COMPUTERS FOR KIDS
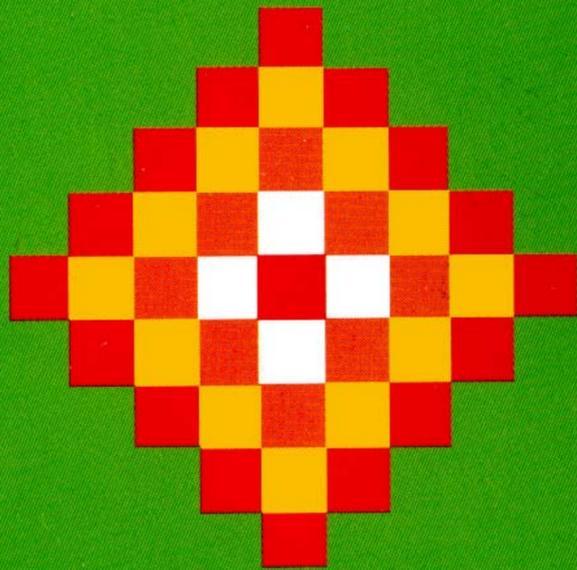
TRS-80 Edition

Sally Greenwood Larsen

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

COMPUTERS FOR KIDS

TRS-80 EDITION

COMPUTERS FOR KIDS

TRS-80 EDITION

SALLY GREENWOOD LARSEN

TABLE OF CONTENTS

SECTION 1: WHAT IS A COMPUTER?

When cavemen and women had work to do, they had no machines or tools to help them. They had to do it all by themselves. Men and women have since invented many tools to help them with their work.

Instead of pounding with our hands, we now use a hammer. The hammer lets us pound harder and longer then we could with our hands alone.

The telescope was invented so that we could see farther into space. We can now see stars we did not know existed before we had the telescope to help our eyes.

Using our brains, we can remember information and solve problems. But there was a need for a tool that would extend the use of our brains, so the COMPUTER was invented.

Just as a hammer cannot do work without a person to hold it, a computer will not 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 on a sheet of paper.

This book is about the TRS-80 computer,
made by Radio Shack. These are special directions for <u>this</u> computer.
They will not work on all other kinds of computers.

The TRS-80 is called a MICRO-COMPUTER, because it is so small.
Many businesses and universities have computers, too,
but theirs have to do many more jobs than our TRS-80,
so they have to be much larger.
Some of the biggest computers are so huge, they fill an entire <u>room</u>!

The TRS-80 speaks a special computer language called 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 page 5 is a flowchart
showing all the little steps in a funny problem.
The directions on this flowchart
are things for you to do.
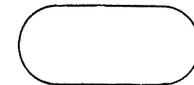They are not directions for the computer!

The shapes drawn around
the steps show
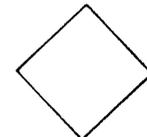what <u>kind</u> of a step it is:

Rectangle

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

Oval

for **START** or **END**.

Diamond

for **YES** or **NO**
questions.

How to Scare Your Mom with an Elephant

START

Do you have an elephant? — **Yes** → Good job!

Good job! — **Yes** → Was she scared?

Was she scared? — **Yes** → Good job! → END

Was she scared? — **No** → Give up! → END

Put it in Mom's bed. → Was she scared?

Bring it home. → Put it in Mom's bed.

Do you have an elephant? — **No** → Is there an elephant store on your block?

Is there an elephant store on your block? — **Yes** → Go in.

Is there an elephant store on your block? — **No** → Go to another block

Go in. → Is there a good elephant in it?

Is there a good elephant in it? — **Yes** → Take it.

Is there a good elephant in it? — **No** → Go to another block

Take it. → Did you pay?

Did you pay? — **Yes** → Bring it home.

Did you pay? — **No** → Go back and pay for it. → Bring it home.

By Stephanie Webb
Age 8

The arrows on a flowchart show you what to do next. One arrow shows you 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.

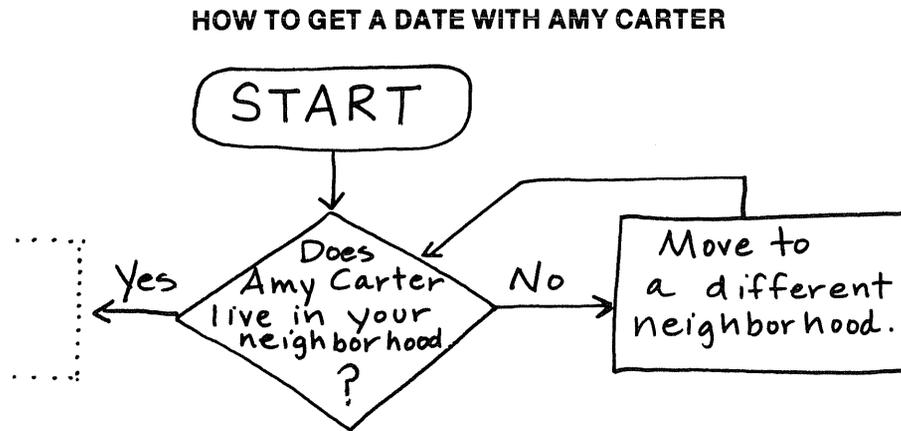You want both questions and statements in your flowchart—
not just a page full of one statement after another.

Here are some suggestions:

1. How to take a bath.   2. How to make your mother scream.

3. How to play kickball.   4. How to buy a birthday present.

5. How to make a peanut butter and jelly sandwich.

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 drawn only <u>part</u> of this flowchart).

HOW TO GET A DATE WITH AMY CARTER

If you follow the directions
for this part of the flowchart,
you will keep moving to a new
neighborhood until you are living
in Amy Carter'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 this example, in order to "get a date with Amy Carter,"
you had to first move into her neighborhood.

We will learn more about DO-LOOPS in Section 6.
In the next sections, you will see how we use flowcharts
to help us write our own computer programs.

8

SECTION 3: RUNNING THE MACHINE ITSELF

The TRS-80 machine has four basic parts:

The Keyboard — This looks like a regular typewriter keyboard.
It has all the electronics for the computer inside it.

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. This is not a regular TV
like you have at your house. It has no channels
and no sound. You can't watch TV programs on it.

The Power Pack—A special power supply for the keyboard
is provided by this little black box.

The Tape Recorder — This is for saving your programs on cassette tapes.

The first time you use the TRS-80,
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 it is done correctly, or you will ruin it.

