want to be done *first*, put parentheses around it. (ATARI always does what's in parentheses first.) Let's say you want  $4+3*2$  to equal 14. If you type:  $4+3*2$  ATARI will give you 10 because multiplication is done before addition.

$$4+3*2$$

first

$$4+6$$

$$10$$

If you want  $4+3*2$  to equal 14, you must use parentheses like this:  $(4+3)*2$ .

first

$$7 * 2 = 14$$

Rewrite each equation below and put parentheses around what ATARI should do first in order to make the equation TRUE. (HINT: You might have to use *two* sets of parentheses for some equations.)

Example:  $7*3+2=35$

$7*(3+2)=35$

1.  $9/2+1=3$
2.  $6*2+2=24$
3.  $3\wedge3-1=9$
4.  $4+2-1*5=9$
5.  $12-3+6/3=9$
6.  $20-10\wedge4+2=10002$
7.  $12/4+2=2$
8.  $9-5\wedge2=16$
9.  $50+10/18+10+2=2$
10.  $10+4-7\wedge2*1+3-3=193$

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_



# COMPONENT 3

## CHAPTER 12

What Else Can ATARI Do for Me? 46

## CHAPTER 13

Flow Diagramming 47

## CHAPTER 14

More About Flow Charts 53

## CHAPTER 15

Double Detours 55

## CHAPTER 16

Looping 59

## CHAPTER 17

Putting It All Together 61

## CHAPTER 18

Printing Whole Equations 67

## CHAPTER 19

A Different Way 69

# CHAPTER 12

## What Else Can ATARI Do for Me?

In Chapter 10 you learned how to teach ATARI to do your math. This is fine as long as your homework is simple arithmetic, but you may be asking, "What else can ATARI do for me?"

Computers deal mainly with numbers. They were invented to do long and tedious arithmetic for people in less than a second. In this way, computers have saved people hours of time.

Computers can help us in other ways as well. Computers can, for example:

1. compare numbers and letters.      Is 97 bigger than 98?  
Does X come before Y in the alphabet?
2. make a decision and then do the right task.      If X = "YES" then PRINT "HELLO"

Computers can also learn to do creative things. You can teach ATARI to:

1. draw a picture;
2. make a game;
3. play a song.

The tricks that you will teach ATARI to do may be very short and simple or very long and complicated. You must first learn how to write the instructions (the PROGRAM) that tell ATARI what to do. You will need to learn how to do some more PROGRAMMING in the BASIC language.



# CHAPTER 13

## Flow Diagramming

When you learn how to play a new game, you must read a set of instructions. These instructions are written in a clear and orderly step-by-step manner. If the instructions are mixed up and out of order, you won't understand how to play the game.

The same is true for ATARI. When you write a program to teach ATARI a trick or to solve a problem, the instructions in your program must be in a clear, step-by-step order. If you don't plan your steps carefully, ATARI will not understand your program.

There is a process you can use when you write a computer program that will help you write your steps clearly and in the correct order. This process is called **FLOW DIAGRAMMING**.

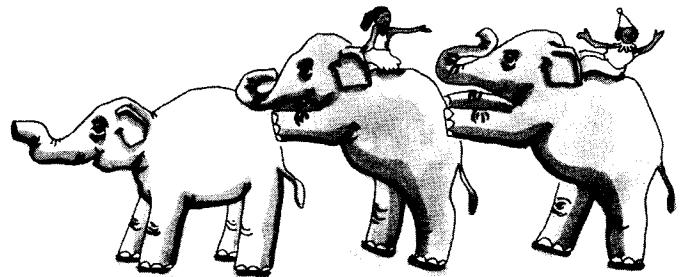
An **ALGORITHM** (al' gor ith m) is a step-by-step method you use to solve a problem. Every problem has a certain algorithm that you can use to solve it. For example:

### Problem

Your front door is locked.

### Algorithm

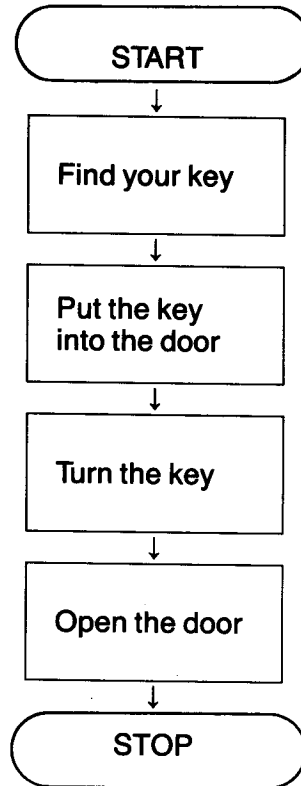
1. Find your key.
2. Put the key into the door.
3. Turn the key.
4. Open the door.





By following the algorithm, you can solve the problem of being locked out of your house.

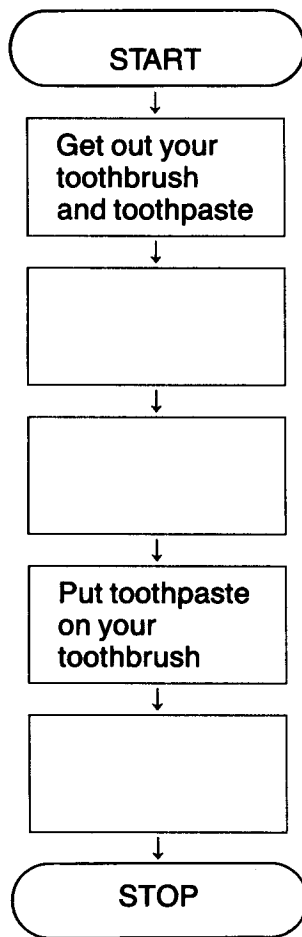
When we do flow diagramming, we show how our algorithms work by putting them into **FLOW CHART** form. Here is how we would write our algorithm in a flow chart.



A flow chart is a diagram which shows all of the steps of an algorithm in the correct order. The arrows in a flow chart show how the steps are connected.



Below is a flow chart that shows an algorithm of how to brush your teeth. Think about which steps at the side of the flow chart would fit in the blank boxes.



### Missing Steps

Brush your teeth  
Wet your brush  
Unscrew toothpaste cap



---

Notice that the boxes in a flow chart have different shapes. What shape are the START and STOP boxes? We usually begin a flow chart with a (START) instruction and end with a (STOP) instruction.

The boxes that tell you to do something are shaped like rectangles. They are called **PROCESSING BOXES**.

**SHAPE**



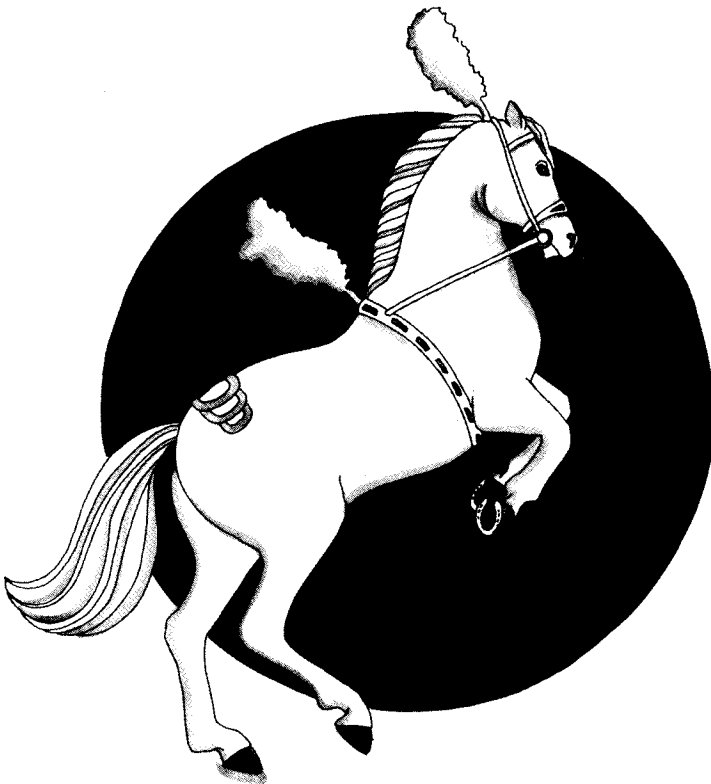
START and STOP box



PROCESSING box

Let's practice writing algorithms and putting them into flow chart form.

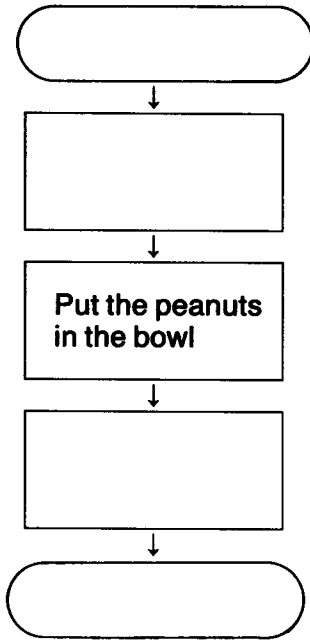
**to do:** Programmer's Pastime #5, #6, #7



# PROGRAMMER'S PASTIME #5

For each flow chart, fill in the blank boxes with the step you think would fit. Make sure your steps are in the right order.

## ALGORITHM / FLOW CHART #1 HOW TO FEED YOUR PET ELEPHANT



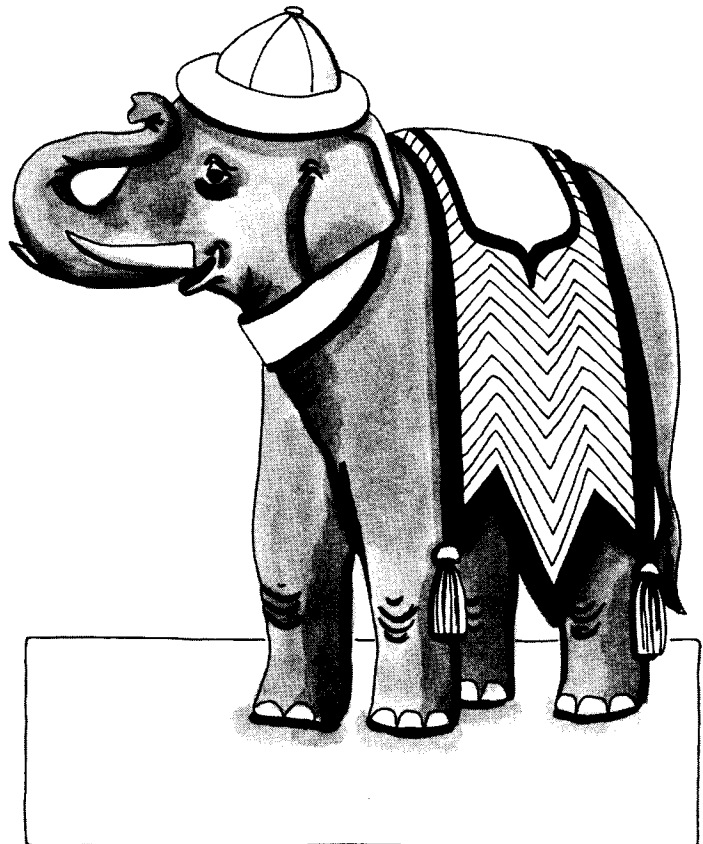
### Missing Steps

STOP

Call your pet elephant

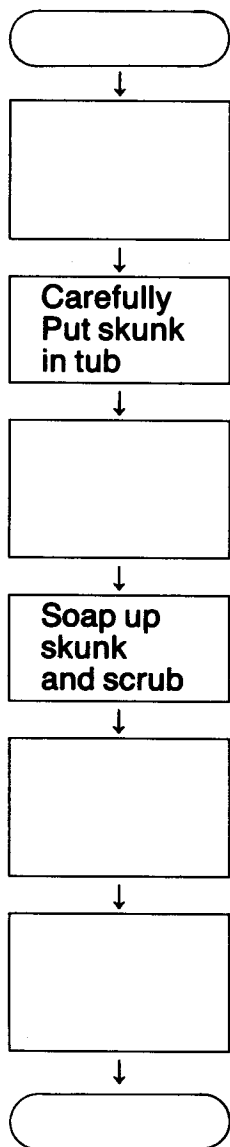
START

Get out the peanuts



---

ALGORITHM / FLOW CHART #2  
HOW TO WASH YOUR PET SKUNK



**Missing Steps**

Get skunk wet  
STOP  
Rinse skunk  
Fill tub with warm water  
START  
Dry skunk



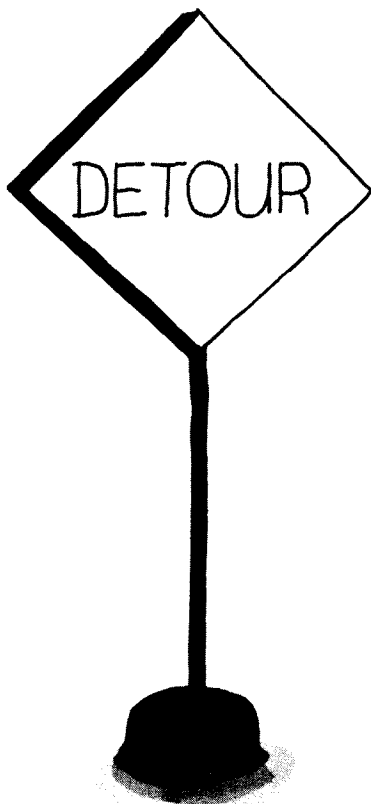
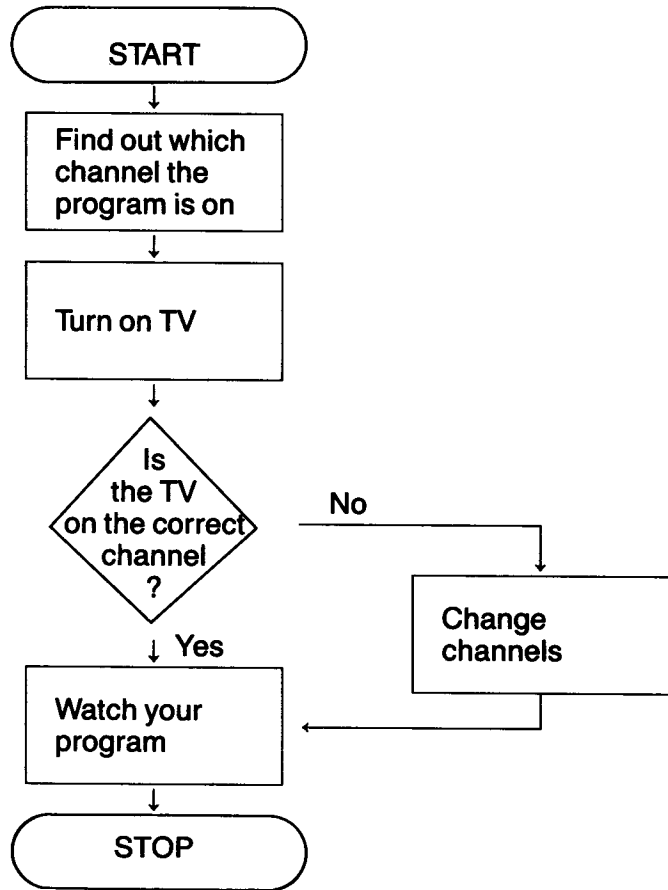
# CHAPTER 14

## More About Flow Charts

Sometimes we will have a step in our algorithm that asks a question. In a flow chart, a question is written in a diamond-shaped box. This is called a **DECISION** box.

Example:

### ALGORITHM / FLOW CHART ON HOW TO WATCH A TV PROGRAM



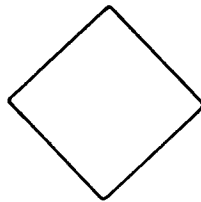
---

In this flow chart, we have to make a decision. The DECISION box asks us a question: "Is the T.V. on the correct channel?"

If the answer is YES, we will stay on the main path of the flow chart. If the answer is NO, we will take a detour in our flow chart and follow a different path. During our detour, we must do a task—change the channel. Notice how the detour comes back to the main part of our flow chart before we stop.

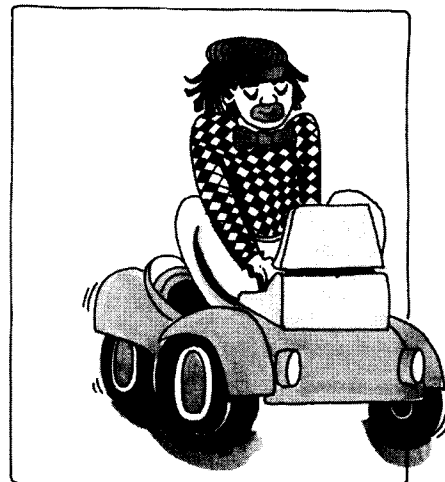
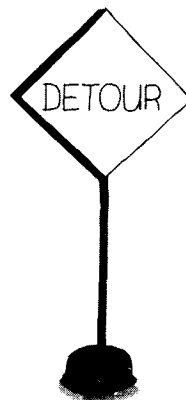
When there is one detour from a decision box in a flow chart, the flow chart is said to have a **SINGLE-ALTERNATIVE DECISION STEP**.

**SHAPE**



DECISION box

**to do:** Programmer's Pastime #8, #9, #10

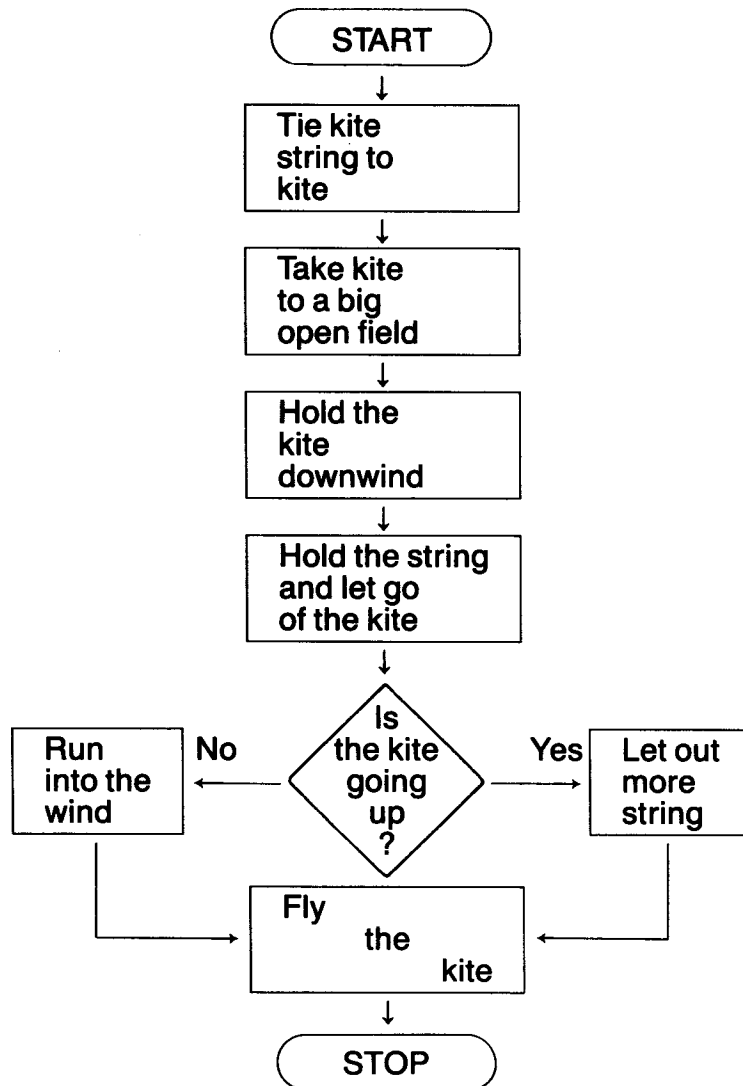


# CHAPTER 15

## Double Detours

Sometimes a flow chart will have a decision box that has a detour for both the YES and NO answers. If the answer is YES, we do a certain task. If the answer is NO, we do a different task.

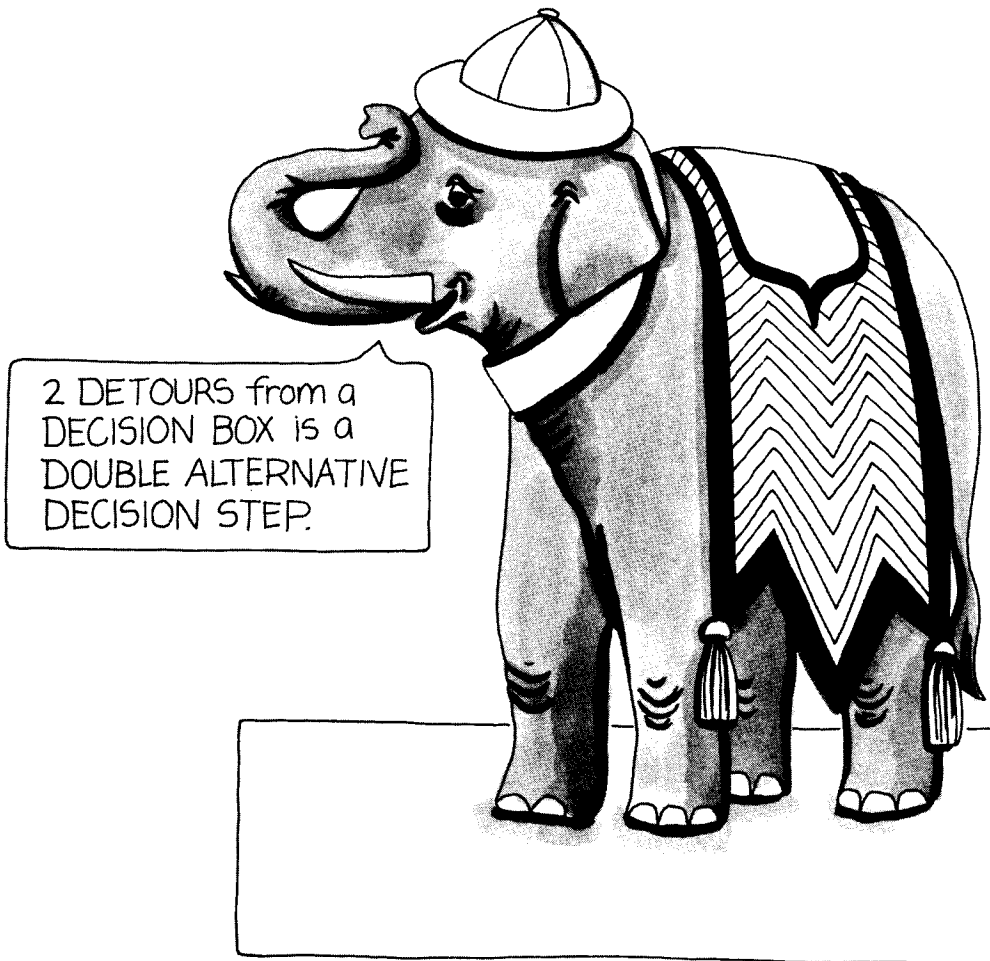
### ALGORITHM / FLOW CHART ON HOW TO FLY A KITE



---

Our flow chart asks the question: "Is the kite going up in the air?" If the answer is YES, we take a detour that tells us to "Let out more string." If the answer is NO, we take a different detour that tells us to "Run into the wind."

Whenever there are two detours from a decision box in a flow chart, the flow chart is said to have a **DOUBLE-ALTERNATIVE DECISION STEP**.

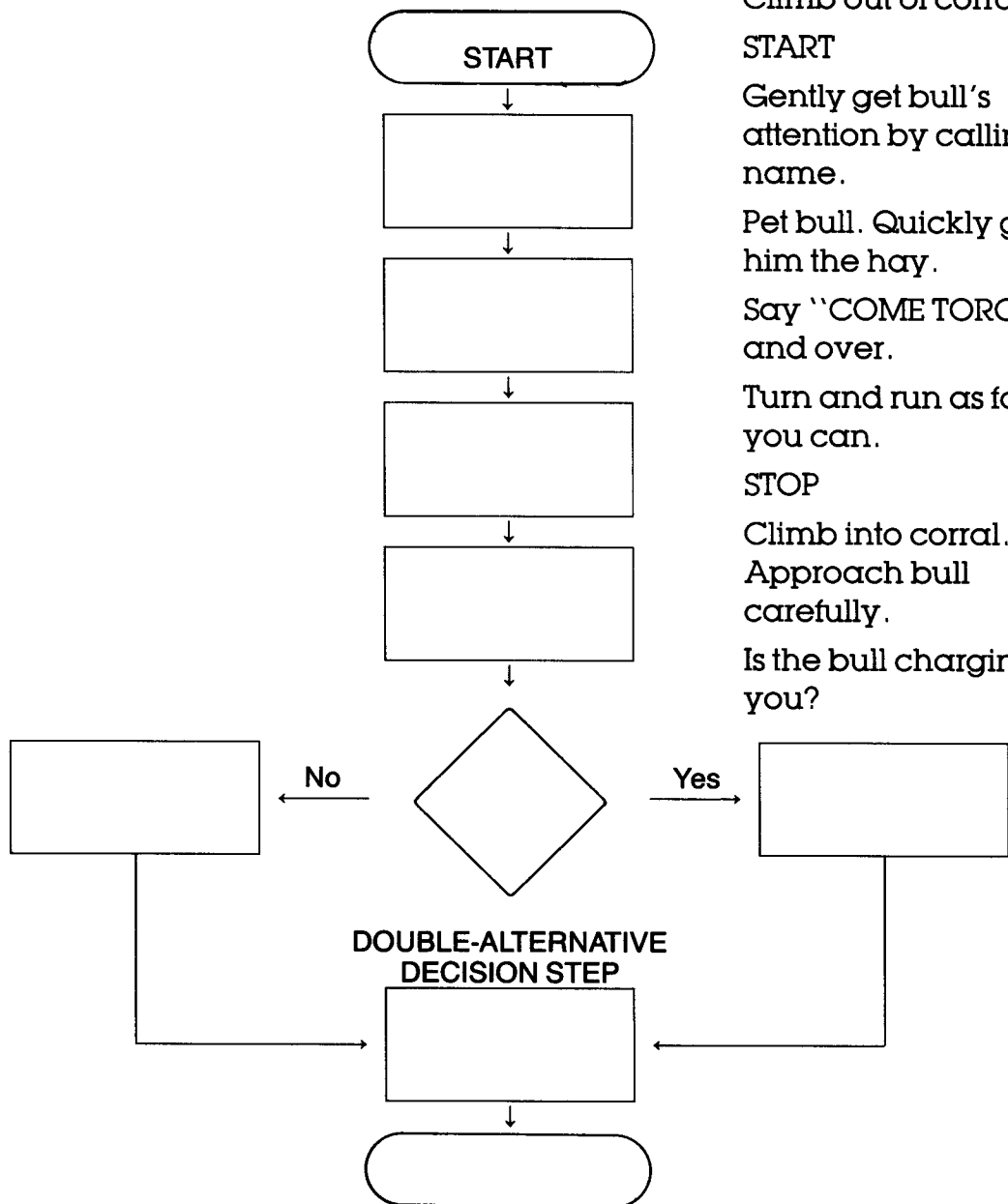


**to do:** Programmer's Pastime #11, #12

# PROGRAMMER'S PASTIME #11

For each flow chart, fill in the blank boxes with the step you think would fit. Make sure your steps are in the right order.

ALGORITHM / FLOW CHART  
HOW TO TEACH YOUR PET BULL TO COME WHEN  
YOU CALL



## Missing Steps

Hold out handful of hay.

Climb out of corral.

START

Gently get bull's attention by calling his name.

Pet bull. Quickly give him the hay.

Say "COME TORO" over and over.

Turn and run as fast as you can.

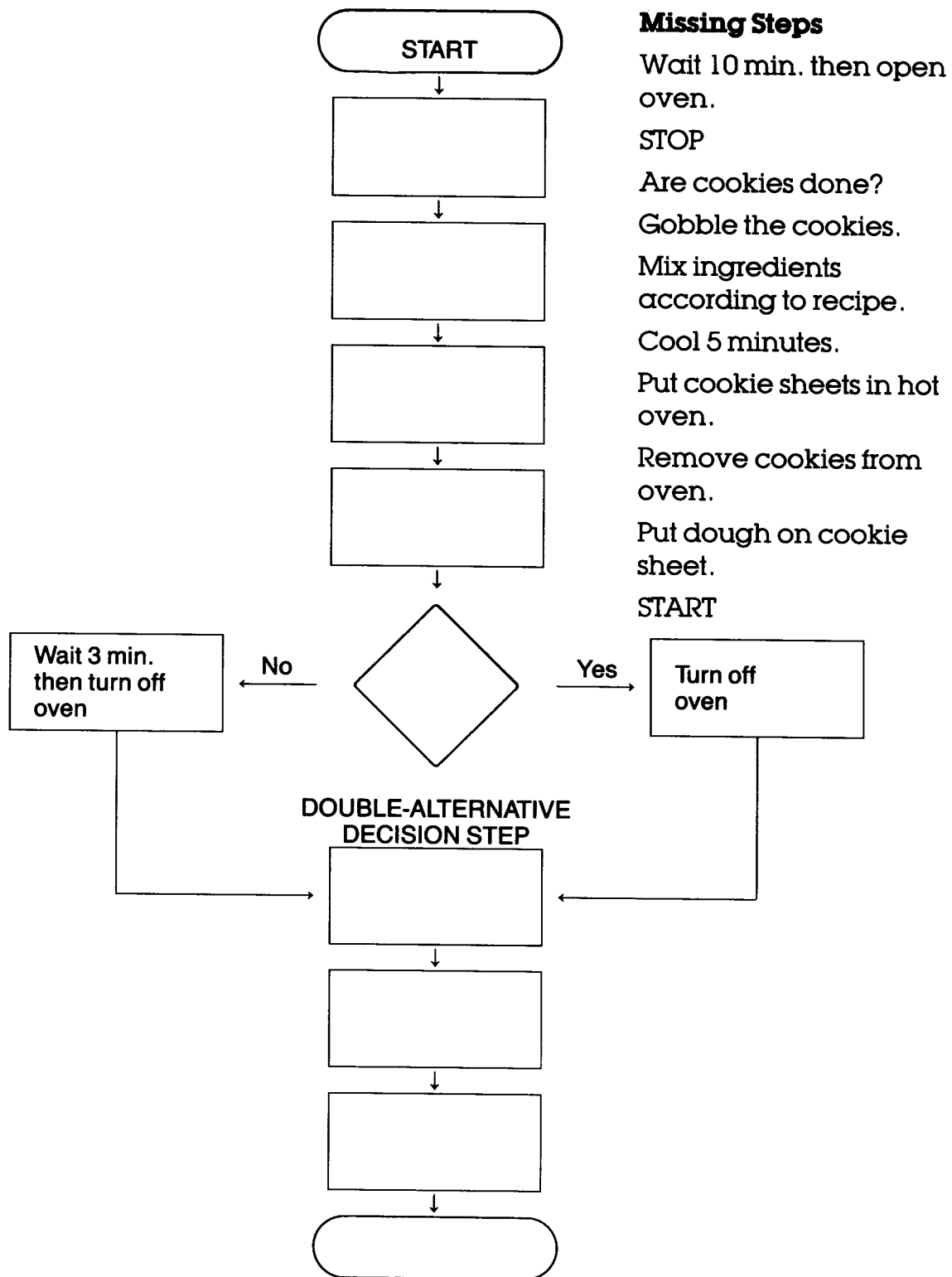
STOP

Climb into corral.

Approach bull carefully.

Is the bull charging at you?

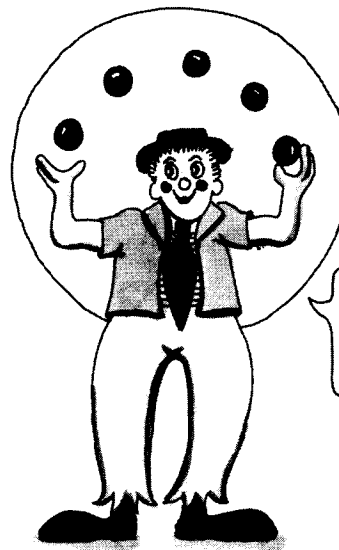
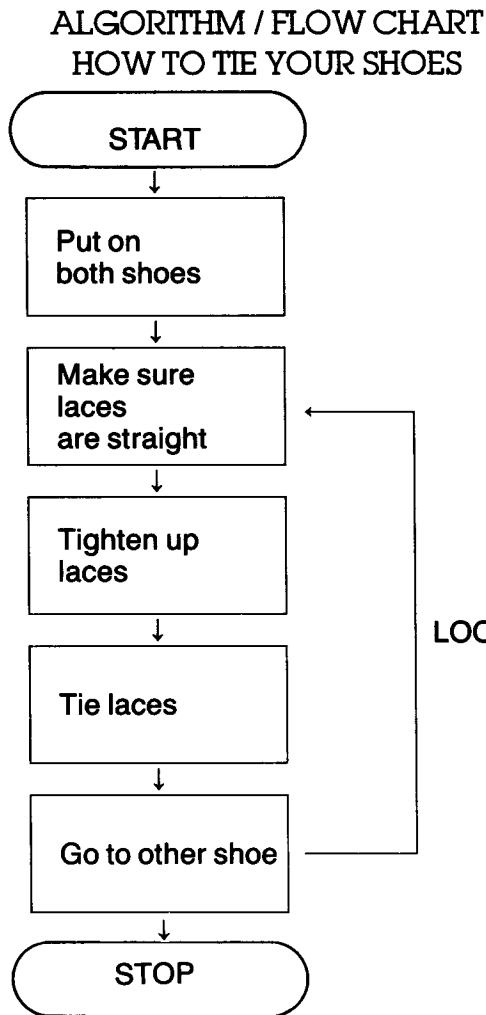
ALGORITHM / FLOW CHART  
HOW TO BAKE COOKIES



# CHAPTER 16

## Loop de Loop

Sometimes we will use an algorithm that repeats a certain step over and over. When we write the flow chart for the algorithm we use a **LOOP** arrow to show that the step is repeated.

