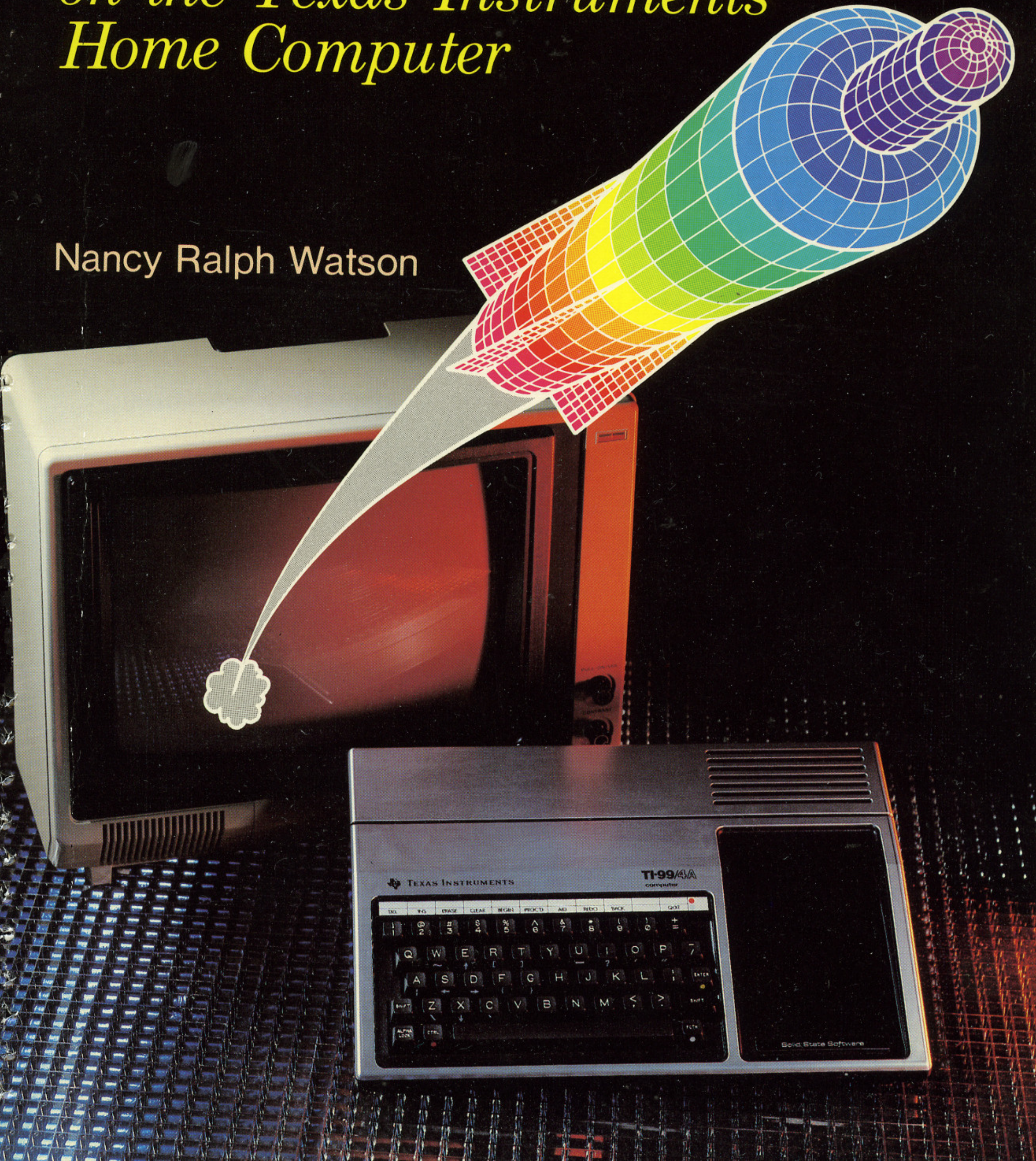
Taking Off with BASIC

on the Texas Instruments Home Computer

Nancy Ralph Watson



Taking Off with BASIC

Publishing Director: David T. Culverwell
Acquisitions Editor: Susan Love
Production Editor/Text Designer: Roberta L. Glencer
Manufacturing Director: John A. Komsa
Art Director/Cover Design: Don Sellers
Assistant Art Director: Bernard Vervin
Typesetting: Alexander Typesetting, Inc., Indianapolis, IN
Printing by: Fairfield Graphics, Fairfield, PA
Typefaces: Century Schoolbook (text), Eurostile (display), OCR-B (programs)

Taking Off with BASIC on the Texas Instruments Home Computer

Nancy Ralph Watson

Robert J. Brady Co.
A Prentice-Hall Publishing
and Communications Company
Bowie, MD 20715

Taking Off with BASIC on the Texas Instruments Home Computer

Library of Congress Cataloging in Publication Data

Watson, Nancy Ralph.
Taking off with BASIC on the Texas Instruments Home Computer.

Includes index.
Summary: "A self-paced, hands-on approach for learning to program" a Texas Instruments home computer in BASIC.
1. TI 99/4A (Computer)—Programming—Juvenile literature. 2. Basic (Computer program language)—Juvenile literature. [1. TI 99/4A (Computer)—Programming. 2. Basic (Computer program language) 3. Programming (Computers)] I. Title. II. Title: Taking off with B.A.S.I.C. on the Texas Instruments home computer.
QA76.8.T133W38 1984 001.64'24 83-25834
ISBN 0-89303-870-9

Prentice-Hall International, Inc., London
Prentice-Hall Canada, Inc., Scarborough, Ontario
Prentice-Hall of Australia, Pty., Ltd., Sydney
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Prentice-Hall of Southeast Asia Pte. Ltd., Singapore
Whitehall Books, Limited, Petone, New Zealand
Editora Prentice-Hall Do Brasil LTDA., Rio de Janeiro

Printed in the United States of America

84 85 86 87 88 89 90 91 92 93 94 10 9 8 7 6 5 4 3 2 1

TABLE OF CONTENTS

to George, with love

Limits of Liability and Disclaimer of Warranty

The author and publisher of this book have used their best efforts in preparing this book and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regards to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Trademarks of Material Mentioned in This Text

Texas Instruments 99/4A is a registered trademark of Texas Instruments, Inc.

Note to Authors

Do you have a manuscript or a software program related to personal computers? Do you have an idea for developing such a project? If so, we would like to hear from you. The Brady Co. produces a complete range of books and applications software for the personal computer market. We invite you to write to David Culverwell, Publishing Director, Robert J. Brady Co., Bowie, MD 20715.

PREFACE

TAKING OFF WITH BASIC is a self-paced, hands-on approach for learning to program a computer in a language called BASIC. That is, learning how to get your Texas Instruments computer (TI for short) to do what you want it to do. The computer does not have a mind of its own—it is only as clever as you make it. When you learn to write programs giving the computer instructions to do certain things, you have gained much power, for now you are in control of a machine that has many capabilities.

BASIC stands for Beginners All-Purpose Symbolic Instruction Code and was invented in 1965 at Dartmouth College. It is thought to be the most widely used computer language in the world. This is probably due to its simplicity compared to other computer languages.

Throughout this manual you will discover all kinds of things that the computer can do, and how you can make those things happen using BASIC. Once you learn each different instruction, statement, and command that the computer understands, you will probably want to try some of your own ideas instead of just using the ones in this manual. That is exactly the purpose of this book. By creating your own programs you will find out how much power you really do have. You will also realize that you are smarter than the computer and will probably invent many new ways of getting the computer to execute your instructions.

The first seventeen chapters teach you how to follow a plan instructing the computer to make an object on the screen look like it is taking off. You will be the designer of the object. There will be a countdown, sound effects, and color, too. Each chapter adds new instructions until, at the end of Chapter 17, you have one complete program.

Each chapter also has a CHECKPOINT which helps you evaluate how well you are learning the material. The answers to each CHECKPOINT are found in Appendix A.

The second part of the book has seventeen supplemental chapters. Each supplement expands the ideas or introduces advanced concepts related to the first seventeen chapters with the same number.

Appendix B is a listing of Musical Tones for use with the sound capabilities of the TI and Appendix C is the Character Code Chart.

For your convenience, a list of the most frequently used statements and commands, with page numbers for quick reference, can be found on the inside front and back covers. A detailed Index is also included.

A friendly word of advice: don't let this manual limit your ideas in any way. Depending on the programmer's personal style, there are many different ways to write the same program and produce the same results. Use the suggestions in this manual that make sense to you, and if you have a better idea, try it! Be sure to write down each new idea in case you want to use it later.

Be prepared to be challenged. You will be pleased with the results. Most of all, enjoy this new learning experience.

ACKNOWLEDGMENTS

To Betty Bryan for freely volunteering evenings and weekends to assist in the preparation of the manuscript.

To Tracy Hugo for offering her invaluable comments at the most critical time.

To my son Matthew for his creative designs and programs for the graphics found throughout.

To George Watson for his patience and unselfish sharing of time, encouragement, and expertise while helping to see this project to completion.

To Sue Love whose enthusiasm and support made this a truly enjoyable project.

TIPS ON HOW TO USE THIS BOOK

Taking Off with BASIC is divided into two sections: Chapters 1-17 and Supplements 1-17. They can be used in two ways: read and complete Chapter 1, and then complete Supplement 1; do the same for Chapter 2, Supplement 2, etc. This is recommended for advanced users. A second way to use the supplement chapters is to read them after the first seventeen chapters have successfully been completed, treating the supplements almost as a second book. This is recommended for beginning users.

Saving the Program

If you have a cassette tape recorder or disk drive, it is recommended that you save the TAKE OFF program after completing each chapter so you can load it into the computer each time you are ready to begin working again.

If you do not have a device to save the program, there is a PROGRAM UPDATE at the end of each chapter. Here is how to use it:

1. After completing a chapter, edit the PROGRAM UPDATE at the end of the chapter so that it corresponds with the program lines you wrote. (The PROGRAM UPDATE is just an example, and yours will not be identical to it.)
2. If you plan to continue to the next chapter immediately, you will encounter a problem. At the beginning of each new chapter, there are practice programs to enter so you can learn the concepts being introduced. If you enter them as shown, they will interfere with the TAKE OFF program. Most of the time you will not want to erase the TAKE OFF program and have to re-enter it. You can avoid this with a little trick. For the examples that are not related to the TAKE OFF program, first type in this line:

   ```
   1 GOTO 800 (Press ENTER)
   ```

   and add 800 to the line numbers given in the examples. This causes the computer to actually jump over the TAKE OFF program and doesn't bother it at all. When you are instructed to LOAD TAKE OFF FILE NOW enter all of the line numbers you used for the examples, one at a time, and press ENTER. For example,

```
1       (Press ENTER)
800     (Press ENTER)
810     (Press ENTER)
820     (Press ENTER)
```

and so on. This eliminates those lines, and the TAKE OFF program is again ready to be used.

3. If you finish a chapter and are not ready to continue with the next, write the differences in your program on the PROGRAM UPDATE and turn the machine off. When you return for another session, follow the instructions in the new chapter until you reach LOAD TAKE OFF FILE NOW. At this time, enter the entire PROGRAM UPDATE in the previous chapter before proceeding with the rest of the chapter.

Supervising Children

If you plan to assist children in the use of this book, these suggestions may be helpful:

1. The reading level is approximately at the 6th grade level. It has been used successfully with gifted students beginning at age 9.
2. Because of the individualized style of the book, children should be allowed to establish a comfortable pace for themselves.
3. Depending on the level of experience of each user, help to determine if and when a child is ready for the supplemental chapters.
4. Each chapter makes a nice lesson in terms of the time it takes to complete (approximately 30 minutes) and its orientation toward success.
5. One or two children may work together with the text at the computer. However, they should complete the CHECKPOINTS separately. If their understanding of the concepts is greatly different, or if one child is going faster than the other, it may be best for them to work alone or with someone closer to their level.
6. If the children save their work at the end of each chapter, help them to select a special file name to use so that there is no chance of it being chosen for another project. For example, first names or initials are not always safe if a number of files are saved on one disk.
7. It is helpful for some children to have an introduction prior to beginning work on a new chapter. This can take the form of a discussion of the concepts and examples, allowing for questions and

answers that will eliminate many of the questions that arise during the work session.

Pressing ENTER

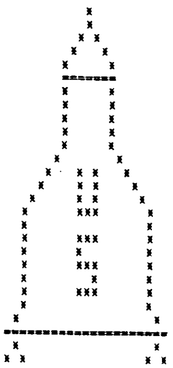
After typing an instruction into the computer, the ENTER key must be pressed in order to place the instruction in the computer's memory.

For the first several chapters you will be reminded of this. However, eventually you will be on your own. You will probably have the experience of typing in an instruction and sitting there, waiting for something to happen, not realizing that you have not pressed ENTER. The computer does not know when you have finished writing an instruction–the only way it knows is when the ENTER key is pressed. Beware of this occurrence.

Introduction to Chapters 1-17

TAKING OFF WITH BASIC has been written to make BASIC easy to understand for those with little or no programming experience. The chapters are short, and provide practice for each concept that is introduced. By working through the frequent TRY THIS examples in each chapter, you will build a program that includes a countdown and take-off of an object you design yourself complete with color and sound. While it is not necessary for you to have a device to save the work you do in each chapter, you are encouraged to do so if you have either a cassette tape recorder or disk drive. Otherwise, you may enter the PROGRAM UPDATE at the end of the chapter you have completed before continuing with a new chapter.

Good luck!



1 Your Microcomputer

You will notice that the Texas Instruments 99/4A microcomputer, or TI as it is usually called, runs on electricity. If there's a cord to plug in, then there is usually an ON/OFF switch–right? Well, the TI has an ON/OFF switch for each cord that comes with each piece of computer HARDWARE. **HARDWARE** is any equipment used with the microcomputer. Make sure all cords are plugged in.

The television the computer is hooked up to is one piece of HARDWARE which needs to be turned on. It is also sometimes called a monitor or CRT. Another piece of HARDWARE which needs to be turned on is the KEYBOARD. The **KEYBOARD** is something like a typewriter because it has keys with the alphabet and other characters on it. It is not just a typewriter, however, because underneath the keys are the parts that make it a computer.

To turn on the KEYBOARD, find the small black rectangle on the front right-hand corner and slide it to the right until you see the red light. If you did not already do so, turn on the monitor (television). Now you are in business!

Look at the monitor. Do you see the colorful screen and the name TEXAS INSTRUMENTS, along with other things? This is like a title page in a book. Notice also that it asks you to PRESS ANY KEY TO CONTINUE. Another word for the information you see on the screen is **OUTPUT.** That is what is coming from the computer to you. If you press a key, it is called **INPUT.** That is what is going from you to the computer. These two words, INPUT and OUTPUT, will be used frequently. Be sure that you understand what they mean.

For example, if you want the computer to work a math problem, you must write some instructions. Are the instructions INPUT or OUTPUT? Hopefully you said INPUT, since that is information you are PUTting INto the computer.

After the instructions to do a problem are entered into the computer, an answer will be shown. Is the answer INPUT or OUTPUT? Of course it is OUTPUT, because the computer is PUTting OUT information on the screen for you.

In the Preface, writing a PROGRAM was mentioned. Programs are also called **SOFTWARE.** Everytime you want the computer to do something, you must give it instructions in a language that the computer understands. In English, we write sentences. In BASIC, a language the computer understands, we write a series of STATEMENTS. **STATEMENTS** are similar to sentences, but we cannot just tell the computer to PRINT COMPUTERS ARE FUN TEN TIMES and expect it to do it. It is simple, but not that simple!

English has very specific rules, but so does BASIC. This manual will teach you how to communicate with the computer in BASIC, plus explain many of the rules of the language. You must spell everything correctly, or the computer will not understand. Luckily, most of the words it knows are short and easy, like END, PRINT, IF and STOP, to name a few.

The computer is just an extension of your brain. What you tell it, it will do, as long as you do so in its language. After completing this manual you will know one more language–but this computer still knows only one–BASIC. (Some computers know several.)

Don't worry if you make mistakes–that's the quickest way to learn to program. It won't take long before you become comfortable with the hardware used with the computer and the special vocabulary introduced.

CHECKPOINT INFORMATION

If you are able to answer most of the questions in the CHECKPOINTS correctly, then you are ready to go on to the next chapter. If you have trouble with some of them, go back and read the chapter again until you can answer the CHECKPOINT questions successfully.

CHECKPOINT 1

1. The television connected to the TI-99/4A is also called a ______________
or ______________.

2. Any piece of equipment used with microcomputers is called ________________________________.
3. The ON/OFF switch of the computer is on the front of the ____________________.
4. We usually call the Texas Instruments microcomputer the ________________ for short.
5. Information which you give to the computer is called ________________.
6. Information that the computer gives to you is called ________________.
7. Sentences in English are similar to ________________ in BASIC.
8. Writing instructions for the computer in BASIC is called ________________.

TO CHECK YOUR ANSWERS SEE PAGE 143 IN APPENDIX A

```
    ******
   *      *
  *########*
 *  $ $ $   *
*    $ $     *
*     $      *
 *    $     *
  *        *
   *      *
    *    *
     ----
    (   )
    (   )
    (   )
    (   )
     ----
```

2 Programming Style

Style refers to the particular way in which you do something. For example, dancing and talking are done with a personal style. When you write a program, you will develop your own style with which you feel comfortable. This chapter offers many suggestions that will be of great help as you learn to program.

Get in the habit of making a plan for what you want to instruct the computer to do. There are four steps to a good plan; 1) set a goal; 2) write a program to carry it out; 3) check to see if it does what you want it to; and 4) if it doesn't work, correct it until it does. The word used by programmers for correcting a program is **DEBUGGING.**

Line Numbers

So that the computer will know in what order you want the instructions to be followed, it is necessary to number each statement you write. An important style rule is that you number each statement by tens. The first line number will be 10, the second will be 20, and so on. If you start with number 1, then use number 2, it would be ok, until you decide you want to add something in between 1 and 2. Sorry, but there is no such line number as 1½. Line numbers must be whole numbers. Numbering by tens gives room to make mistakes, change your mind, and add new ideas. A finished program rarely is numbered perfectly by 10's, but it should always begin that way.