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 keyboard and the TV. (If this doesn't work,
   you probably forgot to plug them into the wall socket.)

2. Our computer has a <u>Random Access Memory</u>. (RAM for short.)
   This means that the computer will hold data in its memory
   only as long as the machine is left on, and 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.)

3. Keep your feet away from the electrical cords.
   You will make 'fuzz' on the screen, and you will lose your program
   if you accidentally kick loose an electrical plug.

4. If you put your cassette tape near the power pack,
   the magnetic field set up by the power pack might
   erase your tape.

5. Take it easy with the keyboard—no pounding, please!

6. Turn off both the TV and the keyboard when
   you are not using them.

SECTION 4: 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 <u>program statements,</u>
and you'll learn more about them in Section 5.

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 <u>commands</u>— they are not part of a program.

You type them in and press ENTER
and the computer does them right away.
(I wrote ENTER in a box because it has a special key
all its own on the keyboard, like BREAK .)

Program statements all have <u>line numbers</u> in front of them,
to tell the computer which statement should be done first.
(You will see these line numbers in Section 5.)
Remember that commands do <u>not</u> have a line number
because they are not part of a program.

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

CLS   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 is no longer stored inside the computer!

NEW   This erases your last program from the memory
        so you can start a new program with a 'clean' memory.

LIST   This command prints out, in order, whatever program
        statements you have typed into the memory so far.
        If your program has more lines
        than will fit on the TV screen all at once,
        push the ▲ key, and the next group
        of statements will scroll up so you can see them.

RUN   This tells the computer you have finished typing
        all the instructions in your program,
        and now you want the computer to do the job
        for which you gave the directions. This is called
        executing the program. When the computer is finished
        executing the program, it will print READY on the screen.

| BREAK | If the computer is in the middle of executing your program and you want it to stop, push the BREAK key. |

| CONT | If you change your mind and want the computer to continue executing the program after you pushed BREAK, then type in CONT and press ENTER. |

| CSAVE | This command is used when you want to save a program you have written on the computer, so you can use it another time without having to type it all in again. You save your program on a cassette tape, using the tape recorder: Follow these directions: |

SAVING YOUR PROGRAM ON A CASSETTE

1. Advance your cassette tape to the spot
   where you want to record your program. Remember the number
   on the Tape Recorder Counter! You will need to pull
   the first gray plug on the side of the recorder to do this.

2. Plug the gray plug back in.

3. Press down the (PLAY) and (RECORD) keys until they stay.

4. Type in CSAVE slowly and carefully, then press [ENTER].

5. When the computer prints READY on the screen,
   turn off the recorder.

6. Your program is now recorded on the cassette.
   Make sure the location and title
   of your program is written down on the cassette.

NOTE: You must use a good quality cassette tape,
and it must be new. You cannot record one program
over the top of another, or erase
an old music tape and use it for computer programs.

CLOAD This command lets you load a program on a cassette back into the memory of the computer, so you can use it again. Follow the directions:

LOADING A CASSETTE PROGRAM INTO THE COMPUTER

1. Advance your tape to a few numbers before where your program is stored. You will need to pull the first gray plug to do this. (Example: if your program is stored at 30, advance the tape to 28, to be sure you don't miss the beginning of your program.

2. Plug in the gray plug again.

3. Push the (PLAY) button.

4. Type in CLOAD, slowly and carefully, then press ENTER.

5. When your program is loading, ** will appear at the top of the screen, and the * on the right will blink.

6. When the screen says READY, the program is in the memory of the computer, ready to be used.

7. Turn off the recorder.

8. Type in LIST and press ENTER.

SECTION 5: PRINT AND VARIABLES

Let's begin by writing a program using the PRINT statement.
Our first statement in any program on the TRS-80 should be:

```
5   CLS
```

This clears the screen, to get rid of any 'garbage' which might mess up your program later. Remember—CLS only clears the TV screen—it does not erase your program from the memory. (CLS is one of the few commands that can be given a line number and become a program statement.)

Now we'll use a PRINT statement to print out the following message:

```
5   CLS

10 PRINT "HELLO!  I AM THE TRS-80 COMPUTER!"

15 PRINT "THIS MUST BE YOUR FIRST PROGRAM. "

20 END
```

...and the last line will be an END statement to show the computer where the program ends.
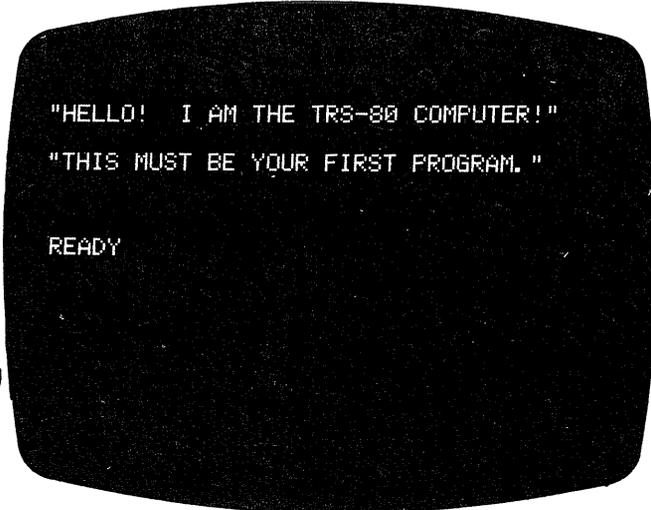
Now—when we type in RUN and press ENTER,
this is what the computer will print on the screen:

You try typing it in.

Remember to press ENTER
each time you finish
typing a line.

RUN has no number in front of it.

See if it turns out the same
as the screen I've drawn here.

"HELLO! I AM THE TRS-80 COMPUTER!"

"THIS MUST BE YOUR FIRST PROGRAM."

READY

The computer prints READY at the end of this program,
even if you didn't tell it to. This is the computer's way
of telling you it is finished executing this program,
and is 'ready' to do something else.

Whenever you use a statement like PRINT "Hello," the computer
will print out <u>exactly</u> what you put between the quotation marks.
Even if what you put is silly! Even if it is spelled incorrectly.