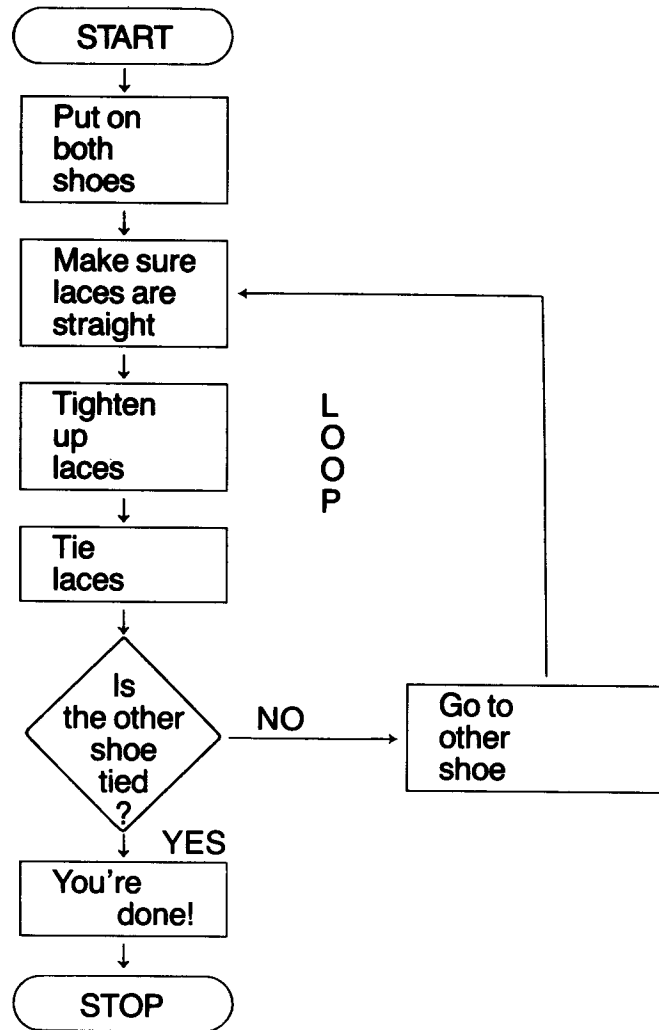


Using a loop helps do the same thing over and over.

After we have tied one shoe, we are told to go to the other shoe. Then the loop arrow takes us back up to the second step. Now we repeat the steps as we tie the other shoe.

**LOOPING** is handy because it helps us to keep our flow chart short. Imagine how long this flow chart would be if we didn't use a loop.

Looping also works nicely with a decision step. We can improve our flow chart by using a single-alternative decision step.



We go through the flow chart twice. Once to do one shoe, and again to do the other shoe. The first time through, the answer is NO and we follow the loop detour. The second time through the answer is YES, and we are done.

**to do:** Programmer's Pastime #13, #14



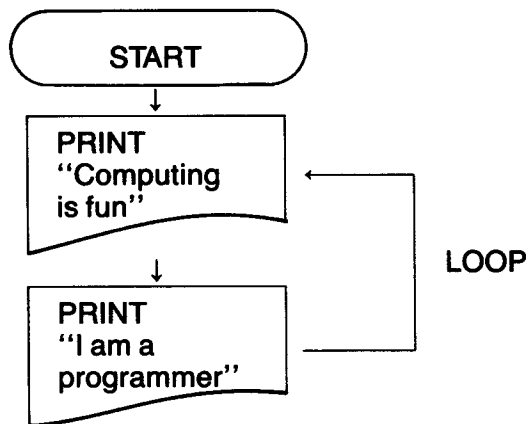
# CHAPTER 17 Putting it all Together


Now that you know how to change an algorithm into a flow chart, let's learn how to change a flow chart into a program that ATARI can understand.

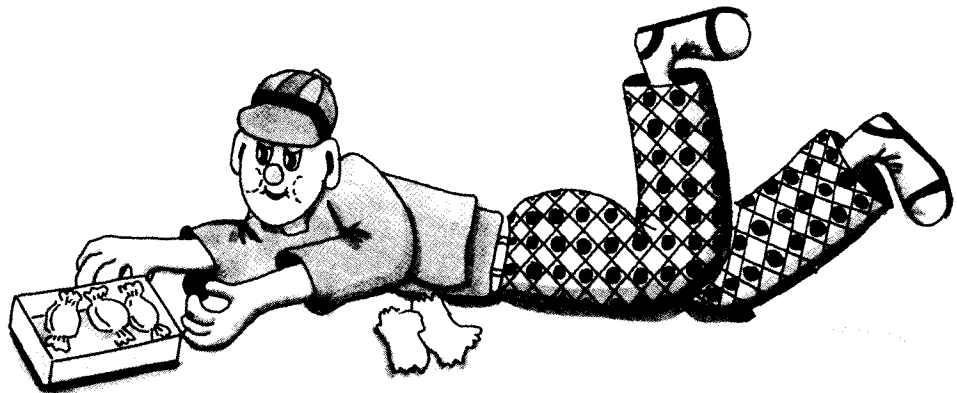
Let's tell ATARI to print over and over:

COMPUTING IS FUN  
I AM A PROGRAMMER

The algorithm and flow chart will look like this:



Because we want ATARI to print something over and over again, we will need to use a loop. Notice that this flow chart never stops. The loop goes on forever. (Also notice that a  box is used in a flow chart to mean PRINT.)



Here is how you program the algorithm/flow chart in BASIC:

```
10 ? "COMPUTING IS FUN"  
20 ? "I AM A PROGRAMMER"  
30 GOTO 10
```

(Remember, the question mark (?) is a quick way to write PRINT.)

```
COMPUTING IS FUN  
I AM A PROGRAMMER  
COMPUTING IS FUN  
I AM A PROGRAMMER  
COMPUTING IS FUN  
I AM A PROGRAMMER  
.  
.  
.
```

Notice how the loop from the flow chart is represented in the BASIC program. Line 30 is where the loop happens. The command to loop is GOTO. After the word GOTO is the number of the line that you want ATARI to go back to.

Thus, we have programmed ATARI to print the information on lines 10 and 20. Then on line 30, we have told ATARI to go back to (GOTO) line 10 and start over again. (Notice that we did not type an END command. By including a loop in our program we told ATARI to print the same thing over and over *without* end. Therefore we did not need an END command.)

ATARI operates in two different ways, or **MODES**. In Chapter 10, we learned about the direct or immediate mode. If we want ATARI to print "Computing is fun" in the immediate mode, we type: ? "COMPUTING IS FUN"  . When we press  , ATARI immediately does what we told it to do—it prints "COMPUTING IS FUN."



? "COMPUTING IS FUN"

COMPUTING IS FUN

READY



When you type line numbers, this tells ATARI you are making a program. ATARI will know to use the **PROGRAM MODE**. When you type a PRINT statement with a line number, nothing happens when you press RETURN.

10 ? "COMPUTING IS FUN"



Nothing happens because ATARI knows to switch to the program mode. In the program mode, you must type RUN to make ATARI do the program.

10 ? "COMPUTING IS FUN"

RUN

COMPUTING IS FUN

READY




Whenever you print a line number in front of a statement, ATARI takes that statement and stores it in memory.



---



Every time you type RUN, ATARI will remember to type the statement, "Computing is fun" because it is stored in ATARI's memory.

```
10 ? "COMPUTING IS FUN"  
RUN  
COMPUTING IS FUN  
  
READY  
RUN  
COMPUTING IS FUN  
  
READY  
RUN  
COMPUTING IS FUN  
  
READY  

```

This program stays in ATARI's memory until you clear the memory by typing NEW, or turn off ATARI.

# REMEMBER

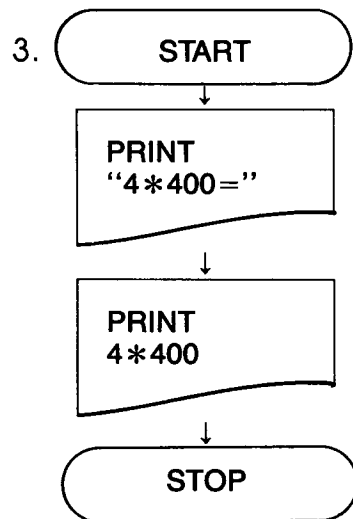
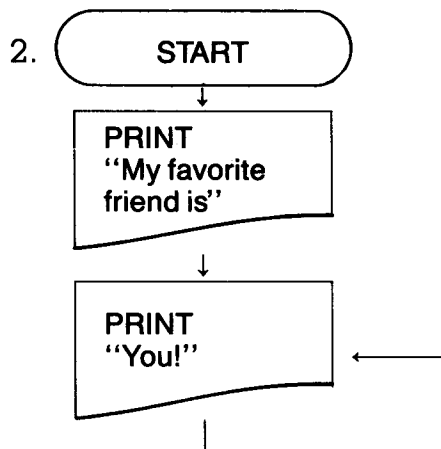
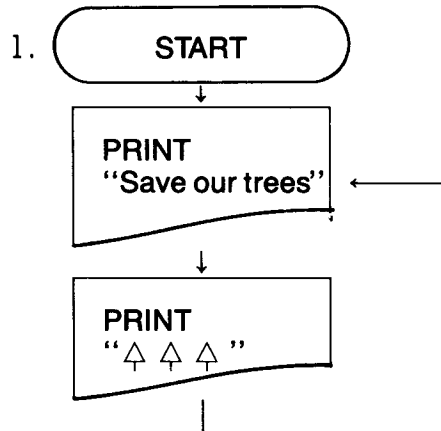
## Remember

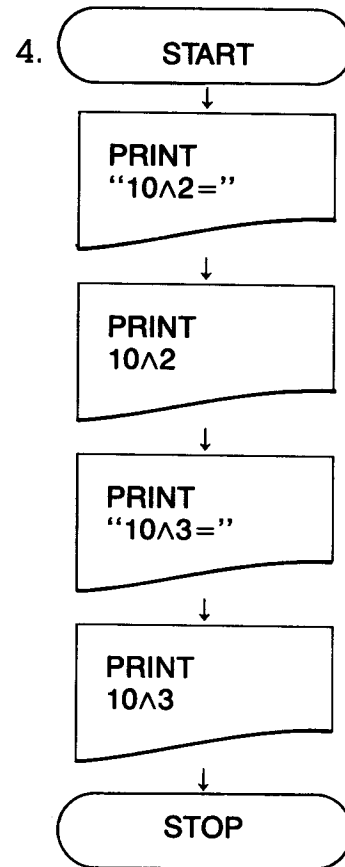
1. Begin every line in a program with a line number.
2. ? tells ATARI to PRINT.
3. Put " " around what you want ATARI to say.
4. GOTO tells ATARI to loop to a certain line.
5. Type RUN to see ATARI do the program.
6. Press  when you want the program to end.
7.  is used in a flow chart to mean PRINT.

**to do:** Programmer's Pastime #15, #16

# PROGRAMMER'S PASTIME #15

For each flow chart, write a BASIC program in the space below. (HINT—Pressing `CTRL` and `:` will give a tree-like graphic.)





# CHAPTER 18

## Printing Whole Equations

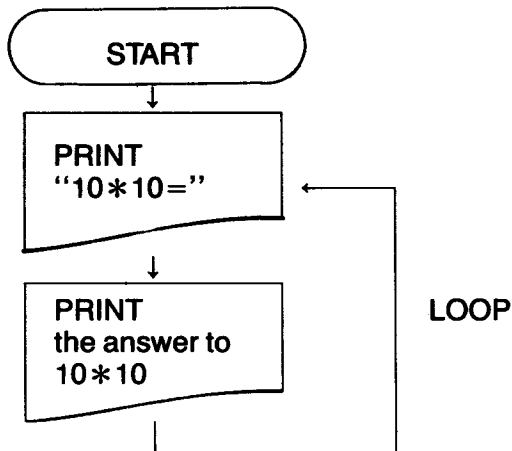
ATARI can give us answers to equations. ATARI can also print the whole equation AND give us the answer. This simple trick is done by using quotation marks (` `').

Let's write a program that tells ATARI to print:

$10 * 10 = 100$

over and over. Since we want ATARI to do it more than once we will need to use a loop.

### Flow chart

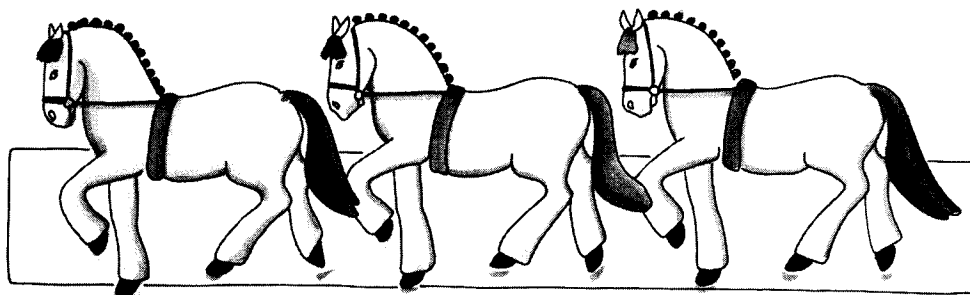


### Program

```
10 ? ``10*10=``  
20 ? 10*10  
30 GOTO 10
```

Line 10 tells ATARI to print the equation.  
Line 20 tells ATARI to print the answer.  
Line 30 tells ATARI to loop to line 10 and repeat the steps.

```
10*10=  
100  
10*10  
100  
.  
.  
.
```

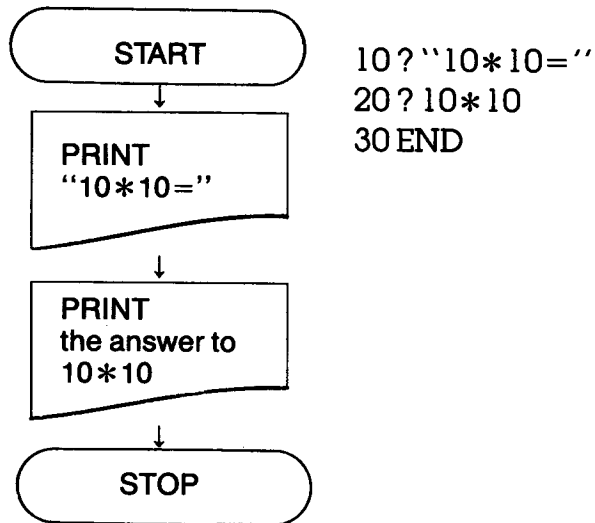


If we want ATARI to do the problem only once, we use the **STOP** instead of the loop for our flow chart.

We use the **END** statement instead of the **GOTO** statement for our program.

Here is what the algorithm/flow chart and program would look like if we wanted

$10 * 10 =$  to be printed only once:  
100



### Important!

1. When you use " " ATARI will print what's inside.
2. When you print the equation without " " ATARI will print the answer.

**to do:** Programmer's Pastime #17



# CHAPTER 19

## A Different Way

In the last chapter you learned how to program ATARI to print a whole equation.

### For this program

```
10 ? "60*6="
20 ? 60*6
30 END
```

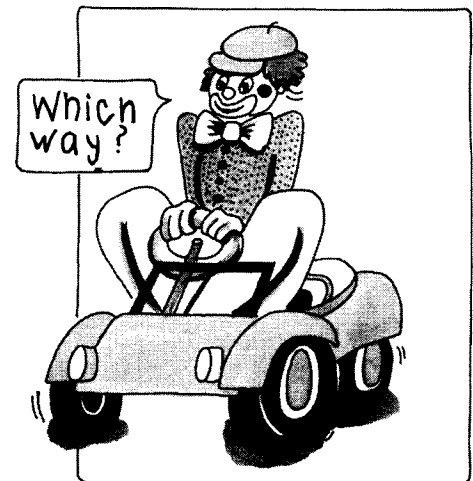
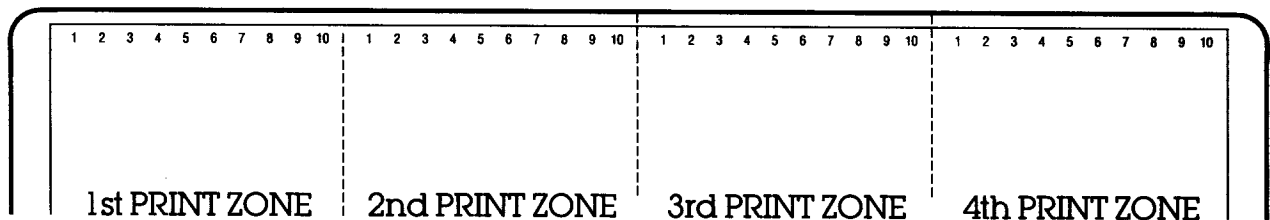
### ATARI would print

```
60*6=
360
READY
□
```

Our equation appears on two separate lines on ATARI's screen. Why? Because we have two PRINT statements in our program (line 10 and line 20). Each PRINT statement tells ATARI to print on a new line. This is why **60\*6=** is on one line and **360** is on the next line.

Can we make ATARI print the equation altogether on one line? Yes! For this simple trick, we will use either a **COMMA (,)** or a **SEMI-COLON (;)**.

ATARI's screen is divided into **PRINT ZONES** or **FIELDS**. Each print zone can hold 10 characters.



If we use a COMMA in our equation, the answer will be printed in the next print zone.

**For this program**

```
10 ? ``60*6=`` , 60*6  
20 END
```

**ATARI would print**

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
6	0	*	6	=						3	6	0							

(1st Print Zone) (2nd Print Zone)

60\*6= was printed in the first print zone because it is at the beginning of the PRINT statement. The answer, 360, was printed in the second print zone because of the comma before it.

A comma tells ATARI to go to the next print zone and then begin printing. (Two commas tell ATARI to move over two print zones.)

This comma method, however, makes our equation look too spaced out. Let's learn a different method that uses a semi-colon.

If we use a semi-colon in our equation, the answer will be printed in the next space.

**For this program**

```
10 ? ``60*6=``; 60*6  
20 END
```

**ATARI would print**

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
6	0	*	6	=		3	6	0											

(1st Print Zone) (2nd Print Zone)

Notice that space 6 in the first print zone is used by the 3. If a space is wanted between the = sign and the following number, a blank space must be left inside the `` `` marks. For example: 10 ? ``60\*6= ` ` ``. The ` symbol is used to represent a blank space.

**For this program**

```
10 ? ``60*6= ` ` `` , 60*6  
20 END
```

**ATARI would print**

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
6	0	*	6	=			3	6	0										

(1st Print Zone) (2nd Print Zone)

Using a comma or semi-colon is good because:

1. It makes our equation get printed on one line.
2. It makes our program shorter.

A PRINT statement with nothing after it leaves a blank line.

### For this program

```
10 ? ``4*4*2=``; 4*4*2
20 ?
30 ? ``4+4+2=``; 4+4+2
40 END
```

### ATARI would print

1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
4	*	4	*	2	=	3	2												
4	+	4	+	2	=					1	0								

(1st Print Zone)

(2nd Print Zone)

# REMEMBER

## Please Remember

1. A COMMA tells ATARI to go to the next PRINT ZONE and then begin printing.
2. A SEMI-COLON holds the cursor at the end of the last thing printed. Then it prints after the next PRINT command.

**to do:** Programmer's Pastime #18  
Component 3 Fun Page  
Evaluate yourself





# COMPONENT 4

## CHAPTER 20

ATARI's Memory 74

## CHAPTER 21

Using Variables 76

## CHAPTER 22

Using Variables in Equations 79

## CHAPTER 23

Important Information 82

## CHAPTER 24

A Shortcut 86

## CHAPTER 25

What Types of Numbers Does  
ATARI Like? 87

# CHAPTER 20

## ATARI's Memory

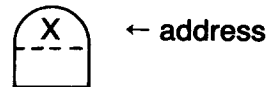
ATARI has a memory in its "brain" just like you do. Without a memory, ATARI would be no more than the average calculator. Memory is what allows ATARI to do many of the special tricks that we teach it.

The memory of ATARI's brain is much different from the memory of your brain. While the memory of your brain is made up of human tissue and nerves, ATARI's memory can be thought of as **ELECTRONIC MAILBOXES**. Each mailbox has its own **ADDRESS**, and can store one piece of information.

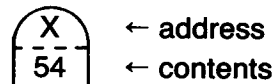
You, the programmer, decide what is to be stored in ATARI's memory. In the programs you write, you tell ATARI what information to remember, and in which mailbox, or **MEMORY CELL**, to store that information. The trick for doing this is very simple:

10 LET X=54

The **LET** statement tells ATARI to pick a mailbox in its memory and call it X. Thus, X is the address of the mailbox or memory cell.

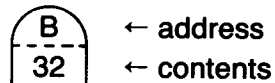


This LET statement also tells ATARI to put the number 54 into the memory cell or mailbox. Thus, 54 is the **CONTENTS** of memory cell mailbox X.

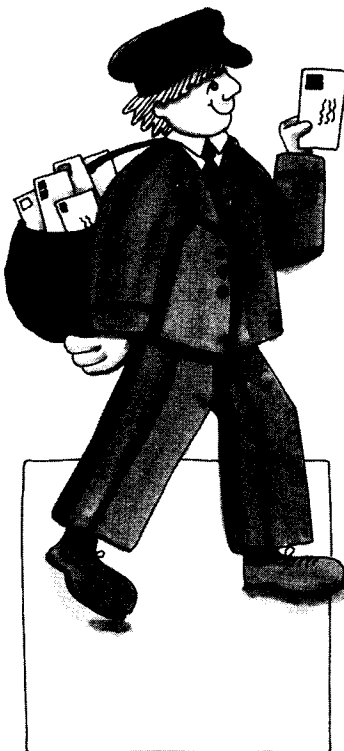
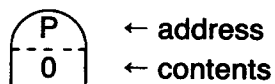


We can use many different letters, or even letters and numbers together, as the address of a memory cell mailbox. For example, we can say:

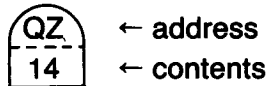
10 LET B=32



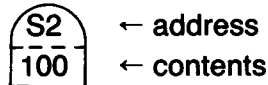
20 LET P=0



30 LET QZ = 14



40 LET S2 = 100



Because an address can have many various names, it is called a **VARIABLE**. In the program above, B, P, QZ, and S2 are all variables. Each variable stores a number value as the contents of its memory cell mailbox.

We will use a different type of variable to store a letter or word as the contents of a memory cell mailbox. We'll learn about these special variables later.

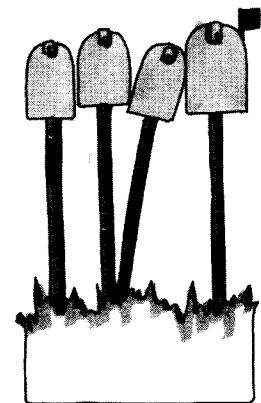
There are several rules that must be followed when writing variables for the ATARI computer.

1. Variables can include both letters and numbers, but must begin with a letter.
2. Only capital letters may be used.
3. Up to 120 letters and numbers may be used.

Even though 120 characters may be used, variables are usually written as single letters (X, Y, A), double letters (AA, DE, XY), or as one letter and one number (A1, Z5, G8).

## Rules for Writing Variables

1. Begin with a letter.
2. Use only capital letters.
3. Numbers can be used.
4. Up to 120 characters can be used.
5. *Usually* only one or two characters are used.
  - a. Single letter    A
  - b. Two letters    AB
  - c. Single letter and a single number    A6



**to do:** Programmer's Pastime #19

# CHAPTER 21

## Using Variables

Variables are very handy to use in a program. They allow us to store information or **DATA** and then **REFER** back to it later in the program. For this reason, you will be using a lot of variables when you write programs.

### Example:

```
10 LET X=5
20 LET Y=7
30 ? X
40 ? "IS THE CONTENTS OF X"
50 ? Y
60 ? "IS THE CONTENTS OF Y"
70 END
```

In line 10, 5 is the number assigned as the contents of memory cell mailbox X.

In line 20, 7 is assigned to Y.

In line 30, ATARI is asked to refer back to X and PRINT the contents.

In line 40, ATARI is told to PRINT a phrase.

In line 50, ATARI is asked to refer back to Y and PRINT the contents.

In line 60, ATARI is told to PRINT a different phrase.

In line 70, the program ends.

Whatever ATARI prints is called **OUTPUT**. The output for the above program would be:

```
5
IS THE CONTENTS OF X
7
IS THE CONTENTS OF Y

READY
☐
```

If we tell ATARI to: ? X (PRINT X)  
ATARI will print: 5 because 5 is the contents of memory cell mailbox X.





---

If we tell ATARI to: ? ``X'' (PRINT ``X'')  
ATARI will print: X because X is inside the  
quotation marks.

Let's use commas and semi-colons to change  
how the OUTPUT would look for our program.

**Program**

```
10 LET X=5
20 LET Y=7
30 ? X,
40 ? ``IS THE CONTENTS OF X''
50 ? Y;
60 ? ``IS THE CONTENTS OF Y''
70 END
```

**Output**

```
5      IS THE CONTENTS OF X
7 IS THE CONTENTS OF Y

READY
□
```

Let's change our program to make our output  
easier to read.

**Program**

```
10 LET X=5
20 LET Y=7
30 ? X,
40 ? ``IS THE CONTENTS OF X''
50 ?
60 ? Y;
70 ? ``IS THE CONTENTS OF Y''
80 END
```

**Output**

```
5 IS THE CONTENTS OF X
7 IS THE CONTENTS OF Y

READY
□
```

**to do:** Programmer's Pastime #20, #21

# PROGRAMMER'S PASTIME #21

Read each program. Then write what ATARI would print as the output. If you can, check your answers by running the programs on ATARI.

## Program

## Output

1. 10 LET RB4=40  
20 LET RB5=50  
30 LET RB1=10  
40 ? RB5; " IS ";  
50 ? RB1; " MORE";  
60 ? "THAN "; RB4  
70 END
2. 10 LET T=5  
20 LET V=25  
30 ? "THE SQUARE  
ROOT OF";  
40 ? V; " IS "; T  
50 END
3. 10 ? "MY FAVORITE  
NUMBER IS";  
20 LET D=333  
30 ? D  
40 GOTO 30
4. 10 ? "MY FAVORITE  
NUMBER IS";  
20 LET D=333  
30 ? D;  
40 GOTO 30



## CHAPTER 22 Using Variables in Equations

We can use variables to help us calculate math equations.

**For example:**

### Program

```
10 LET A=5
20 LET B=6
30 ? A+B
40 END
```

### Output

```
11
READY
□
```

We can use quotation marks and a semi-colon to make ATARI print the whole equation.

### Program

```
10 LET A=5
20 LET B=6
30 ? "A+B=", A+B
40 END
```

### Output

```
A+B=11
READY
□
```

OR

### Program

```
10 LET A=5
20 LET B=6
30 ? A, "+", B, "=", A+B
40 END
```

### Output

```
5+6=11
READY
□
```

Notice in line 30 how the semi-colons (,) are used. It is OK to mix variables (A and B) with symbols for arithmetic processes (+ and =), but they must be separated by semi-colons. If you forget to do this, ATARI will give you an error message.

Using variables in equations can be very helpful, especially if we need to do many equations with the same numbers.

**For example:**

**Program**

```
10 LET X=3
20 LET Y=9
30 LET Z=12
40 ? "X+Y+Z="; X+Y+Z
50 ? "Z-Y-X="; Z-Y-X
60 ? "X*Z/Y="; X*Z/Y
70 END
```

**Output**

X+Y+Z=24

Z-Y-X=0

X\*Z/Y=4

READY



If we wanted ATARI to print each equation using the number values instead of the variables, we would use the quotation marks differently:

**Program**

```
10 LET X=3
20 LET Y=9
30 LET Z=12
40 ? X, "ø+ø"; Y, "ø+ø"; Z, "ø=ø"; X+Y+Z
50 ? Z, "-"; Y, "-"; X, "="; Z-Y-X
60 ? "X, "*"; Z, "/"; Y, "="; X*Z/Y
70 END
```

**Output**

3 + 9 + 12 = 24

12-9-3=0

3\*12/9=4

READY



(Notice how the output for lines 50 and 60 is spaced differently than the output for line 40. Blank spaces (ø) must be used inside the quotation marks to get additional space.)

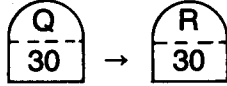
We have learned that a variable can have a number value.

10 LET J=16



A variable can also have another variable's value IF the other variable has already been introduced by a LET statement in the program.

10 LET Q=30  
20 LET R=Q



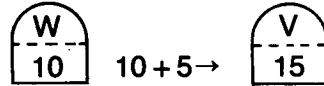
A variable can also have an equation as its value.

10 LET F=7+8



OR

10 LET W=10  
20 LET V=W+5



## Remember!

The LET statement assigns a value to a variable.

? "X" will print X

? X will print the value of X

**to do:** Programmer's Pastime #22, #23



## CHAPTER 23 Important Information

There are some important things to remember about using the LET statement.

**1** The variable must always come before the value (number) in the LET statement.

10 LET S=40      is correct.

10 LET 40=S      is wrong. ATARI will not understand this statement.

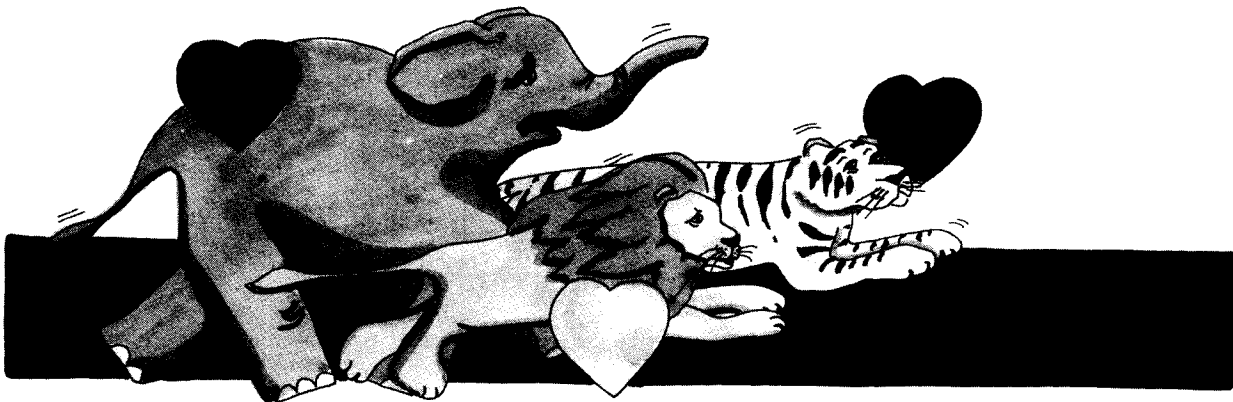
**2** In a program, we must always put our LET statement *before* the statement that tells ATARI to print the variable.

10 LET S=40  
20 ? S              is correct.

10 ? S  
20 LET 40=S      is wrong. ATARI's output will be 0.

If ATARI sees a variable in a program that has not been introduced by a LET statement, ATARI will automatically give that variable a value of zero.

In the second program, line 10 tells ATARI to print S. Since there was no LET statement before line 10 to introduce S, ATARI gave S a value of 0. Even though we tell ATARI that S=40 in line 20, ATARI will print 0 because the PRINT statement comes *before* the LET statement.



**Program**

```
10 LET U=10
20 LET V=20
30 ? U+V      is correct.
```

**Output**

```
30
READY
☐
```

**Program**

```
10 ? U+V
20 LET U=10
30 LET V=20  is wrong.
```

**Output**

```
0
READY
☐
```

**3**

When you introduce the same variable more than once in a program, ATARI will always remember the *last* thing you told it.

**Program**

```
10 LET K=1
20 LET K=2
30 ? K
```

**Output**

```
2
READY
☐
```

We used a LET K statement two times. ATARI only remembers that K=2 because it was the *last* LET statement. We told ATARI to *change* the value of K from 1 to 2.

**Program**

```
10 LET K=1
20 ? K
30 LET K=2
40 ? K
```

**Output**

```
1
2
READY
☐
```

In this program we told ATARI to print the first value of K in line 20 before we changed the value of K in line 30.

**to do:** Programmer's Pastime #24, #25, #26

# PROGRAMMER'S PASTIME #24

Read each program. Then write what ATARI would print as the output. Check your answers by running the programs on ATARI.

## Program

## Output

1. 10 LET PJ=17  
20 LET J2=34  
30 LET J4=PJ+J2  
40 ? J4  
50 END

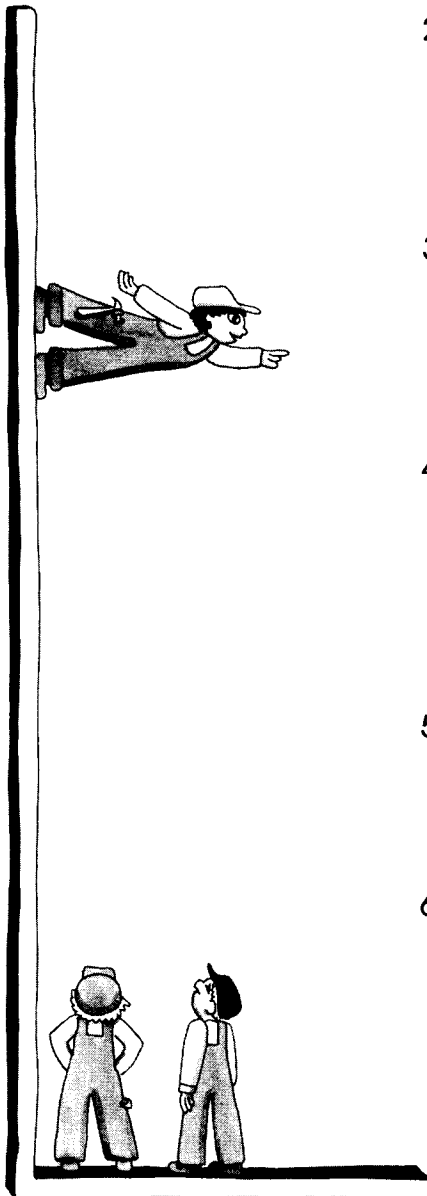
2. 10 LET B=2  
20 ? B  
30 LET B=100  
40 ? B  
50 END

3. 10 LET X1=2  
20 LET X2=X1\*5  
30 LET X3=X2/X1  
40 ? X3  
50 END

4. 10 LET E6=3  
20 LET E7=12  
30 ? "PRODUCT",  
"QUOTIENT"  
40 ? E6\*E7, E7/E6  
50 END

5. 10 LET HI=16  
20 LET HJ=HI+4  
30 ? HJ+10  
40 END

6. 10 LET M=16  
20 LET N=14  
30 ? M+N  
40 LET N=12  
50 ? M+N  
60 END





---

7. 10 LET Z1=8  
20 LET Z2=Z1-2  
30 ? Z2+Z1/2  
40 END

8. 10 LET T1=6  
20 LET T1=7  
30 ? T1  
40 GOTO 30  
50 END

9. 10 LET J=11  
20 LET K=22  
30 LET J=17  
40 ? K+J  
50 END



# CHAPTER 24

## A Shortcut

As computer programmers, we are always looking for helpful shortcuts that will make our programming easier.

We can use colons to shorten our program when we use LET statements.

```
10 LET A=6: LET B=7: LET C=8: LET D=9
```

We can use commas to shorten our use of PRINT statements.

```
20 ? A, B, C, D
```

Both the LET and PRINT statements can be as long as three lines on ATARI's screen. As you approach the end of the third line, ATARI will make a "squawking" sound. You must press RETURN before the end of the third line or ATARI will give an error message.