END Statement

Just as a sentence must have a period at the end, a program should have an **END** statement. Since you don't usually know exactly how long the program is going to be, it is a good idea to choose an END statement line number that is quite large. It is wise to use a number with all 9's, such as 999 or 9999. If you make a habit of always using 9's in the END statement, you will not have trouble remembering what number you used. Be sure that your END line number is larger than the largest line number you intend to use in the program. For example, if the program goes through line number 1050, the END line number cannot be 999, since 999 is smaller than 1050. The number 9999 is the next number with all 9's that is larger than 1050.

REMarks

Another important suggestion for style deals with **REM** statements. REM stands for REMarks. With a REM statement you can write messages to yourself right in the program. These REM statements are entirely for you because the computer completely ignores them when running the program.

Why would you want to write yourself messages in a program? Good question. You speak in English, but write instructions to the computer in BASIC. Sometimes it is easier to say in English what you are trying to do in BASIC. For example, you might write a short program to calculate the area of a triangle. You can write a REM statement that says in English FOLLOWING IS A FORMULA TO CALCULATE THE AREA OF A TRIANGLE and put it right in the program. REM statements are a way to write in English within a BASIC program.

Believe it or not, after writing several programs, you will not remember everything in each one. By using several REMS in each program, you can go back and read any program you have written and know immediately what it is intended to do.

Remember that REM statements are only for your use and that the computer doesn't care a bit about them as long as they have a line number and REM is spelled correctly. In the next chapter you will have a chance to write a program with REM statements.

Programming style depends on each individual but there are several ideas which have been suggested that will increase programming efficiency. Try to use them as you complete the exercises in each chapter.

CHECKPOINT 2

1. You should number your lines by ______________ when writing programs.
2. Every program must have an ______________ statement.
3. If you want to write yourself messages within a program, use ______________ statements.
4. When you first decide to write a program follow the four steps for a good ______________.

TO CHECK YOUR ANSWERS SEE PAGE 143 IN APPENDIX A

```
            !
           i i
          i   i
         i  V  i
        i  V V  i
       i    V    i
      i           i
     i   $$$$$$    i
    i     $$$$      i
   i       $$        i
  i       $$$$        i
 i       $$$$$$        i
i                       i
========       ==========
        # ++++ #
        # ++++ #
        # ++++ #
        # ++++ #
        #      #
        ########
```

3 Commands

Turn on the monitor and keyboard. Refer to page 3 of this manual for help. On the screen is the title page. Press any key. You now have a choice. Press the number 1. This prepares the computer to accept a BASIC program. You will see the words TI BASIC near the bottom of the screen. There also is a little arrow under those words, and a flashing black box. This box is called a **CURSOR** and is where your program begins.

```
TI BASIC

>■
```

FIGURE 3-1

Throughout this manual there will be TRY THIS examples. When you come to them, type the instructions listed in capital letters exactly as shown and press the **ENTER** key. The ENTER key must be pressed after each instruction is typed. You'll be reminded of this as we go along and will quickly get used to doing it.

Commands

Commands are one-word instructions which, when typed on the keyboard and the ENTER key is pressed, are carried out immediately, except when they are part of a program. There are four commands that are frequently used.

1. **NEW.** Whenever you first begin to enter a program on the computer, it is important to type **NEW** first. If you don't, anything the computer had from a previous program will be a part of the new one. By typing NEW, the computer erases everything it has in its memory so that you can begin completely fresh. Be careful not to type NEW unless you want everything erased that you have been working on. If you do this by mistake, you will probably never make that error again!

Memory is the part inside the computer that remembers the instructions you entered. Typing NEW or turning off the computer completely empties the memory.

2. **LIST.** **LIST** is a command that causes the computer to show each program line that has been typed on the keyboard.

TRY THIS:

```
NEW            (Press ENTER)
```

The screen blinks and the memory is cleared of anything that was in it.

TRY THIS:

```
LIST           (Press ENTER)
```

You receive the message:

```
*CAN'T DO THAT
```

That is because you have not typed any statements yet. There is nothing in the memory to list. Let's try an experiment. Type the following line into the computer, exactly as shown.

TRY THIS:

```
10 PRINT 12345      (Press ENTER)
LIST                (Press ENTER)
```

You should see line 10 appear on the screen. In fact, it should be on the screen twice. Once for the time you typed it, and once after the LIST command. The first line is input, and the second line is output. Do you understand why? It is because the first line you typed in, and the second line you asked the computer to produce for you.

3. **RUN.** **RUN** is another command that the computer will execute as soon as it is typed and the **ENTER** key is pressed. Notice that the two

statements on the screen right now are identical. Each has a line number, the word PRINT, and the number 12345.

TRY THIS:

```
RUN              (Press ENTER)
```

Only the number 12345 is printed. That is because you have asked the computer to follow the instruction in line 10, which told it to PRINT those numbers. You also get the message:

```
**DONE**
```

which lets you know the computer has finished following all the instructions.

TRY THIS:

```
LIST          (Press ENTER)
RUN           (Press ENTER)
LIST          (Press ENTER)
RUN           (Press ENTER)
```

LIST shows the entire instruction, and RUN makes the computer execute (follow) it.

TRY THIS:

```
NEW           (Press ENTER)
LIST          (Press ENTER)
```

Line 10 is now erased from memory.

4. **CALL CLEAR.** There is one more command that is very useful at times. It is the **CALL CLEAR** command.

TRY THIS:

```
CALL CLEAR    (Press ENTER)
```

Do you see what it does? It gets rid of all the things on the screen (except the arrow and cursor). It is not the same as typing NEW, however. It does not erase the program from the computer's memory—just from the screen. Type in line 10 again just as you did before.

TRY THIS:

```
10 PRINT 12345     (Press ENTER)
CALL CLEAR         (Press ENTER)
LIST               (Press ENTER)
```

Line 10 is still there. **CALL CLEAR** only erases the screen, not the computer's memory.

TRY THIS:

```
NEW                     (Press ENTER)
LIST                    (Press ENTER)
```

It is now gone. NEW is very different from the command CALL CLEAR.

Making Corrections

Holding down the key marked **FCTN** on the keyboard, and pressing the right arrow (→) found on the letter **D**, causes the cursor to move to the right. Try moving the arrow to the right several spaces. Now move it to the left by holding down the **FCTN** key and pressing the **S**. The left arrow (←) is on the side of the S key.

Press ENTER and type your name, but spell it wrong. Don't press ENTER. Use FCTN and the arrows to go back and correct your name so that it is spelled correctly. Type over the incorrect letters and they will be replaced with the new letter you input from the keyboard. Use the space bar to erase letters or numbers to the right of the cursor that are no longer needed.

Type in other misspelled words and without pressing ENTER correct them using FCTN and the left and right arrows. Use these arrows to correct spelling errors as you write programs.

NOTE: In the following chapters you will be writing a program which you will want to keep so that you can continue building upon it. If you have a cassette tape recorder or disk drive, follow the instructions in supplements 1 and 3 so you will be ready to save the program beginning with the next chapter.

CHECKPOINT 3

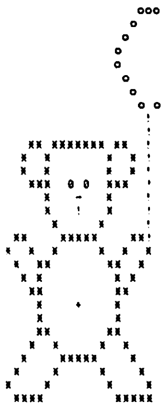
1. When you type the command ____________, the computer erases everything in its memory.
2. Type the command ____________ to see the program you have written so far.
3. When are commands executed by the computer?

____

____

4. If you want the computer to follow instructions in a program, type the command ______________.
5. To erase everything on the screen, but not from the computer's memory, use the command ____.
6. Each time you type in a command, you must press ______________ to let the computer know you are finished.
7. Which key must you hold down first if you want to use the left or right arrow? ______________
8. Use the left and right arrows when you want to ______________________ ____

TO CHECK YOUR ANSWERS SEE PAGE 143 IN APPENDIX A



4 The PRINT and GOTO Statements

Beginning with this chapter we will start writing the TAKE OFF program which will be completed in Chapter 17. If you intend to save your work on a cassette tape or disk, read Supplements 1 and 3 before proceeding. You must have a tape or disk ready in order to save the program.

BASIC programs are made up of a series of statements that must have line numbers. Following the NEW command below is an example of a statement written in BASIC. To type the quotation marks (or any character on the side of a key), hold down the FCTN key and press the key with the character you want. To type the characters above the letters or numbers, hold down the SHIFT key before pressing the key with the character you want.

TRY THIS:

```
NEW              (Press ENTER)
10 PRINT "TAKING OFF WITH BASIC"     (Press ENTER)
```

The line number is 10, the statement (instruction) is **PRINT**, and inside the quotation marks is the message you want the computer to output or show on the screen. The computer does not even look at (or care about) what is inside the quotation marks. It just outputs whatever it finds, exactly as you have entered it.

Write your own line 20. Have the computer PRINT your name and today's date. (Hint: Be sure to put your own name and current date in quotation marks.)

Line 20 will look something like this:

```
20 PRINT "PAT GOLDEN, APRIL 15, 1984"
```

TRY THIS:

```
LIST   (Press ENTER)
RUN    (Press ENTER)
```

Lines 10 and 20 should be showing on the screen as well as the output. If not, type NEW and enter the two lines again. Be sure to press ENTER after you are finished typing each line. If the two lines are on the screen you are ready to continue.

Here is another experiment. Type line 30 with a PRINT statement and inside the quotes type something silly.

TRY THIS:

```
30 PRINT "YOU ARE A BOZO"      (Press ENTER)
LIST                           (Press ENTER)
RUN                            (Press ENTER)
```

Are all three lines (10, 20 and 30) there? Use the LIST command as much as you like to check to see if the computer is remembering everything you want it to.

TRY THIS:

```
30                  (Press ENTER)
LIST                (Press ENTER)
```

What is missing? Line 30! You have discovered a way to erase just one program line. Any time you want to get rid of one line of instructions, simply type the line number and press ENTER. That line will be erased as soon as ENTER is pressed.

TRY THIS:

```
9 REM   TITLE OF PROGRAM      (Press ENTER)
19 REM   NAME AND DATE        (Press ENTER)
LIST                          (Press ENTER)
```

Check to see that there are four lines of instructions on the screen; lines 9, 10, 19 and 20. They are also stored in the memory. Otherwise, when the program is listed, it wouldn't appear on the screen. Note that the computer placed all the numbered instructions in order for you, even if you didn't enter them that way. Now type the command RUN. If you remembered to press ENTER, you should see the following on the screen:

```
TAKING OFF WITH BASIC
PAT GOLDEN, APRIL 15, 1984    (or whatever is in line 20)
```

Where are the REM statements you put in? Remember that the computer ignores REM statements when it runs a program. LIST the program and the REM statements are there, but you do not see them when the program is executed.

Character Graphics

Soon you will discover why this book is called TAKING OFF WITH BASIC. Using **CHARACTER GRAPHICS**, which is a fancy term that refers to making pictures on the screen using the letters and symbols on the keyboard, you will design your own picture, and it will be something that "takes off". For example, it might be a space ship, hot air balloon, helicopter, or anything else you can think of that takes off going straight up. Following is an example of a simple design in CHARACTER GRAPHICS. Type in the lines just as you see them, spaces and all.

TRY THIS:

```
310 PRINT "XXXXXXX"
320 PRINT "  XXX"
330 PRINT " XXXXX"
340 PRINT "XXXXXXX"
350 PRINT "   X"
360 PRINT "  XXX"
```

You may first wonder why these program lines begin with line 310, after the strong suggestion that line numbering be by ten's. It is because in this manual we are going to write a long program with many parts and it is necessary to leave space for them. We already have decided on the entire plan for this program and because of that it is already known what line numbers are needed for each different section of the program.

LIST the program and check it. If there are any errors, correct them by retyping the lines. It is always a good idea to proofread and look for errors in each line before pressing ENTER, so that you can use the arrow keys to go back and make corrections. Once you press ENTER, however, you cannot use the arrow keys to go back to a certain line. There is another method for making corrections after you press ENTER which you will learn in the next chapter.

RUN the program. This CHARACTER GRAPHICS design isn't very exciting. You can surely do better than that!

To design your own CHARACTER GRAPHICS, it is helpful to use the grid in Figure 4-1 on page 21. Notice that each number, letter, or character is in its own box. The design at the beginning of the chapter is a very simple example of a program for an object. Use the large, blank grid on page 22 (Figure 4-2) to design an object of your choice that takes off vertically (straight up). You might want to make a copy of the blank grid to use. The grid is the same size as the screen.

Write a complete program for one design on the grid before entering it: line numbers, PRINT statement, quotation marks, graphic symbols, and quotation marks again. Only use the blank spaces provided so you don't exceed the dimensions of the screen. Throughout this manual are illustrations of taking off objects that have been included to spark your imagination.

There is one important rule to remember when programming a design for this project: begin at line number 310 and do not go beyond line 490. It is OK to count by fives for line numbers if you need more room, but don't make it larger than the grid. Also, at line 300, write a REM statement describing the space ship (or whatever object you design.) Run and debug the program until you are satisfied with it.

GOTO Statement

Our next plan is to make the object take off. Only one statement is needed to make the object take off.

TRY THIS:

```
495 GOTO 310          (Press ENTER)
```

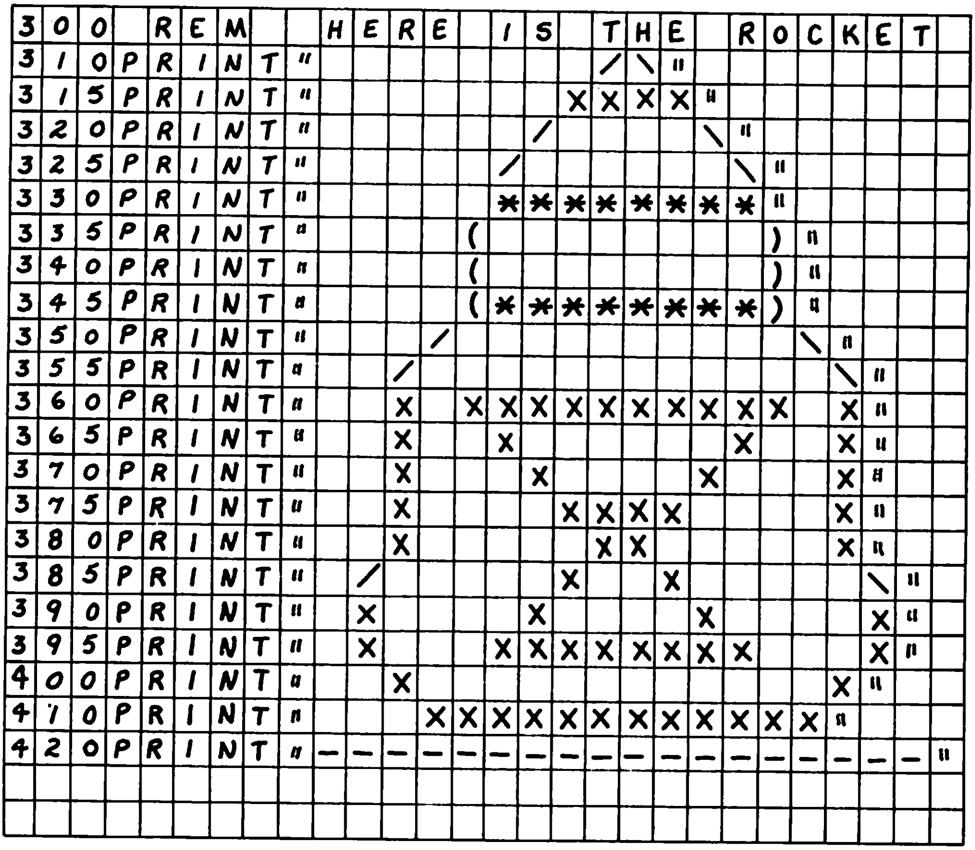
Literally this line is saying "Go back to line 310 and do what it tells you." It will run all the lines between 310 and 495, which tells it to go back to 310 again. This makes the object look like it is taking off.

TRY THIS:

```
RUN                   (Press ENTER)
```

Press FCTN and the number 4 to stop the object. You have learned how to get it going, but have no BASIC instruction for stopping it yet. A better way to stop it will be introduced later but for now this is how you must stop it. **FCTN 4** (labeled above the 4 key as CLEAR) is a command telling the computer to stop its execution of the program immediately. After giving the command FCTN 4, the program stops and the computer tells you at which line number the BREAKPOINT occurred. The BREAKPOINT just means where it stopped. Sometimes having this information is very helpful in correcting mistakes (BUGS), as you will discover.

