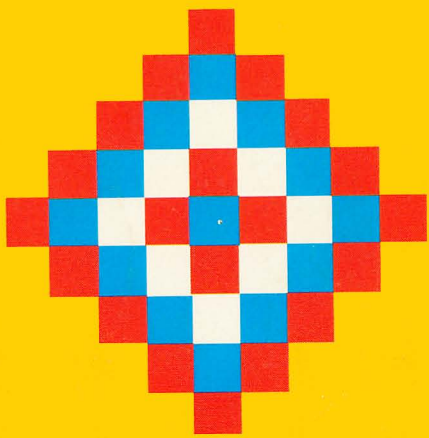


Atari Edition

# COMPUTERS FOR KIDS

Sally Greenwood Larsen



A BASIC programming manual written just for kids.  
Special section for teachers and parents.



# **COMPUTERS FOR KIDS**

**ATARI EDITION**





# **COMPUTERS FOR KIDS**

**ATARI EDITION**

**SALLY GREENWOOD LARSEN**

**creative computing  
press**

Copyright© 1981 by Creative Computing.

All rights reserved. No portion of this book may be reproduced—mechanically, electronically, or by any other means, including photocopying—without written permission of the publisher.

Library of Congress Number: 81-66614  
ISBN: 0-916688-22-4

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

Creative Computing Press  
39 E. Hanover Avenue  
Morris Plains, New Jersey 07950

**To my parents,  
Jack and Donna Greenwood,  
who have always been  
there when I needed them.**

With special thanks to  
Robert Callan for  
translating this book to use  
with the Atari personal computer

# TABLE OF CONTENTS

Section	page
1: What Is a Computer? _____	1
2: Flowcharting _____	4
3: Running the Computer Itself _____	9
4: Saving your Programs with a Cassette Recorder _____	14
5: Saving your Programs with a Disk Drive _____	18
6: Getting Ready to Program _____	25
7: PRINT and Variables _____	28
8: GOTO and INPUT _____	41
9: IF-THEN and FOR-NEXT _____	44
10: Graphics Programs _____	51
11: Sample Programs _____	64
12: Glossary of Statements and Commands _____	66
Notes for Teachers and Parents _____	77



## SECTION 1: What is a Computer?

When a caveman had work to do, he had no machines or tools to help him. He had to do it all by himself. Man has since invented many tools to help him with his work.


Instead of pounding with his hands, he now uses a hammer. The hammer lets him pound harder and longer than he could pound with his hands alone.

Man invented the telescope so that he could see farther into space. He can now see stars he did not know existed before he had the telescope to help his eyes.

Using his brain, man can remember information and solve problems.

Man wanted to invent a tool so that he could extend the use of his brain, so he invented the COMPUTER.

Just as a hammer can't do work without a person to hold it, a computer cannot do work without a person to run it and tell it what to do. This person is called a PROGRAMMER.



Even the best hammer cannot do all the different things our hands can do. And even the best computer cannot do everything our brains can do.

A computer cannot feel emotion. It cannot feel happy or sad, as we can.

A computer cannot combine ideas the way our brain can. It can't put two ideas together and take the best parts of each one to make a brand new idea.

But . . . a computer **can** do some of the simpler jobs our brains can do. And it can do some of them even faster than we can!

A computer can remember many more things than most of us can with just our brains, especially things like long lists of names or numbers. This information is kept inside the computer in the **MEMORY**. Computer programmers call this information **DATA**.

A computer can **compare** data to see if one thing is bigger than another, or smaller, or the same. It can also put things in order.

A computer can sort many pieces of data and put together the things that are alike.



And a computer can look in its memory to find the data a programmer wants, and print out that data on a video screen or a sheet of paper.

This book is about the ATARI computer made by ATARI, Inc.

These are special directions for *this* computer. They will not work on all other kinds of computers.

The ATARI is called a MICROCOMPUTER, because it is so small. Many businesses and universities have computers, too, but theirs have to do many more jobs than the ATARI, so they have to be much larger. Some of the biggest computers are so huge, they fill an entire *room*!

The ATARI uses a special computer language called ATARI BASIC®. It is an easy language to learn, because it uses words we hear every day.

Some bigger computers use languages called FORTRAN or COBOL. You might hear about other languages when you find out more about computers.

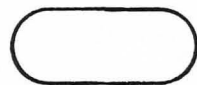
## SECTION 2: Flowcharting

When you want the computer to do a job for you, you must break down the job into small steps, so the computer can understand what to do. One big job may have many small steps, and sometimes it is hard to keep track of all the steps.

One way of keeping track is with a **flowchart**. A flowchart shows all the steps in a problem, shows what choices there are, and in what order the steps must be done.

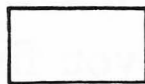
On the next page is a flowchart showing all the little steps in a funny problem. The directions on this flowchart are things for you to do. They are not directions for the computer!

The shapes drawn around the steps show what **kind** of a step it is:



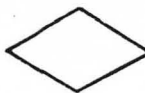
OVAL

— for START or END.



RECTANGLE

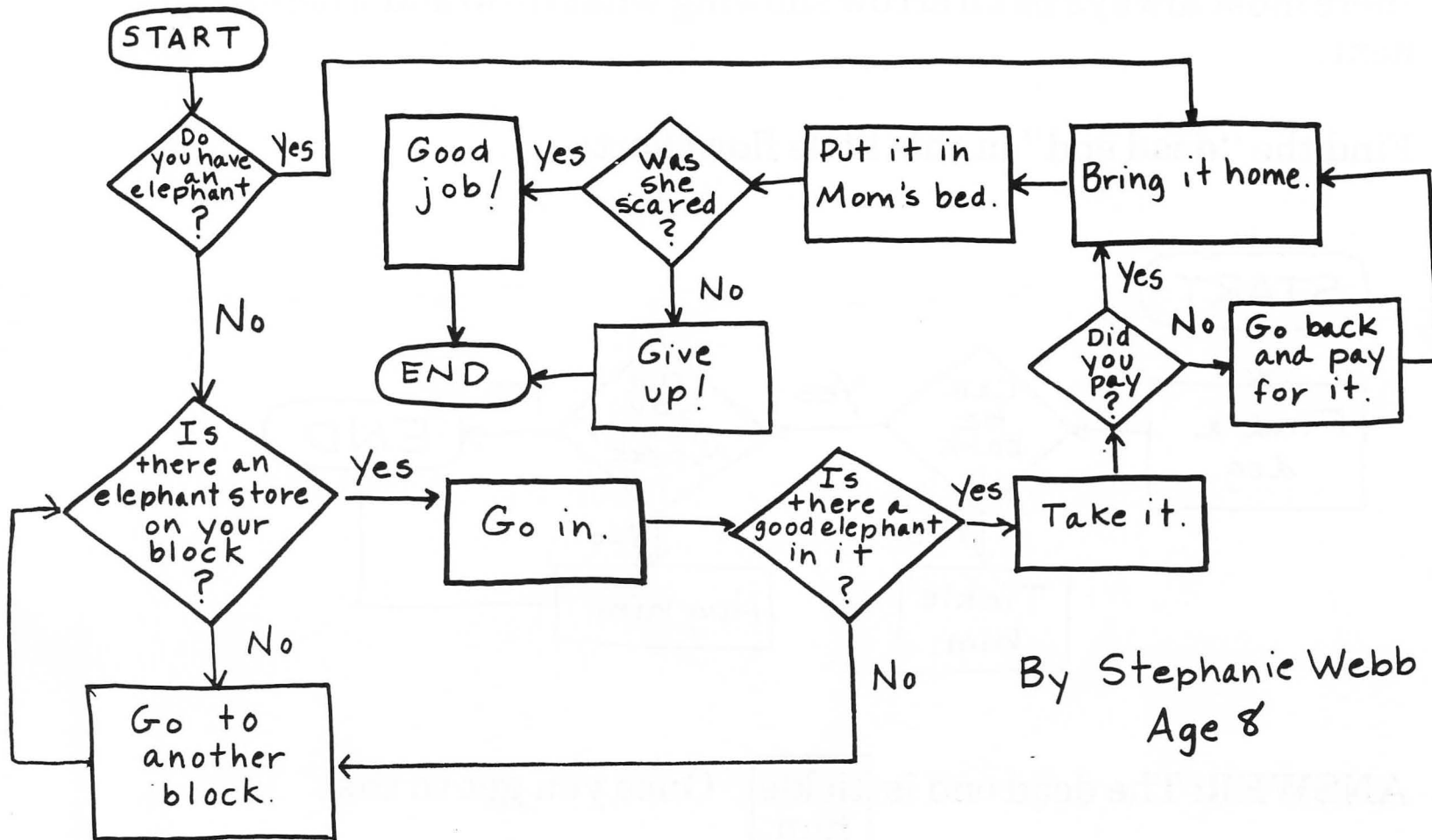
— for statements telling exactly what to do (you have no choice).



DIAMOND

— for yes or no questions.

## How to Scare Your Mom with an Elephant

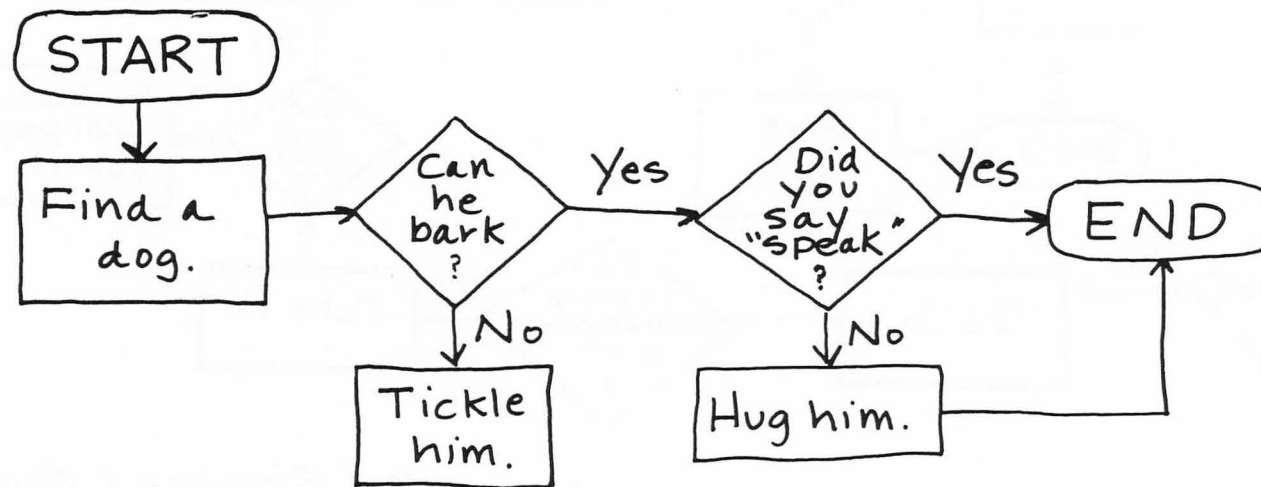


By Stephanie Webb  
Age 8

The arrows on a flowchart show you what to do next. One arrow shows what to do if the answer is *yes*. The other arrow is for *no*.

There must be no “dead ends” in a flowchart. This means that there must always be an arrow showing what to do and where to go next.

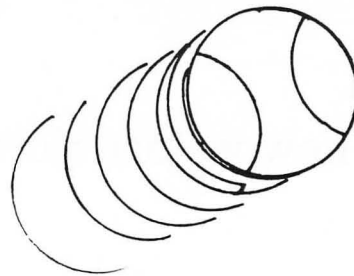
Find the “dead end” in this little flowchart:



**ANSWER:** The dead end is tickle him. Once you get to that statement, there is no arrow showing you where to go next.

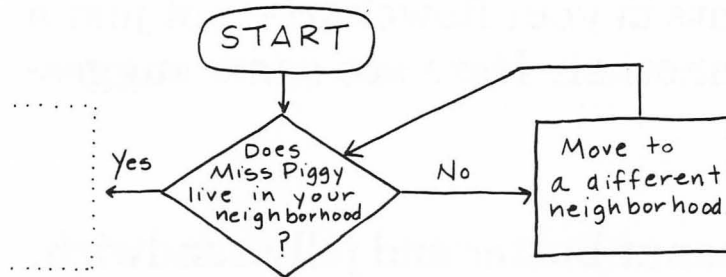
When you write your own practice flowcharts, pick a subject you know something about. Also, your flowchart will be much more interesting if you pick a topic which has some choices in it. You want both questions *and* statements in your flowchart—not just a page full of one statement after another. Here are some suggestions:

1. How to make a peanut butter and jelly sandwich.
2. How to take a bath.
3. How to make your mother scream.
4. How to play kickball.
5. How to buy a birthday present.