Using these shortcuts, you can take a long program like this:

```
10 LET S=66  
20 LET T=33  
30 LET U=99  
40 ? S+T  
50 ? T+U  
60 ? U+S  
70 END
```

And shorten it to this:

```
10 LET S=66: LET T=33: LET U=99  
20 ? S+T, T+U, U+S  
30 END
```

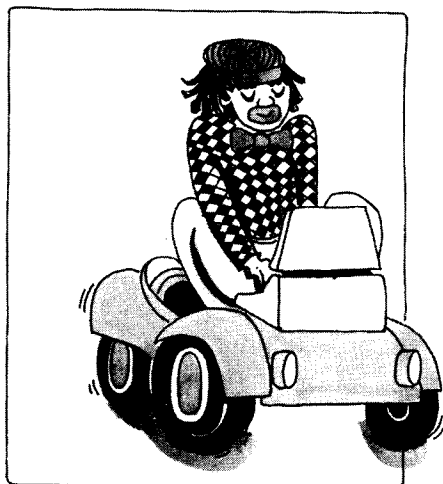
Or, line 10 can be written *without* the LET statement.

```
10 S=66: T=33: U=99
```

### Remember:

1. Put all of your LET statements on 1 line.
2. Put all of the things you want to print with 1 PRINT statement.

**to do:** Programmer's Pastime #27



# CHAPTER 25

## What Types of Numbers Does ATARI Like?

So far, we have asked ATARI to deal mainly with **WHOLE NUMBERS** (0, 1, 2, 3...). We know that ATARI can also handle **NEGATIVE NUMBERS** (-1, -2, -3...).

ATARI can also work with decimals (.09, 1.23) and simple fractions ( $\frac{1}{4}$ ,  $\frac{1}{2}$ ), but cannot understand compound fractions ( $1\frac{1}{2}$ ,  $2\frac{1}{4}$ ). If you need to have ATARI do some math with compound fractions, you will need to change the fractions to their decimal equivalents, either by dividing them yourself, or by using the PRINT (?) statement. After you have changed the fractions to their decimal equivalents, add them to the whole numbers.

**For example:**

For  $1\frac{1}{2}$ , change  $\frac{1}{2}$  by  $2 \overline{)1.0} = .5$  or typing ? 1/2.  
 $1\frac{1}{2} = 1.5$

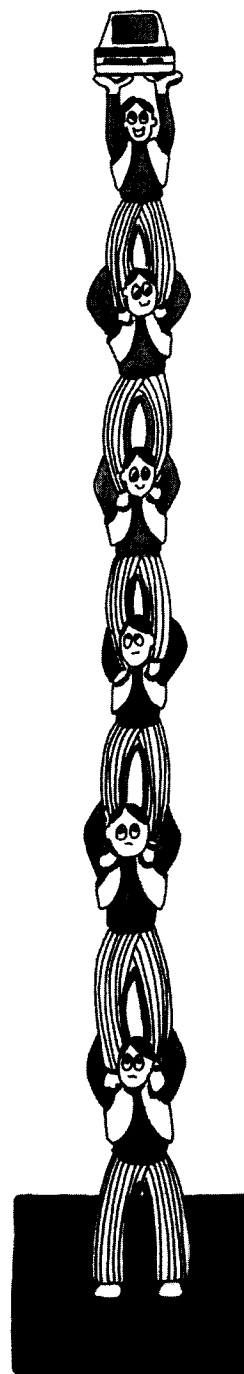
For  $1\frac{1}{4}$ , change  $\frac{1}{4}$  by  $4 \overline{)1.00} = .25$  or typing ? 1/4.  
 $1\frac{1}{4} = 1.25$

If you want to use numbers that are very large or very small, ATARI will change them into something called **E NOTATION** or **FLOATING POINT NOTATION**. For example, if you wanted to use a number that has 12 digits, like 420000000000, ATARI would print it as **4.2E+11**. The E+11 means that the decimal point belongs 11 more places to the *right*.

Notice that we do not use commas with large numbers. We write 42000000 not 42,000,000. Commas used inside numbers will confuse ATARI and cause an error message.

If you wanted to use .0000009876 ATARI would print **9.876E-07**. The E-07 means that the decimal point belongs 7 more places to the *left*.

Don't get worried about E notation because you'll only have to use it when you are dealing with very small or very large numbers.



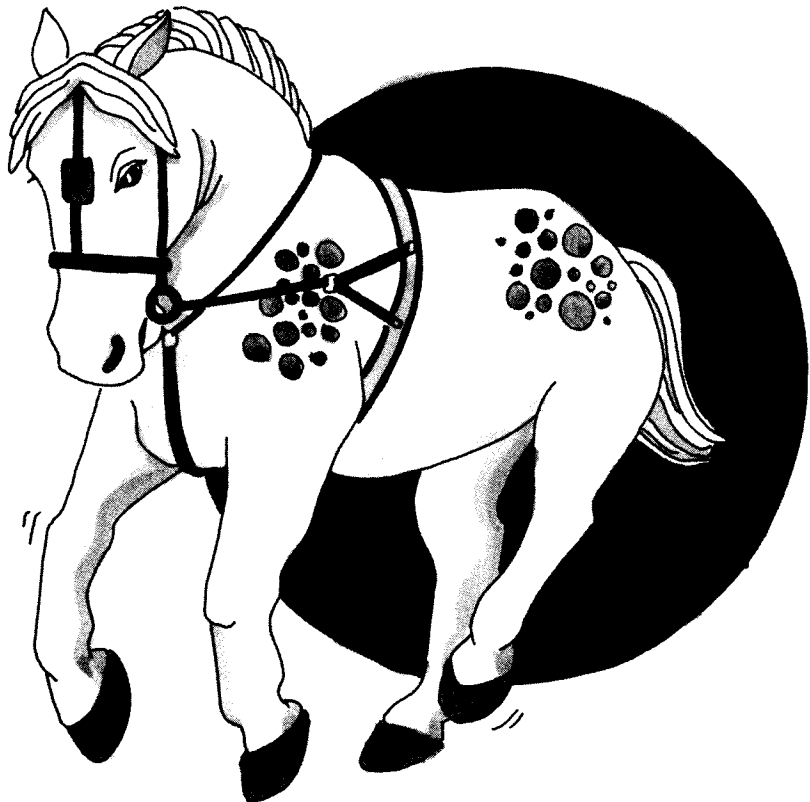
If you are using decimals like .041, ATARI will switch to E notation if there are two or more zeros after the decimal point.

**Example      ATARI prints**

.041	.041
.0041	4.1E-03
.00041	4.1E-04
.000041	4.1E-05

**to do:** Programmer's Pastime #28  
Component 4 Fun Page  
Evaluate Yourself

E Notation stands for "Exponential Notation." It is a helpful shortcut you can learn to use.

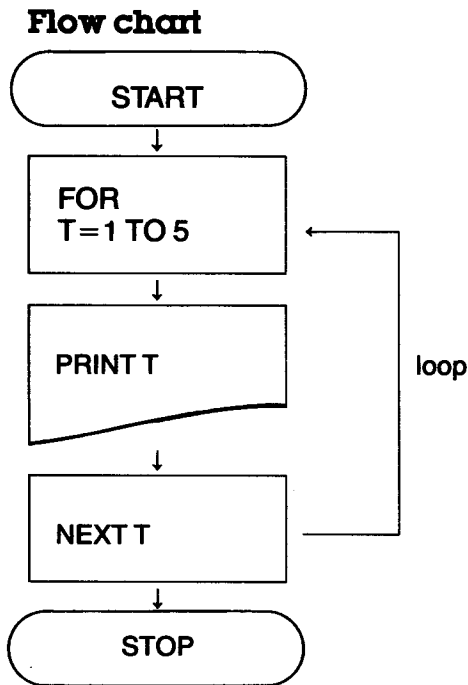


# COMPONENT 5

<b>CHAPTER 26</b>	
FOR-NEXT Looping	90
<b>CHAPTER 27</b>	
Stepping	96
<b>CHAPTER 28</b>	
A Counter	99
<b>CHAPTER 29</b>	
A Clean Trick	102
<b>CHAPTER 30</b>	
Blinkers	106
<b>CHAPTER 31</b>	
Special Commands	109
<b>CHAPTER 32</b>	
Debugging	112

# CHAPTER 26 FOR-NEXT Looping

Another type of loop we will use in our programming is called a **FOR-NEXT LOOP**. We use it to create **COUNTER-CONTROLLED LOOPS** in a program. These loops allow us to repeat program instructions a certain number of times. For example:



## Program

```
10 FOR T=1 TO 5  
20 ? T  
30 NEXT T  
40 END
```

loop  
is done  
5 times.

## Output

```
1  
2  
3  
4  
5  
READY  
□
```

Let's trace the program to see how it works!

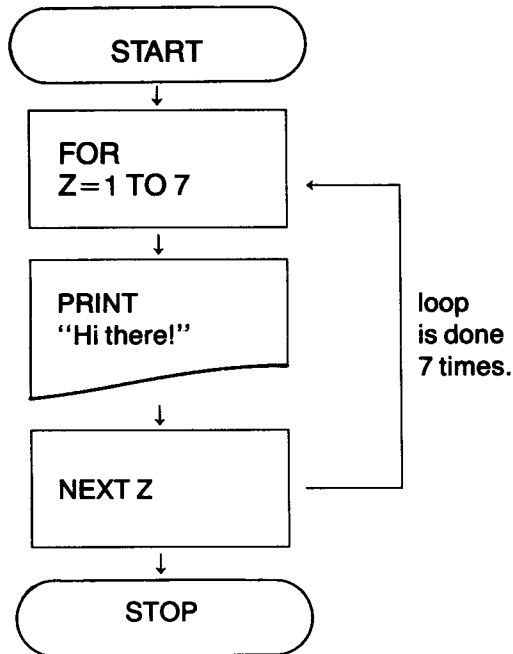
Line Number	What Happens	Contents of T
10	Tells ATARI to start counting from 1 to 5.	1
20	Tells ATARI to print T (which is 1).	1
30	Tells ATARI to go back to line 10.	1
10	Tells ATARI to count to the next number.	2
20	Tells ATARI to print T (which is now 2).	2
30	Tells ATARI to go back to line 10.	2

This looping continues until T=5 and ATARI has printed the numbers 1, 2, 3, 4, and 5. Then the program goes to line 40 and ends.

The FOR-NEXT steps make ATARI count from 1 to 5. Because the PRINT statement in line 20 is between the FOR and NEXT statements, it is in the middle of the loop. T will be printed each time the loop is done.

### New example:

#### Flow chart



#### Program

```
10 FOR Z=1 TO 7
20 ? "HI THERE!"
30 NEXT Z
40 END
```

#### Output

```
HI THERE!
HI THERE!
HI THERE!
HI THERE!
HI THERE!
HI THERE!
HI THERE!
```

```
READY
□
```

ATARI is told to count to 7 and print "Hi there!" each time. The variable Z in line 10 is called a **COUNTER**. Line 10 starts the counter, Z, at 1. Each time ATARI comes to line 30, it counts the next number until it has reached 7. If the counter is already 7 when the computer comes to line 30, the computer will go on to line 40 and END. Thus, the counter controls how many times the loop is done.

### Important

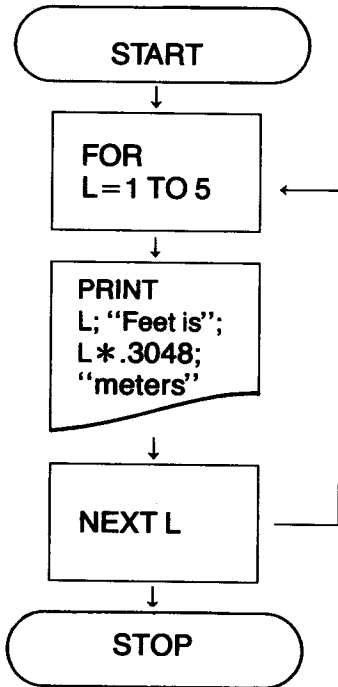
Every FOR statement *must* have a NEXT statement after it somewhere in the program.

Any statements in between the FOR statement and the NEXT statement are in the **BODY** of the loop. These statements will be done each time the loop is run.



A FOR-NEXT loop is handy to use in a program that **CONVERTS** or changes one type of measurement into another. The following program converts feet into meters.

### Flow chart



### Program

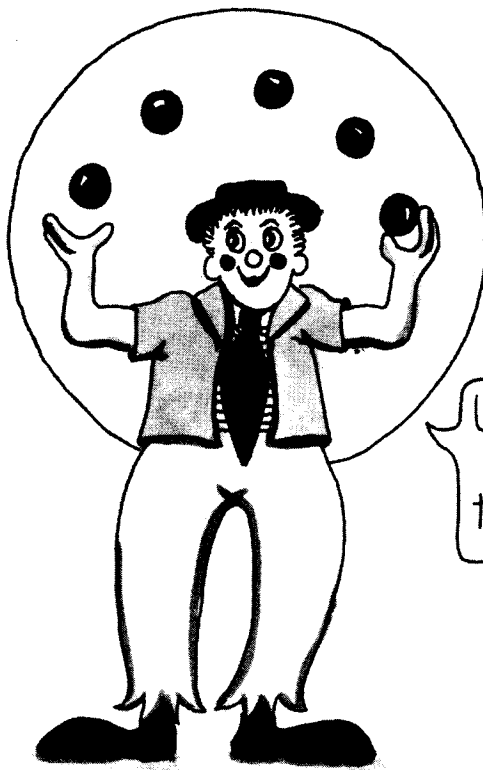
```
10 FOR L=1 TO 5
20 ? L, " FEET IS ";
   L*.3048, " METERS"
30 NEXT L
40 END
```

The program code is shown with a bracket indicating the loop body, which consists of lines 20 and 30.

### Output

```
1 FEET IS .3048 METERS
2 FEET IS .6096 METERS
3 FEET IS .9144 METERS
4 FEET IS 1.2192 METERS
5 FEET IS 1.524 METERS
```

READY

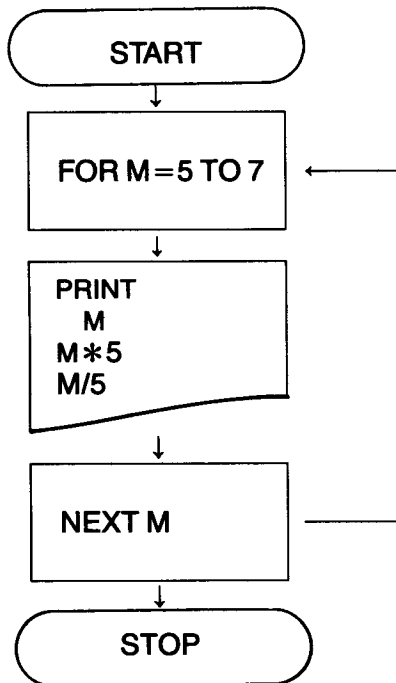


Using a loop helps  
do the same  
thing over and over.



A FOR-NEXT loop can also allow a program to do arithmetic, using a new number each time the loop is done. The loop in the following program causes the numbers 5, 6, and 7 to be printed, multiplied by 5, and divided by 5.

### Flow chart



### Program

```
10 FOR M=5 TO 7
20 ? M
30 ? M*5
40 ? M/5
50 NEXT M
60 END
```

A bracket on the right side of lines 20 through 40 is labeled 'loop body'.

### Output

```
5
25
1
6
30
1.2
7
35
1.4
READY
□
```

to do: Programmer's Pastime #29, #30, #31



# PROGRAMMER'S PASTIME #29

Read each program. Then write what you think ATARI would print as the OUTPUT. Run the programs on ATARI to check your answers.

## Program

## Output

1. 10 FOR Q= 2 TO 6  
20 ? Q  
30 NEXT Q  
40 END
2. 10 FOR Q=2 TO 4  
20 ? "Q=", Q  
30 NEXT Q  
40 END
3. 10 FOR A=1 TO 5  
20 ? "HELLO FRIEND!"  
30 ? "HOW ARE YOU?"  
40 NEXT A  
50 END
4. 10 FOR D=1 TO 3  
20 ? D  
30 ? D+10  
40 NEXT D  
50 END
5. 10 LET P=3  
20 FOR Q=1 TO 3  
30 ? P, "+", Q, "=", P+Q  
40 NEXT Q  
50 END
6. 10 FOR B=1 TO 5  
20 ? "B", "B+B", "B\*B"  
30 ? B, B+B, B\*B  
40 NEXT B  
50 END



---

7. 10 ? "MULTIPLICATION TABLE

FOR 7"

20 FOR K=1 TO 12

30 ? K, "TIMES 7 = ", K\*7

40 NEXT K

50 END

8. 10 FOR G=1 TO 10

20 ? "♥"

30 NEXT G

40 END

9. 10 FOR G=1 TO 10

20 ? "♥",

30 NEXT G

40 END

10. 10 FOR S=1 TO 10

20 LET S=S\*S

30 ? S, S/S

40 NEXT S

50 END

Can you explain how this program works? \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

# CHAPTER 27 Stepping

When you were younger you learned to count in patterns

like: 5, 10, 15, 20 . . . (by fives),  
or: 10, 20, 30, 40, 50 . . . (by tens).

ATARI can learn this trick too. If you want ATARI to count in a certain pattern, use the **STEP** statement. For example,

STEP 5 would tell ATARI to count by fives and  
STEP 10 would tell ATARI to count by tens.

The STEP statement goes on the same line as the FOR statement. Study the following programs:

## Program

```
10 FOR Z=0 TO 25 STEP 5
20 ? Z
30 NEXT Z
40 END
```

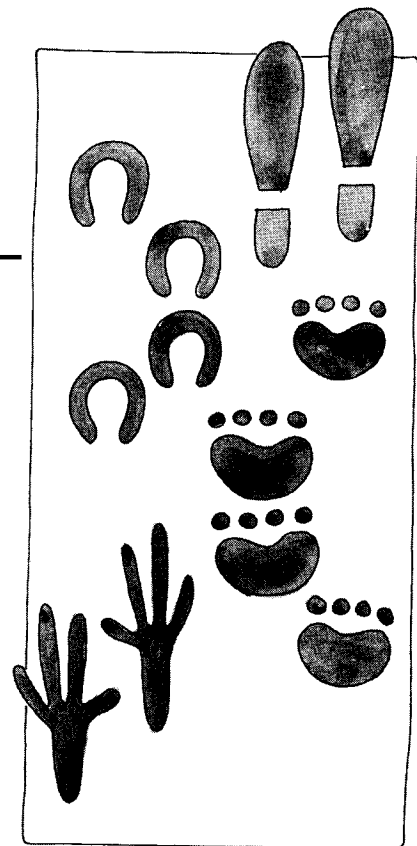
## Output

```
0
5
10
15
20
25
READY
☐
```

```
10 FOR Z=1 TO 25 STEP 5
20 ? Z
30 NEXT Z
40 END
```

## Output

```
1
6
11
16
21
READY
☐
```



How are the two programs different? If you want ATARI to count by fives, you must make the FOR statement say: FOR Z=0 TO 25 STEP 5

In the second program, ATARI started counting with 1 and added 5 to it to get 6. In this program ATARI is not counting "by fives," but adding 5 to each number—beginning with 1. The last number printed was 21. Because  $21 + 5 = 26$ , which is more than 25, ATARI won't print 26.

ATARI can also count backwards.

#### Program

```
10 FOR R=5 TO 1 STEP -1
20 ? R
30 NEXT R
40 END
```

#### Output

```
5
4
3
2
1
READY
☐
```

R starts counting at 5. The STEP of -1 makes R count backwards, subtracting 1 each time.

You can write some fun programs by using the STEP statement.

#### Program

```
10 ? "STAND BY FOR BLAST OFF"
20 FOR D=5 TO 1 STEP -1
30 ? D; "SECONDS"
40 NEXT D
50 ? "BLAST OFF!"
60 END
```

#### Output

```
STAND BY FOR BLAST OFF
5 SECONDS
4 SECONDS
3 SECONDS
2 SECONDS
1 SECONDS
BLAST OFF!
READY
☐
```



Backward stepping can also be used to print words a certain number of times.

### Program

```
10 FOR P=20 TO 5 STEP -5
20 ? "GOING BACKWARDS"
30 NEXT P
40 END
```

### Output

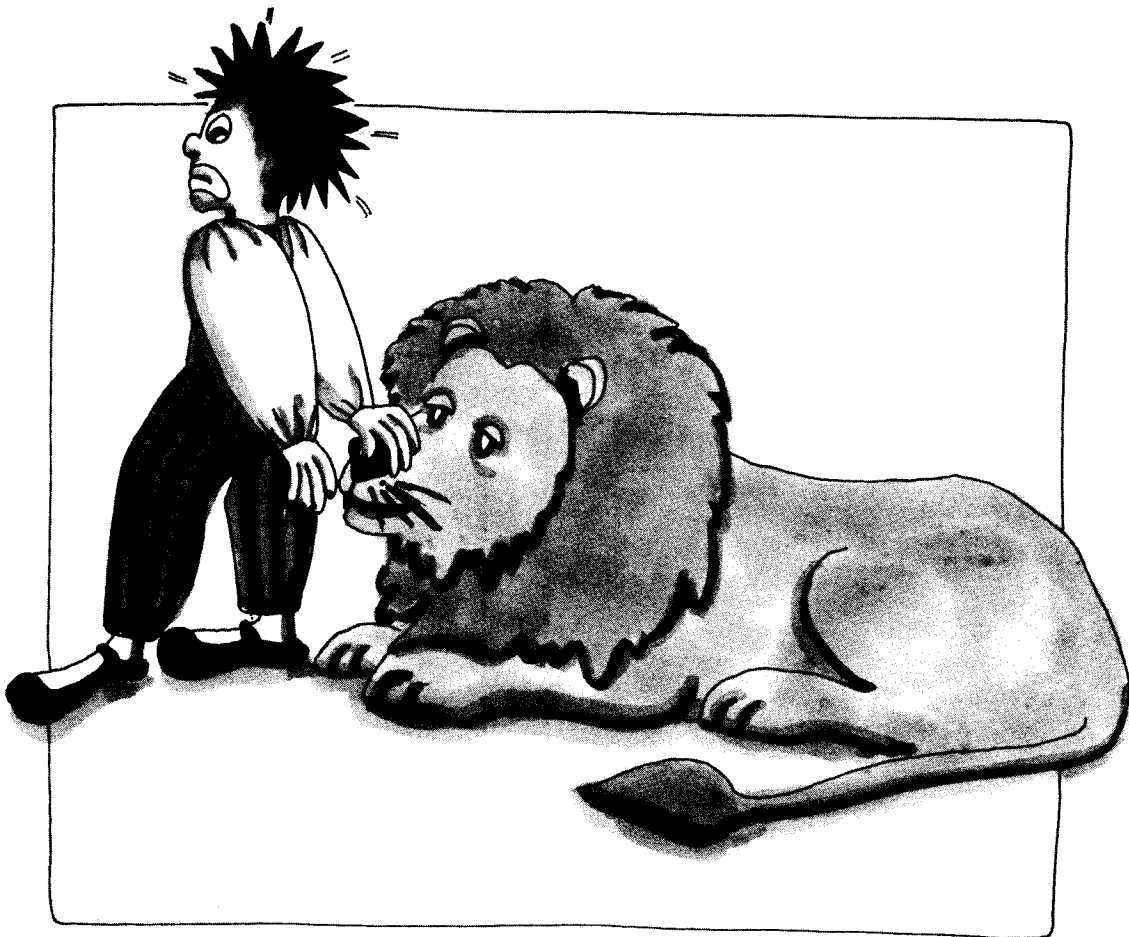
```
GOING BACKWARDS
GOING BACKWARDS
GOING BACKWARDS
GOING BACKWARDS
```

READY



"Going backwards" is printed four times because it takes four runs of the loop to go from 20 to 5 in steps of -5.

**to do:** Programmer's Pastime #32, #33



# CHAPTER 28

## A Counter

Sometimes it is handy to use a counter in your program to help you keep track of how many times you have done a loop. For example:

### Program

```
10 LET N=0
20 ? "BUZZ OFF"
30 ? N
40 GOTO 20
```

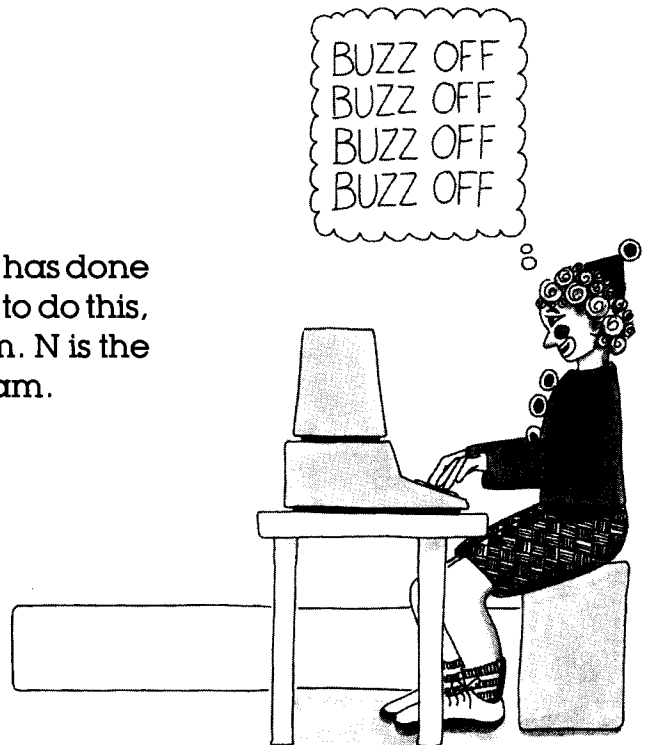
### Output

```
BUZZ OFF
0
BUZZ OFF
0
BUZZ OFF
0
.
.
```

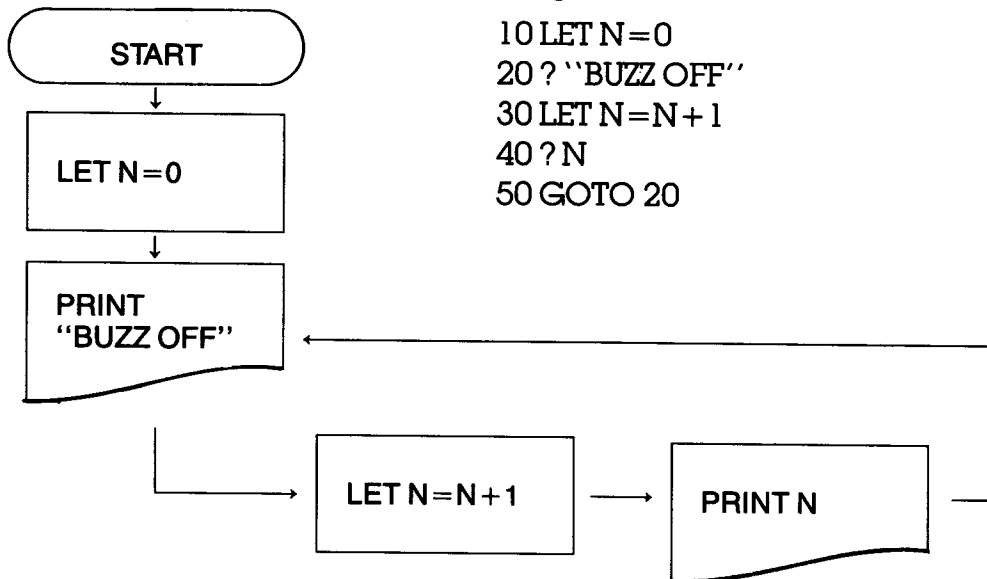
This program has a never ending loop that prints "Buzz off" over and over. If we could get ATARI to print:

```
BUZZ OFF
1
BUZZ OFF
2
BUZZ OFF
3
.
.
```

we would know how many times ATARI has done the loop and printed "Buzz off." In order to do this, you must put a counter in your program. N is the counter variable in the following program.



### Flow chart



### Program

```
10 LET N=0
20 ? "BUZZ OFF"
30 LET N=N+1
40 ? N
50 GOTO 20
```

### Output

```
BUZZ OFF
1
BUZZ OFF
2
BUZZ OFF
3
BUZZ OFF
4
.
.
```

### PROGRAM TRACE

Loop	Line Number	What Happens	Contents of N
1	10	N is introduced at 0	0
1	20	ATARI prints "BUZZ OFF".	0
1	30	COUNTER adds 1 to N.	1
1	40	ATARI prints N (which is 1).	1
1	50	ATARI goes back to line 20.	
2	20	ATARI prints "BUZZ OFF".	1
2	30	COUNTER adds 1 to N.	2
2	40	ATARI prints N (which is 2).	2
2	50	ATARI goes back to line 20.	2
3	20	ATARI prints "BUZZ OFF".	2
3	30	COUNTER adds 1 to N.	3
3	40	ATARI prints N (which is 3).	3
3	50	ATARI goes back to line 20.	3



The statement that makes N increase by 1 each time the loop is done is:

```
30 LET N=N+1
```

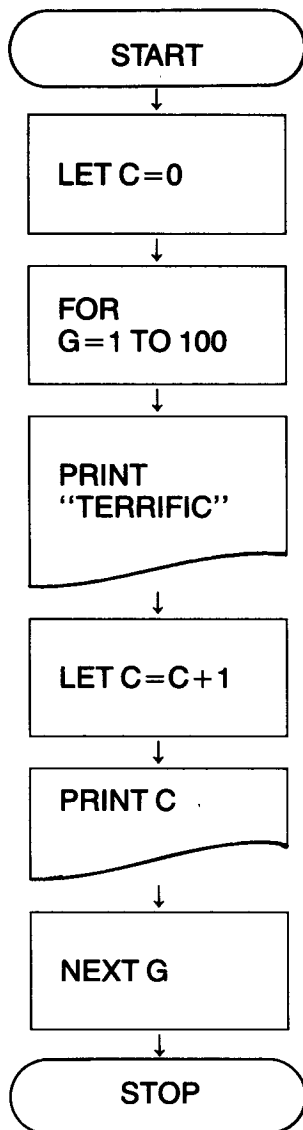
It *must* be in the loop body. This statement is called a counter.

When we push  to stop the program, we will know how many times ATARI has done the loop and printed "Buzz off."

We can also use a counter in a FOR-NEXT loop.



### Flow chart



### Program

```
10 LET C=0
20 FOR G=1 TO 100
30 ? "TERRIFIC"
40 LET C=C+1
50 ? C
60 NEXT G
70 END
```

### Output

```
TERRIFIC
1
TERRIFIC
2
TERRIFIC
3
.
.
.
TERRIFIC
100
READY
☒
```

We must be careful when we use two different variables in a program. In the previous program, the variable C stands for the counter. The variable G stands for the FOR-NEXT loop. It is important to keep these variables separate so we can better understand what our program is doing.

### A GOOD IDEA

1. Use the variable C to stand for COUNTER.
2. Use the variable FL to stand for a FOR-NEXT loop.



**to do:** Programmer's Pastime #34



# CHAPTER 29




## A Clean Trick

A good trick to teach ATARI is how to clear the screen before, after, or during a program.

When you clear the screen you press  and hold it down while you press . This makes the screen blank.




To teach ATARI how to do this in a program, a step must be included as part of the program:

### Program

```
10 ? "    "
```




### What appears on the screen

```
"↵"
```

Notice that first you use a PRINT statement and quotation marks. Next you press , and then hold down  while pressing the  key. ATARI puts an arrow (↵) on the screen to let you know that you have typed the statement correctly.

This trick can be used to clear the screen before a program is run. In this case, the step must be the very first line of the program. For example:

### Program

```
10 ? "    "  
20 ? "WATCH ME RUN"  
30 GOTO 20
```

### Output

(screen is cleared)







```
WATCH ME RUN  
WATCH ME RUN  
WATCH ME RUN  
WATCH ME RUN
```

·  
·  
·




You can also use this trick to clear the screen in the middle of a program run. For example:

#### Program

```
10 ? "    "  
20 ? "MY NAME IS ATARI  
30 ? "    "  
40 ? "WHAT'S YOURS?"  
50 END
```







#### Output

```
(screen is cleared)  
MY NAME IS ATARI  
(screen is cleared)  
WHAT'S YOURS?  
READY  

```


When you run this program, you will notice that ATARI writes "My name is ATARI" and then clears the screen so fast that you cannot read it.

What can you do to slow ATARI down? Add a **FOR-NEXT TIME LOOP** in line 25 to use up time and make ATARI wait. For example:

#### Program

```
10 ? "    "  
20 ? "MY NAME IS ATARI  
25 FOR T=1 TO 1000: NEXT T  
30 ? "    "  
40 ? "WHAT'S YOURS?"  
50 END
```

#### Output




```
(screen is cleared)  
MY NAME IS ATARI  
(ATARI counts to 1000)  
(screen is cleared)  
WHAT'S YOURS?  
READY  

```

In line 25 the program stops running while ATARI counts to 1000. When ATARI is through counting, the program continues.

We used a shortcut to write both the FOR and NEXT statements on the same line. There must be a colon (:) between the two statements to keep them separate.



If you want ATARI to wait longer, change 1000 in the FOR-NEXT time loop to a larger number. If you want ATARI to clear the screen more quickly, change 1000 to a smaller number.

This clean trick can also be used to clear the screen at the end of a program, although this is not often done. In this case, you would put the ? "    " statement at the end of the program—right before the END statement.

There is another trick in which ATARI is told to print on a lower line. ATARI will always print on the NEXT available line unless you program it differently. For example:

### Program

```
10 ? "    "  
20 ? "TOP LINE"
```

### Output

(screen is cleared)




TOP LINE

READY



After the screen was cleared, ATARI printed "top line" on the first available line, which was the top line.

If you want ATARI to print on a lower line, you can use the PRINT (?) statement. Each time ATARI comes to PRINT or ?, it will skip a line. For example, you can move the top line program down like this:

```
10 ? "    "  
20 ?  
30 ?  
40 ? "TOP LINE"  
50 END
```

(screen is cleared)

TOP LINE




READY



---

The more PRINT (?) statements you use, the more lines ATARI skips.

Here's a shortcut. You can put several PRINT statements on one line by using colons (:) to separate them, like this:

```
10 ? "    "  
20 ?::?:?:?  
30 ? "TOP LINE"  
40 END
```

(screen is cleared)

TOP LINE

READY



**to do:** Programmer's Pastime #35, #36



# CHAPTER 30

## Blinkers

We can use the FOR-NEXT time loop to make things blink on and off ATARI's screen. For example:

### Program

```
10 ? "    "  
15 ?  
20 FOR T=1 TO 100: NEXT T  
30 ? "WOW"  
40 GOTO 10
```

### Output

WOW

The secret to the blinking is in lines 20 and 40.

In line 20, FOR T=1 TO 100 makes ATARI's counting/waiting period brief.

In line 30, "WOW" is printed.