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

```
5  CLS

10 PRINT "I KIN SPELL REEL GOOD."

15 PRINT "GIGGLE!  GIGGLE!"

20 END

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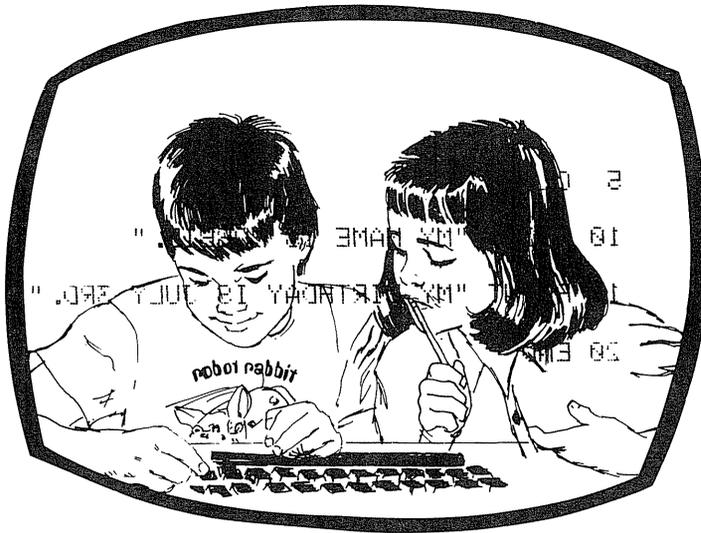
(RUN is not part of the program, but I'm putting
it here so you don't forget to type it in
every time you want to run your program.
Later, I won't write it down each time.)

```
5  CLS

10 PRINT "MY NAME IS JOHN SMITH."

15 PRINT "MY NAME IS MARY JONES."

20 END

   RUN
```

```
5  CLS

10 PRINT "I AM A FRIENDLY COMPUTER."

15 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 in between. 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   CLS
10  PRINT "MY NAME IS ROBBIE."
15  PRINT "MY BIRTHDAY IS JULY 3RD."
20  END
```

Now—if I wanted to put a line in my program
telling how old Robbie is, right after line 1∅,
I could just type in:

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

If I type in CLS to clear the screen, and then type LIST,
the computer will put line 12 into the program,
in the right place, and this is how the program will appear:

```
5  CLS

10 PRINT "MY NAME IS ROBBIE."

12 PRINT "I AM 9 YEARS OLD."

15 PRINT "MY BIRTHDAY IS JULY 3RD."

20 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   CLS

10 PRINT "TODAY IS TOOSDAY."

15 END
```

In this program, line 1∅ has a spelling mistake.
To get rid of line 1∅ completely, all you do is type in: 1∅
and hit the return key. This will erase line 1∅ from the memory.

But suppose you are typing a line and you notice right away
that you've made a mistake, even before you go on to the next line.
Can you erase part of a line? Of course! All you do
is press the ⬅ key until you see the mistake erased
from the screen, and type in the correct letter.
Remember that the ⬅ key only works for the line you are
typing on right then – if you are typing on line 15,
you cannot erase something on line 5 with the ⬅ key.
You will have to type in a whole new line 5.

You may also be wondering why zero
is written with a line through it, like this: ∅
This is done on all 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 TRS-80, if you type in something the computer does not understand, it may print <u>WHAT</u>? or <u>HOW</u>? on the screen. This means you have made a mistake and you should try again. These 'messages' are called <u>error messages</u>.
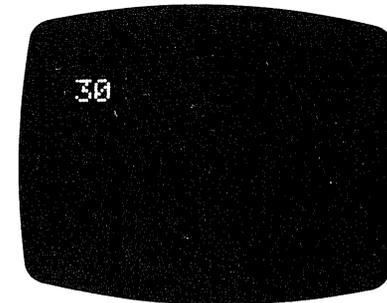
The PRINT statement can also be used to skip a line.

```
5   CLS
10  PRINT "HELLO"
15  PRINT
20  PRINT "GOOD-BYE"
25  END
```

```
HELLO
GOOD-BYE
```

Without the quotation marks in a PRINT statement, the TRS-80 will work like a calculator:

```
5   CLS
10  PRINT 10 + 20
15  END
```

```
30
```

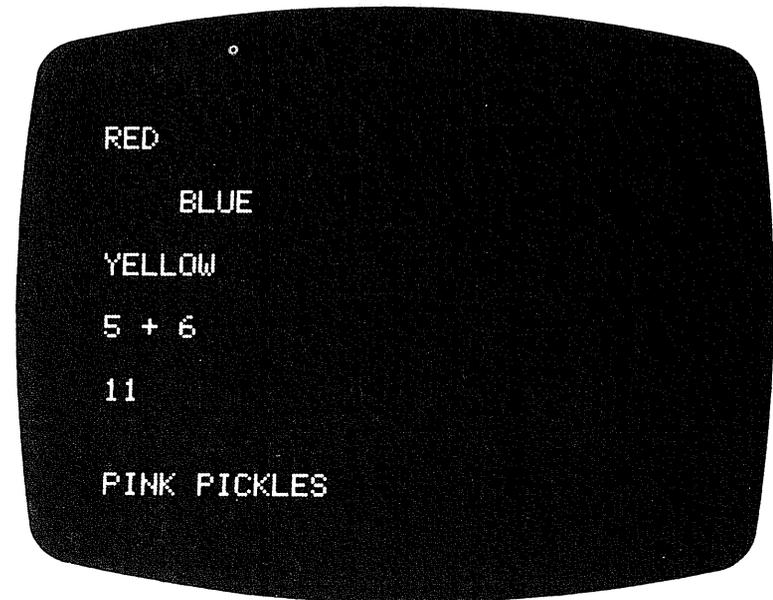This program will print out the <u>answer</u> to 10 + 20, which is 30. If you wanted the computer to print out the actual problem 10 + 20, you write it like this: `10 PRINT "10 + 20"`

Do you notice the difference?

23

Here is a PRINT program and the results on the TRS-80 screen
when the program is executed. Look it over carefully.