When the arrows in a flowchart make you do something over and over again, this is called a *DO-LOOP*. Here is an example: (I have only drawn *part* of this flowchart.)

### How To Get A Date With Miss Piggy



If you follow the directions for this part of the flowchart, you will keep moving to a new neighborhood until you are living in Miss Piggy's neighborhood. This is called a **do-loop**.

In a do-loop, you keep coming back to the question you asked until you finally get the answer you need so you can go on to the rest of the flowchart. In the example on this page, in order to “get a date with Miss Piggy”, you had to first move into her neighborhood.

In Section 8, you will see how we use flowcharts to help us write computer programs.

## SECTION 3: Running the Computer Itself

The ATARI Computer has two basic parts:

The Keyboard - This looks like a regular typewriter keyboard. It has all the electronics for the computer inside it. You type in information and instructions for the computer on the keyboard.

The TV screen - The information you type on the keyboard is printed out on the TV screen, so you can see what you are doing. The keyboard can be hooked up to almost any TV set. However, when you learn to make pictures on the screen, you will see that you need a color TV to make colored pictures. (If you use a black and white TV, the pictures you make will be black and white.)

The computer has a *Random Access Memory* (RAM for short). This means that the computer will hold data in its memory only as long as the keyboard is left on, so it has electricity flowing through it. If you turn off or unplug the keyboard, you will lose your program. (Turning off only the TV won't matter.)



After you have written a program, you will want to be able to save that program some way, so the next time you want to use it, you won't have to type in the whole program again.

The ATARI has two ways to save your programs—with a cassette tape recorder, or with a disk drive. Your computer will probably have either a recorder or both a disk drive and recorder.

The cassette tape recorder is like the kind you use to record music. It is hooked to the keyboard with a special wire.

The disk drive is a small square box, attached to the keyboard with a special wire.

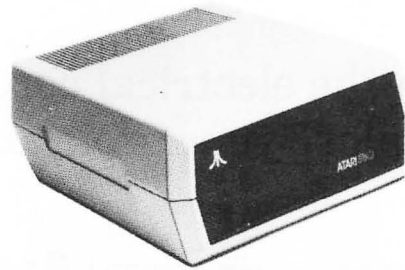
When you have written a program you want to save, you can find the directions for using the cassette recorder in Section 4, and the directions for the disk drive in Section 5.

The first time you use the ATARI, have an adult help you set up the machine and connect all the proper plugs. It is not hard to set up the computer, once you know how, but it is important that the job be done correctly, or you will ruin it.



This is an illustration of your Atari. Follow these four steps to get it ready to program.

1. If you have a disk drive *turn it on first* (see instructions in Section 5).
2. Turn on the TV set and insert the Basic cartridge in the computer.
3. Close the lid.



4. Turn on the computer using the switch on the right hand side.

If the word **READY** doesn't appear on the TV screen you may need to check to see that the TV is set to channel 2 or 3. If the image is fuzzy you can change to the other channel. If you do, you must flip the small switch on the computer in front of the ON/OFF switch.

**IF YOU AREN'T SURE—ASK FOR HELP!!!** It could take weeks to get your computer fixed if you break it.

## Things To Remember

1. Before you start programming, you must turn on both the TV, the keyboard, and disk drive. (If this doesn't work, you probably forgot to plug them into the transformers or the wall socket.) The TV volume should be turned on slightly so you can hear loading sounds.
2. Take it easy with the keyboard—no pounding, please!
3. Keep your feet away from the electrical cords. If you accidentally kick out a plug, you may lose your program.
4. Good programmers never eat or drink while working. The computer will not work well if it is full of cookie crumbs, or has soda spilled between the keys!
5. The only time you have to open the lid of the keyboard is to put in the BASIC ROM cartridge. The power goes off when you lift the lid. Be careful not to let anything fall into this compartment. Be sure to put the BASIC cartridge into the left hand slot.

6. Turn off the TV, the keyboard, and the disk drive, when you are not using them.

Unless you need to save a program right away, skip Sections 4 and 5 about the cassette recorder and the disk drive, and go right to Section 6—Getting Ready to Program.

You can come back to Sections 4 and 5 when you need them.



## SECTION 4: Saving Your Programs with a Cassette Recorder

You now have a computer program typed into the computer, and you want to save it on a cassette tape. Follow these directions carefully.

### Saving Your Program on a Cassette

1. Advance your cassette tape to the spot where you want to record your program. REMEMBER THE NUMBER OF THE TAPE RECORDER COUNTER! Just to be safe, make sure you have gone five numbers past the end of the last program on that tape.
2. Type `CSAVE` on the computer and press the `RETURN` key. The computer beeps twice.
3. Press down both the `PLAY` and `RECORD` buttons on the tape recorder. Hold them down until they stay. Hit `RETURN` again and the recorder starts recording.
4. The cursor ■ will disappear from the screen while the program is being saved on your tape. `READY` will appear on the screen when the recording is over. When the recording is finished the cursor will come back on the screen.

If the computer screen says

ERROR 138 or ERROR 140

you started the tape too far from the start of the program.  
Press RESET , rewind the tape, and try again.

5. Your program is now recorded on the cassette. Make sure the location and subject of your program is written down on the cassette label, so you can find it later. Saving the program on a tape does not erase it from the computer's memory. You must type NEW to do that.

### Special Notes:

- \* To be sure you get a good recording of an important program, you should record the program several times on the tape, with a little space between recordings, of course.
- \* You must use a good quality cassette tape, and it should be new. You should not record one program over the top of another, or erase an old music tape and use it for computer programs. Computer companies sell special short cassette tapes just for this use. They cost less than most tapes, and they work very well. You can save lots of programs on one short tape.

If you have saved a program on a cassette, and now want to load it into your computer, follow these directions carefully:

### Loading a Cassette Tape Program into the Computer

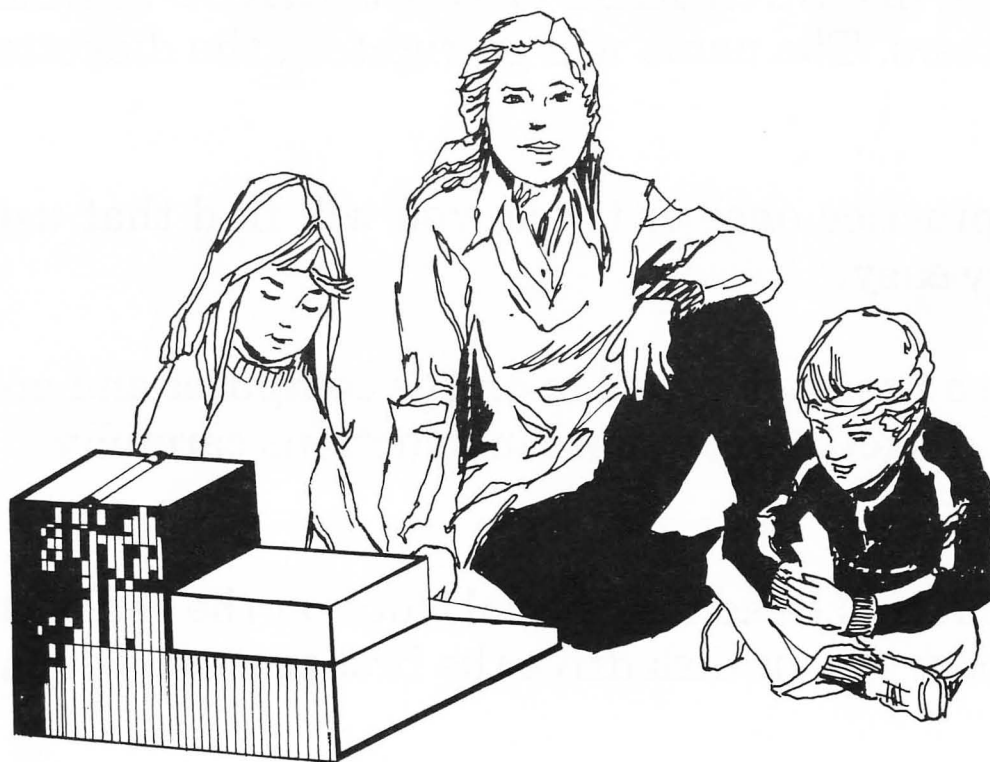
1. Advance your cassette to two or three numbers **before** the location of your program. (If your program is recorded at 85, for example, advance the tape to 82 or 83 on the counter.)
2. Type CLOAD on the computer, and press the RETURN key. The computer will beep.
3. Press down the PLAY button on the tape recorder.
4. Press RETURN on the computer. The computer will beep again.
5. The cursor ■ will disappear from the screen while your program is being loaded into the memory. When it is finished, the cursor and "READY" will appear again.

If the computer screen says

ERROR-138 or ERROR-140

then press **RESET** , rewind the tape, and try again.

6. Your program should now be in the memory of the computer.  
You can LIST it, or RUN it, just like any other program.





## SECTION 5: Saving Your Programs with a Disk Drive

If you have a disk drive with your computer, you will be saving your programs on a round piece of magnetic tape called a **floppy diskette**. In this book we will call it a **diskette**, for short.

Before you save a program on a diskette, you must give your program a name, like SURPRISE or WORMRACE or a name with 8 or fewer letters. The name will go right on the diskette with the program.

After you practice once or twice, you will find that using a disk drive is *very* easy.

If you have a program typed in on the computer and are ready to save it on a diskette, read these instructions carefully.

**IMPORTANT!** Read the Special Notes at the end of this section before you use your disk drive the first time, or you may ruin it.



## Procedure for Inserting a Diskette Into the Disk Drive

Once you have turned on the computer and started to program, it is too late to put a diskette into the drive and save your program. You must turn on the disk drive before you turn on the computer. You must always put your diskette into the drive *first* if you want to save or load programs from your disk.

Make sure everything is off—the computer, disk drive and TV. The door to the diskette should be open. Turn on the disk drive and wait for it to stop spinning. At that time the top red busy light will go off.

Then put the disk in and close the door to the disk drive. Now you can turn on the TV and the computer.



## Saving Your Program on a Diskette

1. Choose a name for your program. It is easier if you pick a short name, such as

SURPRISE

2. Type SAVE "D:SURPRISE" (the name of your program) and press the RETURN key.

3. The red light on the front of the disk drive will come on, and you will hear little noises coming from inside. When the red light goes off, your program SURPRISE is saved on the diskette. Saving a program on the diskette does not erase it from the computer's memory. You must type NEW to do that.

4. You can check to see if the program is really on the diskette by typing


DOS

and pressing the RETURN key. This will turn on the disk drive and make the computer print a menu. To get a list of all the programs stored on that diskette, type A and hit RETURN two times. Next to each program name you will see a number. This is a special code to show how much space each program uses on the diskette. You won't need to use these numbers at all.

Now, suppose you want to find a certain program on a diskette and load it into the computer, so you can use it.

### Loading a Diskette Program into the Computer

1. Type DOS and hit `RETURN`. This will enable you to communicate with the disk drive. You will see a list of instructions with letters in front of them. This is a MENU. It tells you what things the Disk Operating System can do. Type A and hit `RETURN` and it will ask you which programs you want to look at. Hit `RETURN` again; the “busy” light on the disk drive will come on and the computer will print out a list of all the programs on that diskette.
2. Find the name of the program you want. Look carefully at how it is spelled.
3. Hit `RESET`.
4. Type LOAD “D: SURPRISE” if you want to load the program named SURPRISE into the computer. The name of the program must be spelled (and spaced) **exactly** the same as it is on the diskette catalog list, or the computer will print  
ERROR 170  
which means there is no program by that exact name on the diskette. (You will have to try again.)

- 
5. The red light on the front of the disk drive will come on while SURPRISE is being loaded, and you will hear little noises inside the disk drive. The light will go off when the program is loaded.
  6. The program you wanted is now in the computer's memory. You can use it the same way you would use any other program you typed in yourself.

If you decide you don't want a program on a diskette any longer, here is how to erase it:

### **Deleting (Erasing) a Program From a Diskette**

1. Type DOS and hit **RETURN** . This will enable you to communicate with the disk drive. You will see a list of instructions with letters in front of them. This is a MENU. It is a list of things that the Disk Operating System can do. To DELETE (or Erase) a program, type D and hit **RETURN** . The computer will ask "which file to delete?" You must type in the program name, D:SURPRISE, and hit **RETURN** . The computer will say "Type "Y" to delete", then you must type

Y and hit **RETURN** . This will erase the program named SURPRISE from the diskette. The computer now says "SELECT ITEM".

2. To be sure that the program is no longer on the diskette, Type A and hit **RETURN** . The computer will ask you what programs you want to see. Hit **RETURN** again and the "busy" light will come on. The computer will show you the programs on the diskette, and SURPRISE will no longer be there.