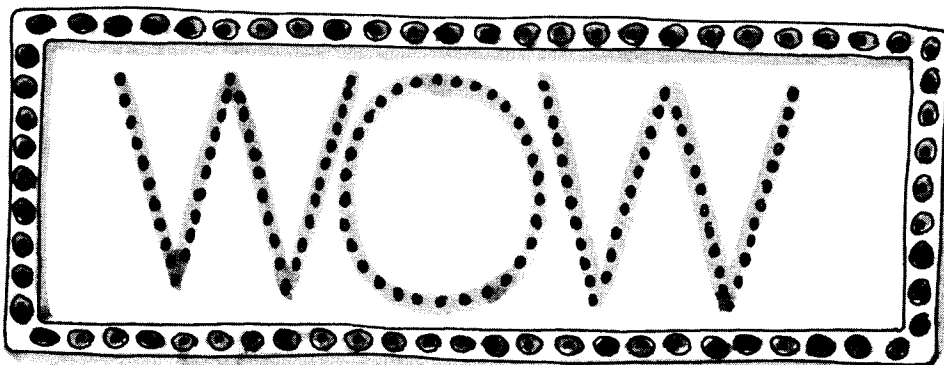
In line 40, ATARI goes back to line 10 and clears the screen. Then ATARI will count very quickly again in line 20 before printing "WOW" in line 30.

BLINK OFF: Line 10 and line 20   
BLINK ON: Line 30 and line 40

In order to make something blink, you *must* have a FOR-NEXT time loop and a GOTO statement in your program.

You can make something blink faster by changing 100 to a smaller number. You can make something blink more slowly by changing 100 to a larger number.

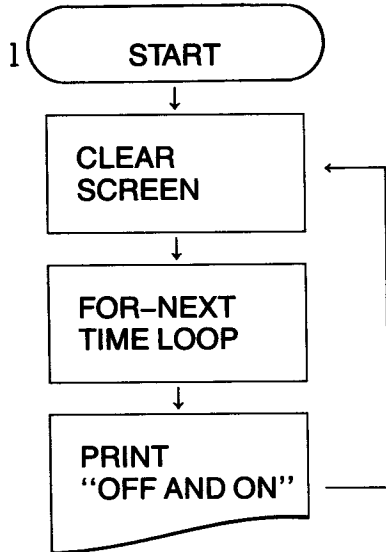
**to do:** Programmer's Pastime #37



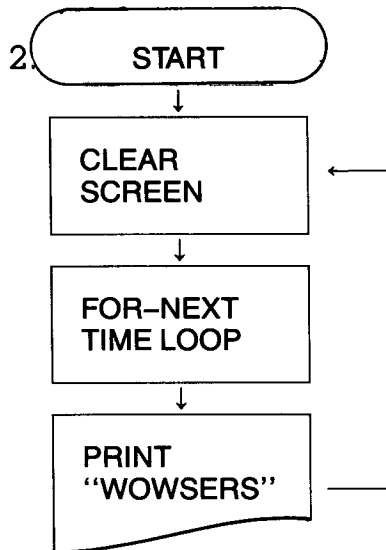
# PROGRAMMER'S PASTIME #37

Write a program for each flow chart, then run your programs on ATARI to make sure they work.

## Flow chart



This is the basic algorithm for making something blink.



---

Use your expertise and imagination to write two of your own programs that make something blink. You can even make graphics or pictures blink! Don't be afraid to experiment.

**Flow chart**

**Program**

1.

2.





# CHAPTER 31

## Special Commands

You have learned to tell ATARI to do many different things by using commands such as NEW, RUN, END, LOAD, and SAVE. Here are a few more commands that you may find useful.

You know that you may stop a program while it is running by pushing the  key. When the program stops, the screen will give you a BREAK message:

"STOPPED AT \_\_\_\_\_"

↑  
the line number at which you  
stopped the program

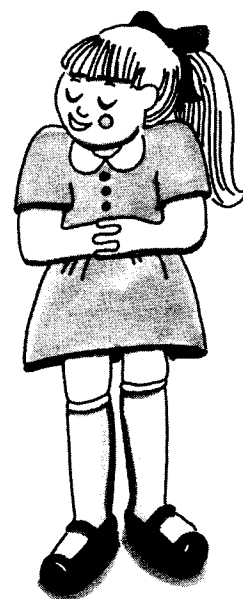
**TO  
STOP**

If you want to start the program again, continuing from where the break occurred, type CONT .

**TO  
CONTINUE**

If you want to re-run the program from the very beginning, type RUN .

You can also cause a program to stop by including a STOP statement in the program. It's sometimes useful to use stop within long programs so that smaller portions of the program can be checked for errors. The program can be started again by typing CONT and pressing .



Sometimes it is necessary to have ATARI display all the lines of a program in its memory. To do this, type "LIST."

## TO LIST

Type LIST

The LIST command is especially helpful when you are writing a program yourself. As you test your program, you can list the whole program or just parts of it to see how it looks.

To list just one line of a program type:

LIST \_\_\_\_\_

↑  
the line number  
you wish to list

For example, "LIST 100" would list only line 100 of your program.

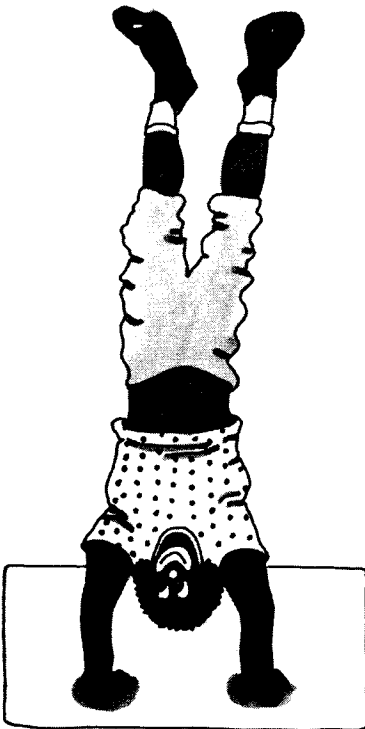
To list sections of your program type:


LIST \_\_\_\_\_

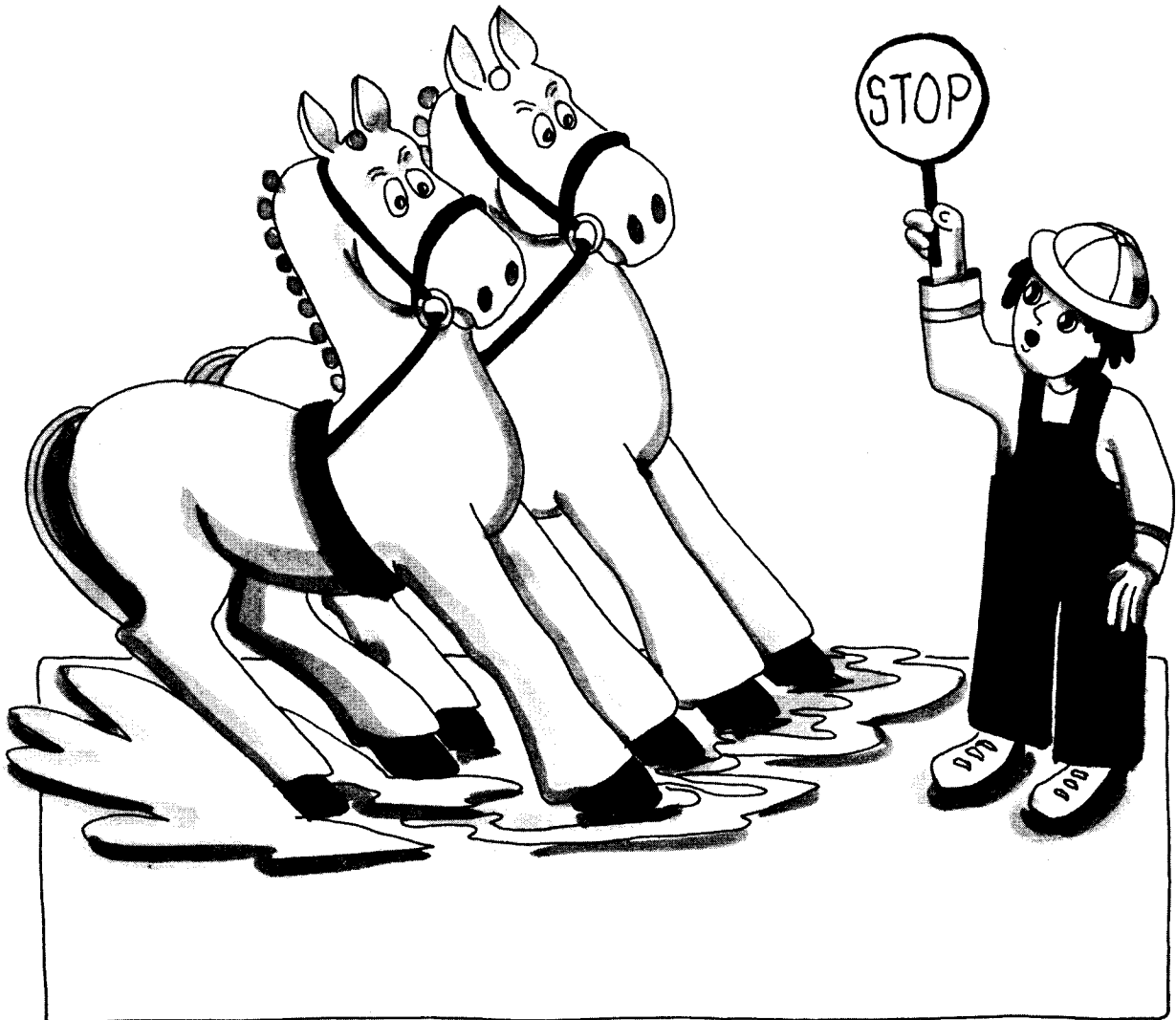
↑  
the first and last line numbers of the  
section you wish to list,  
separated by a comma

For example: "LIST 100,300" would list only those lines between and including lines 100 and 300.

If you are listing a long program, and you want the display on the screen to stop, press  and  together. Press the same two buttons a second time to get the listing to continue.



You may find one other special command useful in programming. You can leave BASIC and go to the MEMO PAD MODE by using the BYE command on the older Atari models. Using this command, you can experiment with the keyboard or leave a message on the screen without changing any program in ATARI's memory. You use BYE differently on the newer XL Atari models. On these, you use it to enter a testing mode and check how well ATARI's memory keyboard and audiovisual systems are working. To get back to your BASIC program, press . Then you will need to run or list your program again.



# CHAPTER 32

## Debugging

Writing a computer program can be a long process. It often takes many tries before we get a program to work properly. This is because there can be **BUGS** in your program. No, there aren't little insects climbing around inside the computer. Bugs are mistakes that you, the programmer, can make when you write a program.

Some examples of bugs might be:

1. forgetting to type a punctuation mark;  
Example: Typing ? "HI" instead of ? "HI"
2. spelling a command wrong;  
Example: Typing "REN" instead of "RUN"
3. putting the steps of your program in the wrong order.

It is very important to check your program for bugs *before* you try running it. Even once you start running the program you may come across more bugs that need to be corrected. The best way to do this is to take turns running the program and listing it. Once you find your mistake in the program listing, use ATARI's edit features to insert or delete to correct the mistake. You may change an entire line, if need be, by retyping it at the end of the listing. You can delete an entire line by typing the line number and pressing RETURN. For example:

110

erases line 110 from the program. There is no longer a line 110.



The entire process of getting rid of program bugs is called **DEBUGGING**.

There are three types of errors you may run across as you work with computers. Sometimes ATARI will tell you what your error is. Other times you will have to figure out yourself what the error is and where it is. The three types of errors are:

### **USER ERRORS**

**1**

A user error happens when you make a typing mistake or fail to communicate with ATARI in BASIC.

### **PROGRAM ERRORS**

**2**

A program error occurs when there are bugs in your program. You will have to debug your program to correct the errors.

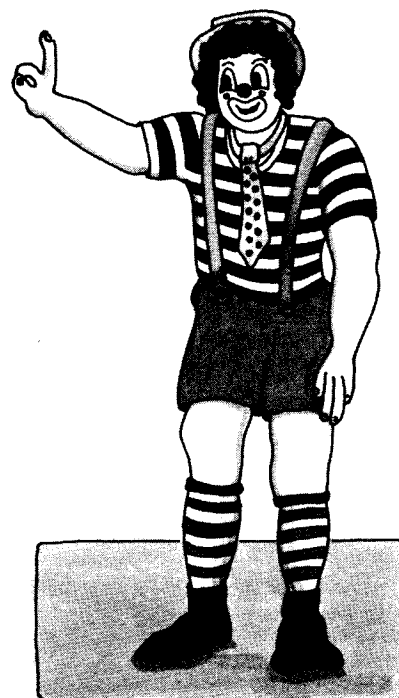
### **COMPUTER ERRORS**

**3**

A computer error could happen if the computer's equipment is not hooked up properly. These errors can be very complicated, but they rarely occur.

To find out what certain errors mean, turn to Appendix B at the back of this book.

**to do:** Component 5 Fun Page  
Evaluate yourself



# COMPONENT 6

<b>CHAPTER 33</b>	
Strings	115
<b>CHAPTER 34</b>	
Input	117
<b>CHAPTER 35</b>	
IF-THEN	121
<b>CHAPTER 36</b>	
Alphabetizing	128
<b>CHAPTER 37</b>	
Remarks	130
<b>CHAPTER 38</b>	
READ-DATA	132
<b>CHAPTER 39</b>	
Problem-Solving Programming	140

# CHAPTER 33

## Strings

Until now, the variables we have been using in our programs have had numbers as their values. For example:

`X=42`

A variable like `X` is called a **NUMERIC VARIABLE** because its value or contents is a number. We learned that there are rules to follow when using a numeric variable safely in a program:

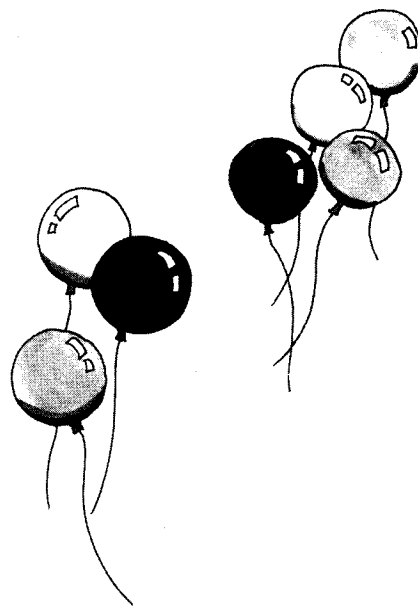
1. Use up to 120 letters and numbers. (It's best to only use one or two letters or numbers.)
2. Use only capital letters.
3. Always begin with a letter.

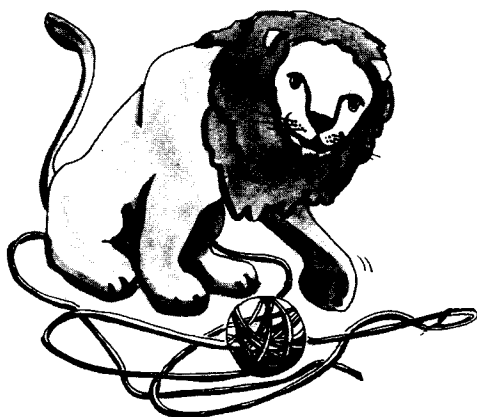
We are now ready to store numbers and letters, words, special characters, and even whole sentences in a variable. This type of variable is called an **ALPHANUMERIC** or **STRING VARIABLE**. A string variable can be written by using any legal variable name (up to 120 characters—capital letters and numbers—beginning with a letter), but the variable must be followed by a **DOLLAR SIGN** (`$`). For example:

`10 LET A$ = "HEY YOU"`

Before we can use string variables, we must set aside a certain number of locations in the computer's memory to be used for the string. We do this by using a **DIMENSION** (abbreviated **DIM**) statement. For example, if `A$` is a string that is to hold the word "hello," we would have to dimension, or reserve, at least 5 spaces before we can use `A$`. We would do so like this: `DIM A$(5)`.

If you do not know for certain how many spaces will be needed, then you have to guess, and dimension an extra large amount. For example, if `N$` is to be used for someone's name, and you don't know how long the name is, simply dimension more spaces than you think could be possibly used—`DIM N$(40)`.





If there are several strings in a program, they can all be dimensioned on a single line at the first of a program by using commas to separate them:

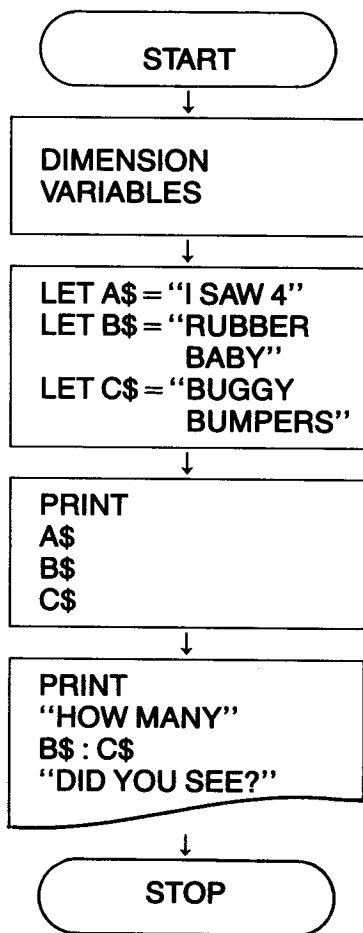
```
10 DIM A$(20), B$(10), C$(15)
```

The contents of a string variable *must* be enclosed in quotation marks, and cannot have more than three screen lines of characters. Letters, punctuation marks, numbers, graphics, or blank spaces may make up the message, and each is counted as one character.

We assign values to a string variable the same way we assign values to a numeric variable—with a LET statement. The only difference is we *must* put quotation marks around the contents of a string variable.

This program shows how you can use strings in programming.

#### Flow chart



#### Program

```

10 DIM A$(20), B$(20), C$(20)
20 LET A$ = "I SAW 4"
30 LET B$ = "RUBBER BABY"
40 LET C$ = "BUGGY BUMPERS"
50 ? A$ : ? B$ : ? C$
60 ? "HOW MANY"
70 ? B$ : ? C$
80 ? "DID YOU SEE?"
90 END
  
```

#### Output

```

I SAW 4
RUBBER BABY
BUGGY BUMPERS
HOW MANY
RUBBER BABY
BUGGY BUMPERS
DID YOU SEE?

READY
  
```

**to do:** Programmer's Pastime #38, #39



# CHAPTER 34

## Input

In our interaction with ATARI so far, we have typed our program on ATARI's keyboard and then sat back and watched it run. The only time we have given information or **INPUT** to ATARI is while we were typing in the program. (Input is anything we put into the computer—either through the keyboard, the cassette tape, or disk drive.)

By using an INPUT statement in your program, you can *interact* with the program while it is running. The INPUT statement lets you type in INPUT or data for your program on the keyboard while the program is running. When you use the INPUT statement, the program becomes an **INTERACTIVE PROGRAM** because you can interact with the computer.

Put an INPUT statement in your program at a point where you want ATARI to stop the program and ask for data or information. When the program is run, ATARI will stop at the INPUT statement and print a ? on the screen to prompt you to answer. At this point you must type the data before ATARI can continue running the program.

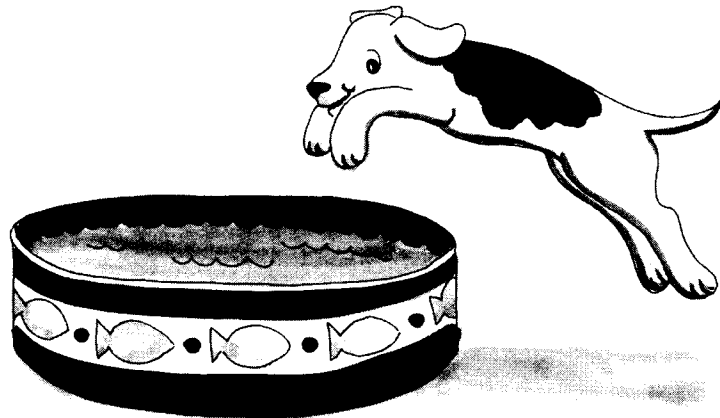
An INPUT statement looks like this:

```
10 INPUT A
```

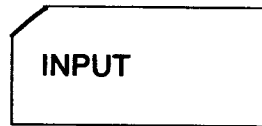
A is the variable where the input numbers will be stored. If the input is to be a word or other alphanumeric data, a string variable must be used:

```
10 INPUT A$
```

“REMEMBER—If ALPHA-NUMERIC data (STRING VARIABLES) are used, the variable must be DIMENSIONED before INPUT can be accepted.”



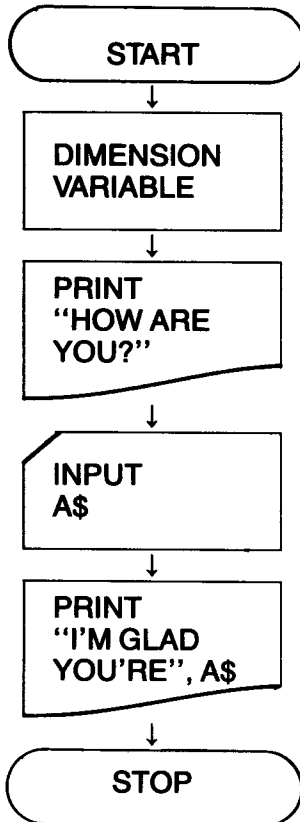
When you develop a flow chart for a program with an INPUT statement, you will use a new shape:



You must write the word "INPUT" inside the box because the box is also used for another statement, which we will learn about later.

This is how an INPUT statement works in a program:

### Flow chart



### Program

```
10 DIM A$(50)
20 ? "HOW ARE YOU?"
30 INPUT A$
40 ? "I'M GLAD YOU'RE", A$
50 END
```

### Output

```
HOW ARE YOU?
?(you type FINE)
I'M GLAD YOU'RE    FINE
READY
☐
```

In this program, ATARI stops at line 30 and prints a ?. After you type in your answer ('Fine'), the program continues running.

Notice that there are two question marks printed—one after “How are you” and another for the INPUT statement. By taking away the ? after “How are you” and using a ; we can make only one question mark appear:

#### Program

```
10 DIM A$(50)
20 ? "HOW ARE YOU";
30 INPUT A$
40 ? "I'M GLAD YOU'RE", A$
50 END
```

#### Output

```
HOW ARE YOU ?
(you type TERRIBLE)
I'M GLAD YOU'RE   TERRIBLE

READY
□
```

An INPUT statement will cause a ? to be printed. That is why we don't need to put a ? at the end of our sentence. In the program above, only one ? is printed, and it is printed on the same line as our question. This is because the ; holds the print zone at the same line as “How are you” and prints the ? on that line.

When we are printing something inside quotation marks and we want it to be followed by the contents of a string, we must be very careful. Using a ; does not leave the first space blank. Look at this program:

#### Program

```
5 DIM P$(10)
10 LET P$ = "JOE"
20 ? "HI"; P$
30 END
```

#### Output

```
HIJOE
READY
□
```

Using a semicolon made the words run together.




If you use a comma, the words will be printed farther apart.

**Program**

```
5 DIM P$(10)
10 LET P$="JOE"
20 ? "HI", P$
30 END
```

**Output**

```
HI      JOE

READY

```

So how do we get the words to be printed with one space in between? There are two ways:

**1**

Leave a blank space at the end of the string or phrase to be printed. `Ø` means blank.

**Program**

```
5 DIM P$(10)
10 LET P$="JOE"
20 ? "HIØ", P$
30 END
```

**Output**

```
HI JOE

READY

```

**2**

Create a new variable which contains a blank.

`B$="Ø"`.

**Program**

```
5 DIM P$(10), B$(2)
10 LET P$="JOE"; LET B$=" "
20 ? "HI", B$, P$
30 END
```

**Output**

```
HI JOE

READY

```

You get a `Ø` (blank space) by pressing the SPACE bar on the keyboard.

**Another shortcut!**

The INPUT and PRINT statements may be combined into one statement:

```
10 ? "What is your name?"
INPUT N$
```

**to do:** Programmer's Pastime #40, #41

# CHAPTER 35

## IF-THEN

In Chapter 12 you learned that computers can do things other than printing numbers and letters and performing arithmetic operations. Computers can:

1. compare numbers and letters      Is 12 greater than 4?  
Does A come before C?
2. make a decision and then do the right task.      IF A=5 THEN PRINT A

You have the skills to set up a flow chart for these types of problems. Now you are ready to write the programs.

You must first understand the signs ATARI will use in making a comparison. Here is a list of the signs and what they mean:

Sign	Meaning	Example
=	equal	$4+5=6+3$
>	greater than	$88>2$
<	less than	$6<46$
>=	greater than or equal to	$33>=32$
<=	less than or equal to	$4<=4$
<>	not equal to	$65<>800$

We use the **IF-THEN** statement when making a comparison in a program. For example:

### IF-THEN Statement

IF A>B THEN ? A

### Meaning

If the value of A is greater than the value of B, then print A.

IF A\$ <= S\$ THEN 20

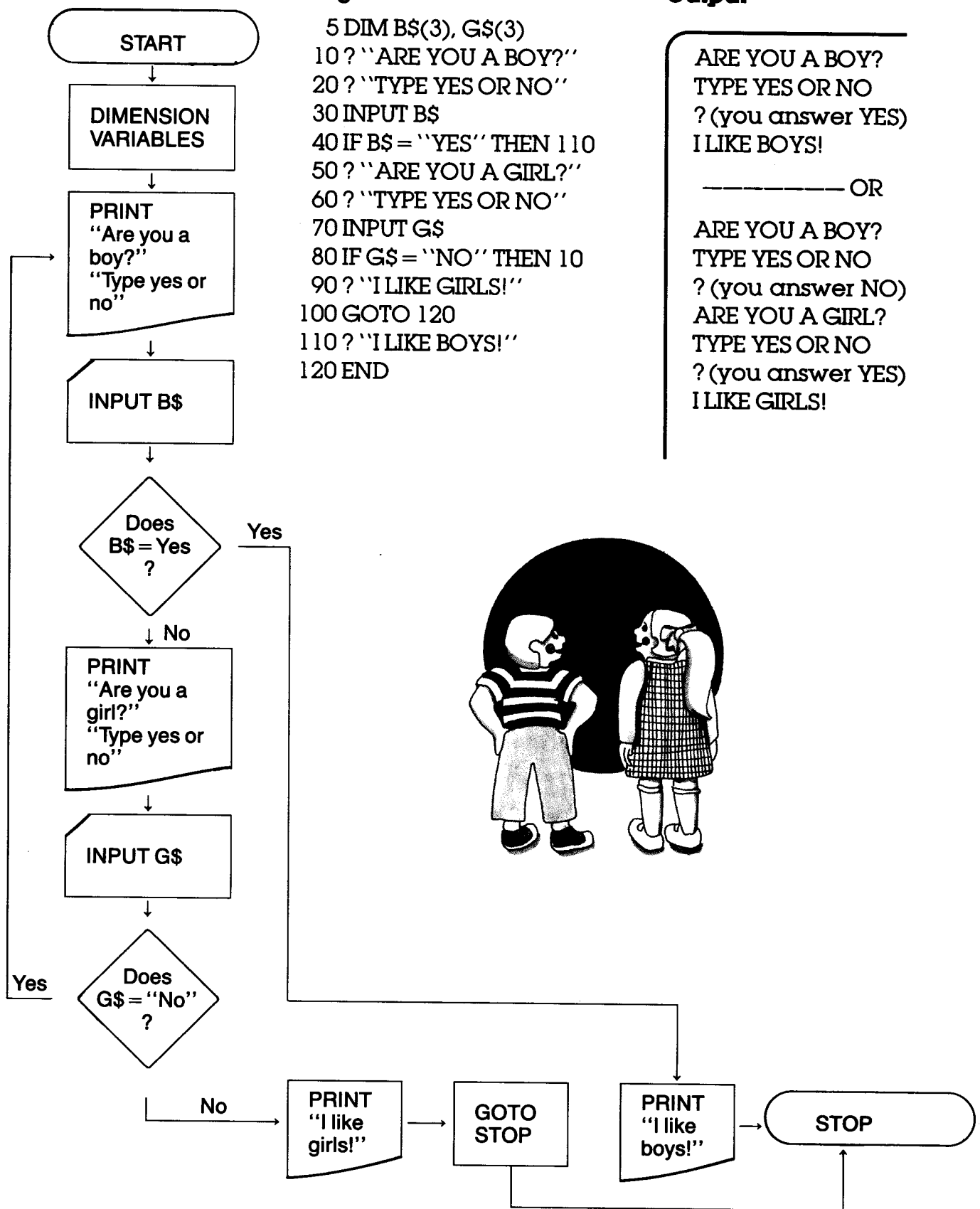
If the contents of A\$ are less than or equal to the contents of S\$, then GOTO line 20 in the program.

Notice that the IF and the THEN are written in the same statement on the same line.

Let's see how the IF-THEN statement can work for us in a program:



## Flow chart



## Program

```

5 DIM B$(3), G$(3)
10 ? "ARE YOU A BOY?"
20 ? "TYPE YES OR NO"
30 INPUT B$
40 IF B$ = "YES" THEN 110
50 ? "ARE YOU A GIRL?"
60 ? "TYPE YES OR NO"
70 INPUT G$
80 IF G$ = "NO" THEN 10
90 ? "I LIKE GIRLS!"
100 GOTO 120
110 ? "I LIKE BOYS!"
120 END
  
```

## Output

ARE YOU A BOY?  
TYPE YES OR NO  
? (you answer YES)  
I LIKE BOYS!

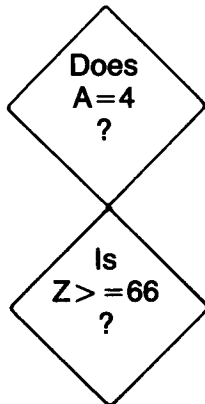
----- OR

ARE YOU A BOY?  
TYPE YES OR NO  
? (you answer NO)  
ARE YOU A GIRL?  
TYPE YES OR NO  
? (you answer YES)  
I LIKE GIRLS!