```
5   CLS

10  PRINT "RED"

15  PRINT "    BLUE"

20  PRINT "YELLOW"

25  PRINT "5 + 6"

30  PRINT  5 + 6

35  PRINT

40  PRINT "PINK PICKLES"

45  END
```

```
                 o


RED

        BLUE

YELLOW

5 + 6

11


PINK PICKLES
```

Notice that line 45 END does <u>not</u> print the word 'END'
on the screen. It just tells the computer
that this is the <u>end</u> 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 the 'mailbox' by using a LET statement.

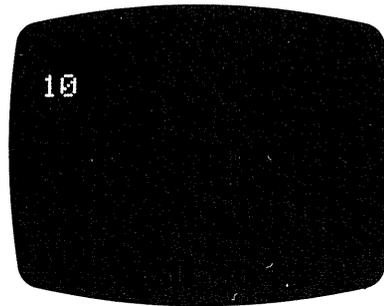| A	B	C	D
5	7	2	0

```
5   CLS
10  LET A = 5
15  LET B = 7
20  LET C = 2
25  LET D = 0
```

Now, whenever you use the statement `30 PRINT "A"`
in your program, 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 say: `WHAT?`

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   CLS
10 LET A = 6
15 LET B = 4
20 PRINT A + B
25 END
```

```
10
```

MEMORY

A	B
6	4

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

NOTE: The TRS-80 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 $6 / 2$

SECTION 6: GOTO, INPUT, and RND

PRINT statements alone don't make very exciting programs, but this section has three 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   CLS

10  PRINT "HELLO"

20  GOTO 10

30  END
```

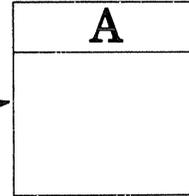Every time the computer gets to line 20, the program tells it to goto line 10.

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!

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

```
5   CLS

10  PRINT "TYPE IN YOUR AGE."

20  INPUT A
```

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:

```
25  PRINT "YOUR AGE IS "

30  PRINT A

35  END
```

Type in 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 in front of it,
telling the person who uses your program
what the computer is waiting for them to type in.

28

RND (which is short for Random) also stores a number
in the memory, but the computer picks the number!

```
5   CLS

10  LET X = RND (25)

15  PRINT X

20  END
```
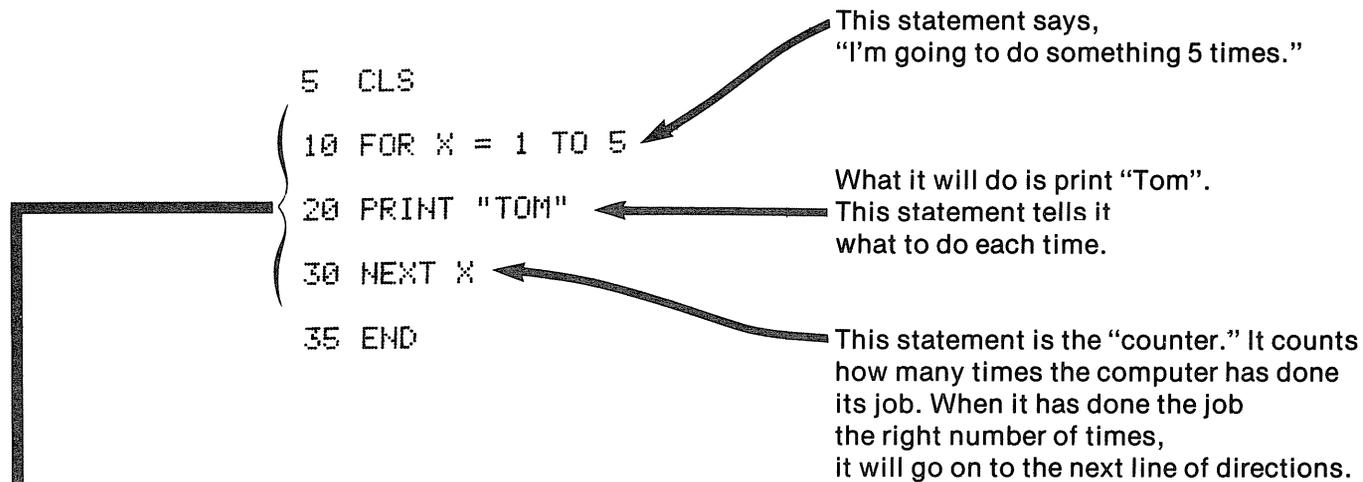
This statement makes
the computer pick a
"secret number" between
1 and 25 and stores it
in memory space X.

RND is a very handy statement for game programs.
After we learn about IF-THEN statements
in Section 7, you will be able to write
your own computer game of "Guess my number!"

SECTION 7: 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  CLS

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:

```
5   CLS

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
```

FOR-NEXT LOOP

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 an error message:

```
10  FOR Q = 1 TO 12

15  PRINT "HARRY"

20  NEXT Q
```

These two variables
must be the same

This program will print "HARRY" 12 times.

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

Name

```
5   CLS

10  FOR Z = 1 TO 100

15  PRINT "SUSAN IS GREAT."

20  NEXT Z

30  END
```

Numbers

```
5   CLS

10  FOR R = 1 TO 100

15  PRINT R

20  NEXT R

25  END
```

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.

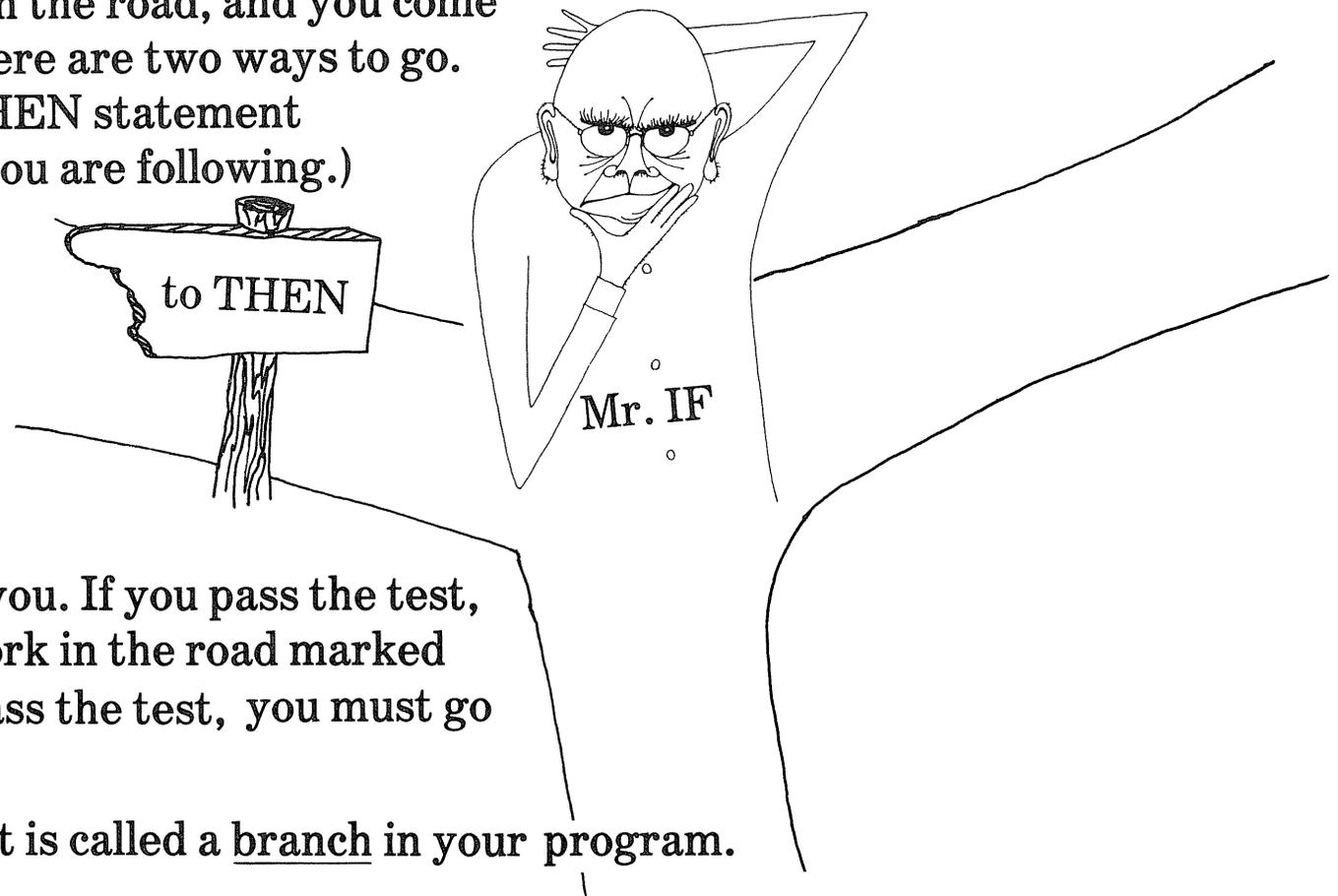## IF-THEN statements provide a "test" for your programs.

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.