### Special Notes

- \* Have an adult show you which way to put the diskette into the disk drive. Be careful not to do it upside down or backwards!
- \* Your diskettes should be kept in their special paper jackets, and should never be left out to get dirty, or be stepped on.

- \* Static electricity from your fingers or a diskette can ruin your disk drive. Especially in the wintertime, or if you are walking over rugs, your body will pick up a static electricity charge. Then if you reach over to put a diskette into the disk drive, that spark will jump from the diskette into the disk drive, which can damage the disk drive.

You can easily get rid of the static charge on your body by touching the metal surface of your television set or monitor.

Touching this surface is called “grounding yourself out.” You must get into the habit of **always** doing this before you touch the disk drive.





## SECTION 6: Getting Ready To Program

When you write a program, you are writing a list of instructions the computer needs to do a particular job, such as printing your name on the screen. These instructions are called **program statements**, and you'll learn more about them in Section 7.

But sometimes you need to tell the machine itself to do something, such as get rid of an old program so you can write a new one, or clear all the printing off the TV screen. These are called **commands**, because they are not part of a program. You type them in and press `RETURN`, and the computer does then right away.

(I wrote `RETURN` in a box, because it has a special key all its own on the keyboard, like `CTRL` and `RESET`.)

Program statements all have **line numbers** in front of them, to tell the computer which statement should be done first. (You will see these line numbers in Section 7.) Remember that commands do not have a line number because they are not part of a program.

Here are some of the **commands** you will need. Remember to press **RETURN** after each one you use.

**SHIFT**  
**CLEAR**

this clears all the printing off the screen, but it does *not* take your program out of the memory. Remember—just because the information isn't printed on the TV screen doesn't mean it isn't stored inside the computer any more!

**NEW**

this erases your last program from the memory so you can start on a new program with a "clean" memory.

**LIST**

this prints out, in order, whatever program statements you have typed into the memory so far.

**CTRL** — **1**

by pushing the **CTRL** key and **1** at the same time, you can control how fast the computer will LIST a very long program that won't fit on the screen all at the same time. Press **CTRL** — **1** and the computer will *stop* listing the program. Press **CTRL** — **1** again, and it will *continue* with the list. You can start and stop as often as you like, by using **CTRL** — **1** each time.



RUN

this tells the computer you have finished typing in all the instructions in your program, and now you want the computer to do that job. This is called **executing** the program. When the computer is finished executing the program, it will print ■ on the screen. This is called the **cursor**, and will show where you will be typing next.

**BREAK**

If the computer is in the middle of executing your program and you want it to stop, press

**BREAK** . The computer will print

STOPPED AT LINE 10

which means that line 10 is the last line the computer worked on before you made it stop. For example, if the computer program was stopped at line 45, the computer would print

STOPPED AT LINE 45

CONT

If you change your mind and want the computer to continue executing the program after you pushed **BREAK** , type in CONT and press **RETURN** . The computer will continue at the place it had stopped.

## SECTION 7: PRINT and Variables

Let's begin by writing a program using the PRINT statement.

The computer will be using the TV screen to print letters or numbers (rather than drawing pictures). So our first statement should be a PRINT statement to print out the following message:



```
10 PRINT "HELLO! I AM THE ATARI!"  
15 PRINT "THIS MUST BE YOUR FIRST PROGRAM."  
20 END
```

... and the last line in the program will be an END statement to show the computer where the program ends.

Now—when we type in RUN and press the **RETURN** key, this is what the computer will show on the screen:



```
READY
10 PRINT "HELLO! I AM THE ATARI!"
15 PRINT "THIS MUST BE YOUR FIRST PROGRAM."
20 END
RUN

HELLO! I AM THE ATARI!
THIS MUST BE YOUR FIRST PROGRAM.

READY
█
```

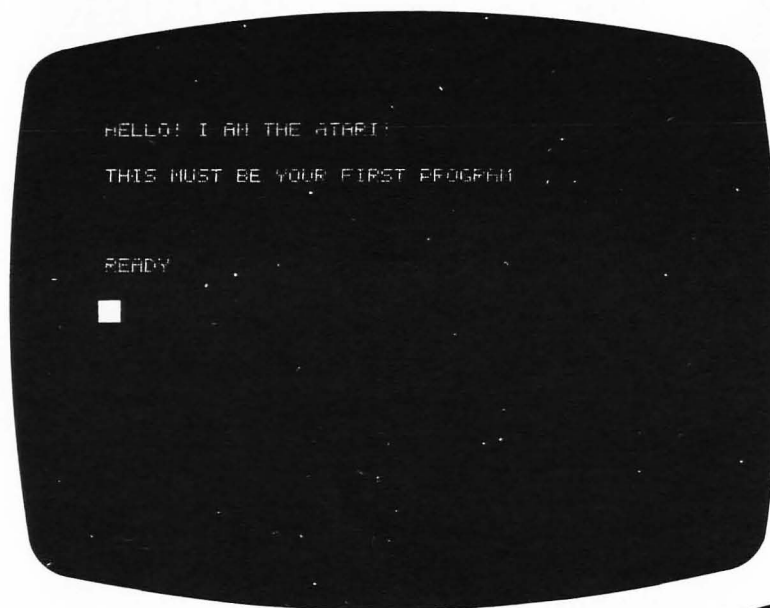
*You try typing it in. Remember to press **RETURN** each time you finish typing a line.*

RUN does not have a line number in front of it.

This is how the computer will follow the directions you gave it in your program.

When the computer is done following your instructions, this little square (called a **cursor**) will appear, which shows that you are ready to type again.

Suppose you want the computer to run the same kind of program, but this time you want the instructions erased from the screen before the program is executed. In other words, you want the computer to erase the screen first, before it follows your instructions. Then when you RUN your program, the screen would look like this:



To do this, all you need is one more statement in your program.

5 GRAPHICS 0

is a statement which tells the computer to erase the screen. Here's where it should go in your program:

5 GRAPHICS 0

10 PRINT "HELLO! I AM THE ATARI!"

15 PRINT "THIS MUST BE YOUR FIRST PROGRAM. "

20 END

GRAPHICS 0 is a very handy statement to remember. It helps you get any “garbage” off the screen that might be in the way when you run your program. You can abbreviate it to GR. 0.

(You also can clear the screen by using **SHIFT** **CLEAR** as a command without a line number. Just hit the **SHIFT** and **CLEAR** keys together. You will use **SHIFT** **CLEAR** and LIST often, when you write long programs of your own.)

Whenever you use a statement like PRINT “HELLO” the computer will print out exactly what you put between the quotation marks. Even if what you put is silly! Even if it’s spelled incorrectly!

Here are some examples for you to try. Go ahead and make up some of your own!

5 GR. 0

15 PRINT "I KIN SPELL REEL GOOD. "

20 PRINT "GIGGLE! GIGGLE!"

25 END

RUN

(Run is *not* part of the program, but I’m putting it here so you don’t forget to type it in every time you want to *run* your program. Later, I won’t write it down each time.)

```
10 GR. 0
```

```
30 PRINT "MY NAME IS JOHN SMITH. "
```

```
40 PRINT "MY NAME IS MARY JONES. "
```

```
50 END
```

```
RUN
```

```
5 GR. 0
```

```
15 PRINT "I AM A FRIENDLY COMPUTER. "
```

```
20 END
```

```
RUN
```

Before we go on with PRINT statements, let's talk about **line numbers**.

Every statement in a program has a number in front of it. This tells the computer which statement to do first. The computer will start with the lowest number and end with the highest number, no matter how many numbers you skip inbetween. Good programmers always count by 5's or 10's when they number their lines, so that if they leave out a line by mistake, they can put it in later, and there will be room. For example:

```
5 GR. 0
```

```
15 PRINT "MY NAME IS ROBBIE. "
```

```
20 PRINT "MY BIRTHDAY IS JULY 3RD. "
```

```
25 END
```

Now—if I wanted to put a line in my program telling how old Robbie is, right after line 15, I could just type in:

```
18 PRINT "I AM 9 YEARS OLD. "
```

If I type **SHIFT** **CLEAR** to clear the screen, then type LIST, the computer will put line 18 into the program in the right place, and this is how the program will appear:



```
5 GR. 0  
15 PRINT "MY NAME IS ROBBIE. "  
18 PRINT "I AM 9 YEARS OLD. "  
20 PRINT "MY BIRTHDAY IS JULY 3RD. "  
25 END
```

This is very helpful if you forget something in your program.

You can use the same idea to **delete** (take out) a line in your program if you make a mistake or just decide you don't want that line anymore.



```
5 GR. 0
15 PRINT "TODAY IS TOOSDAY. "
20 END
```

In this program, line 15 has a spelling mistake. To get rid of line 15 completely, all you do is type in:

```
15
```

and hit the **RETURN** key. This will erase line 15 from the memory, and you can type in a new line 15.

But suppose you don't want to replace a whole line, but only change a letter or two. Can you erase part of a line? Of course. Just press **CTRL** and one of the **↑** **↓** **→** **←** keys until you move the cursor on top of the letter you want to change. Then type the letter you want.

In this case, if you are still on line 15 when you notice you misspelled Tuesday, press **CTRL** and **←** until you are on top of the first 0. Type U and E. Then press **CTRL** **→** until you are at the end of the line, so you won't erase letters that you want to keep.



If you are already at line 25 by the time you notice the mistake, press **CTRL** **↑** until you reach line 15, then **CTRL** **→** until you are over the 0. Type in U and E.

Once you have made the corrections on your line, press **RETURN** and the computer will replace the line in its memory with the changed line. If you do not press **RETURN** the computer will forget the change.

You may also be wondering why zero is written with a line through it, like this:

Ø

This is done on computers so that there is no mix-up between the number zero and the letter O. You should use the special zero when you write your programs on paper, too.

When you type in your own programs on the ATARI, if you type in something the computer does not understand, it may print

**ERROR**

followed by the things that are incorrect.

This means that you have made a mistake in a statement or command, or you have used the wrong statement. These messages from the computer are called **error messages**. They help you figure

out what kind of mistake you have made, so you can fix it. The cursor ■ will appear right after the mistake, to show you where the mistake is.

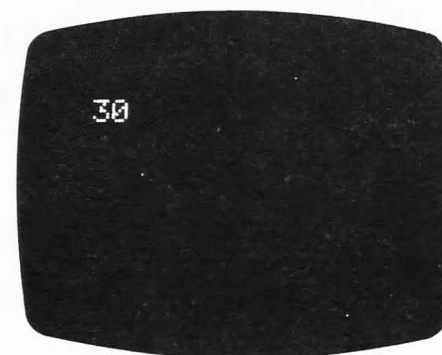
The PRINT statement can also be used to skip a line.

```
5 GR. 0
15 PRINT "HELLO"
20 PRINT
25 PRINT "GOOD-BYE"
30 END
```



Without the quotation marks in a PRINT statement, the ATARI will work like a calculator:

```
5 GR. 0
15 PRINT 10 + 20
20 END
```



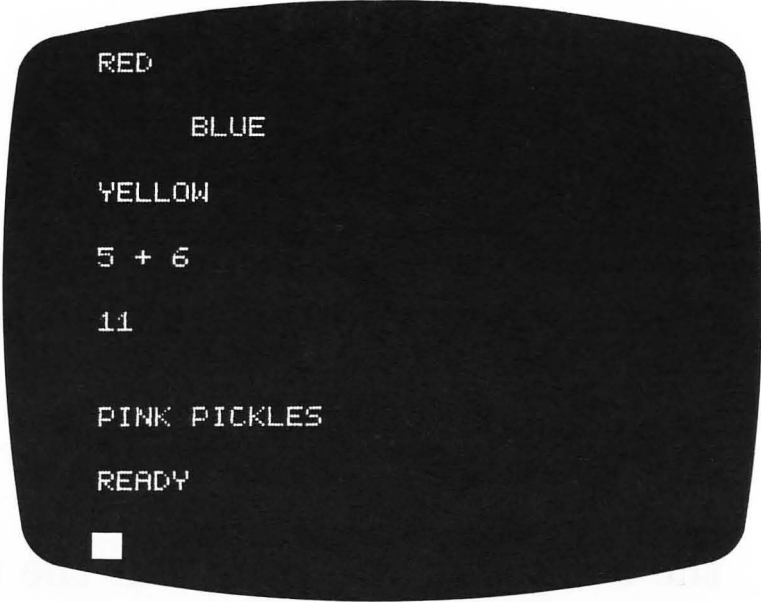
This program will print out the **answer** to  $10 + 20$ , which is 30. If you wanted the computer to print out the actual problem  $10 + 20$ , you would write it like this:

```
10 PRINT "10 + 20"
```

Notice the difference?