Notice that we changed the question in a decision box to an IF-THEN statement. For example:

**Flow chart**

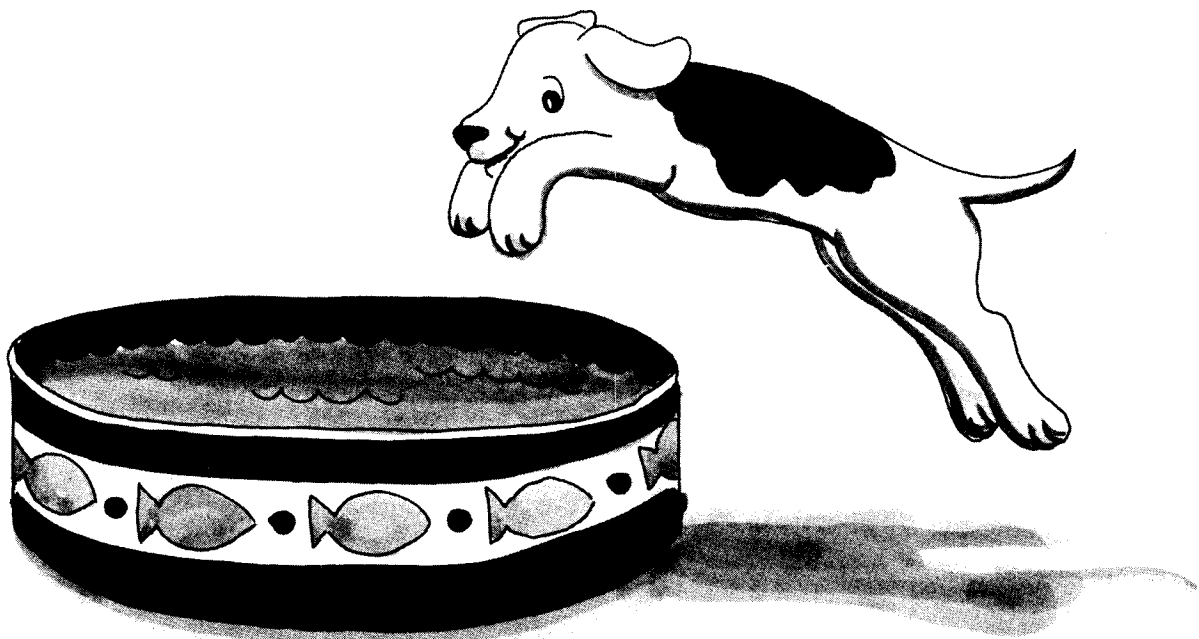


**Program**

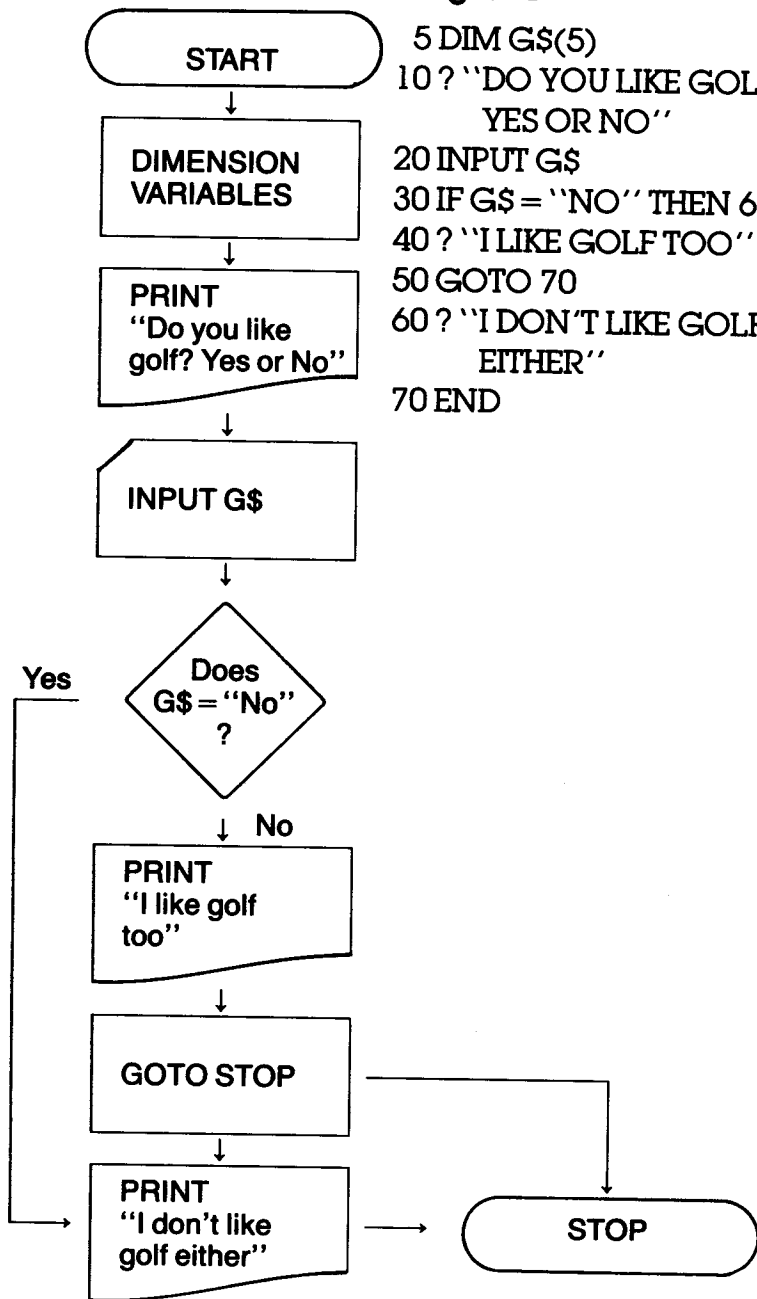
IF A=4 THEN \_\_\_\_\_

IF Z >= 66 THEN \_\_\_\_\_

Study another program:



### Flow chart



### Program

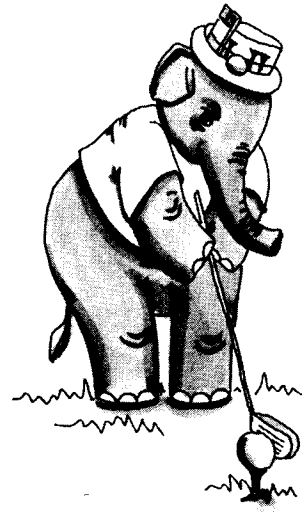
```
5 DIM G$(5)
10 ? "DO YOU LIKE GOLF?
    YES OR NO"
20 INPUT G$
30 IF G$ = "NO" THEN 60
40 ? "I LIKE GOLF TOO"
50 GOTO 70
60 ? "I DON'T LIKE GOLF
    EITHER"
70 END
```

### Output

```
DO YOU LIKE GOLF?
YES OR NO
?(you answer YES)
I LIKE GOLF TOO

----- OR

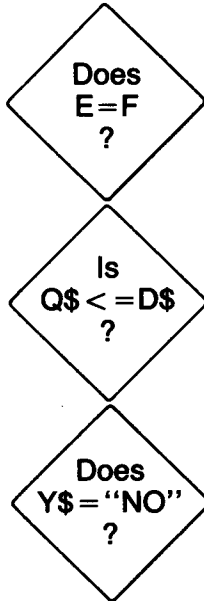
DO YOU LIKE GOLF?
YES OR NO
?(you answer NO)
I DON'T LIKE GOLF EITHER
```





Sometimes we may have to write the **COMPLEMENT**, or opposite, of the question. In this case, we would write the sign that has the opposite meaning. For example:

### Flow chart

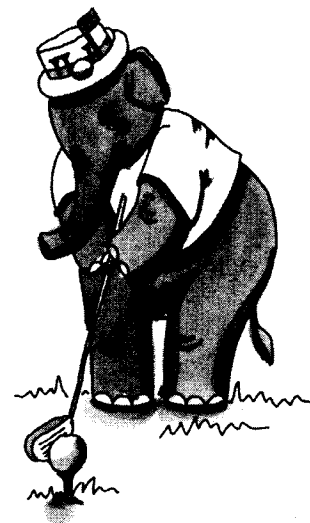


### Complement in the Program

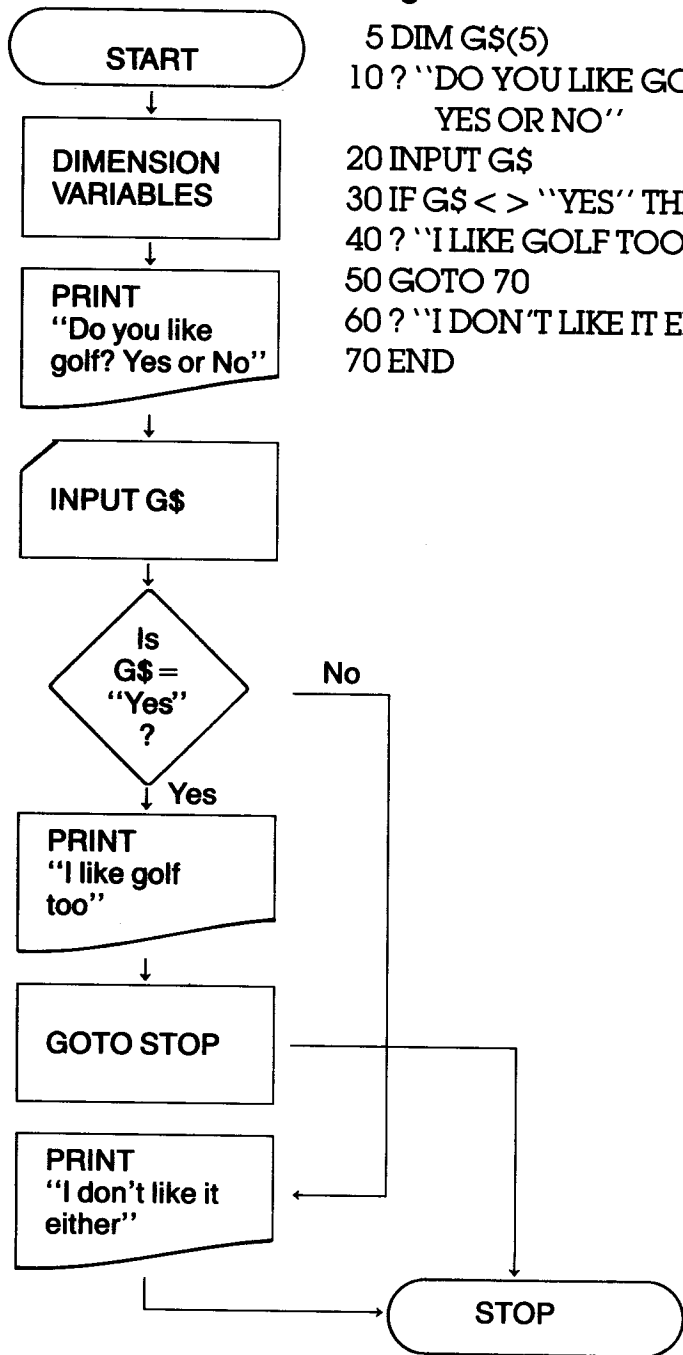
IF E < > F THEN \_\_\_\_\_

IF Q\$ > = D\$ THEN \_\_\_\_\_

IF Y\$ < > "NO" THEN \_\_\_\_\_



### Flow chart



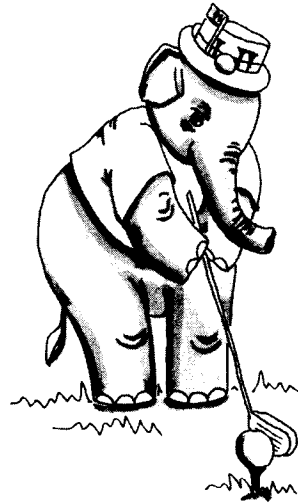
### Program

```
5 DIM G$(5)
10 ? "DO YOU LIKE GOLF?
    YES OR NO"
20 INPUT G$
30 IF G$ < > "YES" THEN 60
40 ? "I LIKE GOLF TOO"
50 GOTO 70
60 ? "I DON'T LIKE IT EITHER"
70 END
```

### Output

```
DO YOU LIKE GOLF?
YES OR NO
? (you answer NO)
I DON'T LIKE IT EITHER

READY
□
```



to do: Programmer's Pastime #42, #43, #44, #45

# PROGRAMMER'S PASTIME #42

Write each equation as an IF-THEN statement.

## Question

## IF-THEN Statement

Example: Is A equal to C?

IF  $A = C$  THEN \_\_\_\_\_

1. Is L\$ equal to "MAYBE"?

\_\_\_\_\_

2. Is F1 not equal to FZ?

\_\_\_\_\_

3. Is GH greater than HI?

\_\_\_\_\_

4. Is S\$ less than or equal to F\$?

\_\_\_\_\_

5. Is X times B less than P times Q?

\_\_\_\_\_

6. Is T divided by W greater than or equal to W times B?

\_\_\_\_\_

7. Is P\$ greater than M\$?

\_\_\_\_\_

8. Is the square root of Y equal to D?

\_\_\_\_\_

9. Is G\$ not equal to "NO"?

\_\_\_\_\_

10. Is 10 divided by 5 less than 14 divided by 2?

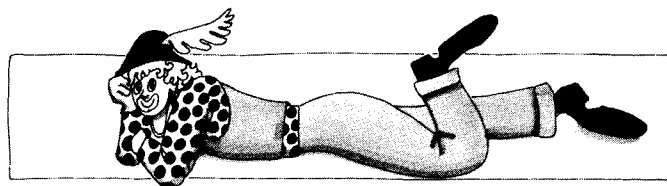
\_\_\_\_\_

11. Is Y\$ equal to the square root of 64?

\_\_\_\_\_

12. Is A plus B greater than D\$?

\_\_\_\_\_



# CHAPTER 36

## Alphabetizing

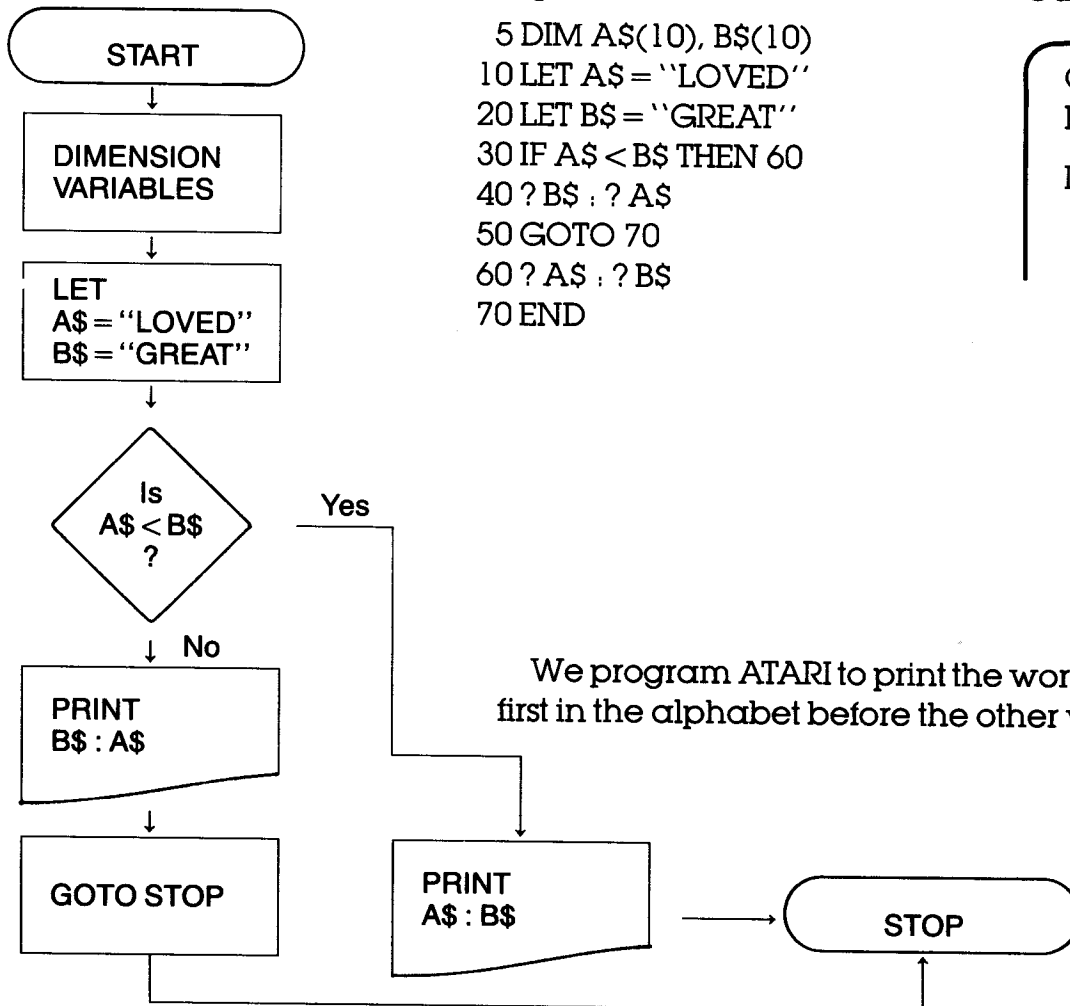
Did you know that ATARI has the ability to compare letters in string variables and alphabetize the words? ATARI already understands that:

"A" < "B" < "C" . . . < "Y" < "Z"

"A" is less than "B" which is less than "C" which is less than "D," and so on, all the way to "Z." In other words, "A" is the smallest letter, and "Z" is the largest letter in the alphabet. A word that begins with "A" is smaller than a word that begins with "Z."

Keeping this in mind, we can write a short program to alphabetize two words. For example:

### Flow chart



### Program

```
5 DIM A$(10), B$(10)
10 LET A$ = "LOVED"
20 LET B$ = "GREAT"
30 IF A$ < B$ THEN 60
40 ? B$ : ? A$
50 GOTO 70
60 ? A$ : ? B$
70 END
```

### Output

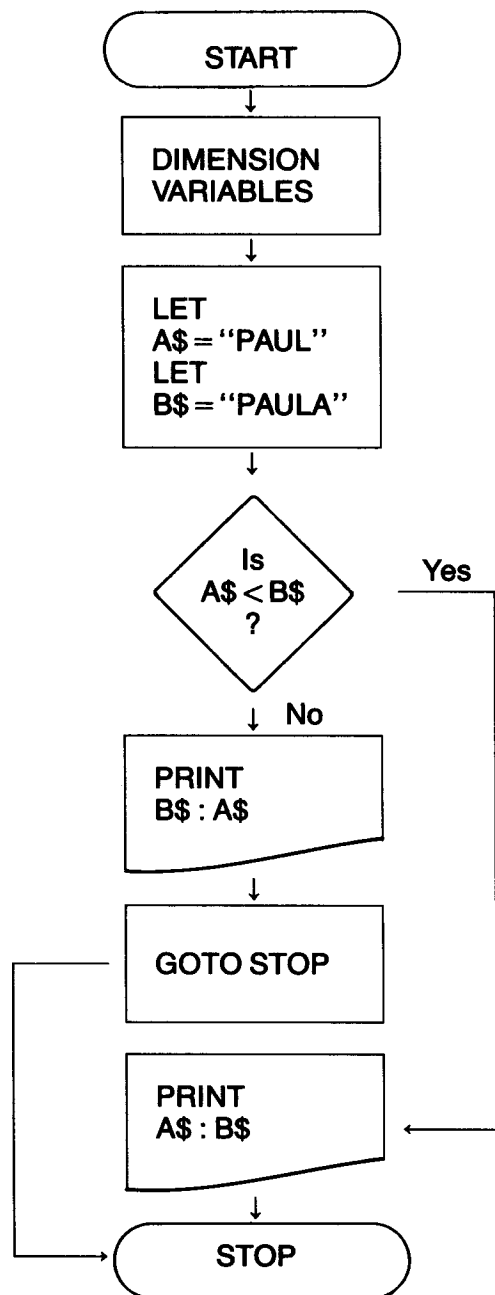
```
GREAT
LOVED

READY
□
```

We program ATARI to print the word that comes first in the alphabet before the other word.

ATARI can also alphabetize words that have the same letters—like Paula and Paul. Both words begin with P-A-U-L, but Paula has an extra letter at the end. The rule for this situation is that the shortest word comes first. ATARI knows this rule too. For example:

### Flow chart



### Program

```

5 DIM A$(10), B$(10)
10 LET A$ = "PAUL"
20 LET B$ = "PAULA"
30 IF A$ < B$ THEN 60
40 ? B$ : ? A$
50 GOTO 70
60 ? A$ : ? B$
70 END
  
```

### Output

```

PAUL
PAULA
READY
  
```

Using this type of algorithm, it is only worthwhile to alphabetize two words. If we needed to alphabetize more than two words, we would have to use a different type of algorithm, which will be considered later.

**to do:** Programmer's Pastime #46

# CHAPTER 37

## Remarks

As you begin writing more complicated programs, you will want to make sure they can be easily read and understood by others who may read them. Writing your programs so they are easy to read is good programming **STYLE**.

One style technique is the use of **REMARK** statements in your program. This is also called **DOCUMENTATION**, which means noting what is happening in your program. For example:

```
10 REM ADD TWO NUMBERS    REM stands for  
                           REMARK.
```

```
20 LET D=4 : LET C=5  
30 ? D+C  
40 END
```

Each REMARK statement describes the purpose of the statements following it. Lines 20 and 30 in the program add two numbers, so the REMARK statement in line 10 should say:

```
REM ADD TWO NUMBERS.
```

When the program is run, ATARI will ignore the REM statement. REM tells ATARI to ignore what is written after it and to go on to the next line number.

REMARK statements will not show up when you run the program because they are ignored. The REM statements only show up when you list the program.

Use REMARK statements at the beginning of your program to tell the name of the program or what it does. You can also add that you are the author of the program. For example:

```
10 REM SPACE ATTACK  
20 REM TRY TO SHOOT DOWN THE ALIENS  
30 REM WRITTEN BY JOE COOL, 1980
```



It is also helpful to use REMARK statements to describe each main section of your program. For example:

### Program

```
10 REM MATH PROGRAM
20 REM WRITTEN BY CHARLIE
   BROWN, 1981
30 REM MULTIPLICATION
40 ? ``5*5=``;
50 INPUT M
60 IF M=25 THEN 90
70 ? ``WRONG. TRY AGAIN``
80 GOTO 40
90 ? ``RIGHT!``
100 REM DIVISION
110 ? ``50/10=``;
120 INPUT D
130 IF D=5 THEN 160
140 ? ``WRONG. TRY AGAIN``
150 GOTO 110
160 ? ``RIGHT!``
170 END
```

### Output

```
5*5= ?(you answer 22)
WRONG. TRY AGAIN
5*5= ?(you answer 25)
RIGHT!
50/10= ?(you answer 12)
WRONG. TRY AGAIN
50/10= ?(you answer 5)
RIGHT!

READY
☐
```

In the above program, we used REM statements to:

1. introduce the program;
2. show the beginning of the multiplication section of the program;
3. show the beginning of the division section of the program.

Be careful that you don't use too many REMARK statements in your programs. Too many can clutter up the program and make it more difficult to read. They could also waste space and use up ATARI's memory.

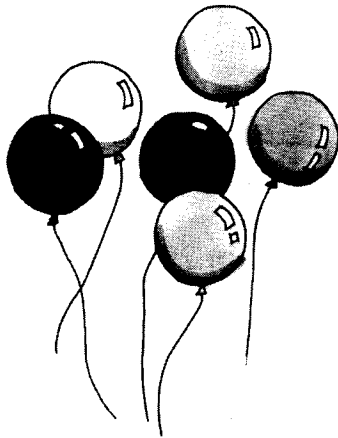
As you practice writing programs, you will become more aware of where REMARK statements should be placed.

REM  
stands for  
REMARK

**to do:** Programmer's Pastime #47

# CHAPTER 38

## READ-DATA

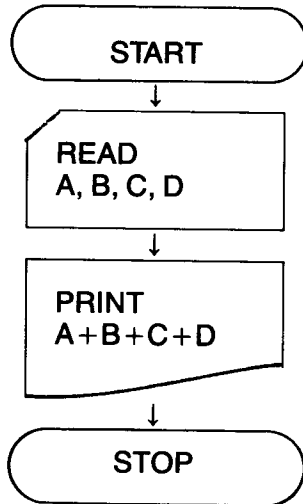


Another programming trick which can save you and the computer time is the use of **READ-DATA** statements.

The READ statement and the DATA statement go together in a program. These two statements make it possible for you to place data in your program as you type it on the keyboard, or even while you are running the program.

This is handy because you can use the same program many times. Instead of writing and typing the program over again for different data, you merely change the information in the DATA statement. This program adds four numbers:

### Flow chart



### Program

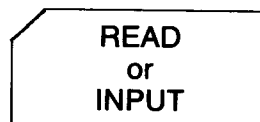
```
10 READ A, B, C, D
20 DATA 6, 7, 8, 9
30 ? A+B+C+D
40 END
```

### Output

```
30
READY
□
```

If you want to use the same program to add four different numbers, just change the DATA statement in line 20:

```
20 DATA 10, 11, 12, 13
```

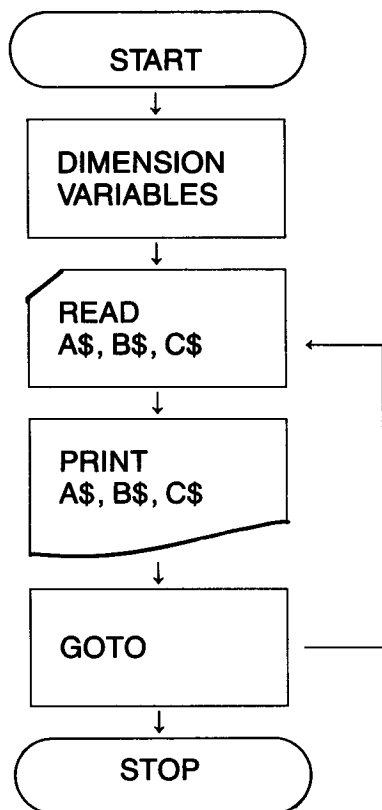


The READ box in our flow chart looks just like the INPUT box. You must label the box as READ or INPUT so it is not confused.



You can also add more data to the DATA statement:

### Flow chart



### Program

```

5 DIM A$(10), B$(10), C$(10)
10 READ A$, B$, C$
20 ? A$, B$, C$
30 GOTO 10
40 DATA "I", "LIKE", "YOU", "YOU'RE", "MY", "FRIEND"
50 END
  
```

### Output

```

I           LIKE      YOU
YOU'RE      MY        FRIEND

ERROR—6 AT LINE 10
  
```



In line 10, ATARI is told to READ enough data to fill up the three variables A\$, B\$, and C\$. SO ATARI looks for a DATA statement in the program and finds one in line 40. It "gobbles up" the first three pieces of data it finds ("I", "LIKE", "YOU") and assigns them to A\$, B\$, and C\$.

```
10 READ      A$,      B$,      C$
```

```
40 DATA     "I"      , "LIKE"  , "YOU"
```

A\$
I

B\$
LIKE

C\$
YOU

You can think of this data as being used up.

In line 20, ATARI prints the contents of A\$, B\$, and C\$. Line 30 tells ATARI to go back to line 10 and read three new pieces of data. The Apple finds "YOU'RE", "MY", "FRIEND" in the DATA statement and again assigns them to A\$, B\$, and C\$. These are the new values of A\$, B\$, and C\$. "I", "LIKE", and "YOU" have been erased from ATARI's memory.

```

10 READ      A$,      B$,      C$
.
.
.
40 DATA...  "YOU'RE" ,  "MY"   , "FRIEND"

```

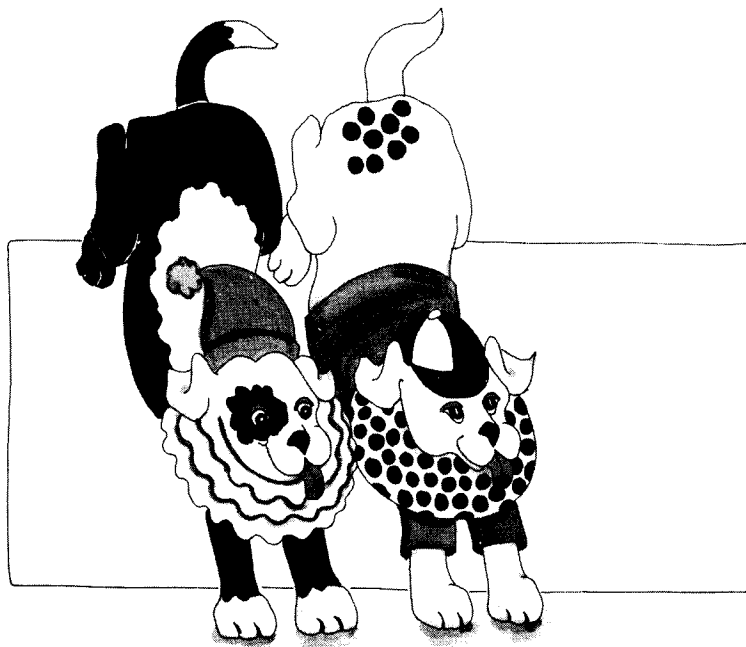
In line 20, ATARI prints the new contents of A\$, B\$, and C\$. Line 30 tells ATARI to GOTO line 10. Since there is no more "fresh" data in the DATA statement, ATARI will print

ERROR—6 AT LINE 10

at the end of the output. This is ATARI's way of saying, "There's no more data to read into A\$, B\$, and C\$!" (See Appendix B.)

A DATA statement may be up to three screen lines long. This means that it can have up to 112 characters (including spaces, commas, and the word DATA itself). If you have more data than can fit in three screen lines, you will need to use more than one DATA statement.

ATARI treats all of the data in a program as one big list. The READ statement has a "pointer" that goes through this data list and "gobbles up" any new data.



The DATA statement can be placed anywhere in a program. There is only one thing you must watch out for. You *must* have the data in your DATA statement in the correct order. For example, if you want the program to print:

HI THERE PAL

the data "HI", "THERE", and "PAL" must be in this order in the DATA statement. If they are out of order, this is what might happen:

#### Program

```
5 DIM L$(10), M$(10), N$(10)
10 READ L$, M$, N$
20 ? L$, M$, N$
30 DATA "PAL", "THERE", "HI"
40 END
```

#### Output

```
PAL      THERE      HI
READY
  □
```

You must also make sure that the data in your DATA statement is separated by commas. Any data not separated by commas will be lumped together as one piece of data by ATARI. Also, be sure to dimension your variables if you are using alphanumeric data.

If you have three variables to READ, ATARI will "gobble up" data in groups of three. Any leftovers will not be printed. For example:

#### Program

```
10 READ X, Y, Z
20 ? X, Y, Z
30 GOTO 10
40 DATA 2, 4, 6, 8, 10, 12, 13, 14
50 END
```

#### Data used

1st time: 2, 4, 6  
2nd time: 8, 10, 12  
left over: 13, 14

#### Output

```
2      4      6
8      10     12
```

ERROR—6 AT LINE 10

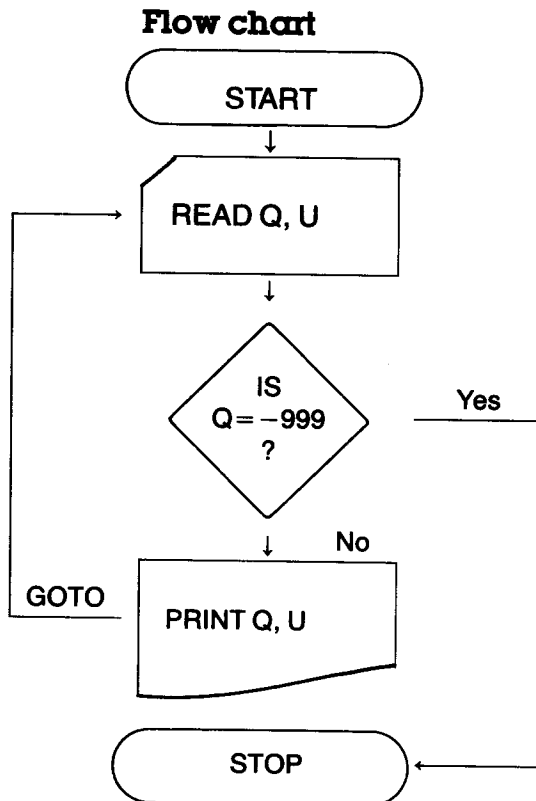


If the data in your DATA statement is not in order, it can really mess up your program.



You may ask, "Is there any way I can get the program to end without printing the ERROR-6 message?" The answer is "Yes!" You need to:

1. Put some **DUMMY** data at the end of your DATA statement. (Dummy data is data you want ATARI to read as a signal that the pointer is at the end of the data list.)
2. Use an IF-THEN statement which directs ATARI to the end of the program as soon as it READS the dummy data.



**Program**

```

10 DATA 48, 6, 8.5, 9,
   -999, -999
20 READ Q, U
30 IF Q = -9999 THEN 60
40 ? Q, U
50 GOTO 20
60 END
  
```

**Output**

```

48      6
8.5     9

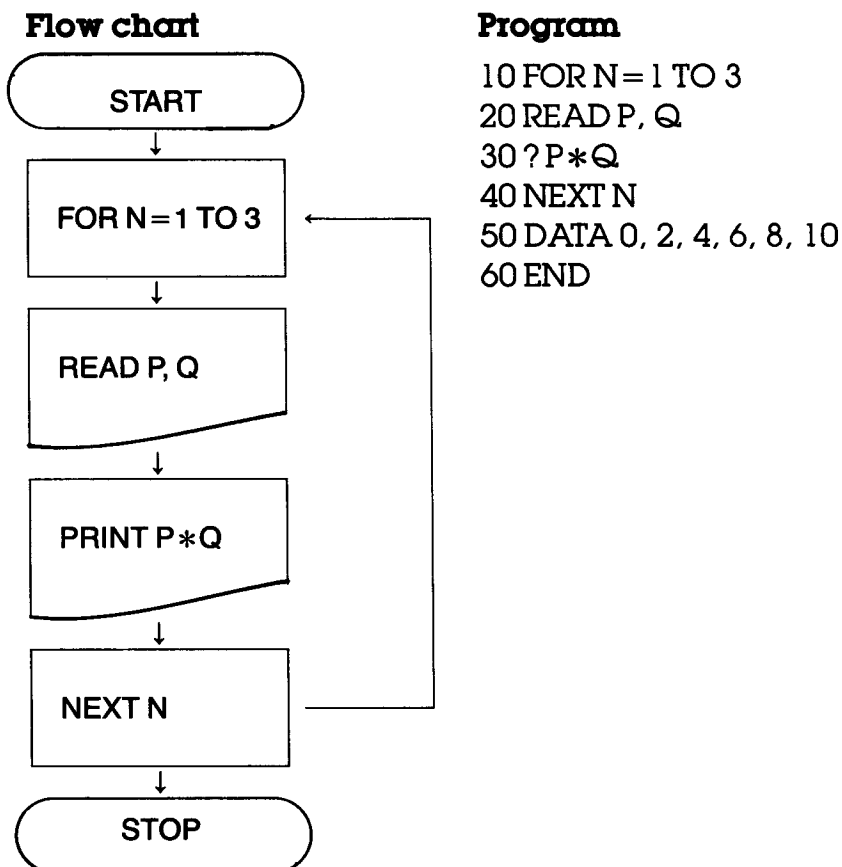
READY
  
```

In the program we used -999 as our dummy data. When you choose your dummy data, select something you know you probably won't be using for data. For example, it is very unlikely that -999 would be data that we would want to use in a program.

The IF-THEN statement in line 30 of the program asks, "Does Q = -999?" after each "gulp" of data is read. When Q = -999, ATARI is directed to the END of the program.

It is important to have dummy data for each variable that ATARI will read. For example, if you have five variables in your READ statement, you must have five pieces of dummy data at the end of the DATA statement. Each variable must have data READ into it every time or ATARI will print the ERROR—6 message at the end of your program's output.

**Another way:** If you don't want to use dummy data in your program, you can use a FOR-NEXT loop.



### Output

```
0
24
80
READY
☐
```

The FOR-NEXT statements cause the program to loop three times. During the first loop, 0 and 2 are READ into P and Q. In the second loop, ATARI READS 4 and 6, and during the third loop 8 and 10 are READ. Since the loop is only done three times, the computer goes to line 60 and the program ends.



## Two More Things

- 1** It's OK to have both numeric *and* string variables in the same READ statement. Just make sure any data in the DATA statement that go with the string variables are in the correct order, and that they have been dimensioned.

Example: 5 DIM D\$(20), F\$(20)

```
10 READ C, D$, E, F$
```

```
20 DATA 10 , KEN , 3 , MITSY
```

If you try to put alphanumeric data in a numeric variable, you will get an ERROR—8 statement. However, you will not get an error if you put numeric data in an alphanumeric (string) variable.

- 2** You can't have an equation like  $5 \div 2$  as data in a numeric variable or ATARI will print an error message. You can, however, list an equation such as  $5 + 2$  as part of alphanumeric (string) data.

to do: Programmer's Pastime #48, #49, #50, #51, #52



# PROGRAMMER'S PASTIME #50

READ-DATA statements can help you write shorter programs. Rewrite each program using READ-DATA statements to shorten them. Try to write each program so you don't get an error message.

## Long Program

## Short Program

1. 10 ? "MULTIPLYING 2 NUMBERS"  
20 LET P=60  
30 LET Q=129  
40 LET R=410  
50 LET S=.6  
60 ? P, Q, P\*Q  
70 ? R, S, R\*S  
80 END
  
2. 5 DIM A\$(10), B\$(10), C\$(10), D\$(10), E\$(10)  
10 ? "TEST SCORES"  
20 ? "NAME", "SCORE"  
30 LET A\$="JOE"  
40 LET A=98  
50 LET B\$="TOM"  
60 LET B=52  
70 LET C\$="KRIS"  
80 LET C=95  
90 LET D\$="GAIL"  
100 LET D=75  
110 LET E\$="BOB"  
120 LET E=72  
130 ? A\$, A  
140 ? B\$, B  
150 ? C\$, C  
160 ? D\$, D  
170 ? E\$, E  
180 END



By now you have discovered that ATARI is a friend who can keep you company when you are bored, entertain you, and help you do your work. The most important thing ATARI can do for you, however, is to help you solve difficult problems.

So far you have learned how to program ATARI to do many things. You have learned most of the BASIC commands and algorithms necessary to write problem-solving programs. In this chapter, you will learn how to put all of these valuable tools to use in order to teach ATARI to solve some difficult problems.

Before ATARI can give you the answer to a problem, there are many things that you must plan for in writing a good program. For example:

### Problem

Joe went to the store to buy some goldfish. He has \$4.83 to spend. The fish bowl costs \$2.25. Sand for the bottom of the bowl costs 49¢ a bag. Fish food is 60¢ for 4 ounces. The goldfish cost 80¢ a piece or two for \$1.35. If Joe buys all of the supplies, how many fish can he afford to buy?



1. THINK about the problem
  - a. What exactly *is* the problem?  
*Can Joe buy 1 or 2 goldfish?*
  - b. Do I understand the problem?
  - c. What kind of answer do I want?  
*The answer should be 1 or 2 goldfish.*
  - d. What do I need to know in order to find out the answer?  
*I need to know how much money Joe will have left over after buying the supplies.  
Then I will know how many fish he can buy.*



## 2. Make a DATA TABLE

- a. What variables will I need to use in the program and what will they stand for?
- b. INPUT VARIABLES are variables that you already know the value of.

### Data Table

#### Input Variables

T=total \$ that Joe can spend	=4.83
FB=cost of fish bowl	=2.25
S=cost of sand	=.49
FF=cost of fish food	=.60
G1=cost of 1 goldfish	=.80
G2=cost of 2 goldfish	=1.35

- c. OUTPUT VARIABLES are the answers that ATARI will give you.