TIP: If you type a PRINT statement that is so long that part of it must go on the next row, it is important that you do not press ENTER until you have finished typing the entire instruction.



```
300 REM  HERE IS THE ROCKET
310PRINT"        /\"
315PRINT"       XXXX"
320PRINT"      /    \"
325PRINT"     /      \"
330PRINT"     ********"
335PRINT"    (        )"
340PRINT"    (        )"
345PRINT"    (********)"
350PRINT"   /          \"
355PRINT"  /            \"
360PRINT"  X XXXXXXXXXX X"
365PRINT"  X  X      X  X"
370PRINT"  X   X    X   X"
375PRINT"  X    XXXX    X"
380PRINT"  X     XX     X"
385PRINT" /     X  X     \"
390PRINT" X    X    X    X"
395PRINT" X   XXXXXXXX   X"
400PRINT"  X            X"
410PRINT"   XXXXXXXXXXXX"
420PRINT"------------------"
```

FIGURE 4-1

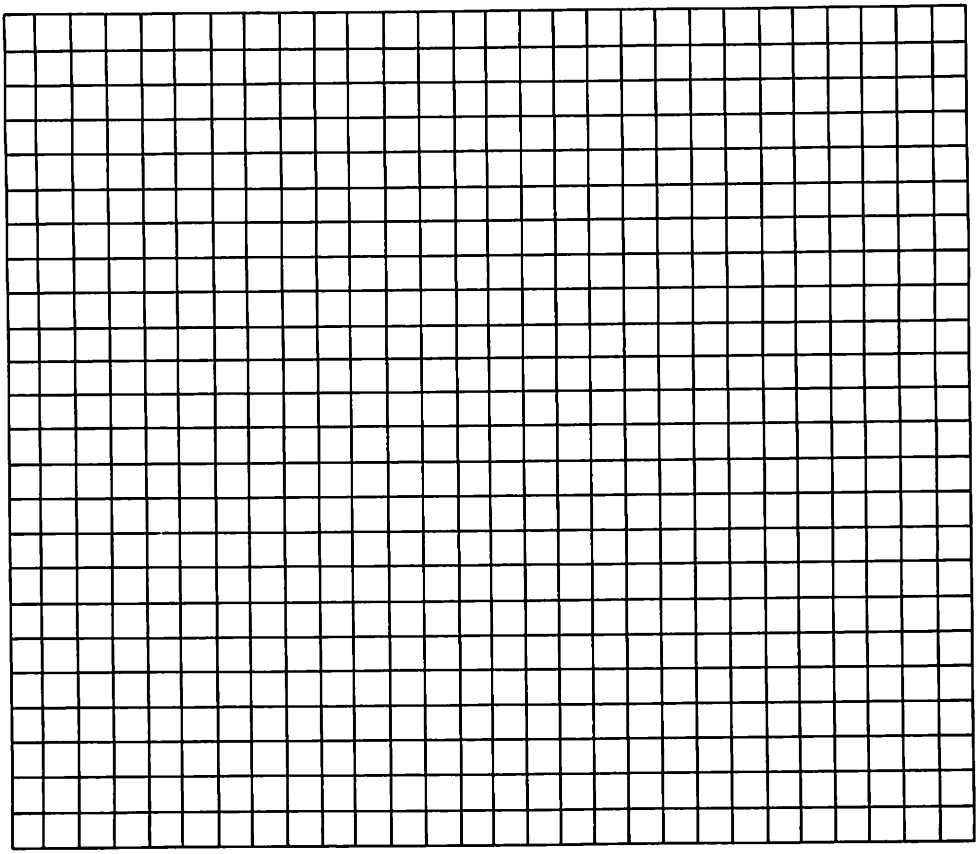


FIGURE 4-2

CHECKPOINT 4

1. If you want the computer to show your name or any words on the screen as part of a program, you must write a ____________ statement.
2. You can write yourself messages in ____________ statements, but they won't show up on the screen when the program is run.
3. A design or picture on the computer is called ____________________.
4. Before each statement you must have a ____________________.
5. To stop the execution of a program hold down the ____________ key and press ____________.

TO CHECK YOUR ANSWERS SEE PAGE 144 IN APPENDIX A

PROGRAM UPDATE

(The PROGRAM UPDATE lists your program up to this point.)

```
9 REM    TITLE OF PROGRAM
10 PRINT "TAKING OFF WITH BASIC"
19 REM   NAME AND DATE
20 PRINT "PAT GOLDEN, APRIL 15, 1984"
300 REM  HERE IS THE ROCKET
310 - 490     (THE CHARACTER GRAPHICS YOU
              DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

NOTE: Save the TAKE OFF program in a file on tape or disk now. Follow the instructions in Supplements 1 and 3.

If you don't have a device to save the program, and are ready to turn the computer off, enter the PROGRAM UPDATE before continuing with Chapter 5. Your program will be a little different from the one in this manual. Write your changes down so that you have a record of them.

Each new chapter introduces new statements and gives you practice lines to enter. They will ruin the TAKE OFF program if it is still in the memory. If you do two chapters in a row, here is a suggestion for you to use as you start a new chapter so the practice lines won't interfere with the TAKE OFF program. Type in the line

```
1 GOTO 800
```

at the beginning of each new chapter. Add 800 to the line numbers suggested in the examples. For example, if it tells you to type in line 10, enter line 810; line 20 would become 820, and so on. Ignore the END statements. When you come to the message

LOAD TAKE OFF FILE NOW

type each line number you were using for practice, entering them one at a time and pressing ENTER after each one. It might look like this:

```
1      (Press ENTER)
810    (Press ENTER)
820    (Press ENTER)
830    (Press ENTER)
```

This gets rid of these lines without ruining the TAKE OFF program.

```
     *             *             *
    ? ?           $ $           # #
   ?   ?         $   $         #   #
  ?     ?       $     $       #     #
  ?     ?       $     $       #     #
  ?     ?       $     $       #     #
  ?     ?       $     $       #     #
  ?     ?       $     $       #     #
  =======       =======       =======
   *   *         *   *         *   *
    * *           * *           * *
     *             *             *
    ***           ***           ***
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

5 Editing

EDITING is a very powerful and useful capability of a computer. The **EDIT** mode allows you to go back and make changes or corrections in any line in the program. The only thing you cannot change is the line number. If you want the line number changed, you must type the entire line over again. The word EDIT simply means to make changes.

NOTE: At the beginning of each new chapter, it is assumed that the TAKE OFF program is not loaded. That is OK since usually there are examples to practice before adding new parts to the program. Always save the program at the end of each chapter and load it only when you see this:

LOAD TAKE OFF FILE NOW

(Yes, it means NOW.)

Here's how the TI editor works. Let's EDIT (make a change in) line 10. To do so you first must put (call) that line into the EDITOR. One way to do this is to type the line number (10 in this case), hold down the FCTN key, and press the up arrow (↑), located on the E key.

TRY THIS:

10 (Hold down the FCTN key and press the letter E)

You should now have line 10 on the screen, ready to be EDITed.

The plan is to put a hyphen between the words TAKING and OFF. Move the cursor by holding down the FCTN key and pressing the right arrow until it reaches the blank space between the two words. Hold down the SHIFT key and type in a hyphen (-); it is located above the ENTER key. Press ENTER. The change has now been made. To check this, LIST the program again and look for the hyphen.

CAUTION: Do not use the space bar to move the cursor to the right because this will erase letters or characters; the arrows do not.

For practice let's change the year in line 20. Type 20 and FCTN E to place it in the EDIT mode. Move the cursor to the 1984, using FCTN and the right arrow. Simply type the numbers you want right over the other ones—use 1985. Press ENTER and the changes are made. By typing LIST 20 you can list just line 20 instead of the entire program.

TRY THIS:

```
LIST 20      (Press ENTER)
```

The change has been made.

TRY THIS:

```
20 FCTN E      (Press ENTER)
```

Press the FCTN key and arrows to change the year back to the correct one. List line 20 to check that the change was made.

What if you decide to add more words in the middle of a PRINT statement? There is a way to move everything over one or more spaces to make room for additions. Put line 10 in the EDIT mode again. Make the line read ROCKET TAKING OFF WITH BASIC. Move the cursor so that it is over the spot where you want to add the word ROCKET. It should be right on the T in TAKING OFF since that is where you want the word ROCKET to begin. Once the cursor is located on the T, hold down the FCTN key and press the 2 (INSERT) key once, and type ROCKET. What happened? Everything moved over, making room for ROCKET. Press the space bar once to add an extra space between ROCKET and TAKING-OFF. Press ENTER and then LIST 10 to see if the change was made correctly.

FCTN 2 warns the computer that you will be adding something to the line and to make room for it. Add your middle name to line 20 using this method. LIST 20 to see if it worked.

What if you decide you do not want your middle name in line 20? You are in luck! Call line 20 to the EDITOR again. Move the cursor to the first letter of your middle name and hold down the **FCTN 1** (DEL for DELETE)) key this time. The letter the cursor was on has disappeared. Hold down the FCTN key and press 1 again. Do this until your entire middle name is deleted (erased). Press ENTER and LIST 20. Your middle name should now be gone.

Erase (or delete) the word ROCKET from line 10. Practice with the EDIT mode for a while to become comfortable with it. If you have any changes you want to make in the graphic object, practice using the EDIT mode to make them. For your quick reference, here is a list of the EDIT Keys.

Line # **FCTN E (↑)** Puts line # given on the screen ready to be edited.

FCTN D (→) Moves the cursor one space to the right

FCTN S (←) Moves the cursor one space to the left.

FCTN 1 This is called the DELETE mode. Use this when you want to erase information from a line.

FCTN 2 This is called the INSERT mode. Use this when you want to add extra information to your line. Use the space bar to move the letters over.

ENTER After making your changes, press ENTER to signal the computer to permanently make changes. You can press ENTER at any position on the line.

CHECKPOINT 5

1. To make changes in program lines you must be in the ____________ mode.
2. When you want to change a line, call for it by typing the ____________, holding down the ____________ key, and typing ____________.
3. To make room for additions in a line in the EDIT mode, hold down the ____________ key, type a ____________, and then enter the information.
4. If you want to remove something from a line in the EDIT mode, hold down the ____________ key and type a ____________.
5. Once you make the changes to the line, you must press ____________ for the computer to put the changes into its memory.

TO CHECK YOUR ANSWERS SEE PAGE 144 IN APPENDIX A

PROGRAM UPDATE

```
 9 REM    TITLE OF PROGRAM
10 PRINT "TAKING OFF WITH BASIC"
19 REM    NAME AND DATE
20 PRINT "PAT GOLDEN, APRIL 15, 1984"
300 REM    HERE IS THE ROCKET
310 -490  (THE CHARACTER GRAPHICS YOU DESIGNED
           ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW

```
     *************
          / /
************************
      *          *
      *          *      ***
   ****   ----   *******  *
   *      :  :            *
   *      :  :            *
   *      ----            *
   ************************
        #         #
       # #       # #
"""""""""""""""""""""""""""""""
```

6 Countdown

Whether your object is a rocket, UFO, balloon, or whatever, it should have a COUNTDOWN so that it takes off on cue. A **COUNTDOWN** usually begins at a high number, over days or hours, but to keep it simple ours will go from 10 to 0.

LOAD TAKE OFF FILE NOW

Remember when you wanted words to be placed on the screen? You used a PRINT statement and put the words inside quotation marks. You needed the quotation marks because the computer does not recognize words that are not part of its vocabulary. However, it does recognize numbers. Therefore, when you want it to print numbers, you don't need to use quotation marks.

TRY THIS:

```
210 PRINT 10
```
(Press ENTER)

If you ran this, it would cause the computer to print a 10 on the screen. Before you run it, add the rest of the COUNTDOWN. To save some room here, we will number the lines by fives.

TRY THIS:

```
215 PRINT 9
```
(Press ENTER)

Go ahead and finish the COUNTDOWN by adding nine more lines. The last line should be

```
260 PRINT 0              (Press ENTER)
LIST 210-260             (Press ENTER)
```

Make any EDITING changes needed. The LIST command will allow you to list a block of lines from the first line number specified to the last line number. As your program grows in length, you will not want to waste time listing the entire program. Use this method to list certain lines in any part of the program you want to look at without having the rest of the program also print on the screen.

Run the program and look for the COUNTDOWN. Stop the program by typing **FCTN 4.**

TRY THIS:

```
270 STOP            (Press ENTER)
```

What do you think will happen when you run the program now? Run it and see if you were correct. The STOP statement lets you run parts of the program to check them out without having to run the entire program, since it will STOP without going on to the next line. Remember this trick as you continue to write the program. You can remove the STOP statement at any time by simply typing the line number of the STOP statement and pressing ENTER.

Add a REM statement at line 200 for the COUNTDOWN. You are probably wondering why the COUNTDOWN went so quickly when stop was not used. That is how quickly the computer reads and executes program instructions. There IS a way to slow it down, but it is a bit complicated. Before learning how to slow it down, there are some other things you must know that will be covered in the next chapter.

CHECKPOINT 6

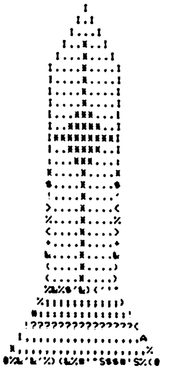
1. When you want the computer to print numbers, it is not necessary to use ____________________ as you do when you want to print words.
2. If you want the computer to run only part of the program, you can make the program stop anywhere as long as you type a ______________ statement with a line number.
3. To get the computer to list only lines 100 through 200 of a program you would type in the command ____________________________.

TO CHECK YOUR ANSWERS SEE PAGE 144 IN APPENDIX A

PROGRAM UPDATE

```
9 REM       TITLE OF PROGRAM
10 PRINT    "TAKING OFF WITH BASIC"
19 REM      NAME AND DATE
20 PRINT   "PAT GOLDEN, APRIL 15, 1984"
200 REM     COUNTDOWN
210 PRINT 10
215 PRINT 9
220 PRINT 8
225 PRINT 7
230 PRINT 6
235 PRINT 5
240 PRINT 4
245 PRINT 3
250 PRINT 2
255 PRINT 1
260 PRINT 0
270 STOP
300 REM   HERE IS THE ROCKET
310 - 490   (THE CHARACTER GRAPHICS YOU
             DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW



7 Variables and LET Statements

The COUNTDOWN in the last chapter is actually quite inefficient. In this chapter we will revise it so that it does the same thing but takes fewer lines. A new statement called LET will be used.

We have talked about the fact that the computer knows numbers. Sometimes it is helpful to give a number a name, especially if it is going to be used more than once in a program. The computer has an area in it for storing numbers that have names–similar to a group of boxes. For example, if you wanted to give the number 10 the name A, you could use a LET statement.

TRY THIS:

```
10 LET A = 10          (Press ENTER)
20 PRINT A             (Press ENTER)
RUN                    (Press ENTER)
```

A is a variable name for the number 10. When the computer comes to this LET statement, it finds an empty box in its memory, names it A, and places a 10 in it. From this point on in the program, instead of using the number 10, you can use the letter A, which is a kind of shortcut. If the numbers in a program are very large, like thousands or millions, you only have to name it in a LET statement and after that call it by the name you gave it. The names you choose for the numbers are called **VARIABLES.**

By using a **LET** statement you can tell the computer what you want a **VARIABLE** to be called. They are called VARIABLES because you can change the name of the VARIABLE at any time. The word VARIABLE comes from the word VARY, which means to change.

LOAD TAKE OFF FILE NOW

Here is an example of how you can add a VARIABLE to your program using a LET statement.

TRY THIS:

```
210 LET C = 0          (Press ENTER)
215 PRINT C            (Press ENTER)
230 STOP               (Press ENTER)
```

Before running the program, delete the following lines (type the line number and press ENTER after each one):

```
220   225   235   240   245   250   255   260   270
```

Run the program. You will notice that all it printed of the COUNTDOWN was a 0. LIST lines 200 through 300 (LIST 200-300) and examine the program. In line 210 you have given the computer a new definition for C using a LET statement. You have told it that from now on, whenever you ask it to PRINT C it will print a 0.

Let's change the value for the VARIABLE named C.

TRY THIS:

```
220 LET C = C + 1          (Press ENTER)
225 GOTO 215               (Press ENTER)
230                        (Press ENTER)
LIST 200-230               (Press ENTER)
RUN                        (Press ENTER)
```

After a few seconds hold down the FCTN key and press 4. Can you figure out what is happening? Why did it keep counting until you stopped it? Lines 210-225 look like this:

```
210 LET C = 0          (Press ENTER)
215 PRINT C            (Press ENTER)
220 LET C = C + 1      (Press ENTER)
225 GOTO 215           (Press ENTER)
```

Line 210 places the value 0 in a box named C. Line 215 instructs the computer to print whatever is in the box named C. Right now C = 0, so 0 is printed on the screen. Line 220 adds 1 to the current value for C; C now equals 1. Line 225 sends it back to line 215.

This time, when the computer adds 1 to the value in box C, it adds 1 to 1; C is now equal to 2. Line 220 prints a 2. This goes on and on adding one each time in an infinite (endless) loop, until you stop it. It never goes beyond line 225 the way it is now written.

One problem with the new program lines for the COUNTDOWN is that it is counting from 0 on up, instead of from 10 down to 0. Can you make changes in the program so that it counts backward instead of forward? Hint: A hyphen, (-), is the subtraction symbol.

TRY THIS:

```
220 LET C = C - 1      (Press ENTER)
RUN                    (Press ENTER)
```

Stop the program (FCTN 4). What happened with the COUNTDOWN this time? It counted backward from 0 until you stopped it. Line 210 tells the computer to begin at 0. The value for C is printed at line 215, then 1 is subtracted from C in line 220 (C=-1). If C started with a value of 10 in line 210, then the COUNTDOWN would begin with the right number (10).

TRY THIS:

```
210 LET C = 10      (Press ENTER)
RUN                 (Press ENTER)
```

Oh no! Now what is wrong? It won't stop counting! Stop it using FCTN 4. You have just created another infinite (or endless) loop. The program will go on forever, or until the computer counts backward as far as it can. At least you got it to go backward starting at 10. Don't worry about the COUNTDOWN not stopping at 0. That problem will be solved in the next chapter.

TIP: Since you have been writing program lines for several chapters now, you will no longer be reminded to press ENTER after entering each instruction. Keep this in mind as you go through the rest of the manual.

CHECKPOINT 7

1. You can put a number into the computer's memory but first you must give the number a ____________ name.
2. To tell the computer what value you want a variable to have you must use a ____________ statement.

3. If you want a variable to increase by one each time it is printed, you would write a LET statement with a +1 −1 (Circle one).
4. When you cause the computer to do something forever, this is called an ____________ or ____________ loop.