Here is a PRINT program and the results on the screen when the program is executed. Look it over carefully.

```
5 GR. 0
15 PRINT "RED"
20 PRINT "    BLUE"
25 PRINT "YELLOW"
30 PRINT "5 + 6"
35 PRINT 5 + 6
40 PRINT
45 PRINT "PINK PICKLES"
50 END
```



```
RED
    BLUE
YELLOW
5 + 6
11
PINK PICKLES
READY
```

Notice that line 50 END does **not** print the word “END” on the screen. It just tells the computer that this is the **end** of your program.

The computer can also keep a number in its memory, and print it out later when you ask for it. Let’s look at how the memory works.

The memory is like a big Post Office, with letters of the alphabet on each “mailbox.” You put a number in a “mailbox” by using a LET statement.

A	B	C	D
5	7	2	0

45 LET A = 5

50 LET B = 7

55 LET C = 2

60 LET D = 0

Now whenever you use this statement in your program,

70 PRINT A

the computer will print out the number or value in the mailbox called A. Of course, if you want the computer to print out the value of A, you must make sure you put a number in mailbox A earlier in your program, or the computer will assume you wanted the value in mailbox A to be *zero*. This works for all the memory locations. If you do not put a number in a memory location, the computer will assume the value is zero.

In computer programs, the letter names you give to the “mailboxes” are called **variables**.

If you write a statement like

```
10 PRINT A + B
```

the computer will look in **A** to see what the value is, then find the value for **B**, and add them together and print out just the answer for you. Here is an example:

```
5 GR. 0
25 LET A = 6
35 LET B = 4
45 PRINT A + B
55 END
```



MEMORY	
A	B
6	4

Later in your program, if you want to change the number stored in mailbox **A**, you can use another **LET** statement. This will erase the old value for **A** and put in the new one.

The ATARI uses a few special symbols for arithmetic:

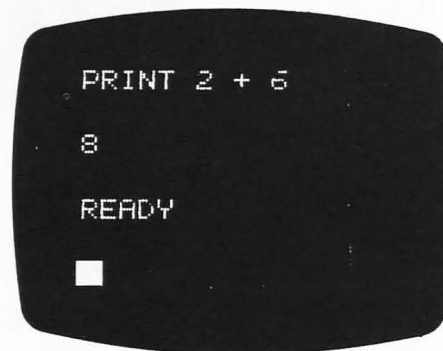
Addition +	3 plus 4 is written as 3 + 4
Subtraction —	5 minus 2 is written as 5 — 2
Multiplication*	6 times 8 is written as 6*8
Division /	6 divided by 2 is written as 6/2

You can use the computer in the **command mode** to do math problems for you.

The **command mode** means that the computer executes each line as soon as you press the **RETURN** key. You are using the command mode when you type in **NEW** or **LIST** or **RUN** when you write programs.

**PRINT** can also be used the same way:

```
PRINT 2 + 6
```



notice—no line number!

The answer to the problem will be printed on the screen as soon as you press **RETURN**.

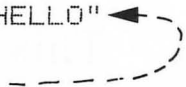
## SECTION 8: GOTO and INPUT

PRINT statements alone don't make very exciting programs. This section has two new statements which make programming more fun!

Let's look at each one, then write some simple programs.

**GOTO** tells the computer to *go to* the line number listed, and do what it says there.

```
5 GR. 0
10 PRINT "HELLO"
15 GOTO 10
20 END
```



Every time the computer gets to line 15, the program tells it to go to line 10.

This program prints "HELLO" over and over and over again. The computer would print HELLO  
HELLO  
HELLO  
HELLO  
HELLO

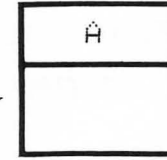
all night long, if you forgot to turn it off! Remember, you can stop the program by typing **BREAK**.

INPUT asks you to type in a number while the program is running.

```
5 GR. 0
```

```
25 PRINT "TYPE IN YOUR AGE. "
```

```
35 INPUT A
```



This sets up a memory space called A, and when you type in your age, it will be stored in memory space A.

Now we can use that information:

```
45 PRINT "YOUR AGE IS"
```

```
55 PRINT A
```

```
65 END
```

This line will print out the number stored in memory space A.

Type this program on the computer and try it yourself. You will notice that when the computer reaches an INPUT statement when it is running a program, it will stop and print “?” until you type in an answer. When you write your own INPUT programs, you must always be careful to put a statement before the INPUT telling the person who uses your program what the computer is waiting for them to type in.



Sometimes you want to stop using a program with an INPUT statement in it. To get the program to stop, type `BREAK` . Now you can LIST your program and fix the mistakes, or type NEW and write another program.



## SECTION 9: IF-THEN and FOR-NEXT

FOR-NEXT statements are *lots* of fun, because you can make the computer do all kinds of work for you!

```
5 GR. 0
```

```
10 FOR X = 1 TO 5
```

```
20 PRINT "TOM"
```

```
30 NEXT X
```

```
35 END
```

This statement says, "I'm going to do something 5 times."

What it will do is print "Tom". This statement tells it what to do each time.


This statement is the "counter." It counts how many times the computer has done its job. When it has done the job the right number of times, it will go on to the next line of directions.

This part of the program is called a **FOR-NEXT LOOP**, because the computer "loops" through that part of the program over and over again, until it has done its job the right number of times.

We can also write a program which has several lines between the FOR and NEXT statements in the loop:

### FOR-NEXT LOOP

```
5 GR. 0
10 FOR X = 1 TO 5
15 PRINT "MY NAME IS JIM LARSEN. "
20 PRINT "I LIKE TO WRITE PROGRAMS. "
25 PRINT "I HAVE MY OWN COMPUTER. "
30 NEXT X
35 END
```

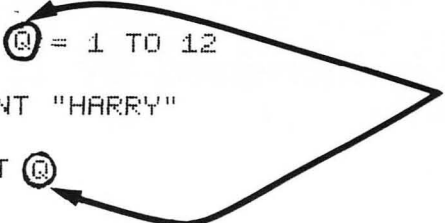


This program will write all three of the PRINT statements each time, until it has gone through the loop five times. It will print a total of 15 lines.

You may use any variable you wish in a FOR-NEXT loop, but the variable must be the *same* in both statements, or the computer will give you the error message ERROR-13.


```
10 FOR @ = 1 TO 12
15 PRINT "HARRY"
20 NEXT @
```

these two variables must be the same



This program will print "HARRY" 12 times.

Let's look at how the counter works in a FOR-NEXT program.



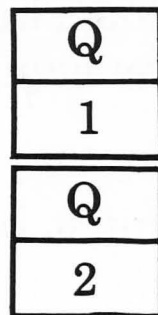
```
10 FOR Q =1 TO 4  
15 PRINT "HELLO"  
20 NEXT Q
```

This statement sets up a memory space named Q. It tells the computer that the **values** stored in Q will start with 1 and end with 4.

```
10 FOR Q =1 TO 4  
15 PRINT "HELLO"
```

Each time the computer goes through the FOR-NEXT loop one time (and prints "HELLO"), the value of Q is increased by one.

After doing line 15 PRINT "HELLO" the first time, the number stored in Q is 1.



Now the computer goes back to line 10 to start the loop again. After it prints line 15 and gets to

```
20 NEXT Q
```

the number in Q is increased by 1. (that's what NEXT Q means.)

This goes on until Q finally gets up to 4. Since the FOR-NEXT loop says

```
FOR Q = 1 TO 4
```

the computer knows that once Q gets to 4, the FOR-NEXT loop is finished, and the computer should go on to the next statement in the program.

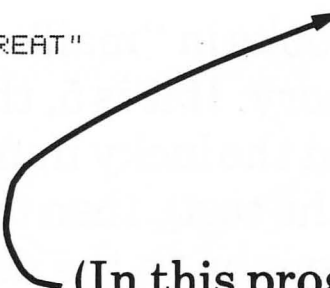
Here are a few sample problems to try. Now take some time and write your own!

### NAME

```
5 GR. 0  
10 FOR Z = 1 TO 100  
15 PRINT "SUSAN IS GREAT"  
20 NEXT Z  
30 END
```

### NUMBERS

```
5 GR. 0  
10 FOR R = 1 TO 100  
15 PRINT R  
20 NEXT R  
25 END
```




(In this program, you can see when the value of "R" changes!)

I have given these programs names, to make them easier to remember. Don't type in the name as part of the program, or the computer will give you an error message. (Of course, if you have a disk drive, you could use these names when you save the programs on a diskette.)

IF-THEN statements provide a "test" for your programs.

```
10 GR. 0
30 PRINT "TYPE IN YOUR FAVORITE NUMBER. "
40 INPUT N
50 IF N = 5 THEN PRINT "YOU HAVE PICKED THE LUCKY NUMBER!"
60 END
```



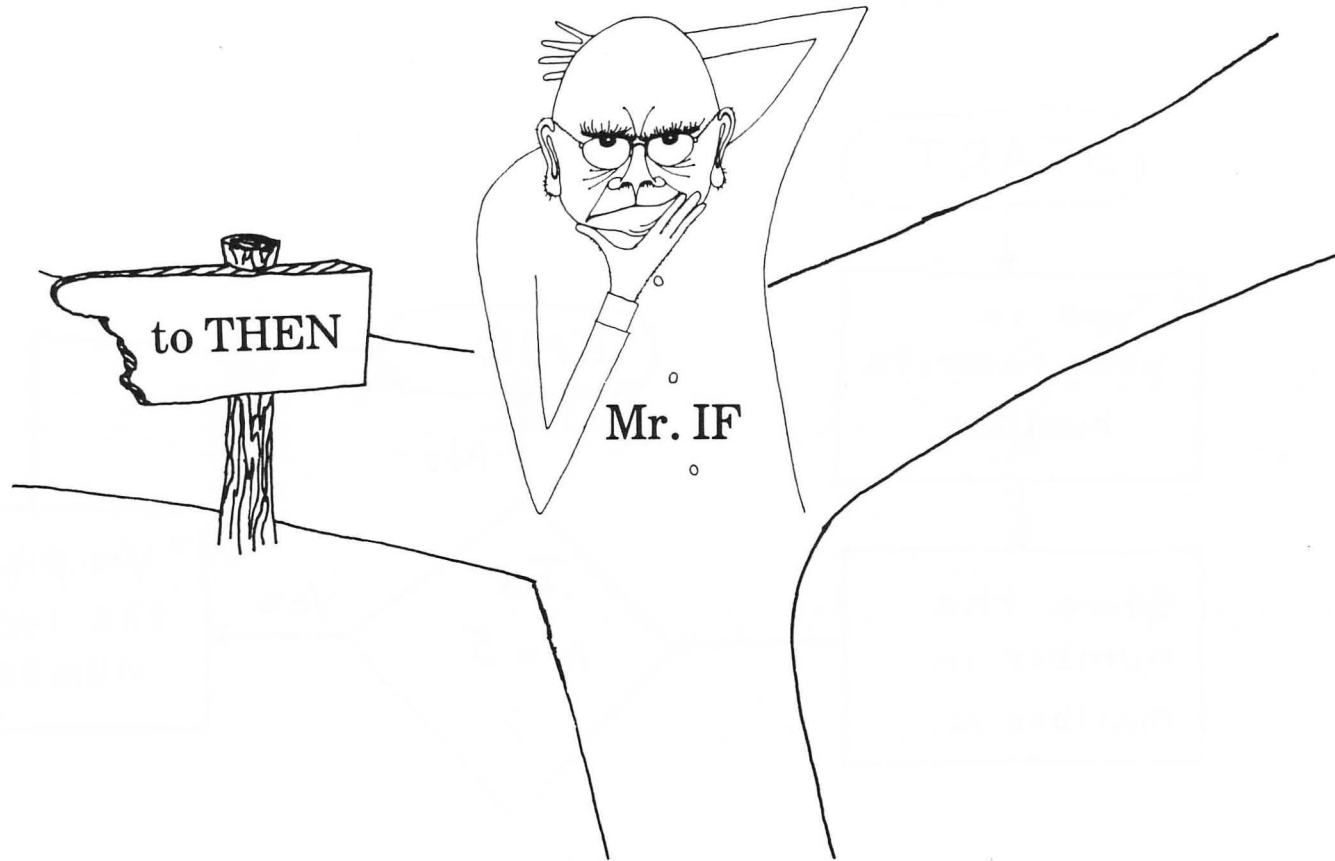
This statement looks in "mailbox" N to see what number is stored there in the memory. If it is 5, then the computer is told to PRINT "You have picked the lucky number!" If the number is *not* 5 (if the number "fails" the test), then the computer ignores the rest of the statement and goes on to the next line.

Let's think about how IF-THEN statements work.

Pretend you are "inside" your program, and you are following all the instructions in the program, just as the computer would.

You are going down the road, and you come to a fork, where there are two ways to go.

This is the IF-THEN statement in the program you are following. Mr. IF has a “test” for you. If you pass the test, you may go down the fork in the road marked “THEN.” If you do not pass the test, you must go the other way.