#### Output Variables

TC=total cost of FB+S+FF

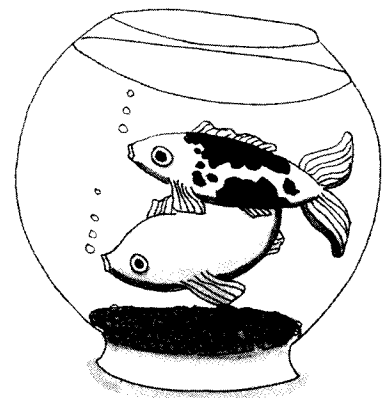
L=money left after buying the supplies

- d. PROGRAM VARIABLES are any other variables that are used in the program to do other things.

*There are no PROGRAM VARIABLES in this program.*

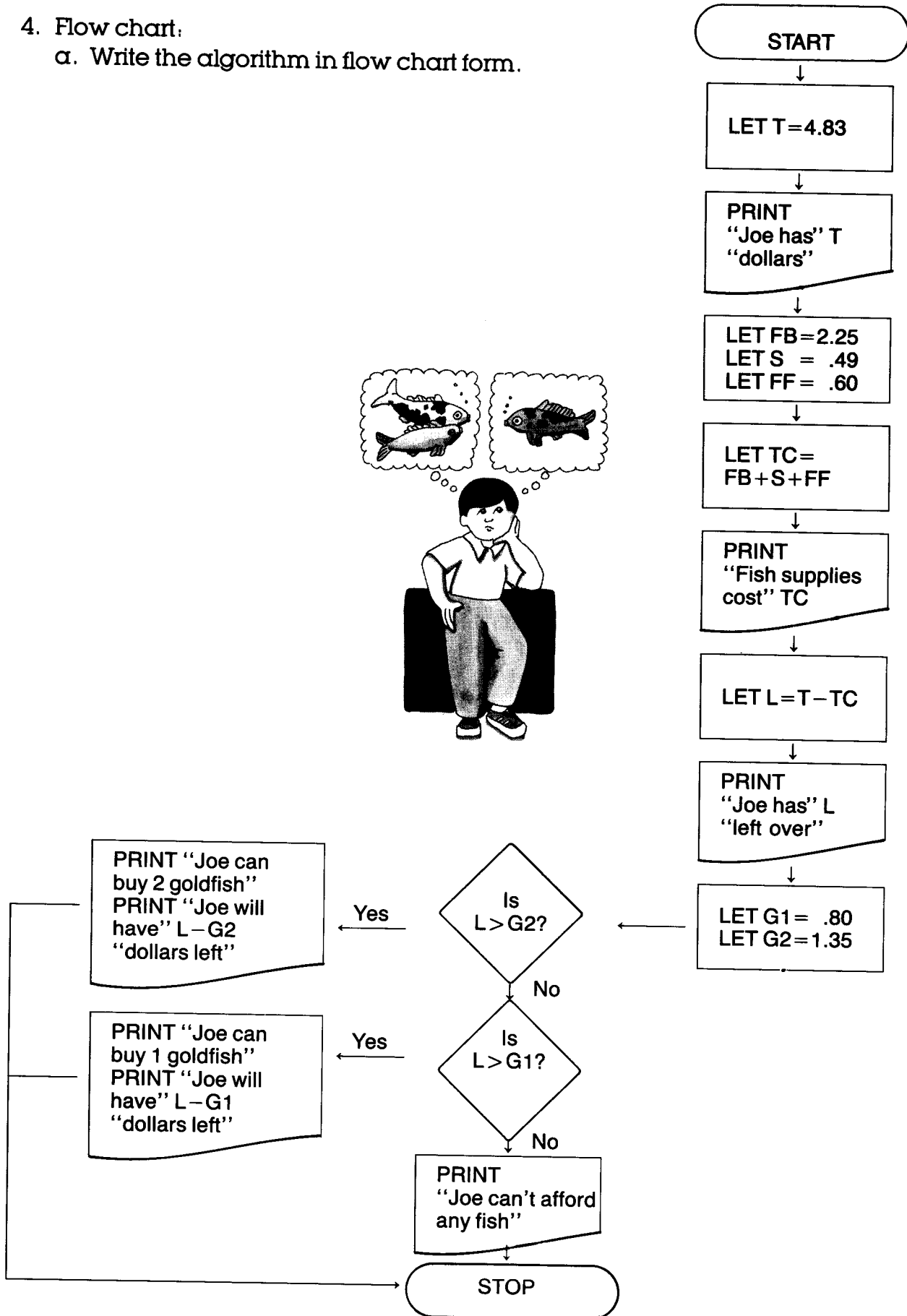
## 3. ALGORITHM

- a. Break the problem into smaller parts.
- b. Figure out the step-by-step process you will use to solve the problem. Decide what operations you will use (+, -, /, and so on).
  1. Find out the TC by adding  $FB + S + FF$ .
  2. Find out L by subtracting  $T - TC$ .
  3. Find out if L is enough to buy one or two goldfish. Ask:  
 $Is L \geq G1?$   
 $Is L \geq G2?$
  4. Tell how many goldfish Joe can buy and how much money he would have after buying both the supplies AND the goldfish.



4. Flow chart:

a. Write the algorithm in flow chart form.



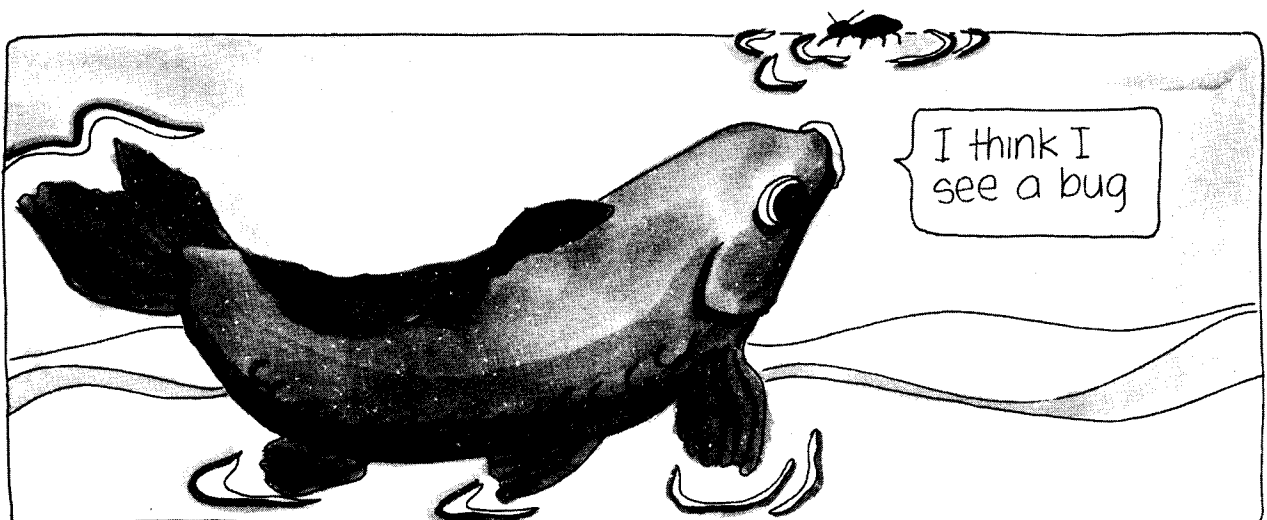
## 5. CODING

a. Write a BASIC program for the flow chart.

```
10 LET T=4.83
20 ? "JOE HAS"; T, " DOLLARS"
30 LET FB=2.25 : LET S=.49 : LET FF=.60
40 LET TC=FB+S+FF
50 ? "FISH SUPPLIES COST"; TC
60 LET L=T-TC
70 ? "JOE HAS"; L, " LEFT OVER"
80 LET G1=.80 : LET G2=1.35
90 IF L >= G2 THEN 130
100 IF L >= G1 THEN 150
110 ? "JOE CAN'T AFFORD ANY FISH"
120 GOTO 160
130 ? "JOE CAN BUY 2 GOLDFISH" : ? "AND
    HAVE"; L-G2, " DOLLARS LEFT"
140 GOTO 160
150 ? "JOE CAN BUY 1 GOLDFISH" : ? "AND
    HAVE"; L-G1, " DOLLARS LEFT"
160 END
```

## 6. DEBUGGING

- Pretend you are a computer. Follow the directions in your program to make sure it works. This is called **TRACING** the program.
- Run the program on ATARI to check for bugs.
- Does the program do what you wanted it to do?



## 7. REVISING

- a. Is there a better or shorter way to write your program?

*Yes. We can write the program using READ-DATA statements.*

- b. Can you use better programming style?

*Yes. We can use REMARKS.*

- c. Can you design your output better?

*Yes. We can clear the screen and leave spaces between the printing.*

### 10 REM CALCULATING PURCHASE OF GOLDFISH & SUPPLIES

20 ? " 

ESC
-----

SHIFT
-------

CLEAR
-------

 "

30 READ T, FB, S, FF, G1, G2

40 DATA 4.83, 2.25, .49, .60, .80, 1.35, 99,  
99, 99, 99, 99, 99

50 IF T=99 THEN 60

60 ? "JOE HAS"; T, "DOLLARS"

70 LET TC=FB+S+FF

80 ?

90 ? "FISH SUPPLIES COST"; TC

100 LET L=T-TC

110 ?

120 ? "JOE HAS"; L, "LEFT OVER"

130 IF L >= G2 THEN 170

140 IF L >= G1 THEN 190

150 ? "JOE CAN'T AFFORD ANY FISH"

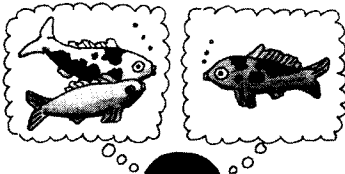
160 GOTO 200

170 ? "JOE CAN BUY 2 GOLDFISH AND HAVE  
"; L-G2, "DOLLARS LEFT"

180 GOTO 200

190 ? "JOE CAN BUY 1 GOLDFISH AND HAVE  
"; L-G1, "DOLLARS LEFT"

200 END



---

Using the READ-DATA statements may be the best way to write this program. Why? If the price of goldfish or supplies goes up, you can change the DATA statement and the program will be updated.

You can write good problem-solving programs for ATARI to solve if you follow these 7 steps:

1. THINK about the problem.
2. Make a DATA TABLE for input, output, and program variables.
3. Create an ALGORITHM—"How can I solve the problem, step by step?"
4. Make a FLOW CHART.
5. CODE the flow chart into a BASIC program.
6. DEBUG.
7. REVISE the program to make it the best!

**to do:** programmer's Pastime #53  
Component 6 Fun Page  
Evaluate yourself



# PROGRAMMER'S PASTIME #53

Use the problem-solving approach to get ATARI to solve the following problems.

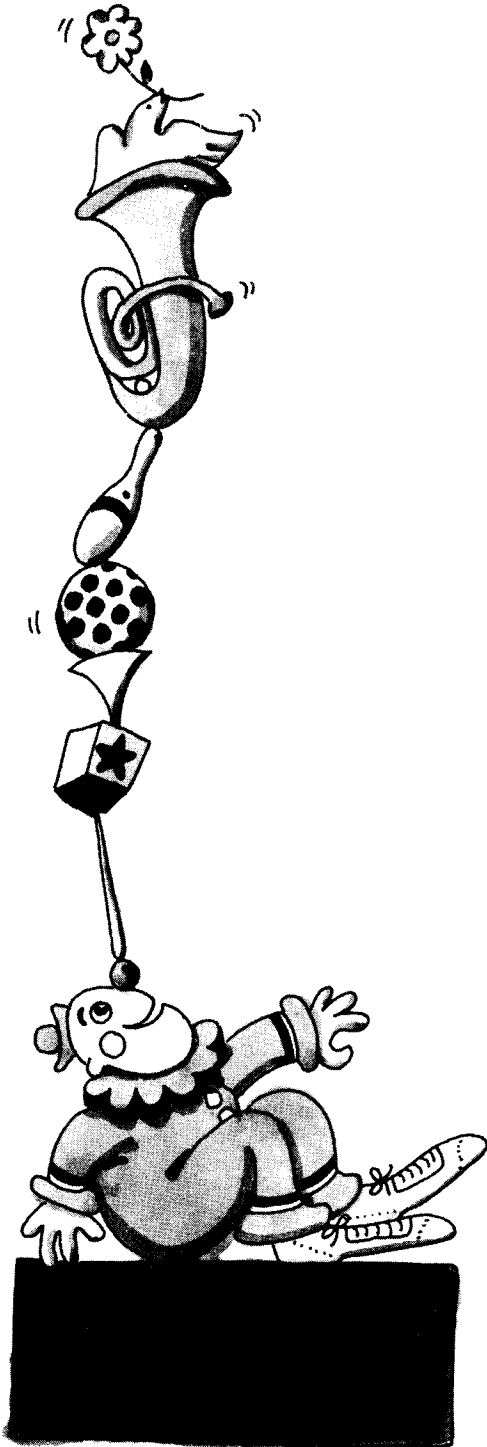
## Problem 1

The teacher gave your class a test on programming the computer. The test scores were:

Jill Jarvis	73%	Your teacher needs to know the average test score.
Katie O'Keefe	98%	
Tommy Templeton	67%	
Susie Sunbeam	82%	
You	90%	

Write a program that tells ATARI to calculate and print the average score. (HINT: To find the average of 5 numbers, add them together and divide by 5.)

1. THINK about the problem.
2. Make your DATA TABLE here.
3. Write the ALGORITHM (steps and equations)
4. FLOW CHART
5. CODE the program
6. DEBUG
7. REVISE



---

## Problem 2

You are the new manager of the "Peppy Pizza" restaurant and you need the help of a computer. Write a program that will allow you to INPUT the number of small, medium, and large pizzas sold during a day.

Have ATARI print out the total number of pizzas sold and how much money you made.

(OUTPUT HINT: HOW MANY PIZZAS:

(SMALL, MEDIUM, LARGE)

? \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_

THERE WERE \_\_\_\_\_ PIZZAS SOLD TODAY.

"PEPPY PIZZA" MADE \$\_\_\_\_\_.)

### Prices:

small	\$4.30
medium	\$5.50
large	\$7.25

1. THINK about the problem.
2. DATA TABLE
3. ALGORITHM
4. FLOW CHART
5. CODE the program
6. DEBUG
7. REVISE

# COMPONENT 7

## CHAPTER 40

Conversions 149

## CHAPTER 41

Random Numbers and Integers 152

## CHAPTER 42

Making Sounds 156

## CHAPTER 43

Graphics 160

## CHAPTER 44

More Graphics 170

## CHAPTER 45

Writing Game Programs 173

## CHAPTER 46

YOU are a Creative Programmer! 174



# CHAPTER 40

## Conversions

ATARI can be especially good at running a program which helps you **CONVERT** one thing to another. **CONVERT** means to *change*, so a **CONVERSION** is changing information to a different type. For example, we can convert:

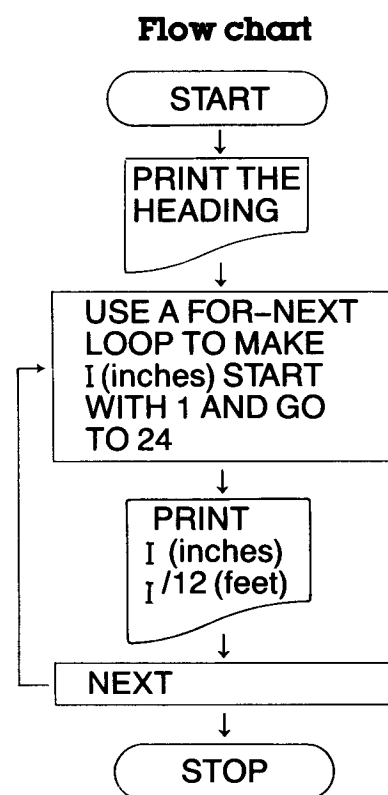
inches to feet

decimals to fractions

feet to meters

miles to kilometers

You can program ATARI to make the conversion, or change, and then print a table that shows how the two types of conversions are equal to each other. For example:



### Program

```
10 REM CONVERTING  
  INCHES TO FEET  
20 ? "INCHES", "FEET"  
30 ?  
40 FOR I=1 TO 24  
50 ? I, I/12  
60 NEXT I  
70 END
```

### Output

INCHES	FEET
1	0.083333
2	0.16666666
3	0.25
4	0.333333
5	0.41666666
6	0.5
7	0.583333
8	0.66666666
9	0.75
10	0.833333
11	0.91666666
12	1
.	.
.	.
.	.
24	2

READY



The output of this program shows how inches compare to feet. You can tell from the program that:

1 inch = 0.083333 of a foot  
3 inches = 0.25 or  $\frac{1}{4}$  of a foot  
6 inches = 0.5 or  $\frac{1}{2}$  of a foot  
12 inches = 1 foot

Line 20 prints the **HEADING** for the output. The heading of a program is usually printed first in the output. It explains the meaning of the numbers that follow. The heading in our program is:

INCHES	FEET
--------	------

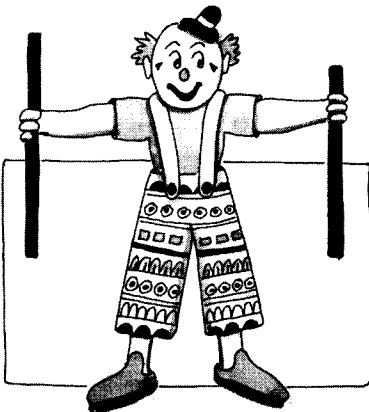
It tells us that the numbers listed under "INCHES" are inches, and the numbers listed under "FEET" are feet.

Conversion programs are very easy to write. They are short because they use a FOR-NEXT loop. The most important part of the program is the **CONVERSION EQUATION**. This equation tells ATARI how to convert from one thing to another. The conversion equation in our inches-to-feet program is  $I/12$ . This tells ATARI that to find feet, ATARI must divide the number of inches (I) by 12.

To write a good conversion program, remember to include:

1. a **HEADING**,
2. a **FOR-NEXT** loop that decides which numbers to start and end with on the conversion output and how many times the program will loop;
3. A **CONVERSION EQUATION**.

**to do:** Programmer's Pastime #54, #55, #56



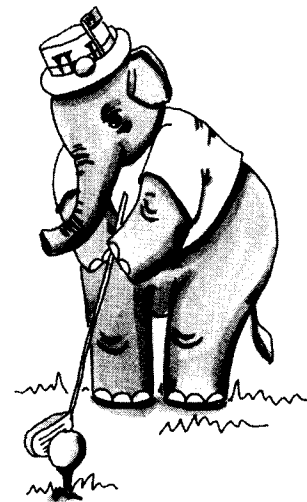
# PROGRAMMER'S PASTIME #55

Write a conversion program for each problem. Make sure your program has a heading, a FOR-NEXT loop, and a conversion equation. Run your programs on ATARI to check for bugs.

**Problem**                      **Program**

**Problem**                      **Program**

1. Convert 1-20 inches to centimeters.  
CONVERSION EQUATION:  
 $\text{Centimeters} = I * 2.5$
2. Convert 1-20 kilometers to miles.  
CONVERSION EQUATION:  
 $\text{Miles} = K / 1.6$
3. Convert 1-20 pounds to grams.  
CONVERSION EQUATION:  
 $\text{Grams} = P * 454$
4. Convert 1-10 liters to quarts.  
CONVERSION EQUATION:  
 $\text{Quarts} = L / 3.8$
5. Convert 0° to 100° Fahrenheit to ° Celsius  
CONVERSION EQUATION:  
 $^{\circ} C = 5 * (F - 32) / 9$
6. Convert 1-100 pounds to kilograms  
CONVERSION EQUATION:  
 $\text{Kilograms} = P * .45$



# CHAPTER 41

## Random Numbers and Integers

The word **RANDOM** means "having no pattern or special purpose." Therefore, **RANDOM NUMBERS** would be a list of numbers that are not in any particular order. An example of a list of random numbers would be: 7, 43, -6, .7, 413. There is no order or number pattern in this list, and the numbers listed have no special purpose or meaning.

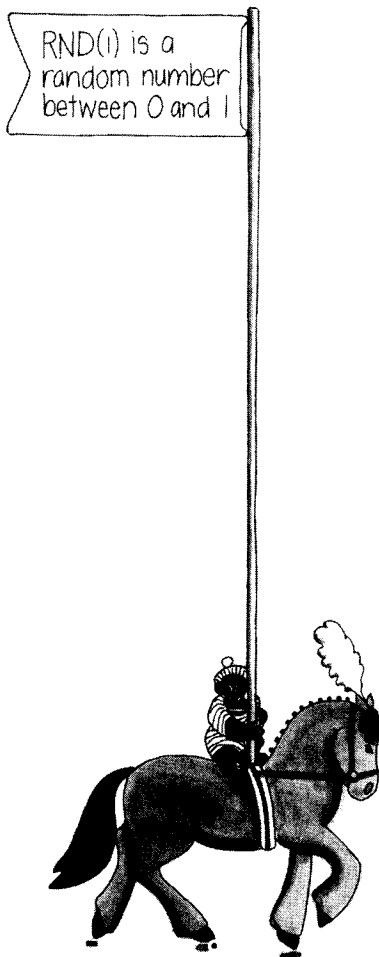
Random numbers are used in two types of computer programs:

1. teaching programs—also called CAI (Computer-Assisted-Instruction);
2. games and simulations (A **SIMULATION** is a "real-life" game. It imitates something the way it would really happen).

In programming, there are certain operations done automatically which are called functions. We use the **RND** function to create random numbers in a program. For example:

```
10 REM CREATE A RANDOM NUMBER BE-  
   TWEEN 0 AND 1  
20 FOR L=1 TO 10  
30 LET X=RND(1)  
40 ? X  
50 NEXT L  
60 END
```

The program we just saw tells ATARI to print any random number between 0 and 1 ten times. ATARI will pick any number it wants each time. There will be no order to the numbers. Each time you run this program, ATARI will print a different list of numbers.



A **SIMULATION** is a "real life" game. It imitates something the way it would really happen

If you want ATARI to print a list of random numbers between 0 and 10, you would change the RND function to:

```
LET X=10*RND(1)
```

If you want ATARI to print a list of random numbers between 0 and 100 you would change the RND function to:

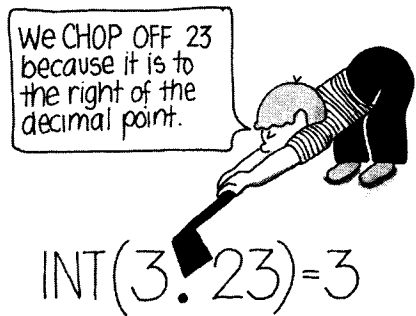
```
LET X=100*RND(1)
```

If you are writing a game program, you will not want 0 to be a random number—especially if the game is simulating the roll of a die. To print any random number between 1 and 101, change the RND function to:

```
LET X=100*RND(1)+1
```

This causes the lowest possible number to be 1.0000 and the highest possible number to be 100.9999.

An **INTEGER** is a whole number. Numbers like .25 and 6.32 are not whole numbers, they are decimals. We use the **INT** function to create whole numbers or integers in a program. For example:



### Program

```
10 REM CONVERTING DECIMALS TO  
  INTEGERS  
20 ? "DECIMAL", "INTEGER"  
30 FOR X=1 TO 5 STEP .5  
40 ? X, INT(X)  
50 NEXT X  
60 END
```

### Output

DECIMAL	INTEGER
1	1
1.5	1
2	2
2.5	2
3	3
3.5	3
4	4
4.5	4
5	5

READY  
□

Notice that the integer for the decimal 1.5 is 1. The integer for the decimal 2.5 is 2, and so on. The INT function rounds the decimal down to the nearest integer.

Sometimes you will want ATARI to print random numbers which are only integers. To do this, use both the RND and INT functions. For example:

```
10 FOR L=1 TO 10
20 LET X=INT(6*RND(1)+1)
30 ? X
40 NEXT L
50 END
```

This program tells ATARI to print a random whole number or integer between 1 and 6. The smallest possible number would be 1 and the highest possible number would be 6.

Let's say you want ATARI to print a random whole number between 2 and 12. The INT and RND function would say:

LET X=INT(11\*RND(1)+2) The smallest  
number that can  
be printed

To create random integers between and including 50 and 85, use:

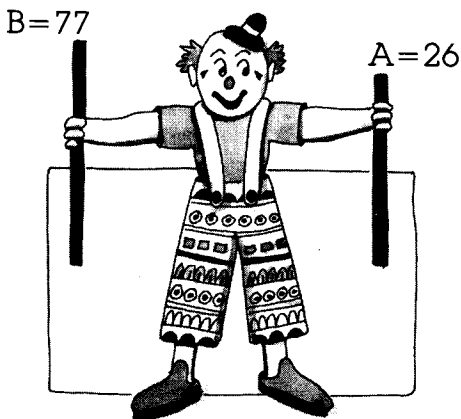
LET X=INT(36\*RND(1)+50)  
85-50=35      ↖      The smallest number  
35+1=36      to be printed.

The formula for creating random integers between A and B (where A is the smallest integer and B is the largest) is:

$\text{INT}((B-(A+1))*\text{RND}(1)+A)$

To create random integers between 26 and 77, use the formula like this:

$\text{INT}((B-(A+1))*\text{RND}(1)+A)$   
 $\text{INT}((\uparrow 77-(\uparrow 26+1))*\text{RND}(1)+\uparrow 26)$   
 $\text{INT}(50*\text{RND}(1)+26)$

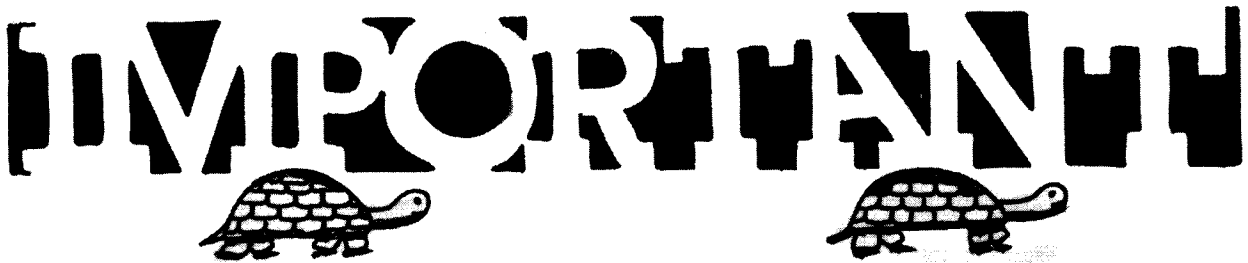


Here is an example of how to use the INT and RND functions in a CAI teaching program which gives the student practice in adding.

### Program

10 LET A1=INT(100*RND(1))	A random integer for A1 is created.
20 LET A2=INT(100*RND(1))	A random integer for A2 is created.
30 ? A1; ``+``; A2; ``=``;	The equation is printed for the student.
40 INPUT S	The student types his/her answer.
50 LET T=A1+A2	ATARI calculates the answer to the equation.
60 IF T=S THEN 90	The student's answer is compared to T.
70 ? ``NOPE. TRY AGAIN``	If the answers are equal, GOTO 90.
80 GOTO 30	If they're not, print the equation again.
90 ? ``RIGHT ON!``	Tell the student they are right.
100 GOTO 10	GOTO the beginning of the program, pick new random integers, and start all over again.

Run this program on ATARI to see how it works.



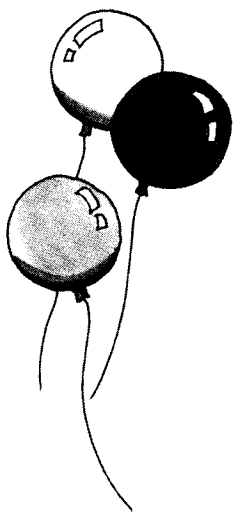
### IMPORTANT

The placement of parentheses in RND and INT functions is very important.

If the parentheses are in the wrong places, the program won't work properly.

**to do:** Programmer's Pastime #57, #58, #59, #60, #61, #62, #63

## CHAPTER 42 Making Sounds!



One of ATARI's fun features is its ability to make sounds and music. The program statement used to do this is **SOUND (SO.)**. (Notice that the shortened form—SO.—is always followed by a period.)

Try running this program on ATARI:

```
10 SO. 1, 121, 10, 8
```

```
20 GOTO 20 (That's right, GOTO 20)
```

As this program is running, adjust the volume on the television or monitor so that it is comfortable for you and others around you. To stop this program from running, you must press the BREAK key, type END, and press RETURN.

You may be wondering about line 20. This is the first time we have had a GOTO statement going to its own line. Try this. Use only line 10 SO. 1, 121, 10, 8 in a program. Listen very carefully when you run it. (You may have to turn up the sound volume of the T.V.) Run it several times and you should hear a very rapid "beep" each time.

Here's what is happening. Line 10 produces a sound, but it is done very rapidly. With no other lines in the program, ATARI assumes the program is over and ends it. When line 20 GOTO 20 is inserted, ATARI does not assume the program has ended, so the sound from line 10 is drawn out.

In the line statement—10 SOUND 1, 121, 10, 8—the four numbers together tell ATARI to make one sound. Each number tells ATARI a different thing about the sound. (If any of the numbers is left out, you will get an error message.) What are these four things that the computer must know in order to make sounds? ATARI must know what **VOICE** to use, what **NOTE** to play, what **TONE** to choose, and how **LOUD** to play. It is also important that each component be placed in the proper order, like this:

```
10 SOUND, VOICE, NOTE, TONE, LOUDNESS
```

Let's learn about each component in the SOUND statement.



**VOICE:** Just as every student in your class has a different voice, ATARI also has different voices, numbered 0, 1, 2, and 3. Run this program to hear ATARI's different voices:

```
10 FOR V=0 TO 3
20 SOUND V, 121, 10, 8
30 FOR T=1 TO 1000: NEXT T
40 NEXT V
50 END
```

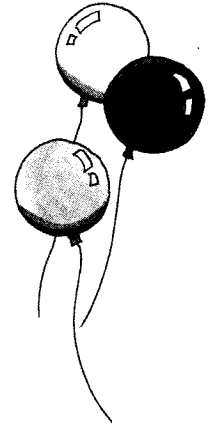
In this program V is a variable that represents Voice. Rather than set the number we want for Voice, we are letting the machine play a variety of voices. Therefore, we use "V" to indicate the voice variable. Line 20 is the part of the program that produces the sound. It has the variable for Voice, as well as numbers that set the Note, Tone, and Loudness. Lines 10 and 40 are a FOR-NEXT loop that causes Voice to move through all four of its levels—0 to 3. Line 30 is a FOR-NEXT loop that slows down the sound. Without it the sound would be made so fast that it would be hard to hear. (Try taking line 30 out and see what happens.)

**NOTE:** When you sing, play, or write music, you use a wide variety of musical notes. ATARI also has a wide variety of notes—from 0 (the highest pitch) to 255 (the lowest pitch). Use the following program to hear some of ATARI's notes:

```
10 FOR N=0 TO 255 STEP 20
20 SOUND 0, N, 10, 8
30 FOR T=1 TO 1000: NEXT T
40 NEXT N
50 END
```

Look carefully at this program and see if you can understand how it can produce a variety of ATARI's notes. Notice how line 10 causes only every twentieth note to be sounded. Could you change the program so more or less notes would be sounded?

The following diagram will give you some idea how the numerical values of ATARI's notes are related to the piano keyboard and musical scale.





---

**LOUDNESS:** ATARI can produce a range of Loudness from 0 (no sound) to 8 (about normal) to 15 (the loudest). As you run the following program, you may have to lower the sound volume of the T.V. so you will not disturb others around you.

```
10 FOR L=0 TO 15
20 SOUND 0, 121, 10, L
30 FOR T=1 TO 1000: NEXT T
40 NEXT L
50 END
```

**COMBINING VOICES:** ATARI can combine up to four voices. Try the following:

```
10 SOUND 1, 121, 10, 8 (the middle C note)
20 SOUND 2, 96, 10, 8 (the E note)
30 SOUND 3, 83, 10, 8 (the G note)
40 FOR T=1 TO 1000: NEXT T
```

When you combine voices, be sure that the voice variable is different (0, 1, 2, and 3) for each statement. When different voices are combined, a chord is played.

**to do:** Programmer's Pastime #64, #65



# CHAPTER 43

## Graphics

We are all familiar with the many video arcade games run by computers. One of the most eye-catching features of most games is the graphics they have. As a beginning programmer you will not be able to program your ATARI to make the graphics of arcade games, but the next two chapters will give you a start at programming some fun and creative graphics.

One of the first programming statements you need to know is **GRAPHICS (GR.)**. Note the necessary period with the shortened (**GR.**) form. ATARI allows you to work in one of nine different graphics modes, numbered from 0 to 8. In fact, you have been working in one of the graphics modes for some time now. Graphics mode 0 makes the regular ATARI screen that we have been using all this time.

There are also two other graphics modes that allow you to work with text material. **GRAPHICS 1** causes letters to be twice their normal width. This is how ATARI would look in Graphics 1.

ATARI

**GRAPHICS 2** causes letters to be both twice the width and twice the height. This is how ATARI would look in Graphics 2.

ATARI



---

Try out these modes with the following program:

```
10 GR. 1
20 ? ``ATARI``
30 END
```

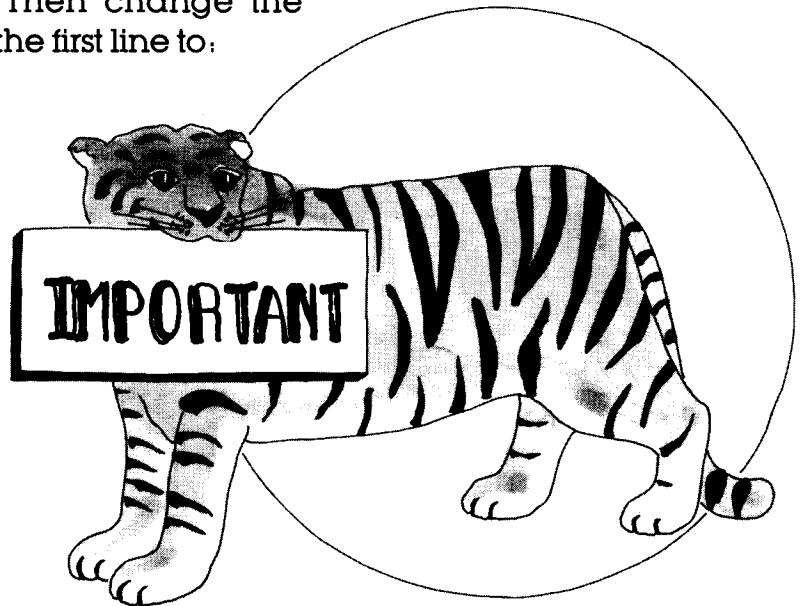
Notice when you run the above program, the television or monitor screen splits into two parts. There is the upper graphics screen, and the lower **TEXT WINDOW**. The text window allows you to write four lines of text material to go with the graphics. When you use the PRINT statement (20 ? ``ATARI``) in graphics mode 1 or 2, the material is printed in the text window. If you want material to be printed on the graphics screen, you must change the statement to read:

```
20 ?#6, ``ATARI``
```

Now change line 20, run the program, and note the size of the letters. Then change the graphics mode by changing the first line to:

```
10 GR. 2
```

and see what happens.