TO CHECK YOUR ANSWERS SEE PAGE 144 IN APPENDIX A

PROGRAM UPDATE

```
9 REM     TITLE OF PROGRAM
10 PRINT "TAKING OFF WITH BASIC"
19 REM    NAME AND DATE
20 PRINT "PAT GOLDEN, APRIL 15, 1984"
200 REM   COUNTDOWN
210 LET C = 10
215 PRINT C
220 LET C = C - 1
225 GOTO 215
300 REM    HERE IS THE ROCKET
310 - 490        (THE CHARACTER GRAPHICS YOU
                  DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW

```
  o o
 o   o
o     o              o o
o     o             o   o
 o   o             o     o
  o o              o     o
   !                o   o
   !                 o o
   !                  !
   !                  !
   !                  !
   !                  !
 ------------------------------
 !             x              !
 !             x              !
 !             x              !
 !xxxxxxxxxxxxxxxxxxxxxxxxxxxx!
 !             x              !
 !             x              !
 !             x              !
 ------------------------------
```

8 IF/THEN Statements and Relations

Another new statement called **IF/THEN** helps to stop the COUNTDOWN wherever we want. Included in the statement there must be a symbol of RELATION. Here is a handy chart for your reference:

RELATIONS

GREATER THAN	>	GREATER THAN OR EQUAL TO	>=
LESS THAN	<	LESS THAN OR EQUAL TO	<=
EQUAL TO	=	NOT EQUAL TO	<>

These **RELATION** symbols are a very important part of computer programming. You should memorize them so that you know which one to use when certain situations arise. One of these symbols is needed in each IF/THEN statement.

Here is a simple example. (Be sure to press ENTER after each instruction is typed.)

TRY THIS:

```
10 LET N = 1
20 PRINT N
30 LET N = N + 1
40 IF N = 5 THEN 60
```

```
50 GOTO 20
60 PRINT "N IS EQUAL TO 5"
99 END
RUN
```

NOTE: If you are adding 800 to these practice lines, you must also add 800 to 60 in the IF/THEN statement and to 20 in the GOTO statement.

This program prints the numbers 1 to 4 on the screen. It prints N IS EQUAL TO 5 when the value for N becomes 5 because of line 40 which literally means, "If the value for N is equal to 5 then go to line number 60." If N is not equal to 5 it goes on to line 50, which sends it to line 20.

Edit lines 40 and 60 to look like these:

```
40 IF N <> 5 THEN 60
60 PRINT "N IS NOT EQUAL TO 5"
LIST
RUN
```

Line 40 is saying, "If N is not equal to 5 then go to line 60." It is not equal to 5 (line 10 tells it N is equal to 1) so only a 1 is printed on the screen, followed by N IS NOT EQUAL TO 5.

How would these lines change the output?

TRY THIS:

```
40 IF N > 3 THEN 60
60 PRINT "N IS GREATER THAN 3"
LIST
RUN
```

The output should be:

```
1
2
3
N IS GREATER THAN 3
```

since it prints numbers until N IS GREATER THAN 3. Then it prints the message N IS GREATER THAN 3 at line 60.

LOAD TAKE OFF FILE NOW

We can use this same idea to stop the COUNTDOWN at 0.

TRY THIS:

```
225
250 IF C <> -1 THEN 215
LIST 210-250
```

The output will be

```
210 LET C = 10
215 PRINT C
220 LET C = C-1
250 IF C <>-1 THEN 215
```

Study the lines carefully.

The first time line 215 is reached, the number 10 is printed because C = 10. Line 220 subtracts one from the value of C, making C equal to 9. Line 250 literally instructs, "If the value for the variable C IS NOT EQUAL TO −1 yet, then go back to line 215, print C again, and subtract one more from the variable C." C is now equal to 8 so it goes back to 215, prints 8 and subtracts one from C again in line 220. C is now equal to 7 so at line 215 the number 7 is printed. C is still not equal to −1 so it goes to 215 again.

After going through this loop eleven times, C will finally equal −1. When this happens, line 250 will no longer be true. Let's say that, at line 215, C is now −1. At line 250 it says, "If C is NOT EQUAL TO −1 then go back to line 215." But it IS equal to −1 so it will ignore the 215 part of line 250, and continue to the next line (which is 300).

Run the program and see what happens this time. You finally have the 10 to 0 COUNTDOWN, just as in Chapter 6 with the countdown PRINT statements, but in a much more efficient manner. The COUNTDOWN still goes by very quickly, however.

Stop the program (FCTN 4). What would happen if you changed line 250?

TRY THIS:

```
250 IF C > -1 THEN 215
260 STOP
```

Run the program. What is different with the countdown? Nothing! Why? You have changed the instruction slightly but not the meaning. Now it is saying, "If the value for the variable C is greater than −1 THEN go back, print C again, and subtract one from C." This is essentially the same instruction you had before, just said in a slightly different way. Here are two more modifications for you to make:

TRY THIS:

```
250 IF C = -1 THEN 300
260 GOTO 215
LIST 200-260
```

Will this accomplish the same thing again? Run it. Stop the program and run it again, watching the COUNTDOWN carefully. Study the output until you understand why there is no change when running the program. Remember that if line 250 is not true, the program will go on to the next line and do whatever it says.

You can see that there are a number of ways you can write this part of the program. Decide on the way in which you want to do it and make the changes. In the next chapter you will learn how to make the COUNTDOWN count in actual seconds, instead of quickly as it does now.

CHECKPOINT 8

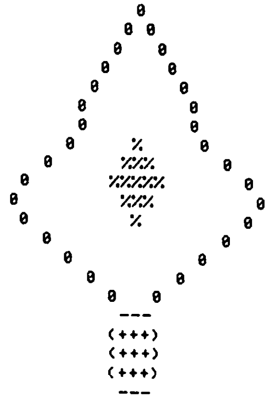
1. The symbol that stands for EQUAL TO is ______________.
2. The symbol that stands for NOT EQUAL TO is ______________.
3. The symbol that stands for LESS THAN is ______________.
4. The symbol that stands for GREATER THAN OR EQUAL TO is ______________.
5. The symbol that stands for LESS THAN OR EQUAL TO is ______________.
6. The symbol that stands for GREATER THAN is ______________.
7. The symbols referred to in this chapter are called ____________________.

TO CHECK YOUR ANSWERS SEE PAGE 145 IN APPENDIX A

PROGRAM UPDATE

```
9 REM    TITLE OF PROGRAM
10 PRINT "TAKING OFF WITH BASIC"
19 REM   NAME AND DATE
20 PRINT "PAT GOLDEN, APRIL 15, 1984"
200 REM    COUNTDOWN
210 LET C = 10
215 PRINT C
220 LET C = C -1
250 IF C = -1 THEN 300
260 GOTO 215
300 REM    HERE IS THE ROCKET
310 - 490    (THE CHARACTER GRAPHICS YOU
              DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW



9 FOR/NEXT Statements and Timers

Finally it is time to learn how to control the COUNTDOWN. Do you have any idea about how you can get a computer to do something slower than it normally does things? There is no instruction called SLOW DOWN on the TI. We have to be very sneaky, but there is a way. We simply instruct the computer to count to itself.

You have seen how quickly it prints 10 to 0 on the screen when C is used as a variable in a LET statement, and one is subtracted from it each time. There is a much easier way to tell the computer to count, using a FOR/NEXT loop. A **FOR/NEXT** loop is actually two statements. The first one is **FOR**, where we tell it to start counting. The **NEXT** is the second part which tells it to continue counting.

Let's have the computer count to 2000 and see how this works. In the FOR/NEXT statement we will use a variable called **T** (for **TIMER**) to tell the computer where to start counting. When you run the program, you will actually see how fast the computer can count.

TRY THIS:

```
10 FOR T = 1 TO 2000
20 PRINT T
30 NEXT T
99 END
RUN
```

In line 10 the word FOR appears, and in line 30 the word NEXT appears. The FOR and NEXT will never appear on the same line, but you must always have the same number of FORs and NEXTs in the program or you will get an error message.

You can see that the computer is actually counting to 2000. Line 10 starts it at 1, it moves to line 20 and prints the value for T (1) and then goes to line 30. NEXT T tells the computer to go back to line 10 and count again. It goes through this cycle until it has counted to 2000, then stops.

Delete line 20 (PRINT T) and run the program again. It is still counting to 2000, but it is not printing the numbers as it counts. Just like us, it can count to itself faster than it can write all the numbers out. That's why it takes a few seconds to tell you it's DONE. You can tell it to count to any number; the larger it is, the longer it takes.

LOAD TAKE OFF FILE NOW

Now you will see why the line numbering has not been following the "by tens" rule. The line numbers have been carefully chosen according to the TAKE OFF program plan. If there were no plan we would have to change many of the line numbers in each chapter to make room for new ones.

TRY THIS:

```
230 FOR T = 1 TO 1000
240 NEXT T
```

Don't run the program yet. In line 230, the instruction tells the computer to count to one. It then goes to line 240 which tells it to go back to 230 unless it has finished counting to 1000 already. The computer has a built-in check to see if it has counted to 1000 yet. Each time it gets to line 240, if it has not reached 1000 yet, it will go back and automatically add one more to the count.

Run this program and try to determine about how many seconds it takes the computer to count from 1 to 1000. You can do this by seeing how long it takes between printing each countdown number, since it counts to 1000 between each number.

After it prints C at line 215, it counts to 1000 (as it is told to do in the FOR/NEXT loop at lines 230 and 240) before getting to line 250. Here, C is checked to see if it has reached -1 yet.

You must decide what number you think it should count to in order to come as close to one second as possible—the count of 1 to 1000 is obviously more than a second. A real COUNTDOWN should have each number on the screen for about one second–right? Experiment by inserting smaller numbers to find which one comes closest to one second and edit line 230 accordingly.

Add a REM statement at line 225 regarding the TIMER loop.

CHECKPOINT 9

1. Every time a FOR statement appears in the program, there must also be a ____________ statement to go with it.
2. FOR/NEXT loops can be used as ____________ to slow down the program.
3. FOR/NEXT loops slow down the computer because it makes the computer ____________ to itself.

TO CHECK YOUR ANSWERS SEE PAGE 145 IN APPENDIX A

PROGRAM UPDATE

```
  9 REM     TITLE OF PROGRAM
 10 PRINT "TAKING OFF WITH BASIC"
 19 REM     NAME AND DATE
 20 PRINT "PAT GOLDEN, APRIL 15, 1984"
200 REM     COUNTDOWN
210 LET C = 10
215 PRINT C
220 LET C = C -1
225 REM     TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 IF C = -1 THEN 300
260 GOTO 215
300 REM     HERE IS THE ROCKET
310 - 490      (THE CHARACTER GRAPHICS YOU
                DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW

```
     ====================
    (                   )
   (                     )
  (                       )
    :::::::::::::::::::::
    :::::::::::::::::::::
   &                     &
  &                       &
 &                         &
   ::::::::::::::::::::::
    ::::::::::::::::::::
     ::::::::::::::::::
     x                x
    x                  x
   x                    x
  x                      x
 x                        x
<++++++++++++++++++++++++++>
 <++++++++++++++++++++++++>
   zzzzzzzzzzzzzzzzzzzzzz
```

10 FOR/NEXT Statements and Variables

FOR/NEXT statements can also be used in ways other than as timers. We can replace the LET statements in the program with a FOR/NEXT loop. Remember how FOR/NEXT statements cause the computer to count to itself? We can use this same idea to have it count from 10 to 0 in place of the LET statements.

FOR/NEXT statements are simply a way to make the computer do something the number of times we tell it to in the FOR statement. We can make it count to itself, print numbers on the screen, or do many other things. In this chapter we will have the FOR/NEXT loop take over the job of the LET statements.

TRY THIS:

```
10 FOR C = 1 TO 10
20 PRINT C
30 NEXT C
99 END
RUN
```

LOOK AT THIS:

```
10 LET C = 1
20 PRINT C
30 LET C = C + 1
40 IF C = 11 THEN 99
50 GOTO 20
99 END
```

Both programs print the numbers from 1 to 10 on the screen. However, the one on the left is shorter, making use of a FOR/NEXT loop. Study each program (enter the one on the right if you like) until you understand how they produce identical output.

We will now edit the TAKE OFF program, taking out the LET statements and inserting FOR/NEXT statements.

LOAD TAKE OFF FILE NOW

TRY THIS:

```
200 REM  COUNTDOWN LOOP
210 FOR C = 0 TO 10
215
220 PRINT C
250 NEXT C
260
LIST 200-260
```

This section of the program now looks like this:

```
200 REM    COUNTDOWN LOOP
210 FOR C = 0 TO 10
220 PRINT C
225 REM    TIMER LOOP
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
```

Look at the lines one at a time:

200 is a new REM statement;

210 tells the computer to begin counting, starting at 0;

220 tells the computer to PRINT C, which right now is 0;

225 is a REMark;

230 is the timer loop, which makes the computer count to 275 beginning with 1;

240 tells the computer to go back to 230 and count to 2; it will keep going from line 230 to line 240 to line 230 until it has finished counting to the number 275. When it finishes counting to 275, it will go on to line 250.

250 tells the computer to go back to line 210 (to the beginning of the C loop), where it was counting with the variable C. Since this is the first time it has gone back to 210, C is now equal to just one.

We have gone through lines 200 to 250 once. The computer will repeat this exact sequence, changing the value of C each time, until C is equal to 10, as specified in line 210. Each time it gets to line 230 it will count from 1 to 275 before getting to line 250 and returning to the beginning of the C loop at line 210. (This is a time delay of about 1 second). When C is equal to 10 it will go on to the instruction following line 250, because it has finished the C FOR/NEXT loop.

This has been a difficult concept. The entire timer (T) FOR/NEXT loop is actually inside the (C) FOR/NEXT loop. Each time the T loop is reached, the computer begins counting from 1 to 275 again, always starting over at 1. It is called a **NESTED LOOP** since it is executed inside another loop. This is completely legal as long as the second (minor) loop is completely within the main loop. If you are unsure of your understanding, go back and read through it again. Once you grasp this FOR/NEXT statement, you have reached an important stage in programming.

Run the program after you have made sure there are no BUGS (errors) in it. Is everything working correctly? Well, almost! We are back to counting from 0 to 10 again, right? Let's see how we can fix this problem. Unless you tell it otherwise, the computer assumes that when you ask it to count, you want it to count by ones, going forward. When it is given the instruction:

```
FOR C = 1 TO 10
NEXT C
```

it counts from 1 to 10 by ones. If we give it the instruction:

```
FOR C = 100 TO 200
NEXT C
```

it counts from 100 to 200 by ones. How would we get it to count by twos if that is what we wanted it to do? Study these lines:

```
FOR C = 100 TO 200 STEP +2
NEXT C
```

This tells the computer to count from 100 to 200 by twos instead of ones. STEP +2 tells it exactly how to count.

Our problem happens to be that we want it to count BACKWARD by ones from 10 to 0. Can you figure out how to make it do that?

TRY THIS:

```
210 FOR C = 10 TO 0 STEP -1
```

Run this program and see if it works. STEP -1 instructs it to count backward, by ones, from 10 to 0. List the program and study it to understand exactly what you have done. Just for fun, make it count backward from 100 to 10 by tens. Try some other combinations. After you are satisfied that you understand this concept, edit line 210 one final time so that it counts correctly from 10 to 0 for the COUNTDOWN.

CHECKPOINT 10

1. In timer loops the NEXT must come right after the FOR statement. Is this true when used for other purposes besides timers? (Hint: Check the UPDATE.) ______________________________
2. What does the computer do when it has finished a FOR/NEXT loop?
____

3. In a FOR/NEXT loop as follows:

   ```
   FOR N = 5 TO 60
   PRINT N
   NEXT N
   ```

   What will the value of N be on the fourth time around the loop? ________

4. Write a four line program (including an END statement) to make the computer count and print the numbers 150 to 300 by threes.

   ____
   ____
   ____
   ____

5. Write a four line program (including an END statement) to make the computer count and print the numbers -50 to -150 by sevens.

   ____
   ____
   ____
   ____

6. Unless you tell it to do otherwise, the computer will count by ________ in FOR/NEXT loops.
7. Explain why the FOR/NEXT statement is referred to as a "loop". ________
   ____

TO CHECK YOUR ANSWERS SEE PAGE 145 IN APPENDIX A

PROGRAM UPDATE

```
9 REM    TITLE OF PROGRAM
10 PRINT "TAKING OFF WITH BASIC"
19 REM    NAME AND DATE
20 PRINT "PAT GOLDEN, 1984"
200 REM    COUNTDOWN LOOP
210 FOR C = 10 TO 0 STEP -1
215 PRINT C
225 REM    TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
300 REM    HERE IS THE ROCKET
310-490        (THE CHARACTER GRAPHICS YOU
                DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW

```
        #
       # #
      #   #
     #     #
    #       #
   #         #
  #           #
 #             #
#               #
######     ######
     i     i
     i     i
     i     i
     i     i
     #######
```

11 TAB

There are two things you can do to the countdown to make it just a little bit better. One has to do with where the countdown appears on the screen. The other deals with the numbers staying on the screen during the countdown and only going away when the object takes off. Wouldn't it be neater if only one number appeared on the screen at a time? Let's clean up these two small problems.

LOAD TAKE OFF FILE NOW

LIST line 220. It should read:

```
220 PRINT C
```

Unless we tell the computer otherwise, it prints information beginning at the bottom left corner of the screen. That is why the countdown appears in that area. There is an extra instruction we can add to the PRINT statement that will make the computer center the information by moving it to the right. It is called TAB.

On a typewriter there is a key called TAB. You can set it so that when you press it, the typewriter carriage moves over to where you want to begin typing. The TI-99/4A computer does not have this key, but you can add such an instruction to the PRINT statement. Here is an example:

```
220 PRINT TAB(10); C
```

Line 220 tells the computer to move 10 spaces to the right and then print the value for C. The semicolon (;) is a required separator between the

TAB instruction and the variable C. Edit line 220 in the program so that it is exactly like the one shown above. LIST the line to check for errors, and then RUN it. Do you like this location for the countdown? If you want it more to the right or left, change TAB(10) to a greater or smaller number until it is exactly where you want it.

To solve the second problem (each countdown number staying on the screen), we will make use of the command CALL CLEAR that has already been introduced. It is going to be of great use now. If you are careful with the line number for the CALL CLEAR statement, it will cause each number of the countdown to disappear before the next one appears. LIST program lines 200-250. Where do you think you should insert the CALL CLEAR line? Actually, there is more than one place which would work.