```
5   CLS

10  PRINT "TYPE IN YOUR FAVORITE NUMBER."

20  INPUT N

25  IF N = 5 THEN PRINT "YOU HAVE PICKED
    THE LUCKY NUMBER!"

30  END
```

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.)
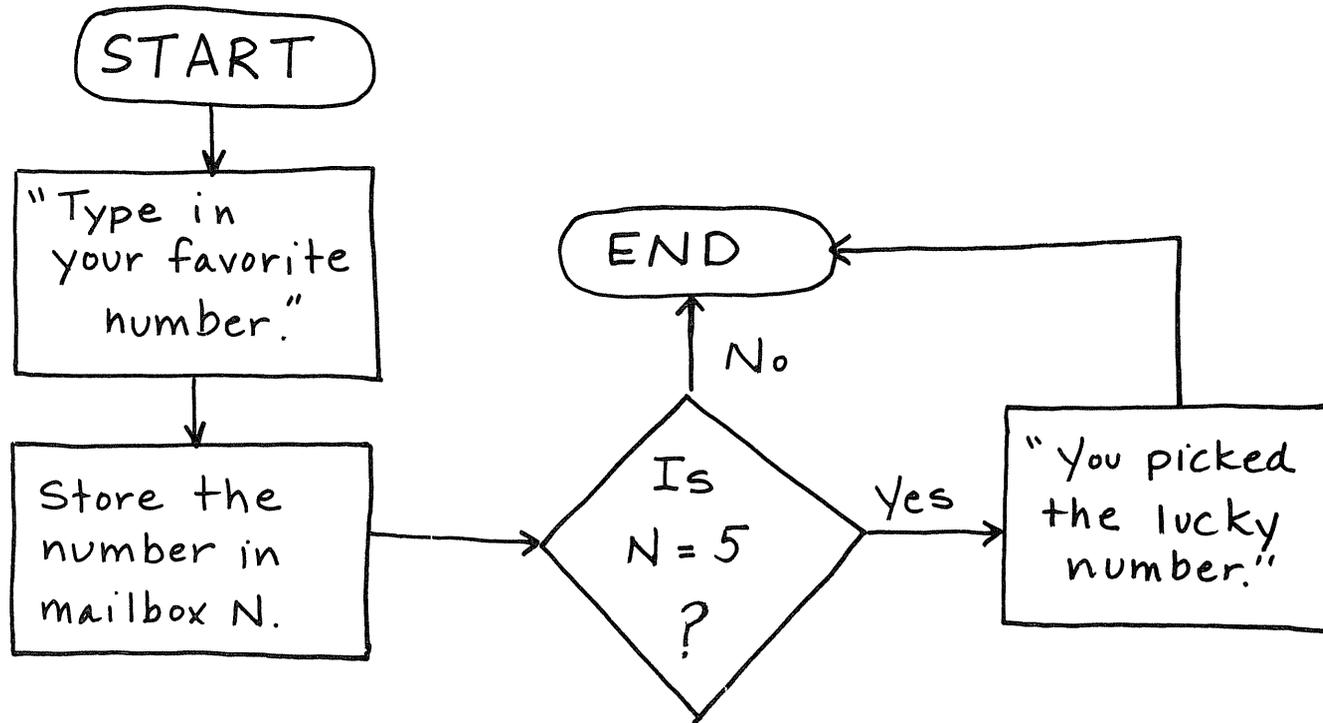
to THEN

Mr. IF

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.
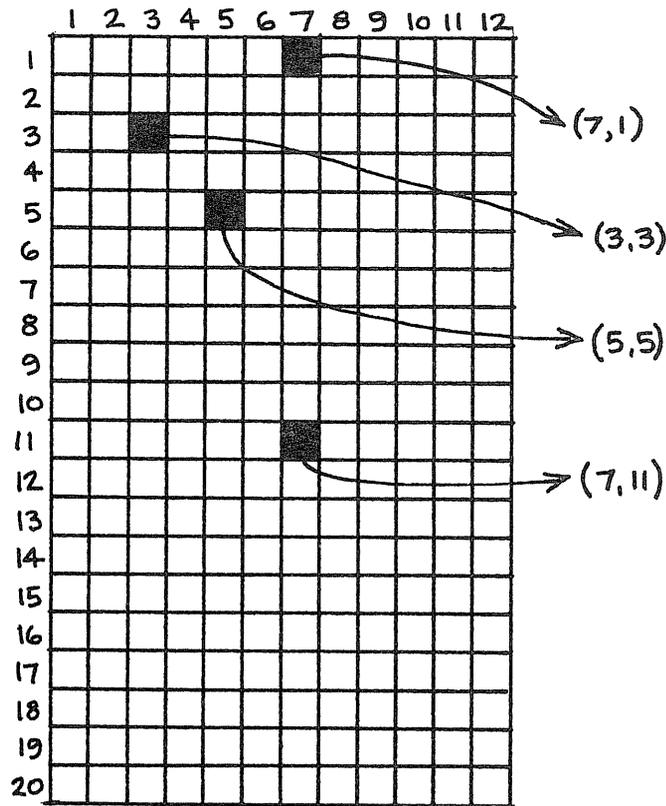
We can also show this with a flowchart:

```
5   CLS

10  PRINT "TYPE IN YOUR FAVORITE NUMBER."

15  INPUT N

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

30  END
```

SECTION 8: GRAPHICS PROGRAMS

Graphics programs let you make pictures on the screen with dots of light. There are 127 dots <u>across</u> each row on the screen, and 47 dots <u>down</u> the side.

Each dot has an address, so you can tell the computer which dot you mean.

Here is a picture of part of the screen. Each of the black dots has an address made up of two numbers.

To get the first number, you count how far <u>across</u> the screen that dot is.

The second number shows how far <u>down</u> from the top of the screen that dot is.

To light up a dot at an address, you use SET.

```
5   CLS

10  SET (7,11)

15  SET (7,1)

20  SET (5,5)

25  SET (3,3)

30  END
```
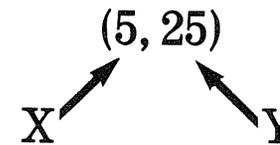
To turn off the dot of light at a certain address,
you use RESET.

```
5   CLS

10  SET (7,11)

15  RESET (7,11)

20  END
```

* A special note: When you write your own graphics programs,
try not to make all your dots in the upper left corner
of the screen. When your program is over and READY
comes on the screen, it will erase them.

When you count out the address
for a dot, we call the number across the screen X.
This is the first letter in the address:

$$(5, 25)$$

X ↗ ↖ Y

We call the number counted
down from the top of the screen Y.
This is the second letter
(or number) in the address.

We need to know about these variables
used to name points, because often you will want
to write an address in a SET statement
as (X, Y), so you can change X and Y easily.

This may sound confusing, but it's not, once you try it.
Here's an example to get you started:

Rain

```
5  CLS
10 LET X = 1
20 LET Y = 1
30 SET (X,Y)
40 X = X + 1        makes this number one bigger
50 Y = Y + 1
60 GOTO 30
70 END
```

Writing a graphics program only with SET statements
that have an address for one single point
can take a very long time, because every single dot
needs a line of its own.

Using (X,Y) as an address makes the job much easier.
Let's look at a way to draw lines,
using (X,Y) and FOR-NEXT loops.

Vertical Line

```
5  CLS
10 FOR Y = 1 TO 47
15 SET (50,Y)
20 NEXT Y
25 END
```

the values of Y will go from
1 to 47 (down the screen).

this line will be drawn at
50 spaces across the screen.

Horizontal Line

```
5  CLS
10 FOR X = 1 TO 127
15 SET (X,24)
20 NEXT X
25 END
```

the values of x will go from
1 to 127 (across the screen).

this line will be drawn
at 24 spaces down.

By putting a GOTO statement in your graphics program,
you can make your graphics blink on and off!

Remember the CLS statement we put
at the beginning of every program,
to clear away any "garbage" on the screen?
If you tell the computer to GOTO
the CLS after it prints out your picture,
the screen will go blank, and then it will print
your picture again. This will make your picture
blink on and off!

Try this program and see:

After the computer
lights up this line across
the screen, it goes to CLS
and starts over.

```
5   CLS

10  FOR X = 1 TO 127

15  SET (X,25)

20  NEXT X

25  GOTO 5
```

SECTION 9: SAMPLE PROGRAMS

Rain

```
5  CLS
10 LET X = 1
15 LET Y = 1
20 SET (X,Y)
25 X = X + 1
30 Y = Y + 1
35 IF Y > 41 THEN 50
40 GOTO 20
50 END
```

Stars

```
5  CLS
10 LET X = RND (127)
15 LET Y = RND (47)
20 SET (X,Y)
25 GOTO 10
30 END
```

Worm Race

```
5  CLS
10 FOR X = 1 TO 127
15 SET (X,24)
20 SET (X,25)
25 NEXT X
30 END
```

Computer Panic

```
5  CLS
10 PRINT "HELP!   THIS COMPUTER IS CRAZY!!!"
15 GOTO 5
```

Pine Tree

```
5  CLS
10 PRINT "     X"
15 PRINT "    XXX"
20 PRINT "   XXXXX"
25 PRINT "  XXXXXXX"
30 PRINT " XXXXXXXXX"
35 PRINT "XXXXXXXXXXX"
40 PRINT "     X"
45 PRINT "     X"
50 PRINT "     X"
55 END
```

Guess my Number