**IMPORTANT:**

You can leave the graphics screen in two ways—either type GR. 0 and press  , or press the  button. Either way, you will return to the regular (GR. 0) screen, where you can list, alter, run, or erase the program.

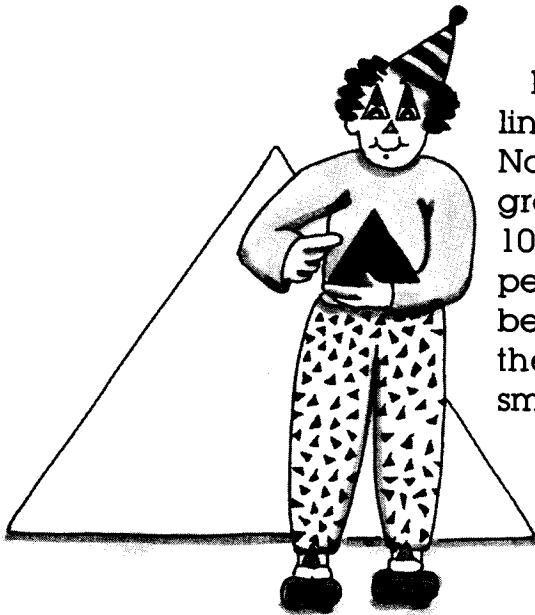
We will not work very much with modes 1 and 2, but will focus more on Graphics 3 through 8, which allow you to create a wide range of graphics while changing color and the thickness of lines.

The line thickness is an important factor in making a variety of graphics. How thick or thin a line can be is dependent upon the number of points that can be placed upon the screen. This in turn varies with the number of (horizontal) rows and (vertical) columns on the screen for each graphics mode.

For example, the **GRAPHICS 3** screen has 40 rows and 20 columns, which can produce 800 ( $40 \times 20$ ) points. On the other hand, the **GRAPHICS 7** screen has 160 rows and 80 columns. This screen can therefore produce 12,800 ( $160 \times 80$ ) points. Because the points are smaller and more numerous in the Graphics 7 mode, you can draw much finer lines than with Graphics 3.

This is somewhat confusing, but maybe the following example will help. Use ATARI to run the following:

```
10 GR. 3
20 COLOR 1
30 PLOT 5,5
40 DRAWTO 39, 5
50 END
```

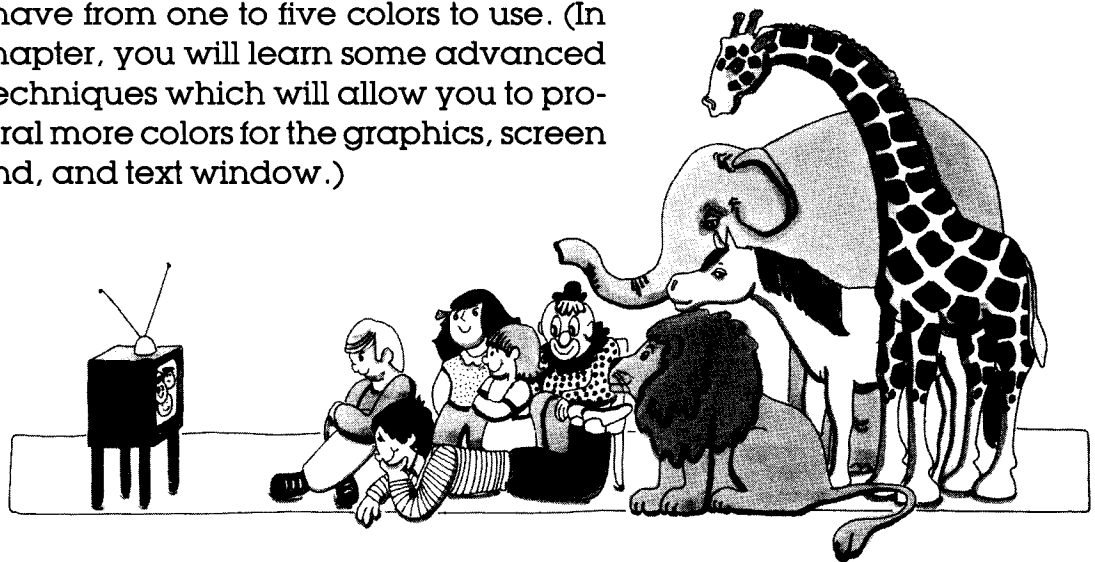


Run this program and you should produce a line going across most of the top of the screen. Notice the thickness of the line. Now, change the graphics mode by changing the first line to: 10 GR. 7. Run this program and see what happens. You should again have a line, but it should be much thinner and shorter now. This is because the Graphics 7 screen has many more, but smaller points than does the Graphics 3 screen.

As you change modes, it takes a different amount of ATARI's memory to produce the different screen sizes. For example, it takes 273 bytes of memory to make the Graphics 3 screen, but 3945 bytes to make the Graphics 7 screen. This difference becomes very important when writing long programs. Therefore, in choosing a graphics mode you will need to consider a variety of factors—the number of points that can be placed on a screen, the amount of memory being used, and the number of colors available. The following chart provides a summary of this information:

GR. Mode	Mode Type	Rows	Cols.	# of Colors	Memory Bytes
0	text	40	24	2	993
1	text	20	20	5	513
2	text	20	10	5	261
3	graphic	40	20	4	273
4	graphic	80	40	2	537
5	graphic	80	40	4	1017
6	graphic	160	80	2	2025
7	graphic	160	80	4	3945
8	graphic	320	160	1	7900

As you may have noticed in the last program, there are several statements which are important when programming graphics. As you probably guessed, the **COLOR (C.)** statement allows you to change the color of the graphics you make. Depending on the graphics mode you are using, you may have from one to five colors to use. (In the next chapter, you will learn some advanced graphics techniques which will allow you to program several more colors for the graphics, screen background, and text window.)



The following chart may help you with using the various colors:

Graphic Mode	Color Number	Color
3, 5, 7	0	same as background
	1	orange
	2	light green
	3	blue
4, 6	0	same as background
	1	orange
8	0	same as background
	1	blue

Note that COLOR 0 in the chart is the same as the background color of the screen. This is handy, for it allows you to go back and erase graphics that you do not want. Also notice that your television or monitor may not give the exact colors as listed. This usually can be corrected by adjusting the color control on your television. Also, on those graphic screens where the points are very small (E.g. 7, 8), it is often difficult to distinguish the various colors. Furthermore, colors tend to change on diagonal and vertical lines.

**PLOT (PL.)** is another programming statement for graphics. This statement allows you to place a point wherever you want on the graphics screen. Try this program:

```
10 GR. 3
20 COLOR 1
30 PLOT 2,2
40 END
```

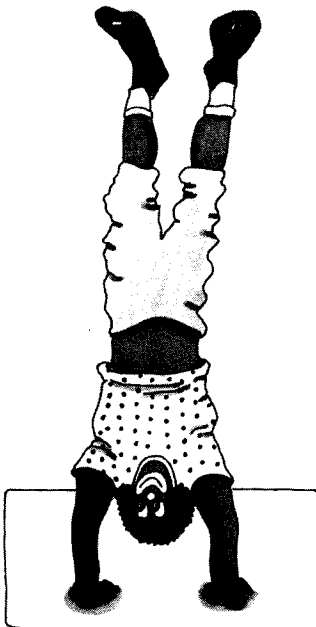
You should get an orange point at the upper left hand corner of the screen. Add another line to the program:

```
32 PLOT 2,18.
```

You should now have another point, but this one is in the lower left hand corner. Add one more:

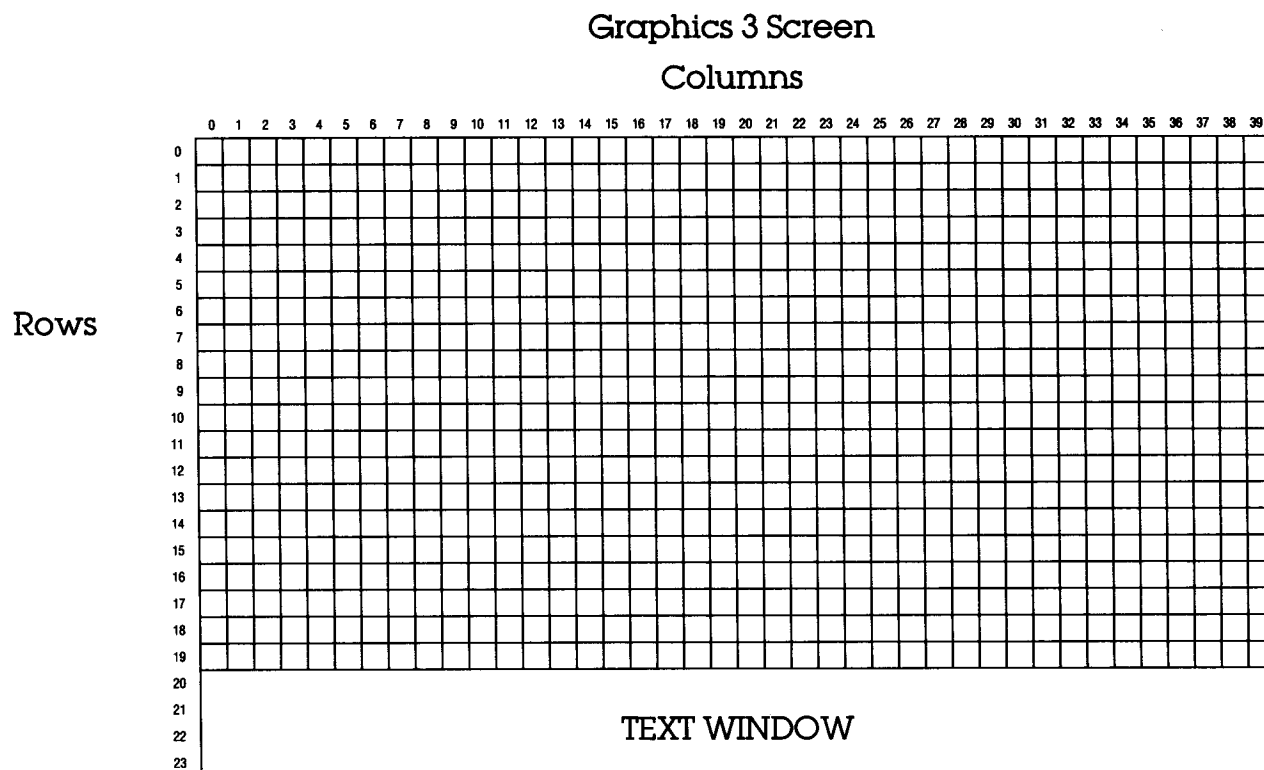
```
34 PLOT 38,2.
```

This will give an upper right hand corner point. Let's see how it works.





You know that in Graphics Mode 3, the screen is divided into 40 vertical columns and 20 horizontal rows. Although you can't see it, ATARI has numbered each of the columns and rows starting with the 0 column and 0 row in the upper left hand corner. If you could see the numbered screen, it would look something like this:

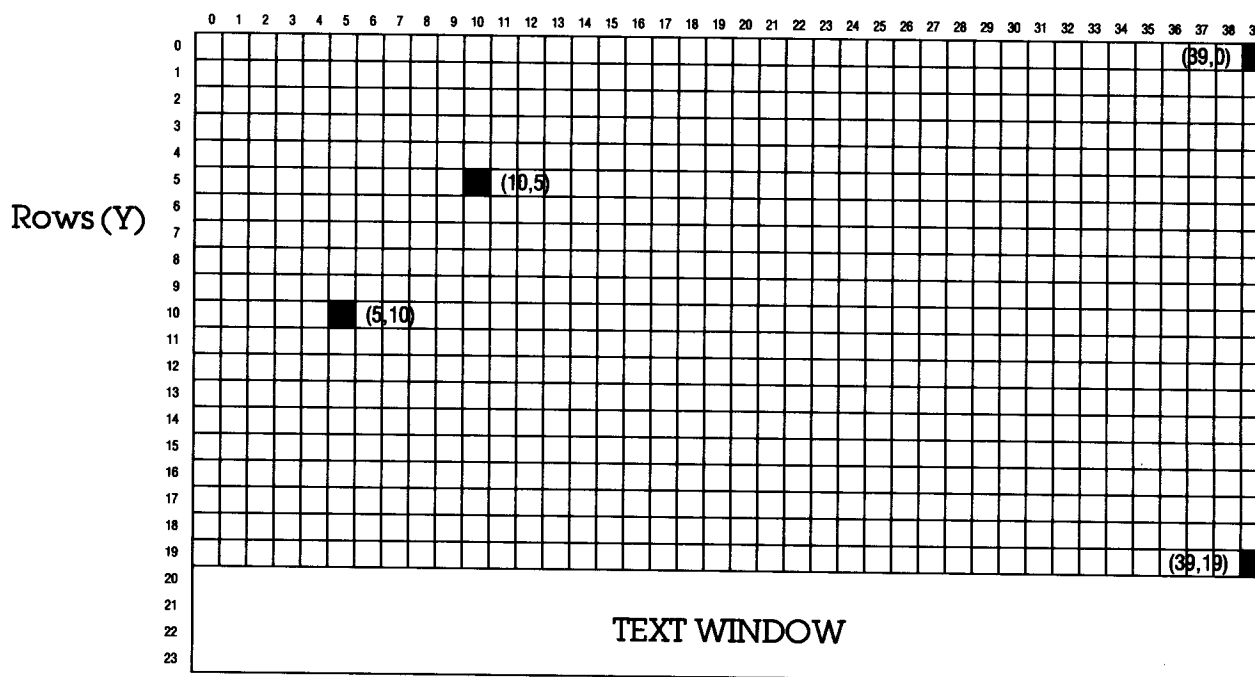


When you use the PLOT statement, the first number tells you the column number, and the second one specifies the row. Another name for the column number is the **X-COORDINATE**. The row number is called the **Y-COORDINATE**.

The X-coordinate or column number is always given first, with the Y-coordinate or row number second. So PLOT 5,10 would cause a point to be placed in the fifth column and tenth row as in the following chart. Also note where PLOT 10,5; PLOT 39,0; AND PLOT 39,19 would be located.

## Graphics 3 Screen

Columns (X)



Try this program:

```
10 GR. 3
20 COLOR 1
30 PLOT 15,23
40 END
```

When you run this program, no point shows on the graphics screen. This is because row 23 is part of the text window, and the point is hidden. Now change line 30 to PLOT 15,24. When this is run, ATARI gives an ERROR 141 message. This is because row 24 is completely off the Graphics 3 screen. If you now change line 10 to GR. 7 and run the program, you no longer get an error message because PLOT 15,24 is well within the larger GR. 7 screen.

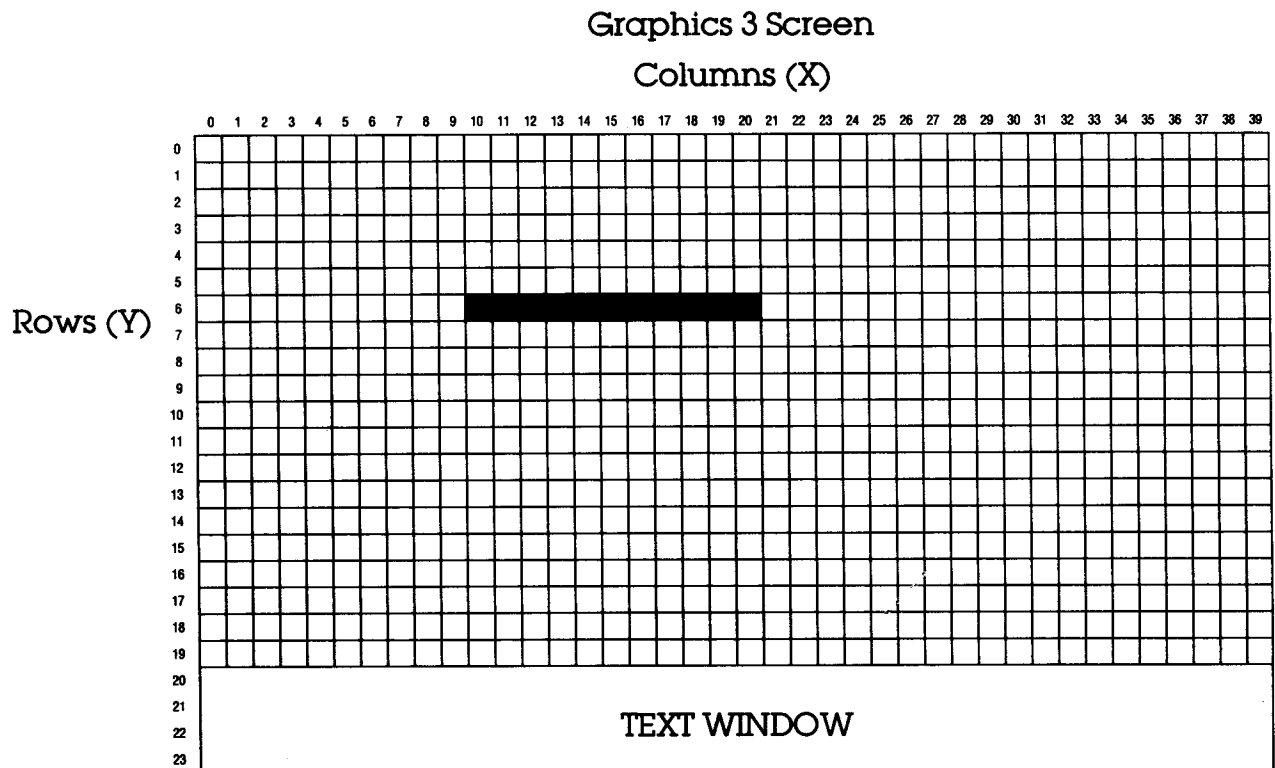
The final programming statement for this chapter is **DRAWTO (DR.)**. Notice that if you use the longer form, DRAWTO is one word. If you leave a space, ATARI will give an error message. Also note that the shorter form (DR.), has a period.



As you might guess, DRAWTO does what it says—it draws a line from one point to another point. For example:

```
10 GR. 3
20 COLOR 1
30 PLOT 10,6
40 DRAWTO 20,6
50 END
```

In this program, line 30 causes a point to be placed at position 10,6. Line 40 tells ATARI to draw a line to position 20,6. When the program is run, you should get an orange line (COLOR 1 from line 20) in the following position:



Add to the program as follows:

```
50 COLOR 3
60 DRAWTO 20,16
```

When run, your output should now include a blue line (COLOR 3) going down to position 20,16.

Now add these:

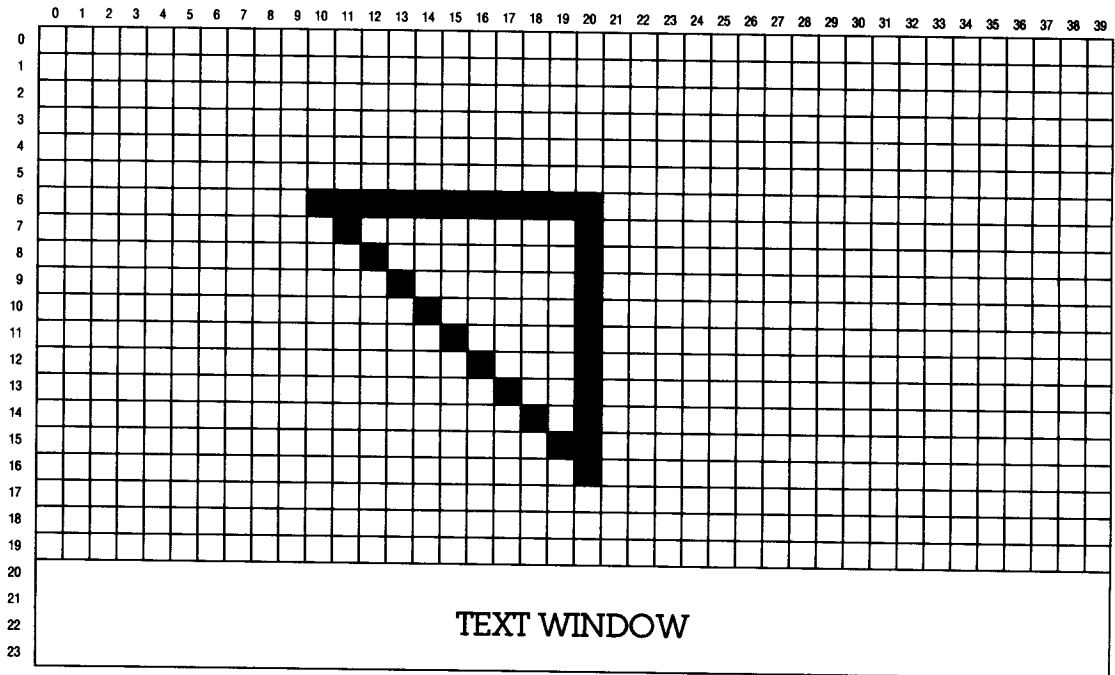
70 COLOR 2

80 DRAWTO 10,6

Graphics 3 Screen

Columns (X)

Rows (Y)



When executed you get a green (COLOR 2) line back to the original point 10,6. Notice how the green diagonal line looks like a staircase. This is because each GR. 3 point is large.

Go to Graphics 7 by changing line 10, and see what happens. You still have a "staircase" effect, but the points are much smaller, and the diagonal now looks more like a straight line.

SHORTCUT: You can save time by placing several PLOT and/or DRAWTO statements on the same line by separating them with a colon (:).

30 PLOT 20,10: PLOT 20,15

40 PLOT 15,5: DRAWTO 15,10

50 DRAWTO 5,15: DRAWTO 5,10

to do: Programmer's Pastime #68



# PROGRAMMER'S PASTIME #68

One enjoyable aspect of graphics is animation—causing the graphics to move. Following is a program for some simple animation.

```
10 GRAPHICS 3
20 COLOR 1
30 FOR X=0 TO 39 STEP 3
35 ? #6; ``    ``
40 PLOT X,7
50 DRAWTO X,10
60 DRAWTO X+3,10
70 DRAWTO X+3,7
80 DRAWTO X,7
85 FOR T=1 TO 100: NEXT T
87 IF X >= 36 THEN GOTO 10
90 NEXT X
```

Here's what the program does. Lines 40 through 80 make a simple square graphic. The X-coordinate is not specified in these lines, but rather it is set as a variable X. Lines 30 and 90 identify the X-coordinate as every third number between 0 and 39. These lines, along with 10 and 20, which determine the graphics mode and color, are the main part of this program. However, notice that the program was improved by adding some more lines after the program was first written. Line 85 is a "timer" so that the graphic remains momentarily on the screen. (Try changing this line for different effects.) Line 87 causes the program to repeat once the graphic has moved completely across the screen. Line 35 causes the screen to be cleared as the graphic starts again. (Remember, #6 must be used with a PRINT statement for the graphics screen!)

1. Run this program, and then modify some line statements to see how you can change the graphics and animation.
2. Use all the graphic techniques that you have learned to this point to make your own animated graphics.




## CHAPTER 44 More Graphics!

Up to this point, you have only worked with a graphics screen with a text window at the bottom. Text windows are useful when you want to include words with your graphics. However, you may want to create graphics with no text. ATARI has a simple way to eliminate the text window—just add **16** to the graphics mode number. For example, if you want to use Graphics Mode 3 without the window, you would use the statement, `GRAPHICS 3+16`. (This also can be written `GRAPHICS 19`.)

Try this program:

```
10 GRAPHICS 3+16
20 COLOR 1
30 PLOT 10,10
40 GOTO 40
```

When this program is run, you have an orange point on a graphics 3 screen, but no text window. Notice how line 40 is a GOTO statement to itself. Run this program without line 40 and see how the screen rapidly changes from the normal one, to the graphics one, and back again. In order to hold the graphics screen without the text window, you must have a GOTO statement such as that in line 40. Also notice that when this program is executed with line 40 in place, it is running continuously. In order to stop the program, you must press the  key.



At this point, you have learned to make a variety of graphics with one to three colors, with or without the text window. The **SETCOLOR (SE.)** statement will give you a much greater variety of options for color graphics with ATARI. In fact, the variety is so great that it is difficult to explain all that SETCOLOR can do.

Let's use the following program to try to understand how the SETCOLOR statement can help you make interesting graphics.

```
10 GRAPHICS 3
20 COLOR 1
30 FOR X=0 TO 15
40 ? : ? : ? X
50 SETCOLOR 0, X, 2
60 PLOT 5,5
70 DRAWTO 25,5
80 FOR T=1 TO 600: NEXT T
90 NEXT X
100 END
```

As you run this program, you will notice how the graphics bar changes sixteen times from gray through several shades of different colors. (Remember, the colors on your screen may not be quite the same as those described here due to the color adjustment on your television set.)

Now change line 50 to SETCOLOR 2, X, 2 and see what happens. (Don't forget, you can change parts of line statements by using the cursor control keys and the screen editing functions. Be sure to press RETURN after you have made changes in a line.)

When this program is run, the graphics bar remains the same color, but the text window goes through 16 color changes.



You can see another variation with SETCOLOR by changing line 50 one more time:

```
50 SETCOLOR 4, X, 2
```

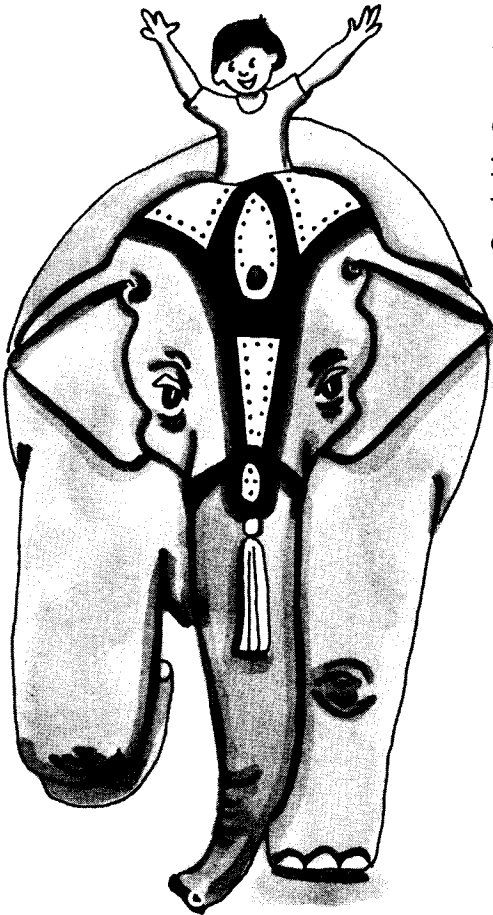
This time both the graphic bar and text window remain the same, but the screen background color changes 16 times.

Let's change both line 30 and 50 this time:

```
10 GRAPHICS 3
20 COLOR 1
30 FOR X=0 TO 14
40 ? : ? : ? X
50 SETCOLOR 4, 4, X
60 PLOT 5,5
70 DRAWTO 25,5
80 FOR T=1 TO 600: NEXT T
90 NEXT X
100 END
```

Notice that now the X variable in line 30 only goes to 14. Also, the position of the X variable in line 50 is changed to the last SETCOLOR component.

When this program is run, "brightness" becomes the factor that changes. This brightness factor is somewhat difficult to use, and probably won't be changed as often as the other SETCOLOR components.





Now you have a general idea of what the SETCOLOR statement can do for ATARI's graphics. The first listed component of SETCOLOR controls where the changes take place—0 changes the displayed graphics, 2 affects the text window, and 4 the background of the graphics screen. The second component of the statement causes 16 different color changes—from 0 to 15. The last component controls 8 brightness variations—using the even numbers from 0 to 14.

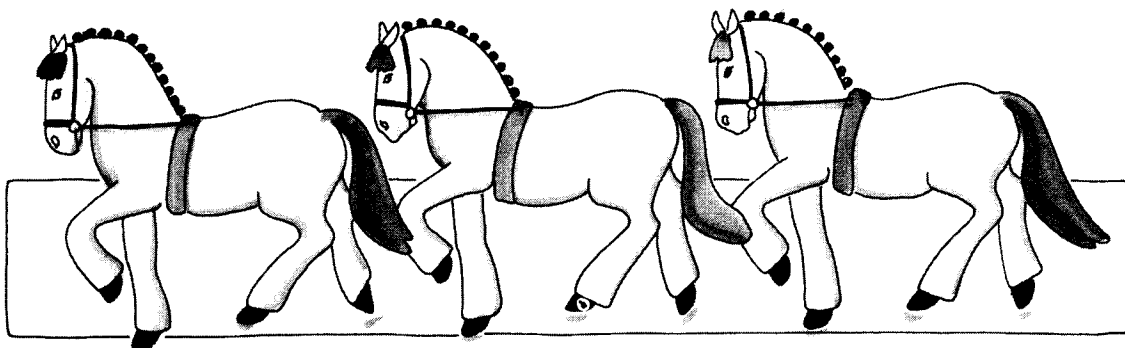
Unfortunately, SETCOLOR's effects will change when the other graphics statements are varied—particularly when color is altered. One way to familiarize yourself with SETCOLOR is to use it only with Color 1, then gradually experiment with changing other factors.

One way to increase your enjoyment of writing graphics programs, is to add sound. Try this program:

```
10 FOR X=1 TO 8: GRAPHICS 7
20 ? :? ``ATARI'S LIGHT AND SOUND SHOW``
30 SETCOLOR 1, 2*X, 8: COLOR 2
40 FOR Y=0 TO 80 STEP X
50 PLOT 0,0: DRAWTO 100,Y
60 FOR Z=210 TO 30 STEP -30
70 SOUND 0, Z, 10, 8
80 NEXT Z: NEXT Y: FOR T=1 TO 100: NEXT T:
  NEXT X
90 END
```

After you have run the program, try to figure out what each line of the written program does. (Notice how several statements are combined on single lines by using colons.)

**to do:** Programmer's Pastime #69, #70



# CHAPTER 45

## Writing Game Programs

Playing computer games can be an enjoyable recreational experience. One of the rewards of learning how to program a computer is being able to write a game program.

There are basically three types of computer games:

1. **Mathematical games:** games involving numbers and/or solving arithmetic or mathematical problems.
2. **Recreational games:** Many different games could fall in this category. I think of "Space Invaders" and "Dungeons and Dragons."
3. **Simulations:** games which imitate real-life situations.

In writing a game program, you must be sure the program will be **USER FRIENDLY**. This means that the program is easy for anyone to use.

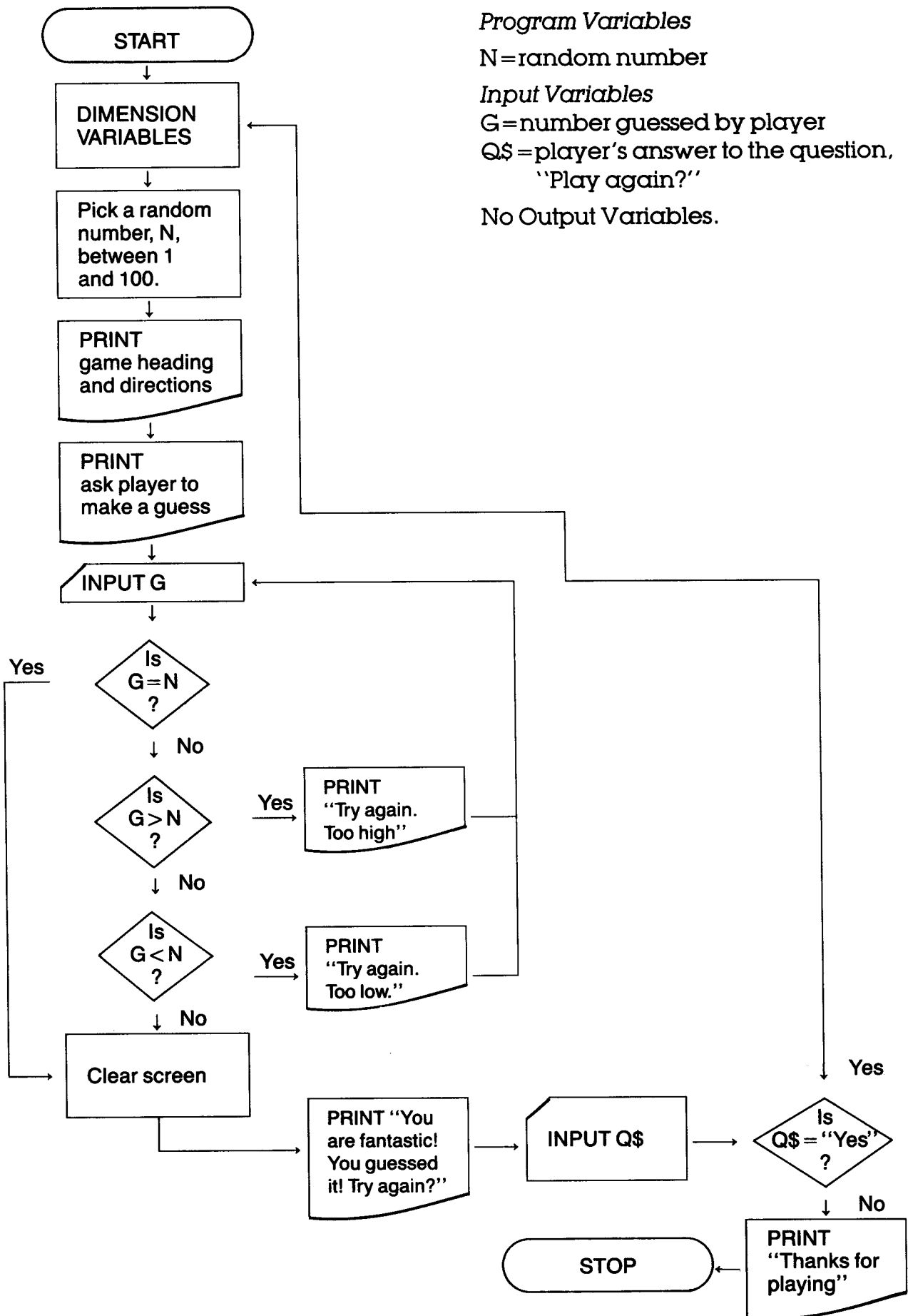
A program that is user friendly should:

1. give clear directions;
2. have easy-to-read screen output;
3. be free of bugs and not be "broken" easily during the run;
4. have fun or interesting graphics;
5. and communicate with the players (tell the players how they are doing through messages or scores).

You have learned all of the programming techniques needed to write a good game program. Study the following game program to get an idea of how a user friendly game should be written.