An IF-THEN statement is called a **branch** in your program.

We can also show this with a flowchart.

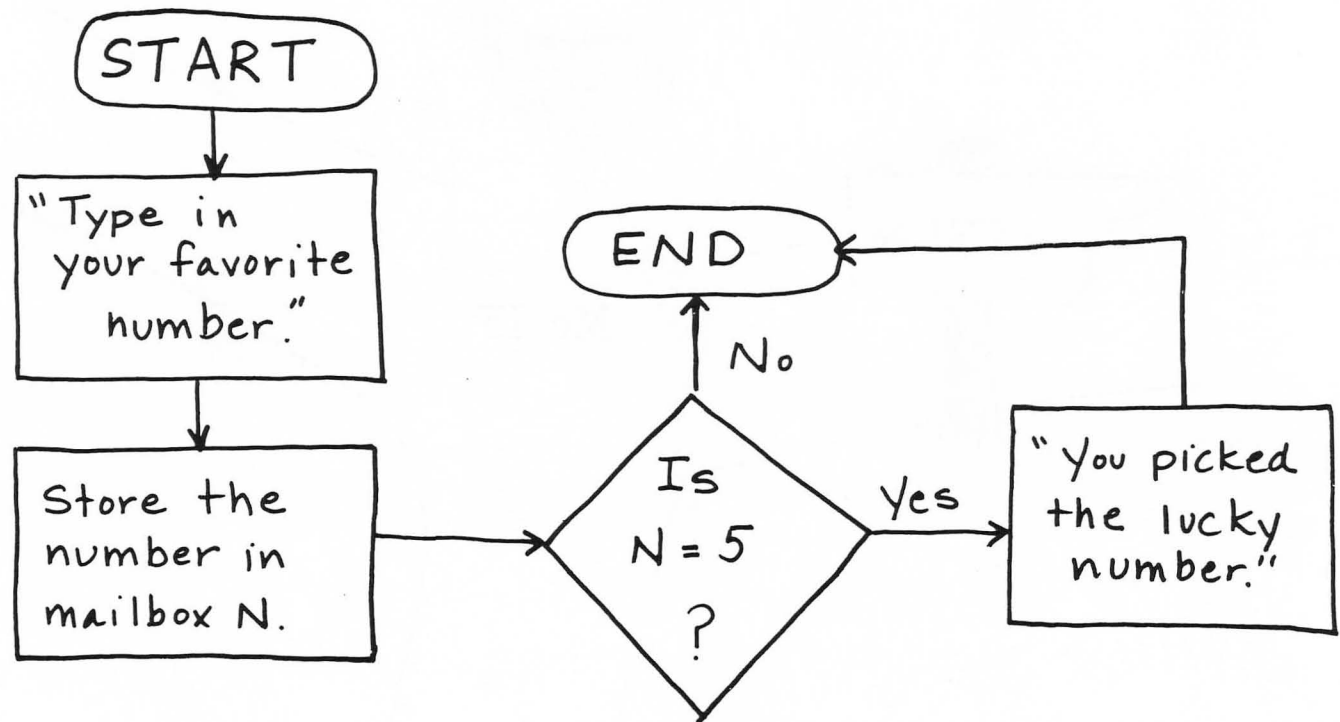
```
10 GR. 0
```

```
30 PRINT "TYPE IN YOUR FAVORITE NUMBER. "
```

```
40 INPUT N
```

```
50 IF N = 5 THEN PRINT "YOU PICKED THE LUCKY NUMBER!"
```

```
60 END
```





## SECTION 10: Graphics Programs

Graphics programs let you make pictures on the TV screen with blocks of light. Each block (really a tiny rectangle called a pixel) has an address of its own. Your graphics program tells the computer which blocks you want lighted.

The ATARI is a special computer, because it has **color** graphics. Not only can you pick which blocks you want, but you have different colors from which to pick!

Your graphics program should begin with a statement such as


```
5 GRAPHICS 19
```

to tell the computer that you will be using all of the TV screen to make a picture. This will draw pictures on the screen using blocks like this:



You can also draw on the screen using small blocks, like this:





To draw with small blocks you must start with a different graphics number. The higher the number, the smaller the blocks. To draw pictures on the whole screen you can use these numbers after the word GRAPHICS:

19, 20, 21, 22, 23, 24

We will use GRAPHICS 19 to draw pictures on the TV screen.

If you want to write a message on the screen, below your picture, you must use different graphics numbers. Here are the numbers you can use: 3, 4, 5, 6, 7, 8. The higher the number, the smaller the blocks are. These let you "PRINT" in the bottom four lines of the screen.

EXAMPLE: 5 GRAPHICS 4

10 PRINT "HI THERE. "

Before you tell the computer which blocks you want it to light up you need to tell the computer more information about how you want that block to look.

To “paint” on the screen, you use brushes dipped into buckets of paint.

There are four buckets you can use in Graphics 19. They are numbered 0, 1, 2, 4. You must first choose which bucket to use.

Second, you must choose which hue you will put into that bucket. There are 16 choices for you which are listed here with their numbers 0-15.

0 GRAY	8 LIGHT BLUE
1 LIGHT GREEN	9 BLUE
2 GREEN ORANGE	10 BLUE GREEN
3 RED ORANGE	11 CHARTREUSE GREEN
4 RED	12 YELLOW GREEN
5 PINK	13 ORANGE GREEN
6 LAVENDER	14 RED ORANGE
7 PURPLE	15 RED

These hues will vary from one TV set to another and from one brightness level to another. Hues 2 and 13, for example, are listed here as Green Orange and Orange Green, but are yellow at the 0 brightness level. Keep this in mind when you write down your own color graphic programs.

Third, you must decide how bright you want the hue to appear on the screen. Brightness ranges from 0 (very dim) to 14 (very bright) but only the even numbers can be used (0, 2, 4, 6, 8, 10, 12, 14). 8 is a medium bright.

The SETCOLOR command is used in a statement to tell the computer your three choices of bucket, hue, and brightness. Here is an example:

```
10 GR. 19  
15 SETCOLOR 1, 9, 8
```

The number 1 tells the computer to use Bucket 1. The number 9 tells the computer to put Blue into bucket 1, and the number 8 tells the computer to make the blue a medium bright.

15 SETCOLOR 1, 9, 8

  
BUCKET HUE BRIGHTNESS

Now that you have the buckets of paint ready you can dip your brushes in the buckets and paint on the screen. This is done with the COLOR command. Type COLOR and the number of the brush you want the computer to use.

Each numbered bucket will only hold a certain numbered brush.  
For Graphics 19 here are the numbered Buckets and Brushes.

### In Graphics 19

USE



WITH



If you are using Bucket 0 you would need to use Brush 1 so you would choose COLOR 1. If you were using Bucket 4 you'd need Brush 0, so you would type COLOR 0.

Try this example:

```
5 GRAPHICS 19
```

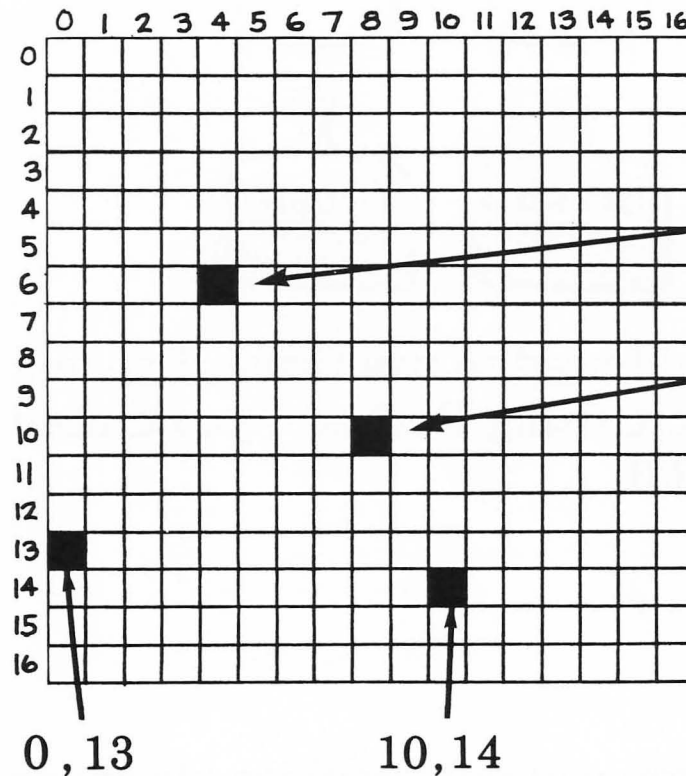
```
10 SETCOLOR 0,9,8
```

```
15 COLOR 1
```

5 Graphics 19 told the computer you will put graphics blocks on the screen. 10SETCOLOR 0, 9, 8 told the computer each block will be from bucket 0(0), a blue hue (9), and of medium brightness (8). 15 COLOR 1 then told the computer to put Brush 1 into Bucket 0.

Now you are ready to paint on the screen.

The GRAPHICS 19 screen is divided up into 40 squares across and 24 squares down. (Part of the screen is shown here.)



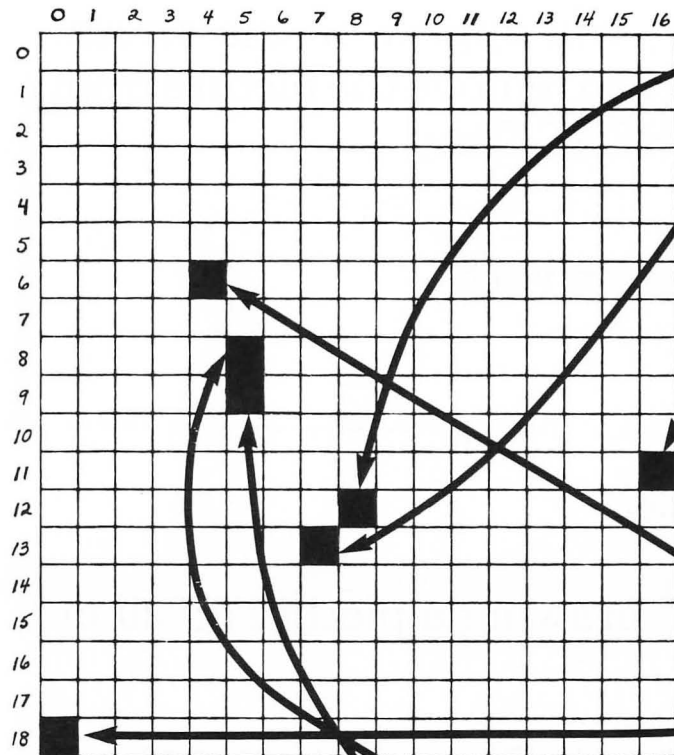
Each square has an address that tells how many squares **over** and how many squares **down** it is.

4, 6 (Four squares over, six squares down.)

8, 10 (eight squares over, six squares down.)

In the address, the numbers always appear in the same order:  
squares **over**, squares **down**

To light up a square, you use **PLOT**.



5 GR. 19

6 SETCOLOR 2,12,10

10 COLOR 3

15 PLOT 8,12

20 PLOT 7,13

25 PLOT 16,11

Let's put some more PLOT statements in this program, but first let's change the hue:

30 SETCOLOR 1,4,10

35 COLOR 2

Now, anything we PLOT after line 35 with COLOR 2 will be pink.

38 PLOT 4,6

40 PLOT 0,18

45 GO TO 45

To plot more green blocks you have only to use COLOR 3 again.


45 COLOR 3

50 PLOT 5,8

60 PLOT 5,9

65 GO TO 65





If you're doing graphics and you want the picture to stay on the screen, you have to use this as the last line in your program. If you don't the computer will be done with your program and will go back to its "ready" state. The cursor will appear at the top left hand corner and the word "READY" will appear on the screen. Line 65 is an *infinite loop*. The only way to remove the picture from the screen is to press the **BREAK** or **RESET** key.