```
5  CLS
8  LET N = RND (100)
10 PRINT "I HAVE A SECRET NUMBER."
15 PRINT
20 PRINT "IT IS BETWEEN 1 AND 100."
25 PRINT "TYPE IN YOUR GUESS."
30 INPUT G
40 IF N = G THEN PRINT "CONGRATULATIONS!  YOU WIN!"
45 IF N = G THEN END
50 IF N > G THEN PRINT "GUESS A HIGHER NUMBER."
55 IF N < G THEN PRINT "GUESS A LOWER NUMBER."
60 GOTO 25
```

SECTION 10: GLOSSARY OF STATEMENTS AND COMMANDS

BREAK — stops the execution of a program.

CLOAD — loads a recorded program from the cassette tape into the memory.

CLS — clears any writing or graphics off the screen (does not erase program from memory).

CONT — continues the execution of a program after you stop it by using BREAK.

CSAVE — saves a program by recording it on a cassette.

END — tells the computer the program is over.

ENTER — this key must be pressed each time you finish typing in a line.

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

    Example: 1Ø FOR X =1TO1ØØ

                       15 PRINT "HELLO"

                       2Ø NEXT X

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

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.

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".

               2Ø INPUT N

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

    Example: 15 LET R = 1ØØ

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

NEW — erases the old program from the memory.

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

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

RND — has the computer pick a number at random.
   Example: 5X=RND(1ØØ) picks a random number X between 1 and 1ØØ.

SET — lights up a certain dot on the screen.
   Example: 15 SET (11Ø, 35) lights up the dot at 11Ø spaces over and 35 spaces down.

RESET — turns off the light at a certain dot on the screen.
   Example: 5 RESET (75, 26) turns off the dot of light at 75 spaces across and 26 spaces down.

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 TRS-80, before you do anything else, read through the manual which comes with the machine. 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 lessons on programming in the Radio Shack manual. It will not be necessary to learn everything in the manual—check the glossary of this book for the most crucial statements and commands.

This book is designed to accompany the Level I machine, although 99% of the material also applies to the Level II machine. For all practical purposes, the book can be used with either machine.

You will see in the manual that the Level I machine will accept one-letter abbreviations for the majority of the statements and commands the children will be learning. I do not teach the abbreviations until late in the year, for I find it is difficult for the children to proofread and trace the function of another person's program when it is a mass of single letters, instead of words such as PRINT and GOTO, which are more readily understood.

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. Stepping 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 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.

46

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. 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. Running the machine itself, behavior guidelines, and scheduling (Save CLOAD and CSAVE for later.)

SECTION 4
6. Beginning programming: CLS, NEW, LIST, RUN with PRINT examples

SECTION 5
7. PRINT statements with quotation marks; error messages
8. PRINT to skip lines; editing
9. PRINT with arithmetics (+, —, *, /)
10. PRINT variables—simple
11. PRINT variables—such as PRINT A + B
12. CSAVE & CLOAD for saving programs on cassette tapes

SECTION 6
13. GOTO
14. INPUT
15. RND

SECTION 7
16. FOR-NEXT with PRINT and arithmetic statements
17. More work on FOR-NEXT
18. IF-THEN test
19. IF-THEN in more complex programs (draw flowchart of program function)

SECTION 8
20. Discussion of SET (X,Y) and plotting of coordinates on paper
21. Graphics worksheets
22. Graphics-make their own initials with SET
23. Graphics-vertical and horizontal lines
24. Blinking graphics; more complex programs

SECTION 9
25. 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.

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: Make pizza; Build a doghouse;
Give a dog a bath; 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 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 cassette player, or the children will be confused and never learn the difference. 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.

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 use the counter properly.
d. They continually fiddle with the volume controls. Find the best settings for your machine, and then tape the volume controls shut, so no one can change them.
e. They don't save one program twice on the same tape, to insure that at least one of them will turn out.

Section 4: 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.

The statement 5 CLS at the beginning of each program prevents a great many problems that can arise when many people use the same computer.

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.

Section 5: 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.

You may delete a line only if you are at the end of the listing of your program. This means that if you have a long program which fills the screen several times when you LIST it, you cannot delete a line until it has scrolled all the way to the end.

I use only variables of one letter with beginning programmers.

Section 6: GOTO, INPUT, RND

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 TRS-80 manual. Someone will ask you sooner or later how to make things PRINT right next to each other on the same line, using several PRINT statements.

RND is great for visually showing a random distribution when you teach probability.

Section 7: 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 get overly concerned with detail when you draw them.

"X is not equal to Y" is written as X<>Y

Section 8: 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 section. This is very helpful, especially for the younger children.

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	1	2	3	4	5
1					
2					
3					

Tracing the RAIN program through its memory changes is helpful for showing how the values of variables change in the program.

When using variables for the coordinates, I teach the children to use X for horizontal and Y for vertical, at all times. This simplifies things, and prepares the children, in a minor way, for Cartesian Coordinates in mathematics.
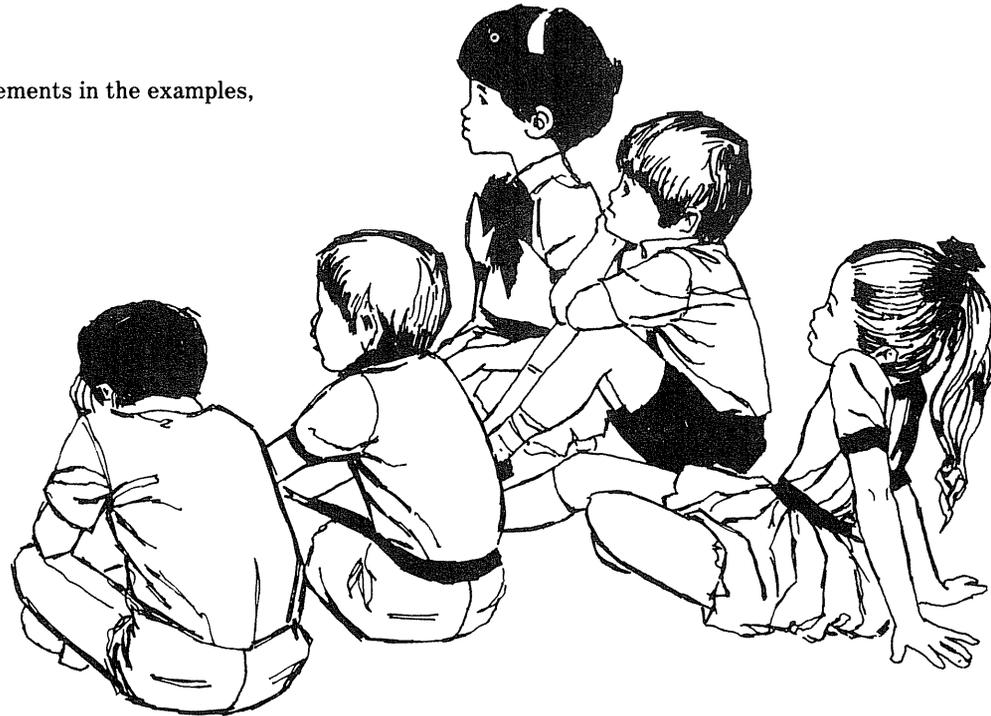
Section 9: Sample programs

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

All 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 time.

Section 10: Glossary

I put line numbers in front of all the statements in the examples, to distinguish them from commands.

NAME

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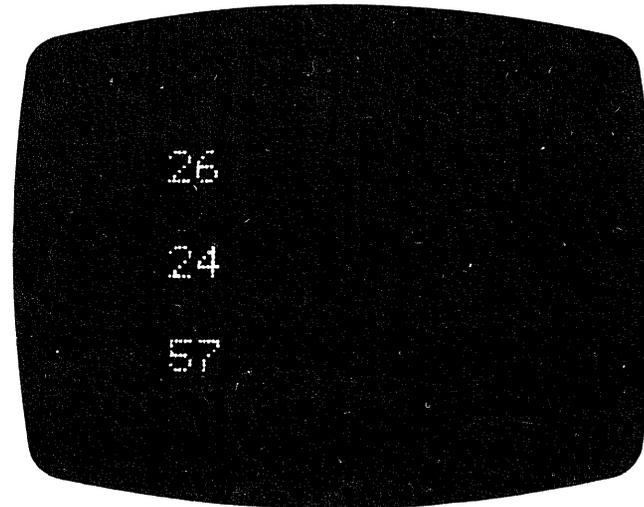your "printout" on the screen.