### Flow chart



### Data Table

*Program Variables*

N=random number

*Input Variables*

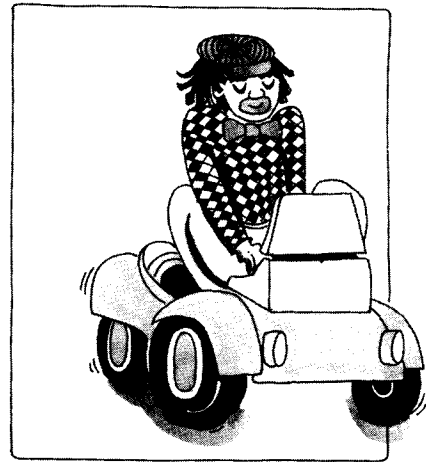
G=number guessed by player

Q\$=player's answer to the question, "Play again?"

No Output Variables.

## Program

```
5 DIM Q$(15)
10 REM ** GUESS A NUMBER GAME **
20 REM ** CHOOSE A RANDOM NUMBER **
30 LET N=INT(100*RND(1)+1)
40 ? "    "
50 REM ** BEGIN GAME **
60 ? "GUESS A NUMBER GAME"
70 FOR T=1 TO 1000: NEXT T
80 ? "    "
90 ? "GUESS A NUMBER BETWEEN 1 AND 100"
100 INPUT G
110 IF G=N THEN 140
120 IF G>N THEN ? "HIGH, TRY AGAIN": GOTO 100
130 IF G<N THEN ? "LOW, TRY AGAIN": GOTO 100
140 REM ** CORRECT GUESS **
150 ? "    "
160 ? "YOU ARE FANTASTIC!"
170 ? :?
180 ? "YOU GUESSED IT!"
190 FOR T=1 TO 1000: NEXT T
200 ? "    "
210 ? "WANT TO PLAY AGAIN"
220 INPUT Q$
230 IF Q$="YES" THEN 30
240 ? :?
250 ? "THANKS FOR PLAYING!"
260 END
```



Does this program have the five elements of a good game program?

1. Clear directions: Lines 60, 90, 120, and 130
2. Easy-to-read output: Lines 170 and 240 space the output.  
Lines 40, 80, 150, and 200 clear the screen.  
Lines 70 and 190 slow down the output.
3. Free of bugs: There is one bug. Look at lines 200-220.

If the user types "YES" the game will start over again. If the user types "NO" or even a positive answer like "SURE," the program will end. The program should be written so that if something other than "YES" or "NO" is typed, ATARI will go back to line 210 and print the question, "WANT TO PLAY AGAIN" another time, instead of ending the program.

4. Fun, interesting graphics or sounds:

This could be improved by adding sounds or graphics when the correct answer is found—perhaps a high musical note when the guess is too high, and a low note when the guess is too low.

5. Messages to the player: Lines 120 and 130 tell the player if the guess is too high or too low.

Lines 160 and 180 congratulate the player for guessing correctly.

Line 210 asks the player if he/she wants to play again.

Line 250 thanks the player.

Run this program to see first hand how it works. Maybe you will have some suggestions on how to make the program even better!

**to do:** Programmer's Pastime #71



# CHAPTER 46

## You Are a Creative Programmer!

You have learned how to use ATARI as a calculator and as a problem-solving tool. You know that ATARI can also help you with your creative projects. Now that computers are available with a color screen, it's a sure thing that we will see more and more people using the computer for creative purposes.

Another creative outlet for computers is animation and sound generation. Did you know that not only can you program computers to make music, you also can program them to talk?

Now that you know how to create visual pictures and designs, ATARI hopes you will continue to learn more about computer animation and sound. The possibilities of what you can do with a computer are endless!

Use your imagination . . . explore . . . try new things . . . ! Your computer is a friend, a tool, and a key to your future!

**to do:** Component 7 Fun Page  
Evaluate yourself



# EPILOGUE

## Congratulations!

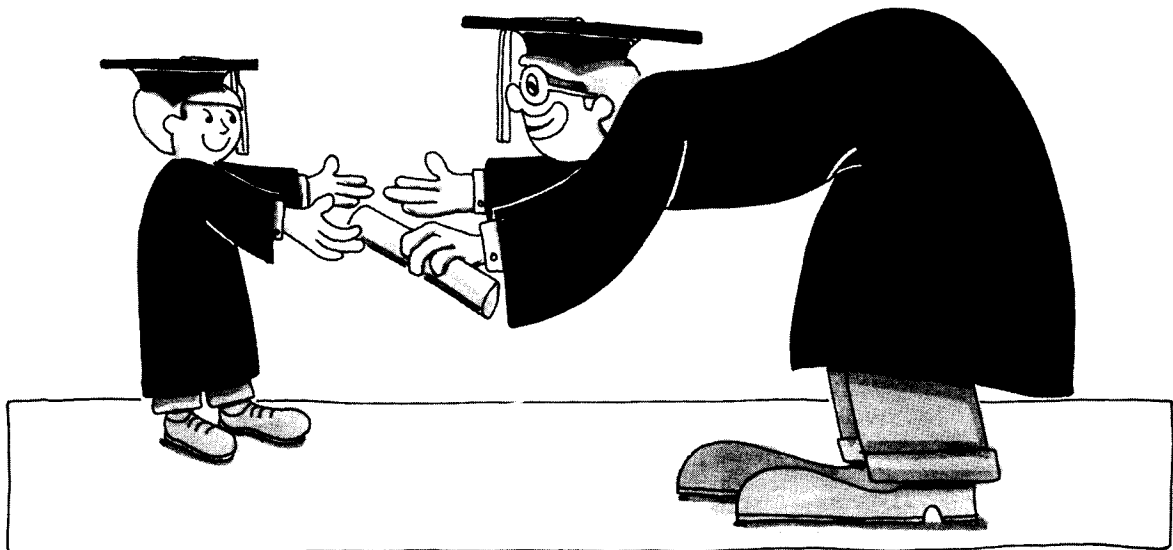
You are now a veteran computer programmer! You've come a long way!

You now have the skills needed to write programs in BASIC to control a computer. You know how to use the computer to solve your problems (problem-solving programming) and to entertain yourself and others (recreational programming). The skills you have learned enable you to create designs, sounds, and new ideas on the computer (creative programming). You should be very proficient at programming the computer to do just about anything!

Sure, there are still many more BASIC programming techniques to learn. Some of them are complicated, but others are shortcuts which will make your programming easier!

Once you are a "pro" at communicating in BASIC, there are other computer languages waiting for you—including PASCAL, LOGO, PILOT, and FORTRAN, just to name a few.

The world of computers is certainly exciting and fascinating. It is the world of the future. Don't you feel lucky to be a part of it now?



# APPENDIX A

## BASIC Commands, Statements and Functions Used in This Book (Abbreviations in parentheses)

Command, Statement, or Function	Purpose	Example
<b>BYE (B.)</b>	Allows BASIC to be exited and puts ATARI in Memo Pad or Testing Mode.	BYE
<b>COLOR (C.)</b>	Changes color of graphics.	COLOR 3
<b>CONT (CON.)</b>	Allows ATARI to continue after BREAK key has been pressed.	CONT
<b>DATA (D.)</b>	Holds data for the variables in the READ statement.	DATA 4, -16, "Y"
<b>DIM (DI.)</b>	Reserves or dimensions the number of characters for string variables.	DIM A\$(10)
<b>DRAWTO (DR.)</b>	Draws lines in the graphics modes.	DRAWTO 15,10
<b>END</b>	Tells ATARI the program run is completed.	END
<b>FOR-NEXT (F.-N)</b>	Creates a loop in the program.	FOR Z= 1 TO 10: NEXT Z
<b>GOTO (G.)</b>	Tells the computer to go to a different location in the program. It can create a loop.	GOTO 10
<b>GRAPHICS (GR.)</b>	Places ATARI in the Graphics Mode.	GRAPHICS 3
<b>IF-THEN</b>	Conditional transfer. IF something, THEN do something else, or go to a different location in the program.	IF Z= 10 THEN ? "HI" IF Z= 11 THEN 500
<b>INPUT (I.)</b>	Tells the computer to ask the user to type in input.	INPUT A, B\$
<b>INT</b>	Integer function tells ATARI to print a whole number (integer).	? INT (P) ? INT (4.69)
<b>LET (LE.)</b>	Assigns a value to a variable.	LET P= 100



<b>Command, Statement, or Function</b>	<b>Purpose</b>	<b>Example</b>
<b>LIST (L.)</b>	Tells the computer to list the statements of a program.	LIST
<b>LOAD (LO.)</b>	Tells the computer to load a program from a tape or diskette into its memory.	LOAD D (for diskette) CLOAD (for cassette tape) (CLOA.)
<b>NEW</b>	Erases unwanted programs from ATARI's memory.	NEW
<b>NEXT (N)</b>	See <b>FOR-NEXT</b>	
<b>PLOT (PL.)</b>	Allows a graphics point to be placed at a specified location.	PLOT 10,5
<b>PRINT (?)</b>	Tells the computer to print output.	PRINT AS ? "HI"
<b>READ-DATA (REA.) (D.)</b>	Tells the computer to use data from the DATA statement for the value of certain variables.	READ Z\$, X
<b>REM (R.)</b>	Allows remarks or documentation to be written into the program without affecting how the program is run.	REM ADDING NUMBERS
<b>RND</b>	Random function tells the computer to pick a random number.	LET R=(10*RND(1)+1)
<b>RUN (RU.)</b>	Tells the computer to start doing or executing the program.	RUN
<b>SAVE (S.)</b>	Tells ATARI to save a program from memory.	CSAVE (to cassette tape) (CS.) SAVE D (to diskette)
<b>SETCOLOR (SE.)</b>	Allows color of screen, text window, or graphics to be changed.	SETCOLOR 2, 10, 6
<b>SOUND (SO.)</b>	Allows ATARI to make sounds and music.	SOUND 1, 121, 10, 8
<b>SQR</b>	Allows the square root of a number to be found.	SQR (49)
<b>STOP (STO.)</b>	Used within a program to stop a program run.	STOP

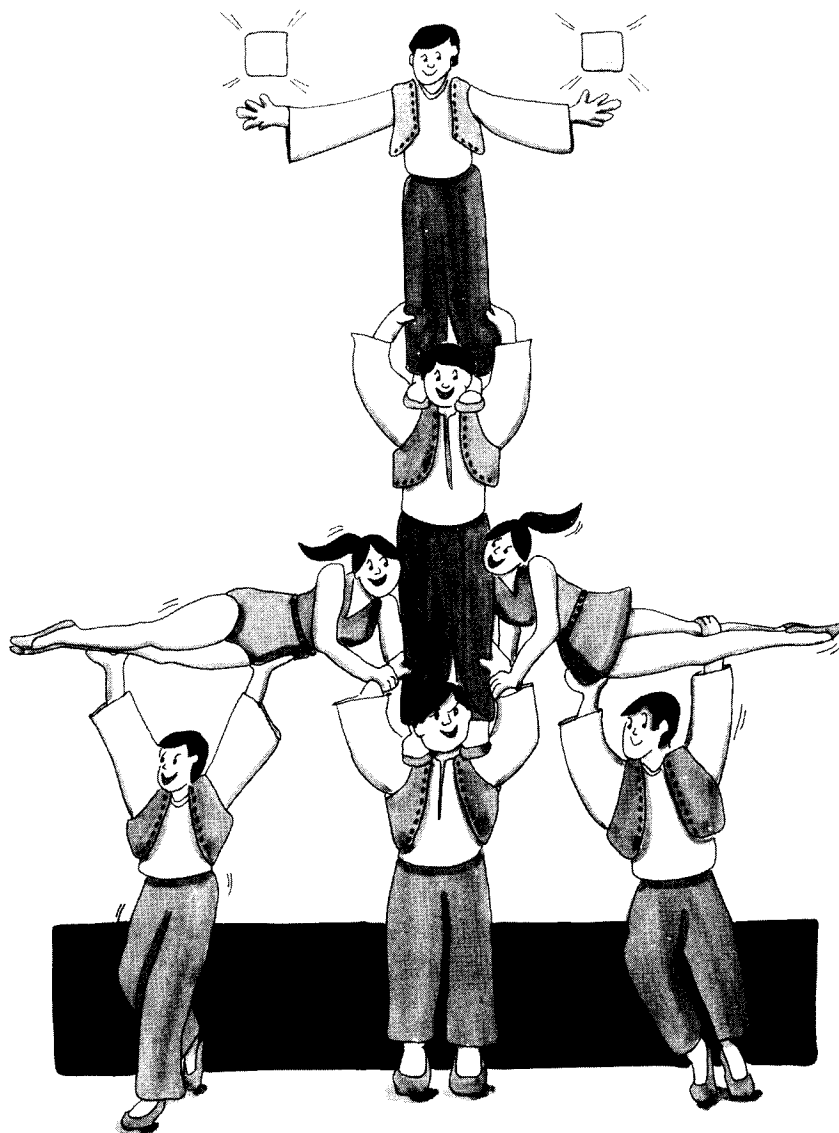
# APPENDIX B

## ERROR MESSAGES

ATARI lets you know if it can't understand what you want it to do by giving an ERROR MESSAGE. This it does in two main ways. Sometimes it will say "ERROR" and place the cursor over the portion of the statement it doesn't understand. Or often ATARI will simply say "ERROR," give an error number, and specify the program line at which the error occurs. Following is a list of some of the more important error message numbers, with a brief explanation of their meanings.

- 2 There isn't enough memory space to do what is asked of ATARI.
- 3 A numeric value is used that is too large, too small, or is negative when it should be positive.
- 4 Too many variable names have been used—only 128 are allowed.
- 5 The number of characters in a string variable is more than have been dimensioned.
- 6 The READ statement tries to read more data than is in the DATA statement.
- 7 A number is negative, or greater than 32767, when it shouldn't be.
- 8 Numeric variables must contain numbers and not letters, graphic characters, punctuation marks, etc.
- 9 A problem with an array or string dimension—often when ATARI tries to use an undimensioned string variable.
- 11 The program tries to divide by zero, or a calculated answer is greater than  $1 \times 10^{98}$ , or smaller than  $1 \times 10^{-99}$ .
- 12 A line statement such as GOTO, GOSUB, IF-THEN, etc. asks ATARI to go to a line that doesn't exist.
- 13 There is no matching FOR statement for a NEXT statement (error reported at the NEXT statement).

- 
- 14 The program statement is too complex or too long.
  - 15 About the same as #13—a FOR-NEXT problem.
  - 17 ATARI finds "garbage" (confused or improper data). May be a problem with the computer itself, or from faulty use of POKE.
  - 18 String variable does not begin with a proper character.



---

The following error messages are INPUT/OUTPUT errors. These happen between ATARI and external devices (cassette tape recorder, disk drive, printer, etc.) Often additional information is provided with the external device.

- 19 The program being loaded is too long for ATARI's memory.
- 21 A program is loaded into a non-load area.
- 128 The BREAK key is pressed while ATARI is in the middle of an input or output operation.
- 130 ATARI is asked to output to a nonexistent device.
- 131 A READ command is given to the printer.
- 132 An invalid command is given to a device.
- 134 An improper device number is used.
- 136 The program is directed to read a file that is not open.
- 137 ATARI finds a data record longer than 256 characters.
- 138 A problem with an external device. Check to make certain all power switches are on, connecting cables are secure, etc.
- 139 The cassette tape recorder or disk drive unit cannot properly perform a command.
- 140 The cassette tape or diskette may be faulty.
- 141 A point is plotted off the range of a graphics screen.
- 142 Similar to #140—perhaps faulty tape or diskette.
- 143 There is a bad recording on the tape or diskette (may be due to faulty tape or diskette).
- 144 A disk error, often because the disk has been protected against writing on it by a tab over the slot.

Although this list is not complete, and some of the errors are difficult to understand, this does give you an idea of the most important and common error messages that you might find while working with ATARI.

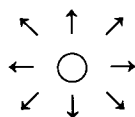
# APPENDIX C

## USING GAME CONTROLLERS

Many games, as well as other programs on your ATARI, use game controllers as input devices (usually to control some part of a game, or to move the cursor). There are three types of controllers—paddles, joy sticks, and keyboards.

All three types of game controllers are plugged into sockets on the lower front of the computer. Most programs requiring game controllers come with instructions on how to use the controllers and where to plug them in. (If not, start with the socket on the left and see which one works for your particular program.)

The joy stick controllers have eight different positions for causing movement on the screen:



All the controllers are sturdy, but can be damaged by dropping or abnormal usage. In particular, the joy stick can be worn out with excessive pressure on the stick as it is pushed and pulled. The joy stick will work with light pressure on the control, and will not work faster or better with heavy pressure.

# GLOSSARY

**Algorithm**—A step-by-step method used to solve a problem.

**ATARI**—A microcomputer made by Atari, A Warner Communications Company.

**BASIC: (Beginner's All-Purpose Symbolic Instruction Code)**—The most popular language used with microcomputers, and fairly simple to use.

**Brain**—The central processing unit and memory bank which make up the internal circuitry of the ATARI computer.

**BREAK message**—The message displayed on the screen, after the BREAK key is struck, describing the line location where the program was stopped.

**Bugs**—Mistakes that a programmer can make while writing a program.

**BYE**—The programming command that allows BASIC to be exited and puts ATARI in the Memo Pad Mode (400, 800 models) or in the Testing Mode (XC models).

**Cassette tape recorder**—A device used to store information from the computer memory, or to send information to the computer memory.

**CLOAD**—The programming statement used to load data from the cassette tape recorder into ATARI's memory.

**COLOR**—The programming statement that allows the color of graphics to be changed.

**Complement**—The opposite of a question or sign. For example, < is the complement of >.

**Computer-Assisted Instruction (CAI)**—Using computers for teaching purposes.

**Computer language**—Sets of symbols used to communicate with the computer.

**CONT**—The command that allows ATARI to continue after it has been stopped because the BREAK key has been pressed.

---

**Conversion**—Changing one type of information to another type. For example, changing a measurement in feet to its equivalent in inches.

**Conversion equation**—The program equation used to change one type of information to another.

**Counter**—A program technique to keep track of the number of times a loop has been executed.

**Counter-controlled loop**—A programming loop that can be executed for a specified number of times.

**CSAVE**—The programming statement used to save data from ATARI's memory to the cassette tape recorder.

**Cursor**—The square of light appearing on the television screen marking the location where data will next appear.

**Cursor control keys**—The keys which allow the cursor to be moved around the screen without erasing what is written on the screen.

**Data**—Information. Also a programming statement. See READ-DATA.

**Debugging**—The process of getting rid of program mistakes.

**Decision box**—The diamond-shaped box in a flow chart that represents a decision to be made.

**Delete**—Tells the computer to erase a character or line.

**DIM**—A program statement used to reserve (dimension) a specified number of characters to be used with string variables.

**Direct or Immediate Mode**—A state of computer operation in which a statement is executed immediately. (e.g. PRINT 5 + 6 would immediately print 11 after the RETURN key is pressed.) This contrasts with a Delayed or Program Mode in which instructions are not executed until a program is run.

**Disk drive**—A device used to store information from the computer memory, or to input information to the computer memory.

- 
- Documentation**—Using REMARK statements to note and clarify what is happening in a program.
- Double-alternative decision step**—A situation in a flow chart in which there are two "detours" from a decision box.
- DRAWTO**—The programming statement used to draw lines in the graphics mode.
- Dummy data**—Data that is read as a signal that the READ-DATA pointer is at the end of the DATA list.
- E (exponential) notation or floating point notation**—A way of representing very large or very small numbers.
- END**—The program statement that tells ATARI a program run is completed.
- ERROR messages**—ATARI's way of telling you that the computer does not understand what you want it to do.
- Filename**—The name given to a program as it is saved to a storage device, like the disk drive. The filename can be any combination of eight letters and numbers, but must begin with a letter.
- Flow chart**—A diagram which shows all of the steps of an algorithm in the correct order.
- Flow diagramming**—The process of illustrating program components in a clear, step-by-step fashion.
- FOR-NEXT**—A program statement that allows counter-controlled loops to be made.
- FOR-NEXT time loop**—A loop using a FOR-NEXT statement that causes a pause in the printing of output on the screen.
- Function**—Certain operations that are done automatically, like a built-in small program.
- Function keys**—Keys which control the mechanical operations of the keyboard such as shift, delete/back space, break, and return.



---

**GOTO**—The program statement that directs the computer to jump to a specified line in the program.

**GRAPHICS**—The programming statement that places ATARI in the Graphics Mode.

**Graphics keys**—The keys which allow graphics symbols to be made.

**Graphics mode**—The state of operation in which graphics can be produced.

**Home**—The position in the upper left corner of the screen where the cursor starts, or returns to under certain conditions.

**IF-THEN**—A program statement used to make comparisons and establish conditional situations.

**Immediate or Direct Mode**—An operational state of the computer in which statements typed on the screen are executed immediately when the RETURN is pressed. These statements do not have line numbers.

**INPUT**—A program statement that allows data to be typed into the program while the program is running. Also, data or information that goes into the computer from the keyboard, cassette recorder, or disk drive.

**Insert**—Creates space within a line to allow the addition of new or corrected material.

**INT**—The program function used to create whole numbers, or integers, in a program.

**Integers**—Whole numbers; without fractions or decimals.

**Interactive program**—A program that allows input to be typed into the program while it is running.

**Keyboard**—The part of the computer used to type in information (input) to the computer brain.

- 
- Letter keys**—The alphabet letter keys.
- Line number**—Any number from 1 to 32767 which precedes a program statement.
- LIST**—The programming command that tells ATARI to display on the screen any program in memory.
- LOAD D: filename**—The programming statement used to load data from the disk drive into ATARI's memory.
- Loop**—A program situation, represented by an arrow in a flow chart, in which a certain step is repeated over and over.
- Memo Pad Mode**—The operating mode that ATARI enters after the BYE command has been given (on older models).
- Memory**—The part of the computer which stores information for future use.
- Microcomputer**—A compact portable computer suitable for school or home use. (Looks much like a typewriter keyboard.)
- NEW**—The program command used to erase unwanted programs from the computer's memory.
- Number keys**—The keys which control the numerals on the top line of the keyboard.
- Output**—Information put out from the computer to external devices (television screen, printer, tape recorder unit, or disk drive unit).
- PLOT**—The programming statement that allows a point to be placed at a specified location.
- Pointer**—An electronic device that marks the location of the data being read from a DATA list.
- PRINT**—The program statement that tells ATARI to print something on the screen. The output printed may be letters, numbers, equations, the results of arithmetic calculations, etc. (A question mark, ?, may be used as an abbreviation for the PRINT statement.)

---

**Print zones or fields**—Areas on the screen in which information is printed.

**Processing box**—The rectangular shaped box in a flow chart that represents "something to be done."

**Program**—The set of directions that tells a computer what to do.

**Program or Delayed Mode**—An operational state of the computer in which statements typed on the screen are placed in the computer's memory when RETURN is pressed. These statements must have line numbers, and are stored in memory as part of a program until ATARI is given the RUN command.

**Programmer**—The person who writes computer programs.

**Random numbers**—Lists of numbers that are in no particular order.

**READ-DATA**—Two programming statements that work together making it possible to place data in a program as it is typed on the keyboard.

**READY**—The signal printed on the television screen when ATARI is ready to receive input.

**REM**—A program statement used to place clarifying notes (remarks) throughout a program. Remarks are not executed as part of the program.

**RUN**—The command that tells the computer to execute or "do" the program.

**SAVE D: filename**—The programming statement used to save data from ATARI's memory to the disk drive.

**Screen**—A television or monitor used to display the information from the computer.

**Screen editing**—Feature of the ATARI which allows you to correct the text on the screen.

**SETCOLOR**—The programming statement that allows the color of the screen, text window, and graphics to be changed.

**Single-alternative decision step**—A situation in a flow chart in which there is one "detour" from a decision box.

---

**SOUND**—The programming statement used with ATARI to make sounds and music.

**Special symbol keys**—These are keys such as +, -, \*, =, ,, :, etc.

**Statement**—An expression in the BASIC language that tells the computer to do something (GOTO, PRINT, FOR-NEXT). ("Command" and "statement" are often used interchangeably).

**STEP**—A program statement that allows counter-controlled loops to be counted in a certain pattern (e.g. by fives, tens, twenties, etc.)

**STOP**—The programming statement used on a numbered line within a program to stop a program run.

**String or alphanumeric variable**—A variable that consists of letters, numbers, or special characters (=, \$, etc.).

**Style**—Using a variety of techniques to develop easy-to-read programs.

**Text window**—The lower four lines of the screen in graphics mode that allows text to be displayed with graphics.

**User friendly**—A program that is easy to use.

**X-coordinate**—The first number in a PLOT statement that specifies the column position.

**Y-coordinate**—The second number in a PLOT statement that specifies the row position.

# INDEX

Addition . . . . .	38, 40
Address . . . . .	74
ADVANCE . . . . .	31
Algorithm . . . . .	47, 141
Alphabetize . . . . .	128
Alphanumeric variable . . . . .	115
Animation . . . . .	169
 BASIC . . . . .	10
Blink . . . . .	106
Body . . . . .	91
Brain . . . . .	4
Break . . . . .	13, 26
Break key . . . . .	109
Brightness . . . . .	172
Bugs . . . . .	112
Bye . . . . .	111
Bytes . . . . .	163
 Calculator . . . . .	39
Caps lowr . . . . .	14
Cassette . . . . .	4
Cassette recorder . . . . .	29, 30
Clear . . . . .	33
Clear screen . . . . .	102
CLOAD . . . . .	33
Coding . . . . .	143
Colons . . . . .	86, 103
Color (C) . . . . .	163
Commas . . . . .	41, 69, 77, 86, 120
Compare numbers . . . . .	46
Complement . . . . .	124
Computer errors . . . . .	113
Computer languages . . . . .	10
CONT return . . . . .	109
Contents . . . . .	74
Conversion . . . . .	149
Conversion equation . . . . .	150
Converts . . . . .	92
Corrections . . . . .	16
Counter . . . . .	34, 91, 100
Counter variable . . . . .	99
Counter-controlled loops . . . . .	90
CSAVE . . . . .	32
Cursor . . . . .	11
Cursor control . . . . .	12
Cursor control keys . . . . .	8

Data	76
Data statement	132, 138
Data table	141, 175
Debugging	113, 143
Decimal equivalents	87
Decision box	53
Decision step	60
Delete	16, 112
Dimension	115, 135
Direct	39
Disk drive	30, 34
Diskette	30, 35, 37
Division	38, 40
Documentation	130
Double detours	55
Double-alternative decision step	56
Drawto (DR.)	166
Dummy	136
E notation	87
Electronic mailboxes	74
End	24, 68
Equation	67
Error message	15
Error numbers	15
Error-6	135
Error-8	138
Errors	113
Fields	69
Floating-point notation	87
Flow chart	48
Flow diagramming	47
For-next loop	91, 90, 100, 137, 150
For-next time loop	103, 106
Function Keys	8
Games	174
GOTO	25, 62, 68, 106, 156, 170
Graphics	8, 25, 160
Graphics 1	160
Graphics 2	160
Graphics 3	162
Graphics 7	162
Heading	150
Home	13
If-then	121, 136
Immediate mode	39
Input box	132
Input statement	118
Input variables	141
Input	25, 117, 120

Insert . . . . .	16
INT . . . . .	153
Integer . . . . .	153
Interactive program . . . . .	117
Keyboard . . . . .	5, 6
Let . . . . .	74, 86
Let statements . . . . .	81
Letters . . . . .	7
Line number . . . . .	22, 26
Line thickness . . . . .	162
LIST . . . . .	33, 110
Load . . . . .	33
LOAD *D . . . . .	36
Loop . . . . .	59, 61, 67
Loud . . . . .	156
Loudness . . . . .	158
Lowercase . . . . .	14
Make a decision . . . . .	46
Mathematical games . . . . .	174
Memo pad mode . . . . .	111
Memory . . . . .	12, 16, 63, 74
Memory cell . . . . .	74
Microcomputer . . . . .	4
Modes . . . . .	62
Multiplication . . . . .	38, 40
Negative numbers . . . . .	87
New . . . . .	20, 22, 26
Note . . . . .	156, 157
Numbers . . . . .	7
Numeric variable . . . . .	115
On-off switch . . . . .	10
Output . . . . .	25, 76
Output variables . . . . .	141
Parenthesis . . . . .	40, 44, 155
PAUSE . . . . .	31
PLAY . . . . .	31
Plot (PL) . . . . .	164
Powers . . . . .	38, 40
Print . . . . .	24, 25, 61, 86, 120
Print zones . . . . .	69
Processing boxes . . . . .	50
Program . . . . .	20
Program errors . . . . .	113
Program mode . . . . .	63
Program variables . . . . .	141
Programming . . . . .	121
? . . . . .	39
Quotation marks . . . . .	79

Random numbers . . . . .	152
Read box . . . . .	132
Read statement . . . . .	132, 138
Read-data . . . . .	132
Read-data statements . . . . .	144
Ready . . . . .	10, 32
REC . . . . .	31
Recreational games . . . . .	174
Reload . . . . .	36
Remark . . . . .	130, 144
Return . . . . .	12, 26, 171
Reverse field . . . . .	13
Revising . . . . .	144
REWIND . . . . .	31, 32
RND . . . . .	152
Run . . . . .	24, 25, 26, 63
RUN return . . . . .	109
Save . . . . .	33
SAVE "D" . . . . .	36
Screen . . . . .	4
Semi-colon . . . . .	69, 77, 79, 119
Setcolor . . . . .	171
Shift key . . . . .	12
Simulation . . . . .	152, 174
Single-alternative decision step . . . . .	54
Skip a line . . . . .	104
Sound (SO.) . . . . .	156, 173
Space bar . . . . .	12
Special symbol keys . . . . .	9
Square root . . . . .	38, 40
Start and stop box . . . . .	50
Step statement . . . . .	96
STOP/EJ . . . . .	31
String variable . . . . .	115
Style . . . . .	130
Subtraction . . . . .	38, 40
System keys . . . . .	14
Text window . . . . .	161
Timer . . . . .	169
Tone . . . . .	156, 158
Tracing . . . . .	143
Typing mistakes . . . . .	15
User errors . . . . .	113
User friendly . . . . .	174
Variable . . . . .	75, 76, 82
Voice . . . . .	156, 157
Writing . . . . .	22
X-coordinate . . . . .	165, 169
Y-coordinate . . . . .	165



# MORE!

## **An Atari for Kids Activity Workbook**

Complete with 91 tear-out worksheets to go with each chapter of the student text, *AN ATARI FOR KIDS* workbook provides practice and reinforcement for skills learned. Most of the activities can be done as seat work without the computer, and students will need the computer only to check their work.

ISBN 0-88056-124-6

91 worksheets  
\$5.95

216 pages/100 illustrations

## **An Atari for Kids Teacher's Guide**

The teacher's guide features 91 worksheets, complete with answers. Additional information and hints for teachers are provided. Information on how to convert the material for other brands of microcomputers and how to use the curriculum is also discussed.

ISBN 0-88056-109-2

91 worksheets  
\$14.95

90 pages

NOTE: For every 25  
copies of *AN ATARI FOR KIDS*  
student texts ordered, receive  
one teacher's guide free!

### **BILL TO:**

Name: \_\_\_\_\_

School: \_\_\_\_\_

Address: \_\_\_\_\_

City, State, Zip \_\_\_\_\_

Phone number \_\_\_\_\_

### **SHIP TO: (if other than bill to)**

Name: \_\_\_\_\_

School: \_\_\_\_\_

Address: \_\_\_\_\_

City, State, Zip \_\_\_\_\_

Date \_\_\_\_\_ P.O. No. \_\_\_\_\_

\_\_\_\_\_ An Atari for Kids, ISBN 0-8856-123-8  
Student Text \$9.95

\_\_\_\_\_ An Atari in the Classroom  
Activity Workbook \$5.95

\_\_\_\_\_ An Atari in the Classroom  
Teacher's Guide \$14.95

\_\_\_\_\_ Check here if your order is over 25  
copies of *An Atari for Kids* to receive  
a free copy of the teacher's guide.

Mail order to: **dillithium Press**  
P.O. Box 606  
Beaverton, OR 97075

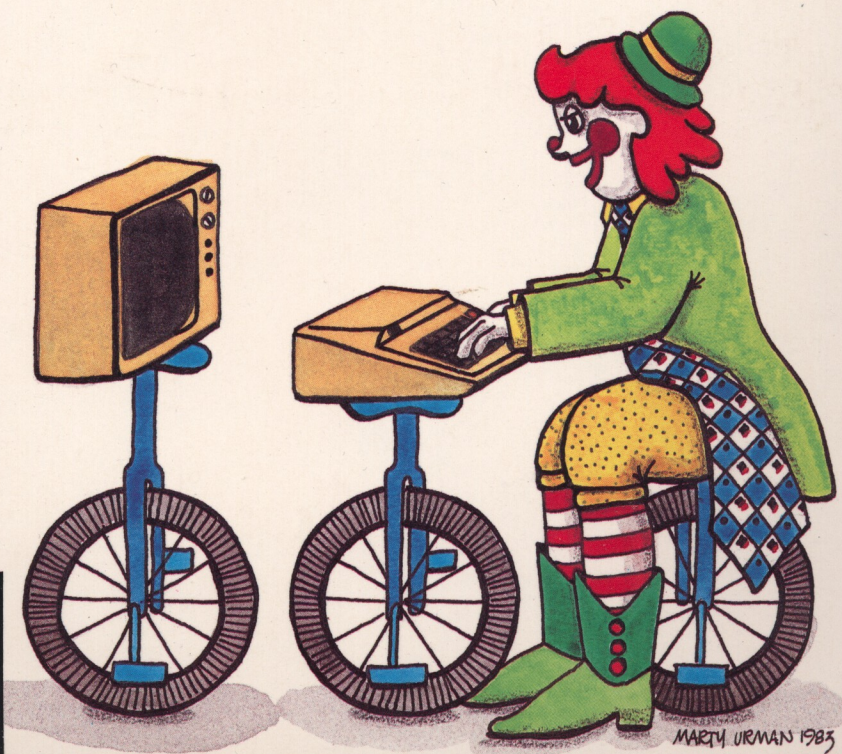
To expedite your order, phone 800-547-1842  
or (inside Oregon 646-2713



**AN ATARI FOR KIDS** is written by teachers who want to teach school-age kids computer operation and programming in BASIC. Using an individualized, self-paced approach, this book encourages kids to be creative programmers while learning good programming techniques.

Full of illustrations and activities to make the learning process fun, **AN ATARI FOR KIDS** is written for the 4th to 8th grade student. It focuses on problem solving, improved thinking skills and creativity. 18 activity worksheets are included to make this a fresh, instructive and fun approach to learning programming.

**AN ATARI IN THE CLASSROOM: Activity Workbook** and **AN ATARI IN THE CLASSROOM: Teacher's Guide** are available to accompany **AN ATARI FOR KIDS**.



ISBN 0-88056-123-8

>>\$9.95



dilithium Press  
PROGRAMMING