TRY THIS:

```
215 CALL CLEAR
RUN
```

Does this work? RUN the entire program again. Experiment with other placements of CALL CLEAR and see what the results are.

Here are a few suggestions for tidying up the program before going to the next chapter. You might want to put a CALL CLEAR at line 2 to start the program off with a clean screen. There may be other places where you would like to insert CALL CLEAR as well. Another thing you could do would be to add a timer loop for the title page so that it stays on the screen briefly (see lines 30-31 in the PROGRAM UPDATE). Go ahead and fix the program in any way you desire. In the next few chapters you will be adding color, sound, a new title page, and making some decisions regarding the readiness of the object for takeoff.

CHECKPOINT 11

1. What command can you insert into the program, along with a line number, that will produce a new, clean screen each time it is encountered by the computer? ______________________
2. By using the word ____________, along with a number in parentheses, you can alter the placement of information on the screen.
3. Write a statement that puts the word BANANA 15 spaces to the right on the screen: ______________________

TO CHECK YOUR ANSWERS SEE PAGE 146 IN APPENDIX A

PROGRAM UPDATE

```
  2 CALL CLEAR
  9 REM  TITLE OF PROGRAM
 10 PRINT "TAKING OFF WITH BASIC"
 19 REM  NAME AND DATE
 20 PRINT "PAT GOLDEN, APRIL 15, 1984"
 30 FOR T = 1 TO 1000
 31 NEXT T
200 REM  COUNTDOWN
210 FOR C = 10 TO 0 STEP -1
215 CALL CLEAR
220 PRINT TAB(14); C
225 REM  TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
300 REM  HERE IS THE ROCKET
310-490           (THE CHARACTER GRAPHICS YOU
                   DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW

12 Color

The TI microcomputer is unique because it has so many colors that it can display. How good the colors are depends on the quality of your monitor or television the TI is connected to. Each color is assigned a code number. Below is a list of these COLOR CODES for the TI:

COLOR CODES

Transparent	1	Medium red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

You may not even know some of these colors. You will get a chance to experiment and see what they look like. The instruction telling the computer to change the color of the screen looks like this:

```
CALL SCREEN (11)
```

The number in parentheses is one of the color codes, dark yellow. Changing the screen color is easy. However, a timer loop is needed in order to see the new screen color for a while. This is how a FOR/NEXT loop is used:

TRY THIS:

```
10 CALL CLEAR
30 CALL SCREEN (14)
40 FOR T = 1 TO 500
50 NEXT T
99 END
RUN
```

This program clears the screen, then changes the screen color to code 14, magenta. The timer loop causes the screen to remain that color for a count of 500 before returning to blue. Do you want to see how all the colors look? Let's try a nested loop to do a demonstration. Edit the program to change line 30 and add lines 20 and 60.

TRY THIS:

```
20 FOR C = 1 TO 16
30 CALL SCREEN (C)
60 NEXT C
```

We have added a FOR/NEXT loop assigning variable C the numbers 1 through 16 since there are 16 colors on the chart. In line 30 these numbers will be substituted for the C, and each different color will stay on the screen for a count of 1 to 500. Run the program and see how it works.

Using the CALL SCREEN instruction makes the TAKE OFF program more interesting and attractive.

LOAD TAKE OFF FILE NOW

TRY THIS:

```
199 CALL SCREEN (7)
RUN
```

What did it do? The color changed to dark red, color code 7, before the countdown. Any time you want the screen color to change just add another line number with a CALL SCREEN statement and the color code number in parentheses. Write a line to make the screen change to dark green for the TAKE OFF. Use line number 299.

Don't worry if the colors aren't quite right. If you want, you can adjust the tint and color knobs on the monitor to improve the quality.

Notice that the natural blue color of the screen is returned after the program finishes running. Add color anywhere you want in the program and experiment with different colors.

CHECKPOINT 12

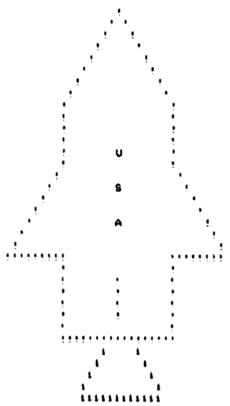
1. How many different colors are available on the TI? ____________
2. What is the statement used in programs to give the screen color? ________________________.
3. How do you specify what color you want the screen to be in the color statement? ____.
4. Where can you put color statements in the program? ____________

TO CHECK YOUR ANSWERS SEE PAGE 146 IN APPENDIX A

PROGRAM UPDATE

```
  2 CALL CLEAR
  9 REM  TITLE OF PROGRAM
 10 PRINT "TAKING OFF WITH BASIC"
 19 REM  NAME AND DATE
 20 PRINT "PAT GOLDEN, APRIL 15, 1984"
 30 FOR T = 1 TO 1000
 31 NEXT T
199 CALL SCREEN (7)
200 REM  COUNTDOWN
210 FOR C = 10 TO 0 STEP -1
215 CALL CLEAR
220 PRINT TAB(14); C
225 REM  TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
299 CALL SCREEN (13)
300 REM  HERE IS THE ROCKET
310 - 490          (THE CHARACTER GRAPHICS YOU
                    DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW



13 String Variables

What in the world are STRING VARIABLES? So far, when we have defined variables, we have chosen a variable name for a NUMERIC (number) value. Often in programming we want to use variables with words or other characters on the keyboard. They are called **STRING VARIABLES.**

STRING VARIABLES are not really variables at all since they do not usually represent numbers. However, they are handled the same in the computer's memory as long as you warn the computer that a STRING VARIABLE is going to be defined. The **dollar sign ($)** is used for the warning. Quotation marks must also surround the word. In the line

```
10 LET CITY$ = "PHOENIX"
```

the computer finds an empty memory box for a STRING VARIABLE named CITY$ and places PHOENIX in it. Just like in the line

```
30 LET A = 20
```

the computer finds an empty box for a NUMERIC VARIABLE named A and places 20 in it.

CITY$ is the name of the STRING VARIABLE. The $ attached to it in the LET statement is the warning the computer needs. If the quotation marks are left out, the computer will be confused since you warned it to expect a word but then didn't use quotation marks. This will produce an error message.

Here is an example of a short program using a STRING VARIABLE.

TRY THIS:

```
10 LET CITY$ = "PHOENIX"
20 PRINT CITY$
99 END
RUN
```

The output for this program would be

```
PHOENIX
```

since the computer goes to its memory and prints whatever is in the memory box named CITY$. In the line

```
30 LET A = 20
```

the computer saw the A and knew you would enter a number because the A did not have a $ as part of its name. In line 10 it saw the variable named CITY$, so it expected a word in quotation marks after the equal sign.

What do you think would happen if you changed line 10 above to read

```
10 LET CITY$ = PHOENIX
```

The computer would give the error message

```
STRING-NUMBER MISMATCH IN 10
```

because CITY$ tells it to look for a word in quotation marks, and the quotation marks are missing.

TRY THIS:

```
30 LET A = "20"
```

You will get the same error message because you are telling the computer to expect a number and instead there are quotation marks. Remember that the computer does not even look inside the quotes so there is no way it can know there is a number there. It is legal to put numbers inside the quotation marks but isn't done very often since a numeric variable name can be used for that purpose.

One way you can remember what the $ is for is that it looks something like the letter S. When you see a variable such as CITY$, we call it CITY STRING. The variable N$ would be called N STRING.

Let's try putting a string variable into the program.

LOAD TAKE OFF FILE NOW

TRY THIS:

```
5 LET TITLE$ = "TAKING OFF WITH BASIC"
10 PRINT TITLE$
```

After making the above changes, run the program to see if the changes made any difference in the way it runs. It should be exactly the same as it was before the editing. Make similar changes for line 20 in the program. It will look something like this:

```
6 LET NAME$ = "PAT GOLDEN"
7 LET DATE$ = "APRIL 15, 1984"
20 PRINT NAME$,DATE$
RUN
```

Notice there are numbers in the STRING VARIABLE in Line 7. When words and numbers are used together it is usually easier to define them altogether with one STRING VARIABLE name.

NOTE: When you list the entire program it will now be too long to show on the screen all at the same time. To stop it in the middle of the listing, use FCTN 4, or just give it certain line numbers to list.

CHECKPOINT 13

1. What is a STRING VARIABLE? ____________________
   ____
2. Write a LET statement using a STRING VARIABLE. ____________
   ____
3. What is the opposite of a STRING VARIABLE? ____________
4. How would you say this variable in English? NAME$ ____________
5. If you want to have a variable to represent a street name, what might you call it? ____________
6. What is wrong with each of the following statements?

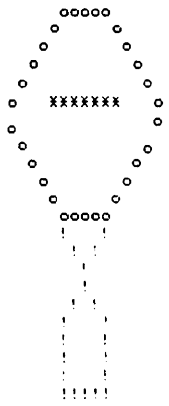
```
    STATEMENT                ERROR
10 LET H = "HEIGHT"  ____________________
20 LET TIME$ = 3  ______________________
30 LET NAME$ = MATT  ___________________
40 LET A$ = "1962"  ____________________
50 LET PHONE = "5558155"  ______________
```

TO CHECK YOUR ANSWERS SEE PAGE 146 IN APPENDIX A

PROGRAM UPDATE

```
  2 CALL CLEAR
  5 LET TITLE$ = "TAKING OFF WITH BASIC"
  6 LET NAME$ = "PAT GOLDEN"
  7 LET DATE$ = "APRIL 15, 1984"
  9 REM  TITLE OF PROGRAM
 10 PRINT TITLE$
 19 REM NAME AND DATE
 20 PRINT NAME$,DATE$
 30 FOR T = 1 TO 1000
 31 NEXT T
199 CALL SCREEN (7)
200 REM COUNTDOWN
210 FOR C = 10 TO 0 STEP -1
215 CALL CLEAR
220 PRINT TAB(14); C
225 REM TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
299 CALL SCREEN (13)
300 REM HERE IS THE ROCKET
310 - 490         (THE CHARACTER GRAPHICS YOU
                   DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW



14 INPUT

A neat thing about string variables is that they allow you to INTERACT with the computer. The word **INTERACT** means that whoever runs the program has a chance to put some things into it as it is running. The user can be permitted to INTERACT with the computer by the use of **INPUT** statements. A skillful programmer can make the user feel like a real part of the programming process. An INPUT statement is a little tricky.

TRY THIS:

```
10 PRINT "WHAT IS YOUR NAME"
20 INPUT NAME$
30 PRINT NAME$
99 END
```

Notice that in line 30 we have instructed the computer to PRINT NAME$ but have not written a LET statement defining what NAME$ is. That is because line 20 allows whoever runs the program to decide what will be in the memory location for NAME$. You must run the program to understand how you can interact with the computer through an INPUT statement.

TRY THIS:

```
RUN
```

The output is what you entered in answer to the question WHAT IS YOUR NAME?

Line 10 prints on the screen WHAT IS YOUR NAME

Line 20 instructs the computer to print a question mark and waits for the user to answer the question. Type in your first name and press ENTER. It now assigns whatever word you entered to an empty memory box called NAME$. Whenever the computer comes to an INPUT statement, it puts a question mark on the next line and waits for the user to answer the question.

Line 30 tells it to print what is in the memory box called NAME$, so your name will appear on the screen.

Let's expand this short program to include a numeric variable as INPUT. Remember that numeric variables do not have a $ in the name.

TRY THIS:

```
40 PRINT "WHAT IS YOUR AGE"
50 INPUT AGE
60 PRINT AGE
```

AGE will be the name of the memory location where the number input for age is placed.

Run the program and answer each question, pressing ENTER after you do so. The output for this 7 line program (if the name is PAT and the age is 20) is:

```
PAT
20
```

As mentioned earlier, even though there were no question marks after the questions in lines 10 and 40, they do appear on the screen after the questions are asked. The computer, as soon as it comes to an INPUT statement, automatically prints a question mark. INPUT to the computer means that a question has been asked, so it takes care of the question mark for you.

LOAD TAKE OFF FILE NOW

It is time to edit the takeoff program again using the INPUT statement. Lines 5 and 10 look something like this:

```
5 LET TITLE$ = "TAKING OFF WITH BASIC"
10 PRINT TITLE$
```

Add line 4 and retype line 5 as follows:

```
4 PRINT "WHAT DO YOU WANT TO CALL THIS PROGRAM"
5 INPUT TITLE$
```

Line 10 remains the same. Notice that TITLE$ is still the name of the program, no matter what the user enters. It might be TAKE OFF OF A HELICOPTER or anything else the user enters. Run the program. What you enter for the title will be words so it must have a STRING VARIABLE name.

As you have seen, numeric variables can also be used with INPUT statements. In the lines

```
40 PRINT "WHAT IS THE DATE"
45 INPUT DATE
```

the computer expects a number since DATE is a numeric variable name.

Use INPUT statements in lines 20 - 70 to allow the user to input his or her own name and current date. To make room for this, you must delete the timer in lines 30 and 31 and place it at lines 80 and 90.

Run and list the program. It is looking pretty impressive!

CHECKPOINT 14

1. What does the computer do when it comes to an INPUT statement in a program? ____________________________
2. What kind of statement should come before each INPUT statement? ____________________
3. What does it mean when we say that the computer interacts with the user? ____________________________
____
4. If you ask a question that will have a numeric answer, the INPUT statement must have a ____________ variable name.

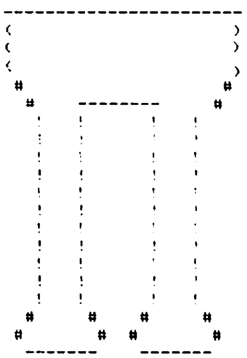
TO CHECK YOUR ANSWERS SEE PAGE 147 IN APPENDIX A

PROGRAM UPDATE

```
 2 CALL CLEAR
 4 PRINT "WHAT DO YOU WANT TO CALL THIS PROGRAM"
 5 INPUT TITLE$
 9 REM  TITLE OF PROGRAM
10 PRINT TITLE$
15 FOR T = 1 TO 500
16 NEXT T
17 CALL CLEAR
19 REM  NAME AND DATE
20 PRINT "WHAT IS YOUR NAME"
```

```
 25 INPUT NAME$
 30 PRINT "WHAT IS THE MONTH"
 35 INPUT MONTH$
 40 PRINT "WHAT IS THE DATE"
 45 INPUT DATE
 50 PRINT "WHAT IS THE YEAR"
 55 INPUT YEAR
 60 PRINT NAME$
 70 PRINT MONTH$;DATE;",";YEAR
 80 FOR T = 1 TO 1000
 90 NEXT T
199 CALL SCREEN (7)
200 REM  COUNTDOWN
210 FOR C = 10 TO 0 STEP -1
215 CALL CLEAR
220 PRINT TAB(14); C
225 REM  TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
299 CALL SCREEN (13)
300 REM   HERE IS THE ROCKET
310-490          (THE CHARACTER GRAPHICS YOU
                  DESIGNED ON THE GRID)
495 GOTO 310
999 END
```

SAVE TAKE OFF FILE NOW



15 Sound

You can surely guess what this chapter is about. The TI has a wide variety of sounds it can make. In order to learn how to use them, there are some vocabulary words that must be defined.

CALL SOUND

This is the statement used to warn the computer to expect further instructions about exactly what sound you want played. In addition to the CALL SOUND statement, there are three numbers that must be included in parentheses. The format of the statement looks like this:

```
CALL SOUND(1000,262,5)
```

DURATION

The first number inside the parentheses refers to how long you want the sound to be played. In the example above, the sound is played for a length of 1000. Each CALL SOUND statement must have the DURATION number immediately following the left parentheses. The number 4250 is the longest number that is allowed for DURATION; the shortest is 1. A DURATION of 1000 is about 2 seconds.

TONE

The note you want to hear is called the TONE and is the second number inside the parentheses. There is a code number for each note the computer can play. Appendix B lists the numbers for many musical notes. A list of Middle C through High C is shown below. In the line on page 65, 262 refers to Middle C (check this with the chart). There must be no spaces between the numbers and commas inside the parentheses.

VOLUME

Loudness is determined by the third number inside the parentheses. In the example, the loudness number is 5. The loudest VOLUME is the number 0, and the softest is the number 30 (which you may have trouble hearing at all!). Remember that before you type the VOLUME number, you must have a comma.

TONE NUMBERS FOR MIDDLE C THROUGH HIGH C

262	MIDDLE C
294	D
330	E
349	F
392	G
440	A
494	B
523	HIGH C

If we wanted to write lines to play the notes Middle C, E, G and High C, they would look like this:

TRY THIS:

```
10 CALL SOUND(1000,262,5)
20 CALL SOUND(1000,330,5)
30 CALL SOUND(1000,392,5)
40 CALL SOUND(1000,523,5)
99 END
```

These lines each have the CALL SOUND statement, parentheses, a duration number of 1000, the number of each different note, and a volume level of 5. Run the program. If you don't hear anything, turn up the volume on the monitor.

We could make each note shorter and softer than the last by editing the lines.

TRY THIS:

```
10 CALL SOUND(700,262,10)
20 CALL SOUND(500,330,15)
30 CALL SOUND(250,392,20)
40 CALL SOUND(100,523,25)
```

Run the program again. Notice how the duration number keeps getting smaller, and we made it softer by increasing the volume number. The tone numbers stayed the same as before.

Now it's time to put some sound into your program.

LOAD TAKE OFF FILE NOW

How about putting in the whole scale to begin with? We'll use a duration of 500 and a volume of 5. The scale will include every note from Middle C to High C. The first line will look like this:

```
510 CALL SOUND(500,262,5)
```

Using line numbers 520 through 580, write the other seven program lines. When you are finished, add a REM statement at line 500 explaining what will be happening in lines 510-580.

After you enter these lines and check them for errors, run the program and see how it sounds. To check it out you will have to omit the GOTO statement that causes your object to continue taking off (line 495). Otherwise, it will never get to the CALL SOUND instructions.

TRY THIS:

```
495        (Press ENTER)
```

The GOTO will be reinserted later.

CHECKPOINT 15

1. What are the two words necessary in the statement that instructs the computer to play a musical note? ______________________
2. What does DURATION mean? ______________________
3. What does TONE mean? ______________________
4. What does VOLUME mean? ______________________
5. What are the numeric limits for VOLUME? ______________________
6. What are the numeric limits for DURATION? ______________________
7. How do you indicate what TONE you want played? ______________________

____

8. Inside the parentheses, what number comes first? ____________ What number comes second? ____________ Third? ____________

TO CHECK YOUR ANSWERS SEE PAGE 148 IN APPENDIX A

PROGRAM UPDATE

```
  2 CALL CLEAR
  4 PRINT "WHAT DO YOU WANT TO CALL THIS PROGRAM"
  5 INPUT TITLE$
  9 REM  TITLE OF PROGRAM
 10 PRINT TITLE$
 15 FOR T = 1 TO 500
 16 NEXT T
 17 CALL CLEAR
 19 REM  NAME AND DATE
 20 PRINT "WHAT IS YOUR NAME"
 25 INPUT NAME$
 30 PRINT "WHAT IS THE MONTH"
 35 INPUT MONTH$
 40 PRINT "WHAT IS THE DATE"
 45 INPUT DATE
 50 PRINT "WHAT IS THE YEAR"
 55 INPUT YEAR
 60 PRINT NAME$
 70 PRINT MONTH$,DATE;",";YEAR
 80 FOR T = 1 TO 1000
 90 NEXT T
199 CALL SCREEN (7)
200 REM  COUNTDOWN
210 FOR C = 10 TO 0 STEP -1
215 CALL CLEAR
220 PRINT TAB(14); C
225 REM  TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
299 CALL SCREEN (13)
300 REM  HERE IS THE ROCKET
310- 490          (THE CHARACTER GRAPHICS YOU
                   DESIGNED ON THE GRID)
500 REM  SOUND EFFECTS
510 CALL SOUND(500,262,5)
520 CALL SOUND(500,294,5)
```

```
530 CALL SOUND(500,330,5)
540 CALL SOUND(500,349,5)
550 CALL SOUND(500,392,5)
560 CALL SOUND(500,440,5)
570 CALL SOUND(500,494,5)
580 CALL SOUND(500,523,5)
999 END
```

SAVE TAKE OFF FILE NOW

```
-------------
      (
      )
    oooooo             /////
   o      o            /   /
  o        o------------   /
  o        o               /
  o        o----------------
   o      o
    o    o
     oooo
       !
  -------------
```

16 Subroutines

A **SUBROUTINE** is a self-contained set of instructions that you may put into a program and cause to be run whenever you want without writing it over each time. For example, remember the sound section you wrote in the last chapter? We can use it in different places in the program as a SUBROUTINE but it only needs to appear in the program once.

LOAD TAKE OFF FILE NOW

Two statements are needed to put a SUBROUTINE into operation: **GOSUB** and **RETURN**. When the computer comes to a GOSUB statement, it might look like this:

```
205 GOSUB 510
```

This is literally saying, "Go run the instructions that begin at line 510." The computer jumps to line 510, ignores everything in between, and executes line 510 and everything that follows that line until it reaches the RETURN statement:

```
590 RETURN
```

When it sees RETURN, it goes back to the instruction immediately following the GOSUB line that sent it.

TRY THIS:

```
205 GOSUB 510
495 GOTO 310
590 RETURN
RUN
```

Hooray! The TAKE OFF is back intact. We are using the musical scale as a SUBROUTINE now. After it finished playing lines 510-580, line 590 RETURNs it to line 210.

Let's put the scale at the beginning of the program just for fun.

TRY THIS:

```
3 GOSUB 510
```

The RETURN at line 590 will automatically send it back to line 4 (since line 4 is the next line following 3).

Here is a problem for you. It doesn't make much sense to have a musical scale play before the TAKE OFF. Why not experiment and see if you can make a better sound for the TAKE OFF? Can you simulate an explosion or the sound of a helicopter? You might need more TONES. The entire list of tones can be found in APPENDIX B. Supplement 16 also has some information about special noises you might find helpful.

CHECKPOINT 16

1. What is a SUBROUTINE? ______________________________

   ____

2. What two statements are necessary as part of the instructions to use a SUBROUTINE? ______________________________

   ____

3. What must come after the GOSUB statement on the same line?

   ____

   What does it indicate? ______________________________

   ____

4. How many times can you use one SUBROUTINE in a program?

   ____

TO CHECK YOUR ANSWERS SEE PAGE 148 IN APPENDIX A

PROGRAM UPDATE

```
 2 CALL CLEAR
 3 GOSUB 510
 4 PRINT "WHAT DO YOU WANT TO CALL THIS PROGRAM"
 5 INPUT TITLE$
 9 REM  TITLE OF PROGRAM
10 PRINT "TAKING OFF WITH BASIC"
15 FOR T = 1 TO 500
16 NEXT T
17 CALL CLEAR
```

```
 19 REM  NAME AND DATE
 20 PRINT "WHAT IS YOUR NAME"
 25 INPUT NAME$
 30 PRINT "WHAT IS THE MONTH"
 35 INPUT MONTH$
 40 PRINT "WHAT IS THE DATE"
 45 INPUT DATE
 50 PRINT "WHAT IS THE YEAR"
 55 INPUT YEAR
 60 PRINT NAME$
 70 PRINT MONTH$,DATE;",";YEAR
 80 FOR T = 1 TO 1000
 90 NEXT T
199 CALL SCREEN (7)
200 REM  COUNTDOWN
205 GOSUB 510
210 FOR C = 10 TO 0 STEP -1
215 CALL CLEAR
220 PRINT TAB(14); C
225 REM  TIMER
230 FOR T = 1 TO 275
240 NEXT T
250 NEXT C
299 CALL SCREEN (13)
300 REM  HERE IS THE ROCKET
310-490            (THE CHARACTER GRAPHICS YOU
                   DESIGNED ON THE GRID)
495 GOTO 310
500 REM   SOUND EFFECTS
510 CALL SOUND(500,262,5)
520 CALL SOUND(500,294,5)
530 CALL SOUND(500,330,5)
540 CALL SOUND(500,349,5)
550 CALL SOUND(500,392,5)
560 CALL SOUND(500,440,5)
570 CALL SOUND(500,494,5)
580 CALL SOUND(500,523,5)
590 RETURN
999 END
```

SAVE TAKE OFF FILE NOW

```
     ***************
    *               *
   *                 *
  *                   *
 *   #################  *
*     # # # # # # #      *
* ##################### *
*   # # # # # # # # #   *
*  ###################  *
 *                     *
  *                   *
   *                 *
    *               *
     *      "      *
      *    """    *
       *   """   *
        $$$$$$$$$
           * *
          *   *
          *   *
          *   *
          *****
          *****
```

17 Finishing Touches

The program you have been creating for 16 chapters is now near completion. Just as you reread a letter you have written to make certain it says just what you want, programmers do the same with their collection of instructions. In reviewing the TAKE OFF program, here are some things to check for:

REM statement. It is easy to put enough REM statements in a program so anyone could read it and follow what the computer is supposed to do. To make REMs easy to see, you can surround them with several asterisks:

```
10 REM ***INITIALIZE VARIABLES***
```

END statement. Make sure you have included an END statement in the program. This should be done early in your writing. On some computers, you cannot save a program if there is no END statement.

Title Page. Run the TAKE OFF program and look at the title page. Using color, formatting, and other statements you have learned, make the title page look impressive. You might even want to design some special characters for your own name. Supplements 10 and 12 have some suggestions that might be helpful.

Color. Insert color throughout the program to make it more interesting.

Sound. Using sound in certain parts of a program can also be very effective. Special tunes before the title page appears, at the time of the take off, and as a finale are appropriate placements.

Interaction. Using input, you might want to allow the user to be more in control of the program. For example, ask the user to decide how many times the object will take off, what colors to use, or what tune to play.

You might ask the user to input certain personal information and then use it in the program.

As you can see, there are an endless number of changes and additions you could make to this program. What you have designed is unique. It reflects your personality as well as your understanding of the concepts you have learned. It is also very interesting to look at others' programs to see how they have written a program similar to yours. You will find many differences, but each one is still correct.

If a printer is available to you, print out a listing of the final TAKE OFF program so that some day in the future you can look at your first major program. Most likely, you will be amazed at how well you did. You will probably be glad that you have many REM statements throughout it.

Hopefully your first experiences at writing a computer program were challenging, fun, and satisfying. Now you can TAKE OFF on your own!

MAKE FINAL CHANGES
AND SAVE TAKE OFF FILE NOW

Introduction To Supplemental Chapters

The seventeen supplemental chapters are for those of you who are in one of the following categories; advanced users who already know the material in Chapters 1-17; or those of you who have successfully completed the first seventeen chapters and want to continue with more advanced BASIC concepts.

Some of the exercises in this section will suggest additions or updates to the TAKE OFF program. However, the main goals of the supplemental chapters are to offer enrichment material, engrain the BASIC concepts presented in chapters 1-17, and encourage the generalization of the BASIC language to applications which are not related to the TAKE OFF program. This section will also serve to round out your knowledge of BASIC programming and increase your comfort level with the TI-99/4A.

NOTE: If you do each supplement after completing each regular chapter, be sure to save the TAKE OFF file before beginning the supplement.

SUPPLEMENT 1 Hardware and Initializing

There is a variety of equipment or hardware which you can purchase for the TI-99/4A. Some of it will be described here, along with instructions for making preparations for using it to save the work you will be doing as you go through this manual. First, a few more vocabulary words will be helpful.

Floppy Disk

Also called a **DISKETTE** or **DISK,** it is about the shape of a 45 record, very flexible, and comes in a square paper envelope. You can buy FLOPPY DISKS with programs (software) already on them, such as games or educational lessons. You may also put your own program on a FLOPPY DISK so that you can save it. The FLOPPY DISK is in a permanent envelope to protect it from scratches, dust, or any other things that can ruin it. It must be treated carefully and should never be placed on top of something that is heated, like the television, or left in hot places. Once a DISK is ruined, it cannot be used again. A blank disk costs about $3 to $5.

Disk Drive

This is a piece of hardware which resembles a rectangular box and is labeled **DISK DRIVE.** When the switch on the back is pressed on its left side, it is in the OFF position. The door on the front can be opened to accept a FLOPPY DISK.

Disk Controller

A silver rectangular box labeled **DISK CONTROLLER** is the piece of hardware necessary to run the disk drive. The switch is on the front right. The OFF position is when the switch is pushed toward the left.

Command Module

A black plastic hand-sized item similar to a cassette tape, the **COMMAND MODULE** has software stored inside it. Like a diskette, it may be a game or educational lesson. The module entitled DISK MANAGER is necessary when you are working with a floppy disk. The DISK MANAGER has information on it that helps you operate the computer.

Peripheral Expansion System

You may have purchased a **PERIPHERAL EXPANSION SYSTEM** for your TI-99/4A which combines the disk drive and disk controller. The word peripheral simply means "something on the outside". Expansion refers to adding hardware to your computer equipment. A System is a collection of units. In this case, it includes the disk drive, disk controller, and can also include extra memory.

Cassette Tape Player

A **tape recorder** can also be used to save and load software on the TI. It is less expensive than a disk drive, but also is slower at loading and saving. It is, however, the only alternative to a disk drive or Peripheral Expansion System and is very nice to have if a drive is not in the budget.

Cassette Tape

For TI systems with tape recorders, purchase **tapes** of good quality, and not longer than C-60 for saving and loading programs. Tapes do not hold as much information as disks and are not as durable.

File

Once a program is written and placed on a floppy disk or cassette tape, it is called a **FILE**. Each FILE has its own name so that you can find what you want very easily.

Initialization

This is the process of preparing a new, blank floppy disk to save programs on. If a disk is not **INITIALIZED**, you cannot save programs on it. It is not necessary to initialize tapes, however. Each microcomputer's memory is a little different so that you cannot take a disk or tape you made on the TI and use it on a different brand of microcomputer.

Initializing a Floppy Disk

The TI microcomputer must be turned on in a specific way or it will not operate properly. Following are the instructions for initializing a new, blank disk, along with the directions for turning on the different pieces of hardware. If you have a Peripheral Expansion System, see the information in parentheses. You should follow these instructions if you plan to save the program to be written in chapters 1-17.

1. Make sure that all switches are OFF on the KEYBOARD, DISK DRIVE, and DISK CONTROLLER (or Peripheral Expansion System).
2. Hold the new disk with your right hand, thumb on the label. (The fingers on your right hand will be on the label.)
3. Open the disk drive door.
4. Gently slip the disk into the drive so that the label is facing up and is the last to go in. (The label should be facing right.)
5. Close the disk drive door.
6. Hold the DISK MANAGER command module in your right hand with the label facing your palm.
7. Find the area to the right of the TI-99/4A label on the top of the keyboard. Firmly push the DISK MANAGER command module into this area.

8. Turn the switches ON in the following order (ORDER IS VERY IMPORTANT):

   A. DISK DRIVE
   B. DISK CONTROLLER
   C. KEYBOARD
   D. MONITOR
   E. (PERIPHERAL EXPANSION SYSTEM)

You could destroy a disk if you do not follow the correct order.

9. The screen should now say

   TEXAS INSTRUMENTS HOME COMPUTER

10. If the screen remains blank, turn everything off and repeat steps 6 and 7. If the screen still remains blank, start over with Step 1.
11. Press any key to go to the next screen.
12. Press the number for DISK MANAGER and wait for the next screen.
13. Press the number for DISK COMMANDS and press ENTER.
14. Press the number for INITIALIZE NEW DISK and press ENTER.
15. Press the number for MASTER DISK. This indicates that your new floppy disk is in the disk drive (or the first one if you have more than one).
16. Type in the name of your new disk. For example, you might want to call it "PATSDISK1". You can choose whatever name you want. Limit your name to less than 15 letters or numbers, and leave NO spaces inside the quotation marks. Press ENTER.
17. Press the letter Y and press ENTER.
18. Hold down the shift key and press the V briefly. The disk drive will now go on and the disk will be initialized. It will take about two minutes to do so.
19. You have now initialized the disk. Remove it from the drive until you are ready to save something on it or load something from it. Label it with a felt pen. (A pencil or hard point pen will damage the disk.) Replace it in its protective envelope and keep it in some kind of container.
20. Instructions are given in Supplement 3 for saving and loading files.

SUPPLEMENT 2
Flow Charting

Programming style is important to develop so that programs are easy to read and follow by anyone looking at them. When you write programs you should not sit down at the computer and begin to write lines of instructions. Before you ever get to the microcomputer, you should know what your goal is and have the entire program largely written. There is a method of organizing a program so that it can be programmed more easily called **FLOW CHARTING.**

Steps to Planning a Program

There are 7 steps to planning a program or part of a program when FLOW CHARTING is used.

1. Decide what the problem is you want to solve or what you want the computer to do.
2. Sketch out a flow chart showing the order you want the computer to follow in carrying out your plan.
3. Write program instructions for each box on the flow chart you have drawn.
4. Enter the instructions into the computer.
5. Run the program to see if it does what you want it to.
6. Debug the program (fix the mistakes.)
7. Repeat steps 5 and 6 until you are satisfied with the results.

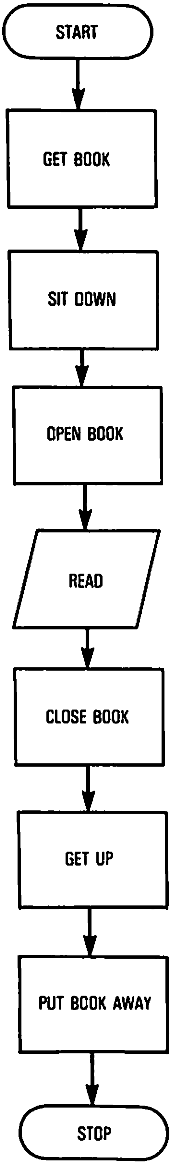


FIGURE S2-1

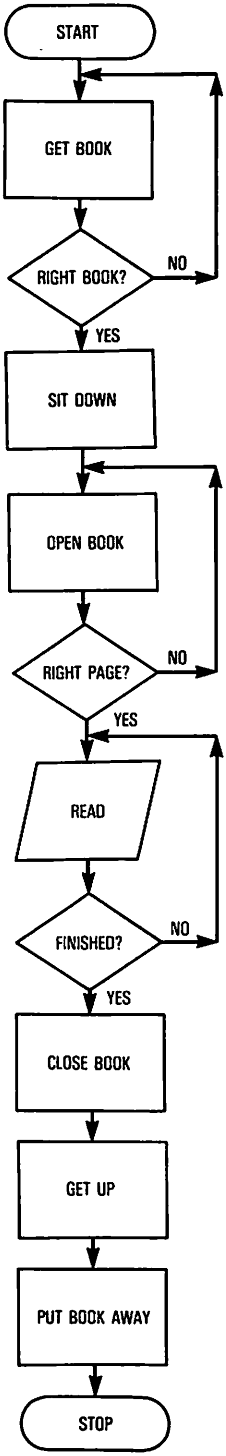


FIGURE S2-2

Sample Flow Chart

First we will draw a **FLOW CHART** for a practical activity that has nothing to do with programming. Follow the flow chart shown on the left of page 84 for reading a book.

The flow chart in Figure S2-1 is very simple. Notice how it differs from Figure S2-2 for the same activity. In Figure S2-2 there is an opportunity to make some decisions. The arrows tell it where to go, whether the answer is YES or NO. The shapes you see around the words are important too. Figure S2-3 defines each shape.

The plan we intend to use for the TAKE OFF program looks like Figure S2-4 on page 86.

It's not as complicated as it may seem. Follow each symbol, reading inside what the computer will do. Pay particular attention to the arrows and you can't get lost.

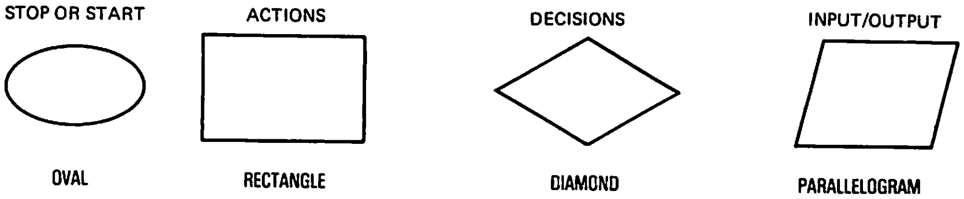


FIGURE S2-3

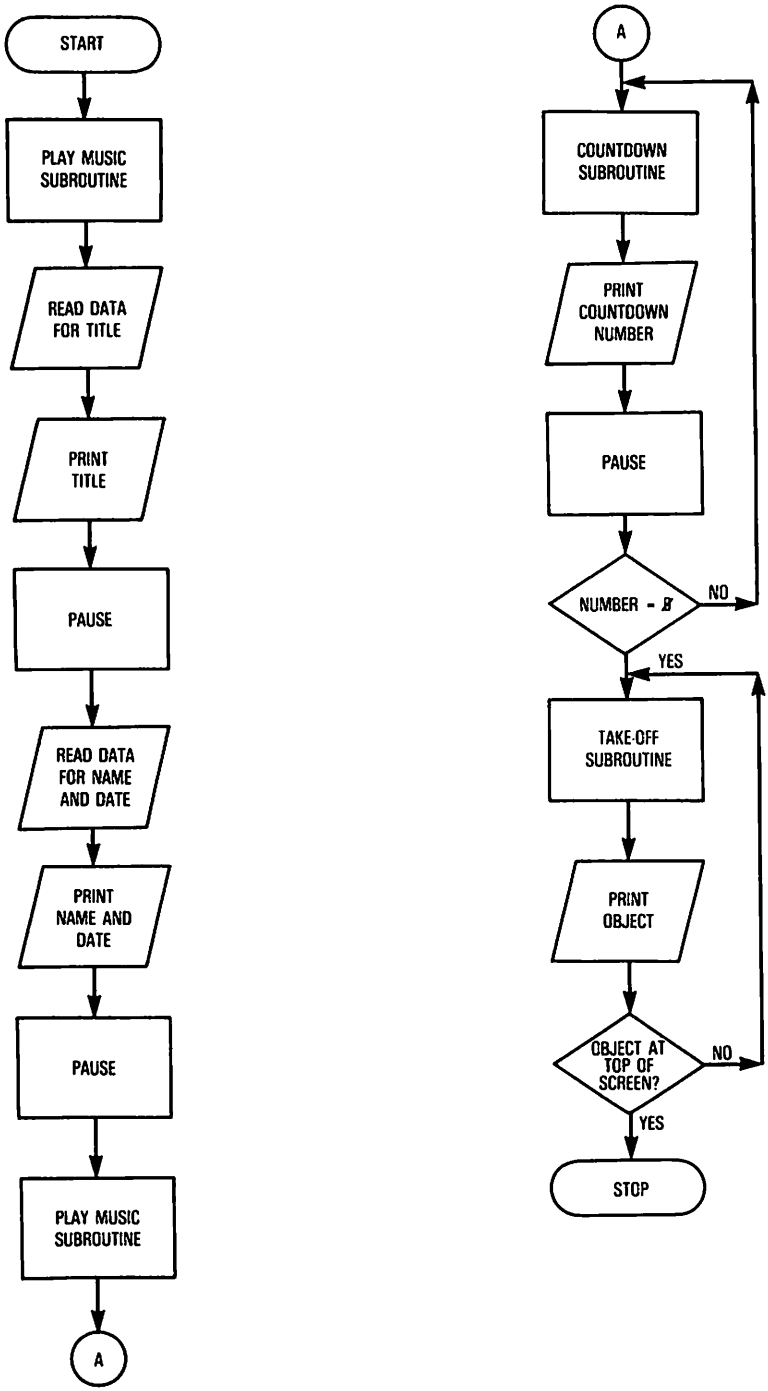

START
PLAY MUSIC SUBROUTINE
READ DATA FOR TITLE
PRINT TITLE
PAUSE
READ DATA FOR NAME AND DATE
PRINT NAME AND DATE
PAUSE
PLAY MUSIC SUBROUTINE
A
A
COUNTDOWN SUBROUTINE
PRINT COUNTDOWN NUMBER
PAUSE
NUMBER - Ø
NO
YES
TAKE-OFF SUBROUTINE
PRINT OBJECT
OBJECT AT TOP OF SCREEN?
NO
YES
STOP

FIGURE S2-4

EXERCISES

1. Make a flow chart for this activity. You want to go to the store on your bike to get a notebook. Put these steps in order and draw the correct shapes around them. Add any other steps which you think are important.

   IS TRAFFIC COMING?
   FIND NOTEBOOK IN STORE
   GET BIKE
   STOP
   IS NOTEBOOK OK?
   UNLOCK BIKE
   LOCK BIKE
   PAY FOR NOTEBOOK
   CROSS STREET
   GET OFF BIKE
   GET ON BIKE
   ENTER STORE
   LEAVE STORE
   AT STORE?
   RIDE BIKE
   INTERSECTION
   AT HOME?
   START
   PUT BIKE AWAY

2. Write a flow chart for the following activities, using your own ideas for the necessary steps to complete it:

   a. Eating a meal
   b. Playing a card game
   c. Going to a movie
   d. Making a sandwich

SUPPLEMENT 3
SAVING and LOADING

Writing programs for a computer is a time consuming undertaking. For long programs, many hours are involved. Even if you write a program in one day, you usually want to come back and make some changes (debug it) at a later time. Consequently, there must be some way to SAVE what you have written so that you can turn off the computer at the end of a session and not lose all your work.

When the computer is turned on, it remembers the instructions you have entered only as long as it stays turned on. Our memory is different. When we go to sleep, we wake up remembering many things. Going to sleep does not erase everything we know from our memory. The computer does not have a living brain like ours, so it is necessary for us to have a way of SAVING what we want it to remember so that it will be there the next time we are ready to continue working on the project.

In Supplement 1 you may have initialized a disk (prepared one to save your work), or perhaps you are using a cassette tape. Disks or tapes take the place of the computer's memory. Just as you can save voices on cassette tapes, or songs on records, you can save a program on a floppy disk or cassette tape. The disk must be initialized on the microcomputer on which you wish to run your program. This is explained in Supplement 1. Remember that you cannot use a TI disk or tape on any other kind of microcomputer.

After you complete each lesson in Chapters 3-17, you will want to SAVE what you have added to the TAKE OFF program on the prepared disk (or tape) if you have the hardware to do so. The following is a practice exercise to save a file on a disk or tape.

Saving a Program on a Disk:

1. Insert the disk in the disk drive and turn on the hardware in this order:

   a. Disk Drive
   b. Disk Controller
   c. Keyboard
   d. Monitor
   e. (Peripheral Expansion System)

   (If the equipment is already on, just insert the disk in the drive.)
2. Put the DISK MANAGER command module in place
3. Type this four-line program and press ENTER at the end of each line:

```
10 REM *PRACTICE PROGRAM TO SAVE*
20 PRINT "THIS IS ONLY A TEST SAVE"
30 PRINT "END OF TEST SAVE"
99 END
SAVE DSK1.TESTSAVE
```

This program tells the computer to save, on a disk (DSK) in drive 1, the 4 instructions you have entered. It names the file TESTSAVE. The drive will go on for a few seconds. Wait until you see the flashing cursor in the lower left corner, which means your program is now saved. You can choose any name for the file, but leave no spaces in it.

Loading a File Saved on a Disk

Let's say that after saving the TESTSAVE file you have finished working on the computer. Remove the disk from the disk drive, and turn off the hardware. It is now the next day and you sit down at the TI and want to see the TESTSAVE file. Follow these instructions:

1. Insert the disk, DISK MANAGER, and turn on the hardware in the correct order.
2. Go to TI Basic and enter this:

```
OLD DSK1.TESTSAVE
```

   This tells the computer to find an old file (all files already saved are called OLD) on a disk in drive 1 named TESTSAVE. The drive will go on and search the disk for a file by that name. When the program has been located you will see the flashing cursor again.

3. Type LIST and TESTSAVE will appear on the screen. You can now add to it or make changes. If you do, then you MUST save the file again or else the changes will NOT be recorded on the disk. To save the new version enter:

   ```
   SAVE DSK1.TESTSAVE
   ```

   You give the same command whenever you want to save a file, new or old. You can name the file anything you want, of course.

Cataloging the Disk

You may forget what programs you have saved on the disk. Follow these steps to get a CATALOG (listing) of the programs on it:

1. Insert the disk in the disk drive.
2. Turn on the hardware if it is not already on.
3. Press any key.
4. Press the number for DISK MANAGER.
5. Wait for DISK MANAGER options to appear.
6. Press the number for DISK COMMANDS and then ENTER.
7. Press the number for CATALOG DISK and then ENTER.
8. Press the number for MASTER DISK and then ENTER.
9. Press the number for SCREEN and then ENTER.
10. Hold down the SHIFT key and press the V.

The contents of the disk will appear. It will list FILENAME, SIZE, TYPE and P. You need only worry about the FILENAME list, since this is what you use when you want to look at or run one of your programs. Remember that you must be in TI BASIC in order to load a program from a disk. Practice SAVING, LOADING, and CATALOGING with your disk. You must use the EXACT name as shown in the catalog.

Saving a File on Tape

1. Insert the tape into the tape recorder.
2. Go to TI BASIC and enter this four-line program exactly as shown. Press ENTER at the end of each line:

   ```
   10 REM *PRACTICE PROGRAM TO SAVE*
   20 PRINT "THIS IS ONLY A TEST SAVE"
   30 PRINT "END OF TEST SAVE"
   99 END
   ```

3. Enter this command:

```
SAVE CS1.TESTSAVE
```

which tells the computer to save on a cassette tape (CS) a file named TESTSAVE.

4. There will now be instructions on the screen which you should follow to complete the saving process.

Loading a File Saved on Tape

Now that the file TESTSAVE is saved on the tape, you can turn off the computer and leave. When you return,. this is how to put TESTSAVE (or any file you have saved) back into the computer's memory:

1. Insert the tape into the recorder.
2. Turn on all the hardware.
3. Go to TI BASIC.
4. Enter

   ```
   OLD CS1.TESTSAVE
   ```

5. Follow the instructions on the screen.
6. When it has been loaded, the flashing cursor will appear in the lower left corner. Simply type LIST and the program will appear. You can now add to it or make changes. If you do, you MUST save it again or else the changes will NOT be recorded on the tape. Use the same file name you used before, and the new version will replace the old one:

   ```
   SAVE CS1.TESTSAVE
   ```

You will be taken through the same process as before with instructions on the screen to follow.

Cataloging the Tape

Cassette tapes cannot be catalogued on a computer. It is very important that you keep a careful record of the files and file names that you save on a tape for your reference.

SUPPLEMENT 4
Immediate Mode

The computer has a special capability called the **IMMEDIATE MODE.** This mode allows you to get immediate action or execution of the instructions you enter. Line numbers are not used.

TRY THIS: (Press ENTER after each line)

```
NEW
PRINT 12345
```

Immediately after pressing ENTER the computer follows the instructions. NEW erases everything from the memory. On the screen are the numbers 12345, because it was told to PRINT them. The PRINT statement is necessary so that you can see the results on the screen.

TRY THIS:

```
CALL CLEAR
PRINT 10 + 20 - 5
```

You are using the TI as a calculator now—an expensive one at that! The major difference between the IMMEDIATE MODE and the PROGRAMMING MODE is that in the IMMEDIATE MODE only one instruction can be executed at a time, no line numbers are used, and it is not necessary to type RUN since each instruction is executed as soon as ENTER is pressed. When you write a program, you accumulate several numbered lines of instructions and each one is executed in order when you give the RUN command.

When using the computer to do math, either in the IMMEDIATE MODE or the PROGRAMMING MODE, there are some important rules to follow. First, study this list of arithmetic symbols the computer knows.

+	ADDITION	-	SUBTRACTION	*	MULTIPLICATION
/	DIVISION	^	EXPONENTIATION	( )	PARENTHESES

The PLUS and MINUS symbols are easy to remember. They were used in the example of the IMMEDIATE MODE above:

```
1 + 2
10 + 20 - 5
```

The asterisk (*) and slash (/) are used by most computers to indicate multiplication and division.

TRY THIS:

```
PRINT 2 * 10 / 5
```

The answer should be 4 since 2 times 10 is 20 and 20 divided by 5 is 4. Do you agree? Predict the answer to this line:

```
PRINT 5 * 3 + 10
```

The answer is 25.

TRY THIS:

```
PRINT 10 + 5 * 3
```

The answer is 25. When doing a math problem the rule is to always do the multiplication and division before doing the addition and subtraction. In the problem:

```
PRINT 5 * 3 + 10
```

it first multiplied 5 and 3 (15) and then added 10 (25). In the problem PRINT 10 + 5 * 3, it multiplied first (5 * 3 = 15) and added 10, no matter in what order the problem was entered. If your answer was 45, it's because you ADDED 10 + 5 first, then multiplied 15 by 3 to get 45. When you use the computer as a calculator you must remember that multiplication and division are done before addition and subtraction. Predict the answer to this line:

```
PRINT 30 - 20 / 10
```

The answer is 28 since the division is done first: 20 divided by 10 is 2; 30 - 2 is 28. Experiment with some problems of your own, each time predicting what the answer will be.

There is a more advanced arithmetic concept which stands for EXPONENTIATION (squaring, cubing, etc.).

TRY THIS:

```
PRINT 4 ^ 2
```

This instructs the computer to SQUARE the number 4. Literally, it means "4 to the second power". When writing this problem by hand, it looks like this: 4^2 and the answer is 16. To SQUARE a number means to multiply it by itself. Predict the answer to this line before pressing ENTER.

TRY THIS:

```
PRINT 3 ^ 2
```

Three times itself (3 * 3 or 3^2 is 9. In higher mathematics exponentiation is used frequently. You will probably not use it very often in everyday math.

The parentheses ( ) are also important when doing arithmetic on the computer. What do you think the answer to this problem will be?

TRY THIS:

```
PRINT (4 + 2) * 3
```

Using parentheses is a way of getting the computer to do a calculation BEFORE any other. The answer would be 18; 4 + 2 (6) is calculated first due to the parentheses surrounding that problem; then 6 is multiplied by 3 (18). If the parentheses were removed from the problem the answer would be 10; 2 times 3 = 6, and 6 + 4 = 10. Check this on the computer without the parenthesis.

Here are some more predictions for you to make:

```
PRINT (8 - 4) / 2
PRINT 8 - 4 / 2
PRINT (8 + 6) / 2 + 4 - 1
PRINT 8 + 6 / 2 + (4 - 1)
```

The order the computer must follow when doing an arithmetic problem is called the ORDER OF OPERATIONS:

FIRST it calculates anything inside the parentheses
SECOND it calculates EXPONENTIATION
THIRD it calculates MULTIPLICATION and DIVISION
FOURTH it calculates ADDITION and SUBTRACTION

It also ALWAYS begins looking at the problem from left to right, so in the problem:

```
PRINT (4+3) * (7-1) / 6
```

FIRST it calculates (4 + 3) (=7) in parentheses;
SECOND it calculates (7 - 1) (=6) in parentheses;
THIRD it calculates 7 times 6 (=42) multiplication on the left;
FOURTH it calculates 42 divided by 6 (=7) division on the right;

Do you see how important it is to know how you want a problem figured out before entering it into the computer? For example, here is a problem and the instruction for it:

There are 5 boys and each buys 2 hamburgers. When they are finished eating, 3 of the boys buy one more. How many hamburgers were purchased?

```
PRINT 5 * 2 + 3
```

A total of 13 hamburgers are purchased. What about this problem?

Two boys come into the restaurant and then 3 more friends join them. They each buy 5 hamburgers (they're very hungry boys!). How many did they buy altogether?