Every time you begin a graphics program with

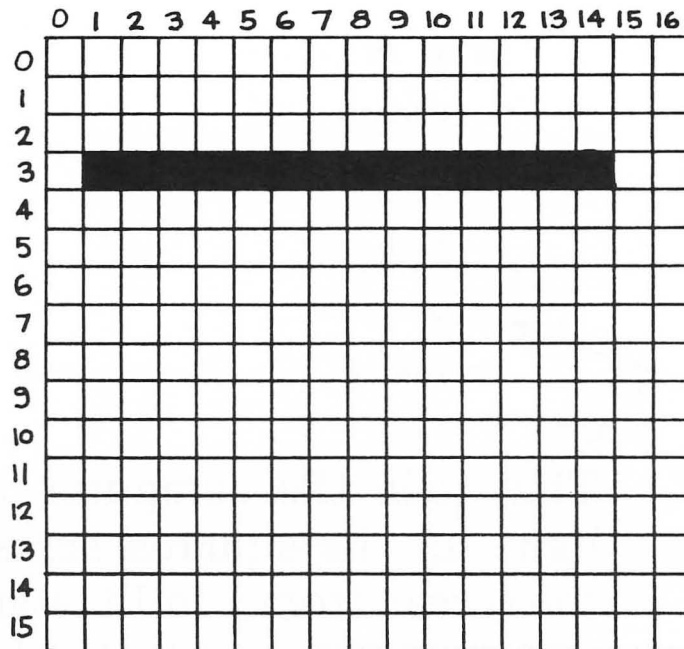
5 GR. 19

you are really doing *two* things at once:

1. Telling the computer you will be drawing a picture on all of the screen.
2. Setting the background color to black. This means that if you forget to put a **COLOR** statement in your program, every square you plot will be black. Since the background color is also black, you won't be able to see what you are drawing!



Be careful you don't forget to choose a hue for your graphics programs!



For a program that would fill most of the screen, you would be up all night typing in PLOT statements!

If you want to light up a line across the screen, you would have to use many PLOT statements.

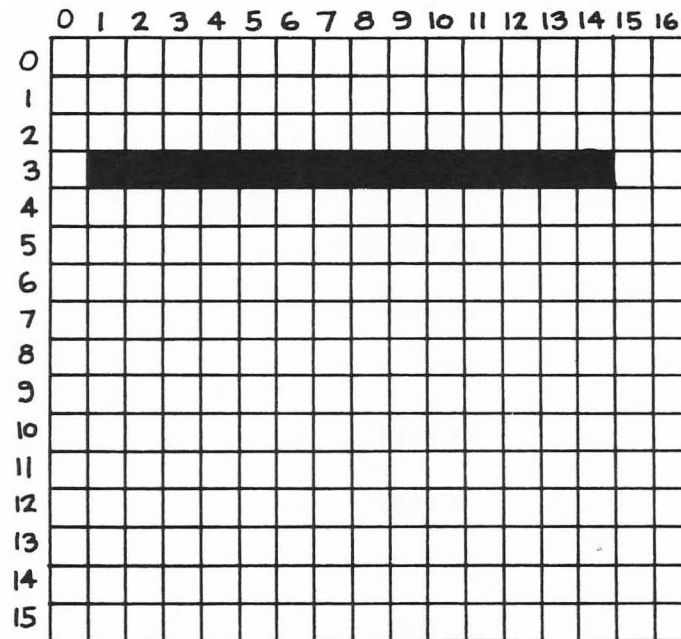
```
5 GR. 19  
6 SETCOLOR 2,13,8  
10 COLOR 3  
15 PLOT 1, 3  
20 PLOT 2, 3  
25 PLOT 3, 3  
30 PLOT 4, 3  
35 PLOT 5, 3  
40 PLOT 6, 3  
45 PLOT 7, 3  
50 PLOT 8, 3
```

•  
•  
•

and so on.

BUT—there is an easier way to do the same thing.

DRAW TO will help us write the same program much more easily:



5 GR. 19

6 SETCOLOR 2,3,8

10 COLOR 3

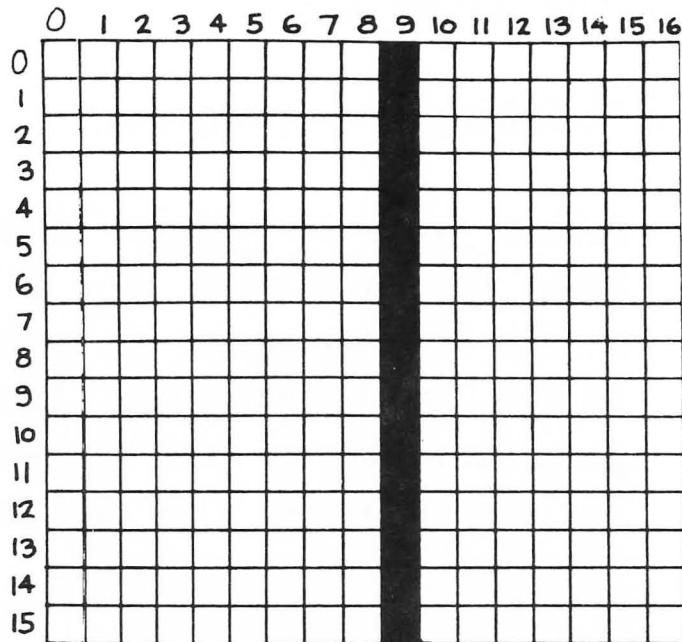
15 PLOT 1,3

20 DRAWTO 14, 3

25 GOTO 25

This tells the computer to light up the squares across the screen from 1 to 14 (start at 1 and end at 14). It also says the line should be 3 squares **down** from the top of the screen.

To light up a line going up and down the screen, we use  
PLOT X,Y: DRAWTO X1,Y1



5 GR.19

6 SETCOLOR 2,3,8

10 COLOR 3

15 PLOT 9,0

20 DRAWTO 9,15

25 GOTO 25

This lights up the squares up and down the screen, beginning at 0 and ending at 15. This line will be 9 squares across the screen.

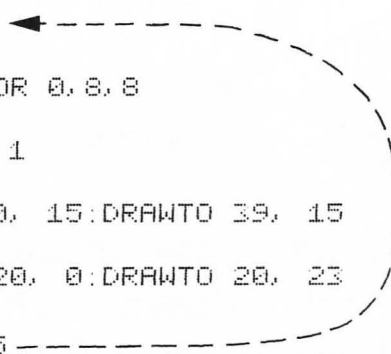
PLOT X,Y and DRAW TO X1, Y1 make graphics programs very easy! By using PLOT and DRAW TO, and lots of hue changes, you can make a picture of anything you like!

Just for fun, you might want to make your picture **blink** off and on. You will need a GOTO statement in your program to do this.

Remember that when the computer comes to a GRAPHICS statement as it is executing your program, it makes the screen black. Your picture has disappeared, because you have changed the GRAPHICS mode, which makes the screen black.

Let's use this information to make a "blinking line" program.

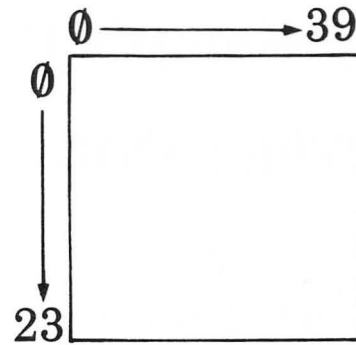
```
5 GR. 19 ← -----  
6 SETCOLOR 0,8,8  
10 COLOR 1  
15 PLOT 0, 15:DRAWTO 39, 15  
20 PLOT 20, 0:DRAWTO 20, 23  
25 GOTO 5 -----
```



After your two light blue lines are displayed, line 25 makes the computer go back to line 5 and start again. The program starts and makes the screen black. This makes the lines blink on and off!

For your first graphics program, you might want to make your initials on the screen. You could make each initial a different color, or you could make them blink on and off.

When you plan your graphics programs, you will find it very helpful to use graph paper. Remember that there are 40 squares across and 24 squares down on the graphics screen in GRAPHICS 19. They are numbered from *zero* to 39, and *zero* to 23. If you try to use



addresses with numbers bigger than 39 or 23, you will get an error message when you run the program:

ERROR-141 at Line 10  
(Cursor out of Range)

When you write a long graphics program, you will need more than those four lines under the graphics screen to LIST your program and look it over before you RUN it.

If you switch back to writing on the screen, hit RESET *before* you type LIST. Then the computer can list the program on the whole screen. You might forget to do this the first few times, but once you get used to it, writing graphics programs will be easier to do.

## SECTION 11: Sample Programs

### COMPUTER PANIC

```
5 GR. 0  
15 PRINT "HELP! THIS COMPUTER IS CRAZY!"  
20 GOTO 5
```

### GUESSING FUN

```
5 GR. 0  
15 PRINT "PICK A NUMBER. TYPE IT IN. "  
20 PRINT "THE CHOICES ARE 1, 2, OR 3. "  
25 INPUT N  
30 IF N = 1 THEN PRINT "YOU WILL BE RICH. "  
35 IF N = 2 THEN PRINT "YOU WILL BE FAMOUS. "  
40 IF N = 3 THEN PRINT "YOU WILL HAVE 13 CHILDREN. "  
45 END
```

## PINE TREE

```
5 GR. 0
10 PRINT "      X"
15 PRINT "     XXX"
20 PRINT "    XXXXX"
25 PRINT "   XXXXXXX"
30 PRINT "  XXXXXXXXX"
35 PRINT " XXXXXXXXXXX"
40 PRINT "      X"
45 PRINT "      X"
END
```

make sure you skip enough spaces in each line!

## COLORING THE SCREEN

```
5 GR. 19  SETCOLOR 2, 3, 8
10 COLOR 3
15 FOR X = 0 TO 39
20 FOR Y = 0 TO 23
25 PLOT X, Y
30 NEXT Y
35 NEXT X
40 GOTO 40
```

## ARITHMETIC

```
5 GR. 0
10 LET A = 1
20 LET B = 2
25 LET C = 3
30 LET D = 4
35 PRINT "A + B ="
40 PRINT A + B
45 PRINT
50 PRINT "C * D ="
55 PRINT C * D
60 END
```

## SECTION 12: Glossary

**CLEAR**—Press **[SHIFT]** and **[CLEAR]** to clear the screen of text. In a Graphics Program, add a new graphics statement to clear the screen.

Example: 40 GRAPHICS 19 will clear the screen

**CLOAD**—loads a program from a cassette tape into the computer's memory.

**COLOR A**—Tells the computer which Brush you are going to use during a GRAPHICS program. A can be number 0, 1, 2, and 3 in GRAPHICS 19. This example lets you paint with Brush 1.

10COLOR 1

(This is used with SET COLOR A, B, C)

**CONT**—Continues the execution of a program after you have stopped it by using **[BREAK]**.

**CSAVE**—Records your program on cassette tape. (This does **not** erase your program from the memory. Only NEW, or turning off the keyboard, will do that.)



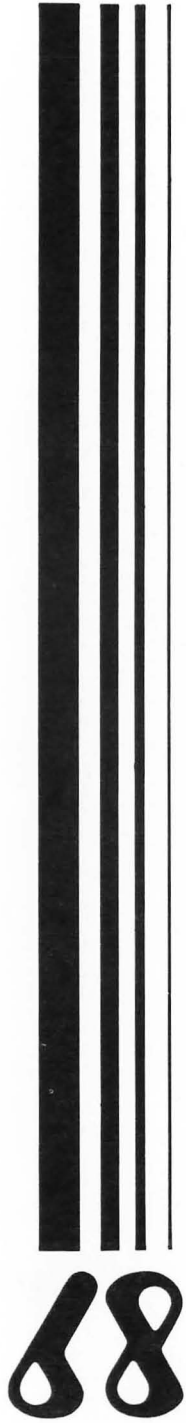
TO DELETE—Typing in DOS, `RETURN`, D `RETURN`, “D:FUNNY”, Y `RETURN` —erases the program named FUNNY from the diskette. For example, to erase “FUNNY” from the diskette type:

DOS `RETURN` —which allows you to communicate with the disk drive. The computer prints a list of commands on the screen. This is a MENU.

D `RETURN` —This command will allow you to delete the program from the diskette. The computer will then ask you which program you want to delete to which you reply:

D:FUNNY `RETURN` —Which is the program you want to delete. The computer will then say “Type “Y” to delete”, which you do.

After you type Y and `RETURN` —the busy light will come on and the drive will whirr. When it stops, hit `RESET` —and you will be back in BASIC.



DOS—allows you to communicate with the disk drive. When you type DOS the Disk Operating System will print out a list of commands it can perform. This list is a MENU. DOS is used to DELETE programs (see DELETE) or to get a list of programs on the diskette. To do this, type the letter next to the command you want and then hit `RETURN` .

For example, to get a list of programs on a diskette, type:

```
DOS RETURN
A   RETURN
RETURN
```

DRAW TO X1, Y1—used in graphics programs to light up a line on the screen. This example lights up a blue horizontal line of medium brightness across the screen from 0 to 23, at 15 squares down from the top of the screen.

```
10 GRAPHICS 19: SET COLOR 2, 6, 8
20 COLOR 3
30 PLOT 0, 15: DRAW TO 23, 15
40 GOTO 40
```

END—tells the computer the program is over.

FOR NEXT—a type of do-loop which has the computer perform some action a certain number of times.

```
Example: 10 FOR X = 1 TO 100
          15 PRINT "HELLO"
          20 NEXT X
```

GOTO—tells the computer to skip to a certain line number in the program.

GRAPHICS A—tells the computer whether you will be drawing pictures, writing, or both. A is a number from 0 to 24.