```
10 CLS
20 PRINT "BIG"
30 PRINT
40 PRINT "YELLOW"
50 PRINT "BLUE"
60 END
```

```
10 CLS
20 PRINT "THE ANSWER"
30 PRINT 30 * 2
40 PRINT 30 + 2
50 PRINT 30 - 2
60 PRINT "THE END"
70 END
```
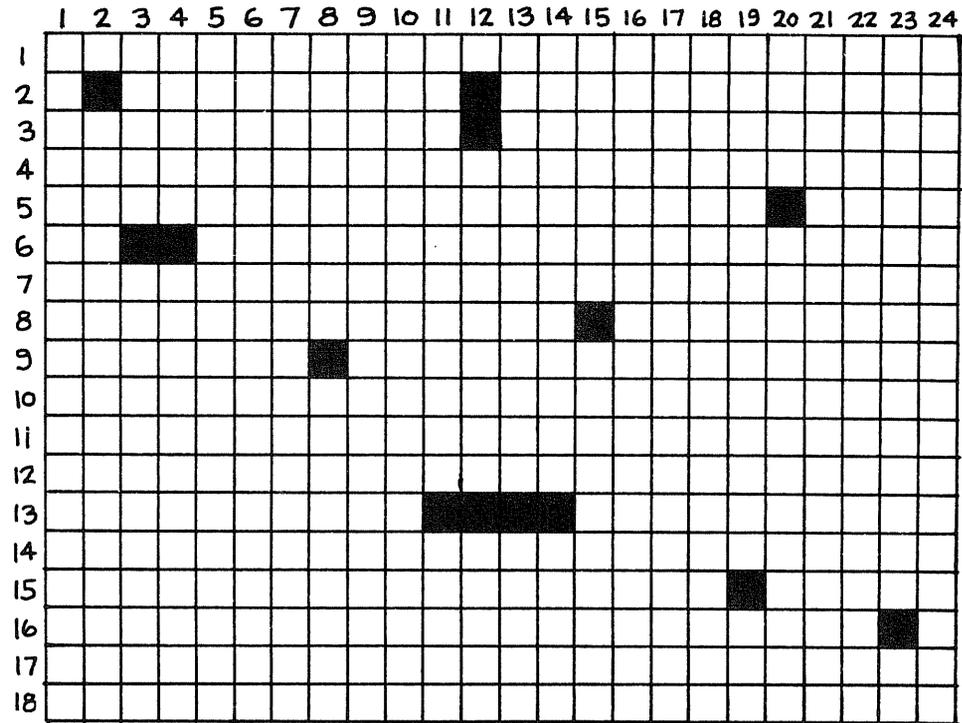
Here is a program and "printout." Find and fix the mistakes
in the programs so a run will produce what is shown on the screen.

```
10 CLS

20 PRINT 20 + 6

30 PRINT 30 + 4

35 PRINT

40 PRINT "60 - 3"

50 PRINT 10 - 10

60 PRINT "HELLO"

70 END
```
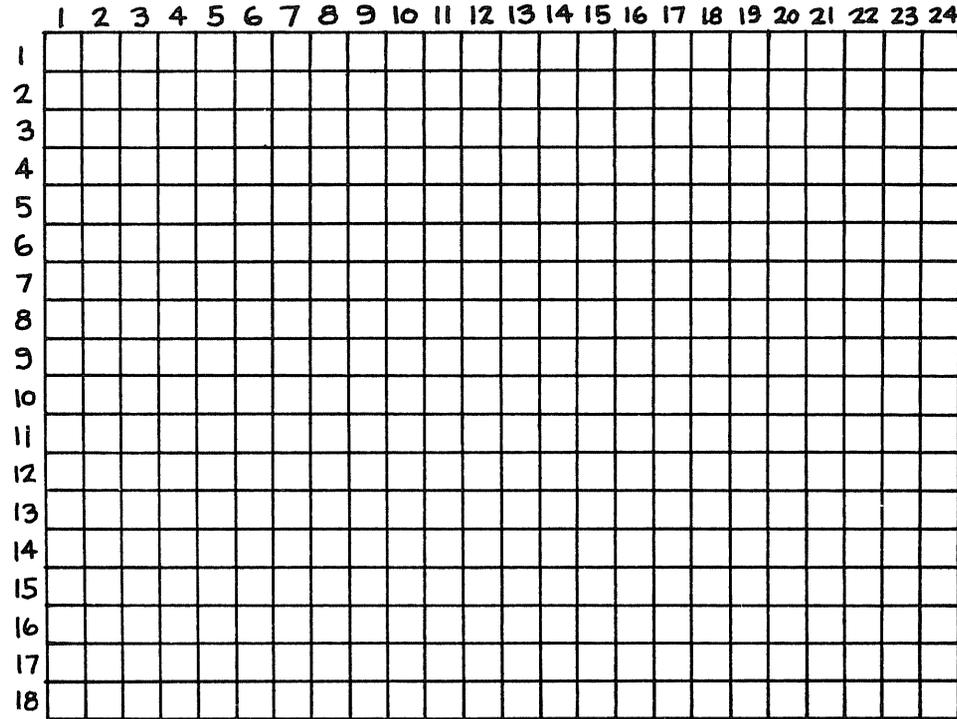


```
26

24

57
```

Write the program to print out the dots shown below.
Don't forget the END statement!

On the graph below, color in each dot which
will light up when you run this program:

```
5   CLS

10 SET (15,12)

15 SET (12,15)

20 SET (1,5)

25 SET (1,6)

30 SET (10,10)

35 SET (20,14)

40 SET (21,14)

45 SET (24,18)

50 END
```

| | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 |

(Grid: 24 columns wide, 18 rows tall, with rows numbered 1–18.)

Notes

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 TRS-80
- 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, Atari, Timex-Sinclair, and Commodore VIC-20 computers.

"There is a special section for parents and teachers with teaching ideas, trouble-shooting hints, lesson plans.

CREATIVE COMPUTING PRESS