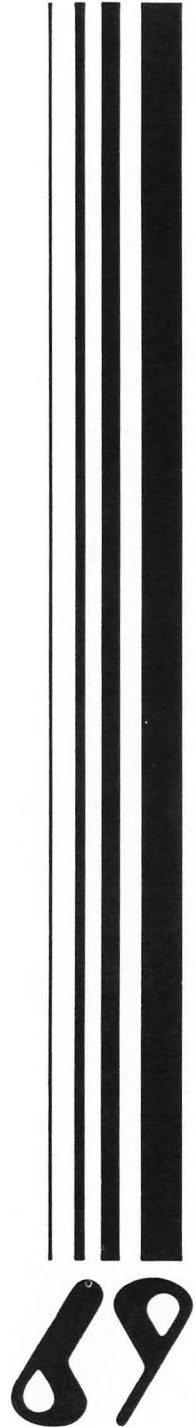
Just Pictures—19, 20, 21, 22, 23, 24

Writing—letters or numbers—0

Pictures and 4 lines of text at the bottom—3, 4, 5, 6, 7, 8

Writing—large letters on the screen—17, 18

Writing—large letters on the screen and 4 lines of text at the bottom—1, 2



**IF-THEN**—a type of branch statement which puts a “test” in the program. If the test is passed, the computer must follow special directions. If the test is not passed, the computer will drop down to the next line in the program.

**INPUT**—types out a question mark when the program is executed, and waits for an answer to be typed in. The answer is stored in a certain memory space.

Example: 15 PRINT “TYPE IN YOUR AGE.”  
20 INPUT N

In this example, the answer is stored in mailbox N.

**LET**—assigns a number to a memory space (or variable)

Example: 15 LET R = 96

**LIST**—Prints out a list, in order, of the program statements you have typed into the memory.

**LOAD “D: SURPRISE”**—loads a program (this one is named SURPRISE) from the diskette into the computer’s memory.

NEW—erases the old program from the memory.

PLOT X, Y—used in a graphics program to light up a point on the screen.

PRINT—tells the computer you want it to write something on the screen.

RESET—immediately stops the execution of your program, but does not erase it from the memory. After you push RESET, you will have to start all over if you want to RUN your program again. Use RESET if you just don't know what else to do, but you don't want to erase your program by turning off the keyboard. (You will use RESET if you goof while loading or saving a cassette tape.)

RETURN—you must press this key each time you finish typing in a line.

RUN—starts the execution of the program. This command is *not* part of the program itself, and does not have a line number.

SET COLOR A, B, C—This command is used to change the color in the “BUCKETS” of paint. These are used with the “BRUSHES” (COLOR statements) to paint on the screen.

A = BUCKET number (0, 1, 2, 4)

B = HUE number (0-15 see Color Chart)

C = BRIGHTNESS (0-15)

In GRAPHICS 19 BUCKET 4 uses BRUSH (Color) 0  
BUCKET 0 uses BRUSH (Color) 1  
BUCKET 1 uses BRUSH (Color) 2  
BUCKET 2 uses BRUSH (Color) 3

The following example sets BUCKET 0 to Blue with medium brightness and puts BRUSH (Color) 1 into it.

```
5 GRAPHICS 19
10 SET COLOR 0, 6, 8
15 COLOR 1
```

NAME \_\_\_\_\_

Simulate these computer runs. Show  
your "printout" on the screen.

```
10 GR. 0
```

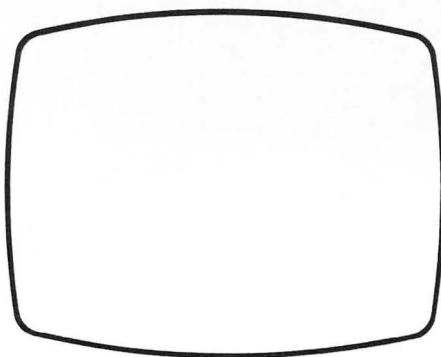
```
20 PRINT "BIG"
```

```
30 PRINT
```

```
40 PRINT "YELLOW"
```

```
50 PRINT "BLUE"
```

```
60 END
```



```
10 GR. 0
```

```
20 PRINT "THE ANSWER"
```

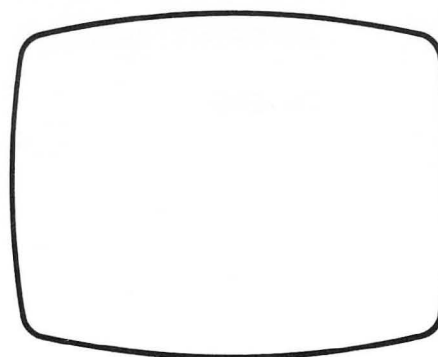
```
30 PRINT 30 * 2
```

```
40 PRINT 30 + 2
```

```
50 PRINT 30 - 2
```

```
60 PRINT "THE END"
```

```
70 END
```



NAME \_\_\_\_\_

Here is a program and "printout." Find and fix the mistakes in the program so a run will produce what is shown on the screen.

```
10 GR. 0
20 PRINT 20 + 6
30 PRINT 30 + 4
35 PRINT
40 PRINT "60 - 3"
50 PRINT 10 - 10
60 PRINT "HELLO"
70 END
```

26

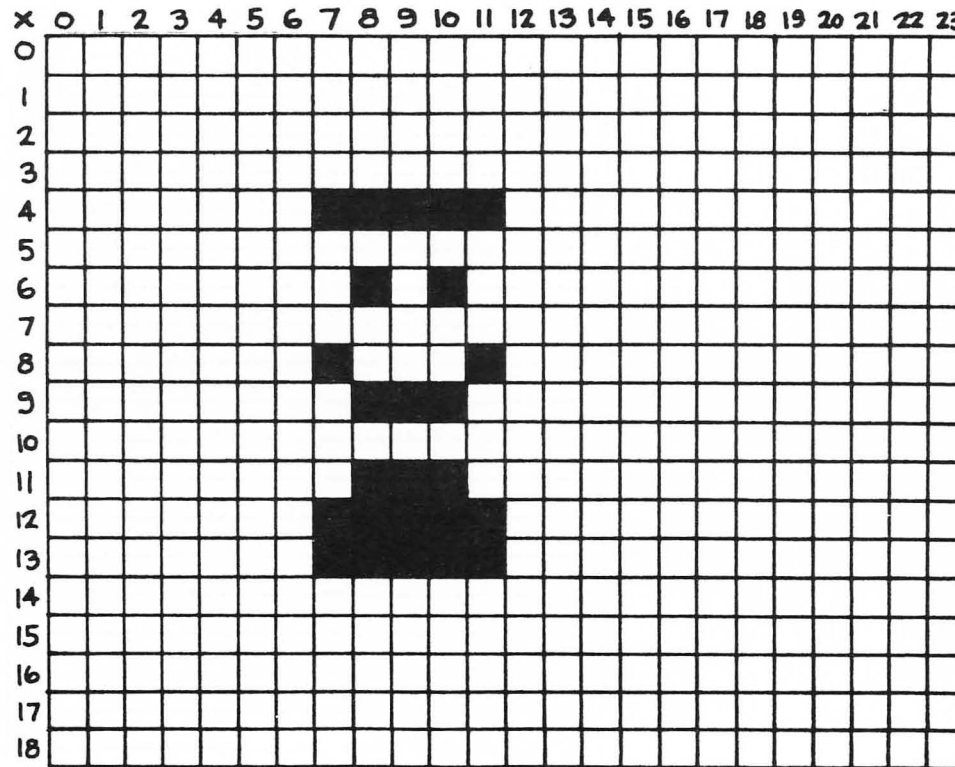
24

57



NAME \_\_\_\_\_

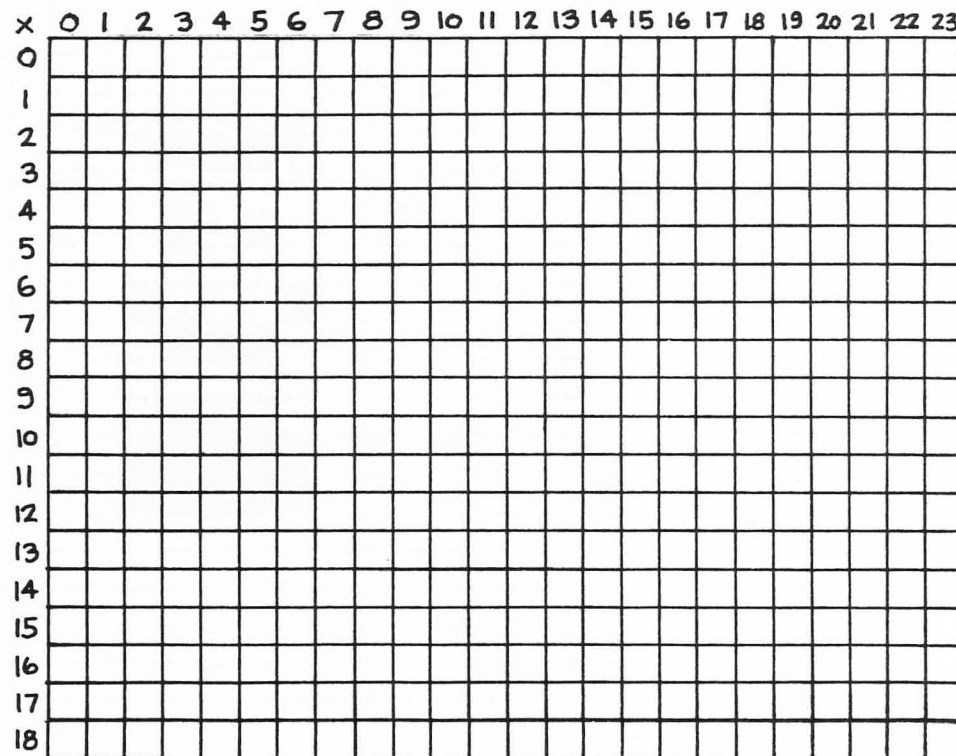
Write the program to print out the pattern shown below. Make the hair yellow, the eyes blue, the mouth red, and the shirt green.



NAME \_\_\_\_\_

On the graph below, color in each dot which will light up when you run this program. Use crayons or colored pencils to show the correct color.

```
5 GR. 19
10 SETCOLOR 0, 6, 8
15 COLOR 1
20 PLOT 5, 5
25 PLOT 6, 5
30 SETCOLOR 1, 3, 8
35 COLOR 2
40 FOR P= 1 TO 10
45 PLOT 3, P
50 NEXT P
55 SETCOLOR 2, 15, 8
60 COLOR 3
65 PLOT 18, 18
70 PLOT 0, 0
75 PLOT 15, 0
80 PLOT 0, 15
90 GO TO 90
```



# Notes for Teachers and Parents

I am not, by any stretch of the imagination, a sophisticated programmer. I took one programming course in college, and have spent the past four years working with elementary school children. I've taught microcomputer programming to nearly 300 children, ranging in age from 4 to 12, and have yet to run into any children who aren't dying to get their hands on the keyboard. Computers are a *natural*—they give immediate feedback, and allow children to create something all their own. I love teaching programming. It is one of the most exciting things I've ever done.

Candidates for teaching programming to young children must have one trait above all others—the ability to interact with the children on a peer level, and learn along *with* them. Computer programming is not just a skill—it is a *tool*. You learn programming not as a study in itself, but because of what you can accomplish with it. In any one computer program, there are many ways to approach and solve the problem at hand. The thing I say most often to my students is, “Run it, and see if it works!”

I won't presume to tell you just how to go about teaching your particular group of children, but I would like to share some of my successful ideas, and some of my failures. These are the things which have worked, or not worked, with every group of children I've taught in the past few years.

One word from someone who's been there: book some computer time for yourself during each week, before you start teaching the children. Once they've had a few lessons, they'll insist you stand in line like everyone else!

## General Hints

If you have had no previous experience with the ATARI, before you do anything else, read through the manuals which come with the machine. (At the time this book went to press, several manuals, some better than others, were available for the novice programmer.) It is especially important to understand the directions for setting up the machine, and the safety precautions. To gain an overview of this book, you may wish to read through the children's portion before you work through the ATARI manuals. It will not be necessary to learn everything in the manuals—check the glossary of this book for the most crucial statements and commands.

This book is designed to accompany the ATARI machine. Because of the relatively large number of directions concerned with operating this particular model of the ATARI computer line, it would be possible, but impractical, to use this text with another

computer. Such use would require considerable adaptation on your part before a child could use the text effectively.

Refer to your manuals for complete instructions on maintaining your computer, but these seem to be the most common pitfalls:

1. If you move the computer around from place to place, it is necessary to open the cover and check to see that the cartridges are firmly seated in the slots *before* you turn on the computer.
2. The cords and wires coming from your computer must be protected from being accidentally jerked out. Make sure you have located the computer in a place where the cords are protected from falling objects, feet, and contending programmers.
3. Have a clear understanding as to which people have the responsibility for opening the cover and working with the hardware inside the computer. (Too many cooks make sauce out of the ATARI.)

**SETTING UP YOUR COMPUTER CENTER:** Since programmers tend to get excited and vocal, I suggest that you locate your computer in a semi-secluded area which is near someone in your school who understands the machine. If the children have any problems with the computer, they are going to come and find you anyway, and it's easier if they don't have to call you down the hall from another room.

Secure the electrical cords in a way that keeps them out of the traffic pattern around the computer. Step-

ping on the cords may cause a fire hazard and will create fuzz on the TV screen.

I schedule only two children at the computer during one time slot. They usually help each other if they encounter difficulties. More than two children at one time may encourage fighting over who will do the typing.

Have enough room for several chairs around the keyboard, and consider where you will place the computer when you teach a group. Sometimes I have used a kitchen timer to keep the children moving, and other times I have run the schedule by the clock. It depends on your group. You will keep your sanity longer if you enforce the rule: "When your turn is over, it's OVER."

A computer notebook for each child is a must. They should learn to take notes on how to do things, or they will never become confident programmers. Discourage them from running to you for answers they should have in their notes.

You may wonder why I have so few sample programs in this book. I have found that the more timid programmers will never pull away from the safety of typing in *my* programs every time they are on the machine, unless I provide very few samples, and force them to think up their own.

It sounds, from the tone of these hints, that I have many problems with children who program. That isn't the case at all. However, a seemingly trivial problem can eat up precious time when 50 children are waiting to use one computer.

The most important thing you as the teacher must do is to give the children an *overall* view of the problem you are trying to program. They must see that a problem can be broken down into sections, and then each section can be accomplished on the computer in several different ways. Teaching only *what* a statement does, without focusing on *why* you would want to use it, creates frustrations for most children.

MY BIGGEST FAILURE OF ALL TIME ... I can't emphasize this one enough. DON'T EVER LET YOUR CHILDREN PLAY COMMERCIAL GAME TAPES UNTIL THEY ARE ACCOMPLISHED PROGRAMMERS! By 'accomplished,' I mean the end of the first year for most children.

Let's face it—playing 'Breakout' or 'Computer Hockey' is much more fun, and a lot less work, than learning to program. Especially for elementary school children. If they discover that they can play game tapes on the school computer, they will lose all interest in doing the work involved in learning to program. This is a sad but true fact, and one I learned the *hard* way. Even with your 12-year-olds, you will regret the day you ever brought a game tape into your computer center. Overnight, they will change from being thrilled about having the chance to try their own programs, to being disappointed because their favorite game tape has been retired. Certainly, they'll have a chance to play games on the computer. But the games should be those that *they* have written themselves.

GROUP INSTRUCTION: Choose for your computer center a room which has effective shades on the windows. When I teach a group, I place the TV facing

them, and I sit at an angle to the keyboard. We talk over what we want to accomplish, and I do the typing. Unless you have a crackerjack typist in the group, it makes lessons unbearably slow if the whole class has to wait while someone hunts and pecks on the keys.

## Suggested Lesson Outline — Once a Week Lessons

Do not go on to the next topic until all the children have had a chance to try out their last lesson on the computer, or they will never remember it. A weekly schedule is imperative for assuring individual children their time at the computer. With the exception of SAVE and LOAD, these lessons follow the sequence of the book.

### SECTION 1

1. What is a computer?

### SECTION 2

2. Introduce flowcharting
3. Practice writing flowcharts
4. More practice on flowcharts

### SECTION 3

5. Using the keyboard, running the machine itself, behavior guidelines, scheduling, RETURN

### SECTION 6

6. Beginning programming:  
CLEAR, GRAPHICS 19, TEXT, CTRL-S,

CTRL-C, CONT, NEW, LIST, RUN with  
PRINT examples

#### SECTION 7

7. PRINT statements with quotation marks, ERROR, TEXT and HOME in programs
8. PRINT to skip lines; editing
9. PRINT with arithmetic operations (+, -, \*, /)
10. PRINT variables—simple
11. PRINT variables such as PRINT A + B

#### SECTION 4 or 5

12. Operation of disk drive or cassette recorder

#### SECTION 8

13. GOTO
14. INPUT

#### SECTION 9

15. FOR-NEXT with PRINT and arithmetic statements
16. More work on FOR-NEXT
17. IF-THEN test
18. IF-THEN in more complex programs (draw flowchart of program function)

#### SECTION 10

19. Discussion of PLOT, SET COLOR, COLOR, and plotting of coordinates on paper
20. Graphics worksheets
21. Graphics—make their own initials with PLOT: DRAW TO

22. Graphics—Horizontal and vertical lines
23. Blinking graphics; more complex programs, such as filling the screen with a color

#### SECTION 11

24. Discuss possibilities for designing and carrying out their own programs

## Teaching Suggestions For Each Section

#### Section 1: What is a computer?

Comparisons to home computer games are helpful. Most children think computers are 'smart', and younger children will think they are 'magic'. Don't overexplain—make arrangements to get the children at the machine as soon as feasible. None of your explanations will really make sense until then. You might want to write a simple INPUT program they can use for some 'hands-on' practice.

#### Section 2: Flowcharting

The objective is logical thinking, not perfection. Have fun! Choose 'How To' type topics, which have built-in choices. They must be about something the children understand.

How to: Give the dog a bath; Make pizza; Build a doghouse; Lose your allowance; Make a phone call to Grandma.



Set a minimum number of do-loops or branches. I usually set a minimum of three. Use manilla drawing paper. Have the children write the words first, *then* draw the boxes around the words. It's easier!

### Section 3: Running the machine itself

Be careful to use the words 'save' and 'load' properly, when you talk about the disk drive or cassette recorder, or the children will confuse the two terms. Typing practice on school typewriters or those at home will help the children accomplish much more during their turn at the computer. Children who practice on manual typewriters tend to *pound* on the computer keys. This will cause typing errors and other keyboard problems.

### Section 4: Saving your programs with a cassette recorder

The cassette player system for saving programs is a headache for many people, but it is my opinion that they have trouble because they do one of these things incorrectly:

- a. They use poor quality recording tape, or try to save programs over old recordings.
- b. They don't leave enough space on the tape between programs.
- c. They don't teach the children how to use the counter properly.

- d. They don't save one program twice on the same tape, to insure that at least one of them will turn out.

### Section 5: Saving your programs with a disk drive

You should record on one side of the diskette only. When you put the diskette into the disk drive, the label goes *up*.

The disk drive has to be turned on. There should be a disk in it.

Static electricity will ruin the disk drive. Stress to the children that they must ground-themselves-out each time they touch the disk drive, especially if you have carpeting in the school, or if the diskette is cold.

When moving the disk drive in its packing box, tape the door shut. If you catch the door on the packing material, it will snap off.

### Section 6: Getting ready to program

Written quizzes on the meaning of the commands and statements accomplish very little. Let the children learn about this while they type in their own programs. If they don't know what they're doing, their program simply won't run.

Teach the difference between *commands* and *statements*. If they do not understand this difference, the children will be frustrated the first few times

they work at the computer. Typically, they will type in a program without any line numbers, and then will not be able to understand why none of the program is in the memory to be listed later.

#### Section 7: PRINT and variables

At the back of the book, you will find samples of PRINT worksheets. This is one of the few areas in which worksheets are of real value.

Discourage the children from using the letter 'O' as a variable. It tends to be confusing.

I use only variables of one letter with beginning programmers. They may get confused, using word-length variables which look to them like statements.

Using Graphics 0 at the beginning of each program, even though it may seem rather cumbersome at first, will prevent a great many problems that can arise when many people use the same computer.

#### Section 8: GOTO and INPUT

Every PRINT line moves the cursor down to the next line. This problem will come up when you

teach FOR-NEXT loops with several PRINT statements in the middle.

Look up the use of the semi-colon in the ATARI manuals. Sooner or later, someone will ask you how to make things PRINT right next to each other on the same line when using several PRINT statements.

#### Section 9: IF-THEN and FOR-NEXT

Reminder: variables must match on FOR-NEXT loops.

The children will have trouble remembering that the computer drops down to the next line if the test is not met in an IF-THEN statement.

Flowcharts may be helpful in tracing the functions of loops and branches in more complex programs. Don't be overly concerned with detail when you draw them.

'X is not equal to Y' is written as  
 $X < > Y$  or  $X > < Y$ .

#### Section 10: Graphics programs

The 'over and down' motion of plotting a graphics point on the coordinates is similar to drawing a large '7' on the screen.



I have included sample worksheets for practicing coordinates in the back of this book. This is very helpful, especially for the younger students.

When the children number the squares of their graph paper, have them put an 'X' in the upper left-hand corner square. This helps prevent mistakes in numbering.

X	0	1	2	3	4	5	6
0							
1							
2							
3							

Tracing the 'Numbers' program through its memory changes is helpful for showing how the values of variables change in a program. (This program may be found at the end of Section 9.)

Use crayons or markers to plan graphics pictures in the actual colors you will use.

### Section 11: Sample programs

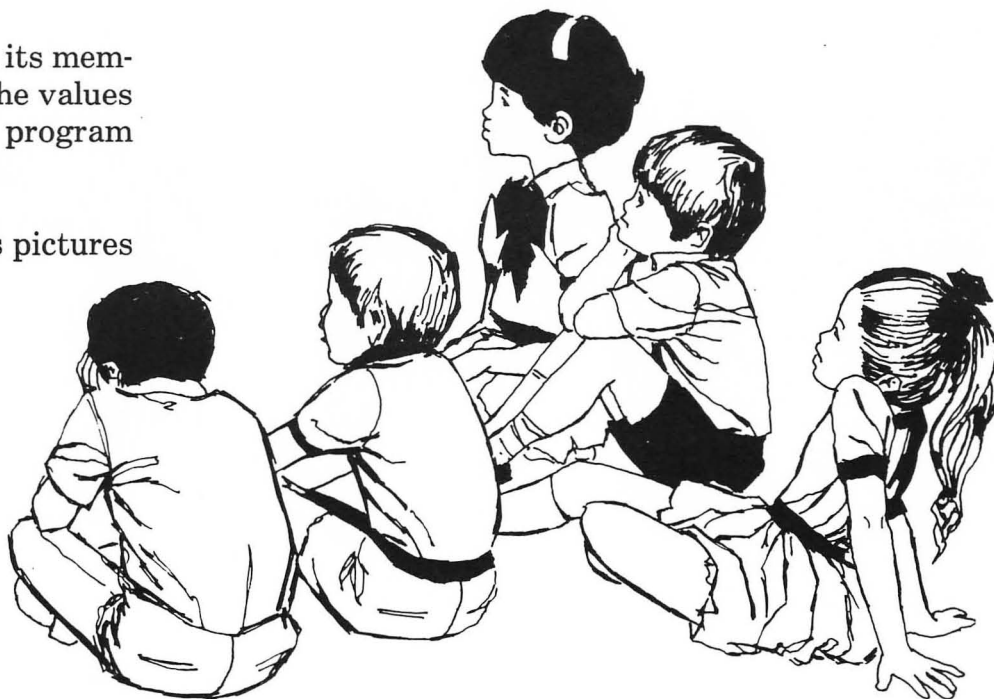
As stated earlier, with the more timid programmers, I find that too many examples inhibit experimentation.

If you are using a cassette recorder, all the children should have their own cassette for saving their programs. Buying some gummed labels is a good investment, so that they will not have trouble trying to locate those programs at a later date.

Go through your diskettes periodically and delete those programs which have been updated, or are no longer being used.

### Section 12: Glossary

For those examples in which I used statements, as opposed to commands, I put line numbers in front of each program line, to help remind the children.







**For the child who is eager to enter  
the exciting world of computers  
and  
For the parent or teacher who is  
eager to help them learn**

**Computers for Kids is here!!**

The Computers for Kids series is designed for children ages 8-13 who are interested in computers but are overwhelmed by the reading level and quantity of material covered in adult level manuals. (Computers for Kids can be considered a children's level manual.)

This entertaining, easy to read book includes sections on:

- How computers work
- How to operate the Atari
- Simplified flowcharting
- Step by step discussion of BASIC
- Statements and how they are used to write programs
- Color graphics
- A child's glossary of BASIC terms

Sally Greenwood Larsen has taught Kindergarten through Seventh grade and is a pioneer in teaching young children to program microcomputers.

She has also written editions in this series for the Apple II Plus, TRS-80, Timex-Sinclair, and Commodore VIC-20 computers.

There is a special section for parents and teachers with teaching ideas, troubleshooting hints, lesson plans.



 **CREATIVE  
COMPUTING  
PRESS**