```
PRINT (2 + 3) * 5
```

First you add the number of boys (in parentheses) then multiply the answer by 5. This looks similar to the instructions for the first problem, but gives a different answer. Do you understand why? If not, go over the examples again.

Always plan ahead to decide how you want to write the PRINT statement so that it does the arithmetic problem in the correct order.

EXERCISES

1. Figure out these problems using the IMMEDIATE MODE by placing the parentheses in the correct spot to get the answer provided:

a.	7 + 4 * 5	Answer = 55
b.	6 / 3 - 2 + 1	Answer = 1
c.	20 - 4 / 4 + 4	Answer = 8
d.	20 + 5 * 10 - 6	Answer = 40

2. A dozen eggs cost 89 cents. How much does one egg cost?
3. Terry washes cars for $2.50 and trucks for $3.00. How much does Terry make if he washes three cars and two trucks?
4. Pat babysits for $2.50 per hour, but after midnight the fee increases to $3.00. How much is made if Pat babysits from 7 p.m. to 2 a.m.?

SUPPLEMENT 5
Editing

The Texas Instruments 99/4A microcomputer has one of the best editors available today. One way of editing a line of instructions was illustrated in Chapter 5. There are actually two other methods that can be used for editing just as easily.

The way you already know is to enter the number of the line you want to edit, hold down the FCTN key, and press the letter E (up arrow). This places the line indicated on the screen and allows you to use certain keys to make changes in it without retyping the entire line.

A second way to use the edit mode is to enter the number of the line you want to edit, press the FCTN key and then the X (down arrow). This also prints the instruction on the screen, ready to be edited, exactly as FCTN E does.

TRY THIS:

```
10 PRINT "HAVE A NICE DAY"        (Press ENTER)
```

For practice, let's change the message in line 10 to HAVE A NICE EVENING. Type 10, press the FCTN key, and then press X to place the line in the edit mode. Move the cursor to the D in DAY using the FCTN key and the right arrow. Simply type EVENING right over the word DAY. Press ENTER and the changes are made. Type LIST 10 to check to see that the change was made.

The third way to edit a line is also easy.

TRY THIS:

```
EDIT 10
```

(Press ENTER)

Again, you can use the FCTN key and arrows to change EVENING to MORNING. Don't forget that the question marks must be at the end of the line. List line 10 to check that the change was made.

Choose whichever of the three methods of editing you find easiest to use since they all accomplish the same thing.

Here is one additional tip that comes in handy. Let's say that you have a ten line program, with line numbers from 10 to 99. You want to edit lines 30, 40 and 50, but for some reason you want to edit line 40 first. You would enter

```
40 FCTN E
```

and edit the line. A shortcut way to get line 30 on the screen for editing is to enter

```
FCTN E     (Up arrow)
```

This places the line just before the one currently in the editor on the screen. Following the same idea, if line 40 is in the editor and you enter

```
FCTN X     (Down arrow)
```

line 50, or the line right after the one currently in the edit mode is placed on the screen. You will find that the FCTN and arrow keys will save you a lot of time when editing programs.

EXERCISE

1. Write a short program to print several things on the screen. Go back and edit them using all three methods to determine which one you feel most comfortable with.

SUPPLEMENT 6
Punctuation

The semicolon (;) and the comma (,) have special meanings in BASIC. When you use the PRINT statement, you can control the way in which it shows the information on the screen. This is referred to as FORMATTING.

The COMMA tells the computer to start printing at the beginning of a FIELD. On the TI, each row of print is divided into two FIELDS of fourteen (14) spaces each.

TRY THIS:

```
10 PRINT 1,2
RUN
```

This instruction prints a 1 at space number 1 and a 2 to its right at space number 15, the beginning of the second FIELD. Let's try some examples.

TRY THIS:

```
10 PRINT 1,2,3,4
RUN
```

When you type a comma between the numbers you can only get two numbers on a line since there are only two FIELDS per row. A number or word can be up to fourteen spaces long, because that is the length of one FIELD.

TRY THIS:

```
10 PRINT "HI","FRIEND"
RUN
```

HI is printed beginning in the first space, then the comma instructs the computer to begin the word FRIEND in space 14, the beginning of the second FIELD.

The SEMICOLON tells the computer to print one number (or word) right next to the other.

TRY THIS:

```
10 PRINT 1;2
RUN
```

Notice that there is one space before the 1, and 3 spaces between the 1 and 2. These spaces are automatically put in by the computer in case a negative sign were needed. If you type -1, the minus sign will take the first space which is now vacant.

TRY THIS:

```
20 PRINT 1;2;3;4
RUN
```

Three spaces are left between each number.

TRY THIS:

```
10 PRINT 1;2,3;4
LIST
RUN
```

There is a large space between the 2 and 3 since a comma was used there.

TRY THIS:

```
NEW
10 PRINT "HI";"FRIEND"
20 PRINT "HI","FRIEND"
RUN
```

Notice how it is printed differently on the screen with a comma than when a semicolon is used. Semicolons leave no space between the words.

TRY THIS:

```
20 PRINT "HI ";"FRIEND"
RUN
```

To insert a space between two words you can put a space inside the quotation marks. The computer prints exactly what is inside the quotation marks—even spaces.

The semicolon always means to print what follows the semicolon, whether it's a word in the very next space or if it's a number on the same line. Before running this program, predict what the output will be.

TRY THIS:

```
NEW
10 CALL CLEAR
20 PRINT "GUM";
30 PRINT "BALL"
99 END
RUN
```

Were you right? Lines 20 and 30 print out on the same line. This is a very important concept. The semicolon at the end of line 20 still has the same meaning as before. It instructs the computer to leave no spaces between GUM and BALL. It doesn't matter that the second word is on a different line of instructions in the program.

EXERCISES

1. Enter this program:

   ```
   10 PRINT "I LIVE IN"
   20 PRINT "PHOENIX"
   99 END
   ```

   Edit line 10, and insert the correct punctuation (comma or semicolon) so that the output looks like this:

   `I LIVE IN PHOENIX` (or whatever your city is)

2. Write a 3 line program using commas and semicolons that gives the following output:

   ```
    2               4
   -8    245                  92
    400             50
   ```

3. Write a one line program to print your first and last names, one space apart, on the same line. Rewrite the program using two PRINT statements to produce the same output.
4. Design a PRINT statement to list the names of a baseball team's players, each player's position, and number of hits. Use one PRINT statement for each separate name, position and number of hits. Format the program so that the output looks like this:

```
PLAYER NAME       POSITION      HITS
  Smith            Pitcher       7
    .                 .          .
    .                 .          .
    .                 .          .
```

SUPPLEMENT 7
READ/DATA

The LET statement is one method of assigning values to variables.

TRY THIS:

```
NEW
10 REM ***ADDING TWO NUMBERS***
20 LET C = 0
30 LET A = 5
40 LET B = 6
50 LET C = A + B
60 PRINT C
99 END
RUN
```

This is a sample program to add 5 and 6 and print out the answer. The output will be 11. Let's say that you want to add two more numbers to C. You could keep writing LET statements giving values and names to the numbers you want to add, but there is a better way.

The **READ/DATA** statement is another way of assigning values to variables. When you use the LET statement, such as LET A = 5, and then want to change the value of A to 6, you would either have to edit the line or type it over. There is a way to change the values of variables inside a program. The READ/DATA statement allows you to substitute different values for one variable without rewriting the LET statements.

TRY THIS:

```
20 LET C = 0
30 READ A
40
50 LET C = C + A
70 GOTO 30
80 REM ***VALUES FOR A***
90 DATA 5,6,7,8
99 END
LIST
```

The entire program looks like this:

```
10 REM ***ADDING TWO NUMBERS***
20 LET C = 0
30 READ A
50 LET C = C + A
60 PRINT C
70 GOTO 30
80 REM ***VALUES FOR A***
90 DATA 5,6,7,8
99 END
```

Run the program. (Don't worry about the error message for now.) Look at the output. Line 30 tells the computer to look for a DATA statement line and whatever the first value is in it, assign it to the variable named A. A is now equal to 5. Line 50 instructs the computer to add variable A to C (0 + 5 = 5). Line 60 instructs that the value of C be printed. Right now C = 5, and so 5 is placed in the memory box C.

Line 70 then tells the computer to go back to line 30 and READ the next value in the DATA statement (which is 6) and assign it to A. In line 50 C = C + A means C = 5 + 6, since the new value for A is 6 and the old value for C is 5. The new value for C is 5 + 6 so 11 is printed and placed into memory location C. The third time around what will variable A have for its value? If you said 7 you are absolutely right! In line 50, C = 11 + 7; 18 is printed. The last time C = 18 + 8 or 26.

Now for the explanation of the error message. The fourth time the computer got to line 70 and then went back to look for the next new value for A there was none left since all the values had already been used. Therefore, the computer tells you

```
DATA ERROR IN 30
```

This is because the data (numbers in the DATA statement) have all been used. There is a way to avoid this problem which will be discussed in the next chapter. The important thing is that you understand what produced the DATA error.

Looking back at the program, if you want to change the numbers being added you need only edit line 90, inserting the new values. The values must be separated by commas, but there is no limit to the number of values in a DATA statement.

The DATA statement can appear anywhere in a program but a good habit and a rule you should follow is to put the DATA statement at the end of the program, right before the END statement. The READ statement must come before you attempt to use any of the variables named in the program. For example, if you try to use the variable A before you have a READ statement, A will have a value of zero (0). The computer assumes all variables are 0 until told otherwise.

TRY THIS:

```
NEW
10 REM ***ADDING NUMBERS***
20 LET Z = 0
30 READ A,B,C,D
40 LET Z = A + B + C + D
50 PRINT Z
60 DATA 5,6,7,8
99 END
```

The computer assigned memory locations to these values: A = 5; B = 6; C = 7; D = 8. When it was told to READ, it placed the first DATA value in memory location A. It then read 6 for B, 7 for C, and 8 for D. This accomplishes the same thing the last program did but in fewer lines. READ/DATA statements are very versatile. Edit the program with these lines.

TRY THIS:

```
15 LET X = 0
40 LET Z = A * B
45 LET X = C * D
55 PRINT X
LIST
```

Run the edited version of the program. This time Z = 30 (A * B or 5 * 6) and X = 56 (C * D or 7 * 8). There is no error message now since only four variables were read and used (A, B, C, and D) and there were 4 values available for them in the DATA statement.

You will be interested to know that the LET in LET statements is optional. For example,

```
15 LET X = 0
```

gets the same results as

```
15 X = 0
```

This is a shortcut which can be used. However, some programmers continue to use LET so that the program is easier to read.

EXERCISES

1. READ five sets of numbers to be multiplied. For example, 2 * 3, 4 * 5, and so on. Each time have the answer printed in the following format:

   ```
   2 * 3 = 6
   ```

2. Using only one READ and one DATA statement, READ one value for each of the variables A, B, C, D, E, F, and then add them together. Print out their sum.
3. Using READ/DATA statements, calculate totals of the following amounts of chemicals needed for five city swimming pools. The output should look like this:

   ```
   TOTAL AMOUNT OF CHEMICAL A = ?
   TOTAL AMOUNT OF CHEMICAL B = ?
   ```

DATA	POOL 1	POOL 2	POOL 3
CHEMICAL A	100 gals	75 gals	125 gals
CHEMICAL B	35 gals	25 gals	43 gals

4. Try to figure out a way to avoid the DATA ERROR message when you use one variable name and several values in the DATA statement. If you have trouble, wait for the next chapter and then come back to do this exercise.

SUPPLEMENT 8
IF/THEN Statements

In this chapter you will learn how to combine READ/DATA statements with other statements.

TRY THIS:

```
20 READ A
30 PRINT A
40 GOTO 20
90 DATA 34,42,56,78,10
99 END
RUN
```

In line 20 the computer reads the first number in the DATA list in line 90 and the value of A becomes 34. Line 30 prints 34 and line 40 sends it back to read the second number in the data list; 42. After reading and printing out the five values in the DATA statement, you get the same error message that you got in the last chapter:

```
*DATA ERROR IN 20
```

The **IF/THEN** statement is one way to avoid this error. The IF/THEN statement allows the computer to regulate a program by skipping or **BRANCHING** to another section of the program if a variable meets certain conditions. We can do this with a FLAG. A **FLAG** is a value in the **DATA** statement that is not intended to be used except to alert the computer that it is the last value. In other words, it is a dummy variable. The FLAG number will be -999.

TRY THIS:

```
95 DATA -999
LIST
```

Depending on what your data looks like, select a FLAG that is unusual. In this program, a number like 56 would not make a good FLAG but -999 is a good number since it is very different from any of the other data. Here is the way to use the IF/THEN statement and flag to eliminate the error message.

TRY THIS:

```
25 IF A = -999 THEN 99
LIST
RUN
```

When the computer goes through each READ and PRINT statement, it must also read line 25. Therefore, once it has READ and PRINTED the five values for A and comes to -999 in the DATA statement, what does line 25 do? Literally, line 25 says that, "If A is equal to -999 then go directly to line 99", which is the END. If A is not equal to -999, then the computer ignores the rest of line 25 and continues on to line 30.

If you put the IF/THEN statement before line 20 (try line 15) would you get the same output? If you are not sure then delete line 25 and add the IF/THEN statement at line 15 to check out your answer. It is important that you be able to predict not only what an IF/THEN statement will do but also what it will do differently depending on where it is placed within a program.

EXERCISES

1. Print all the numbers from 12 to 20 using GOTO, IF/THEN, and READ/DATA statements. Insert a flag to avoid an error message.
2. Write a program that READs in several ages to be assigned to a variable A. If the age is less than 12, print out one message. If the age is between 12 and 20, print a different message. If it is over 20, print a third message.

SUPPLEMENT 9
VCHAR

There is an advanced graphic statement called **VCHAR** that allows you to specify or define a character and then call for it to be printed on a certain row on the screen. The V stands for vertical (up and down). The statement looks like this:

```
10 CALL VCHAR (12,16,42)
```

The first number specifies the ROW. For example, if row 1 were used, the character would be placed at the very top of the screen. Row 12 is 12 lines down the screen.

The second number, 16, refers to the COLUMN. Column 16 is about half way across the screen to the right.

The last number instructs the computer what it is you want printed in ROW 12, COLUMN 16, using a character code number. If you look at page 151, you will see that code 42 is an asterisk (*). If you enter line 10 and run it, the * will be printed on the screen, in the proper row and column, but it happens so quickly it's impossible to see it. A timer loop is essential when VCHAR is used.

TRY THIS:

```
NEW
5 CALL CLEAR
10 REM *PRINT AN * IN ROW 12 COLUMN 16*
20 CALL VCHAR (12,16,42)
```

```
30 REM *PAUSE TO SHOW CHARACTER*
40 FOR TIMER = 1 TO 300
50 NEXT TIMER
60 CALL CLEAR
70 FOR TIMER = 1 TO 200
80 NEXT TIMER
90 GOTO 5
99 END
RUN
```

This is a program that will flash an asterisk on and off the screen. In line 20 the instruction tells the computer to go to row 12, column 16, and print the symbol for the character code 42, which is the *. (The computer really only knows symbols or letters as character codes, as shown in Appendix C.) In lines 40 and 50 the TIMER loop causes the * to remain on the screen for a count of 300. The screen clears at line 60 and line 70 makes the computer count to 200 before returning to line 20 which prints the * again.

Add these lines to the program:

```
55 REM *PRINT A # IN ROW 12 COLUMN 16*
60 CALL VCHAR (12,16,35)
65 REM *PAUSE TO SHOW CHARACTER*
RUN
```

Now the * and # are alternating flashing on the screen. Edit lines 20 and 60 to flash the letters H and I on the screen. Use Appendix C to determine the code numbers.

Are you ready for something really fancy? As you have seen, the **VCHAR** statement requires a **ROW** and a **COLUMN** number. You can add to the COLUMN or ROW numbers causing the character to move from one spot to another using a FOR/NEXT loop. We'll add lines 12 and 90, and edit 20 and 60.

TRY THIS:

```
12 FOR I = 3 TO 20
20 CALL VCHAR (2,I,42)
60 CALL VCHAR (2,I,35)
90 NEXT I
LIST
```

Run the program. Each time the VCHAR is shown on the screen its location will increase by one column, since the value of I increases by one each time the loop is entered. The value for I replaces the I in the VCHAR statements. It stops when I reaches 20, which is near the right edge of the screen.

The same thing can be done by substituting a variable from a FOR/NEXT loop in place of the ROW number. Edit lines 20 and 60 again.

TRY THIS:

```
20 CALL VCHAR(I,I,2)
60 CALL VCHAR(I,I,35)
RUN
```

The value of I from the FOR/NEXT loop is used for both the row and column in the VCHAR statements. Since the I increases by one each time, the row and column also increase by one each time.

EXERCISES

1. Create a program which will cause a letter to go from the lower left of the screen to the upper right of the screen, as if it were walking up stairs. Now make it go the opposite way, down the stairs.
2. Design a program that will show a number flying across the screen from left to right.

SUPPLEMENT 10
HCHAR

If you enjoyed learning how to use the VCHAR statement in Supplement 9, you will have even more fun with **HCHAR**. The **H** stands for horizontal (left and right). Similar to VCHAR, HCHAR allows you to place any character (old or new) on the screen in a particular row and column. But we can add one additional number in the statement which stands for the number of times you want the character repeated across the screen. In the line

```
 1 CALL CLEAR
10 CALL HCHAR (14,5,42,18)
RUN
```

the 14 and 5 are the row and column numbers the character will start in, just like with VCHAR statements. The next number is also the same because 42 is the character code for the asterisk. It tells the computer how many *'s to print beginning at row 14, column 5. Enter line 10 exactly as shown above and run it. Wow! Add line 20 to make another row of *'s under it.

TRY THIS:

```
20 CALL HCHAR(16,5,42,18)
RUN
```

Can you imagine how much you can do with this new statement? By inserting a variable name in the first spot (row), and adding a FOR/NEXT loop, some interesting things can happen.

TRY THIS:

```
NEW
10 CALL CLEAR
20 FOR R = 1 TO 18
30 CALL HCHAR(R,5,42,10)
40 NEXT R
99 END
RUN
```

The row value changes each time it goes through the loop, causing the *'s to be printed on each row from 1 to 18. Are you ready to go one step further? We can also insert a variable name for the column, using the value in the same FOR/NEXT loop so that it also changes each time. This gets a bit tricky. Edit line 30:

```
30 CALL HCHAR(R,R,42,10)
RUN
```

This time the first ten *'s will begin at row 1, column 1 and go to the right. The second time through the loop they will begin at row 2, column 2; the third time at row 3, column 3, and by the time it gets to the last loop they will be printed at row 18, column 18.

Let's try one more thing, getting even more interesting. Edit lines 20, 30 and 40 as follows:

```
20 FOR C = 65 TO 90
30 CALL HCHAR(10,1,C,28)
40 NEXT C
RUN
```

Line 20 now gives values for the character in the HCHAR statement. The first character that will be printed across the entire tenth row will be letter A (character code 65). The second time through the loop character number, 66 (B) is printed over the line of A's. Character code 90 is the letter Z, so this program prints one line of each letter of the alphabet before stopping.

Any number in the parentheses can be substituted with variable names, and a series of FOR/NEXT loops or LET statements can be cleverly inserted to create some interesting effects.

The number for repetition can also be used in the CALL VCHAR statement. The repetitions will start in the row and column indicated and go in a vertical (downward) direction.

EXERCISES

1. Write a program to outline the entire screen in one symbol. Add screen color. Place your name in the middle of the screen.
2. Using FOR/NEXT loops, write a program to fill the screen with symbols, one row at a time.
3. If you are working on the TAKE OFF program, use HCHAR to design an elaborate title page with the title, your name and date inside a special border.

SUPPLEMENT 11
TAB and FOR / NEXT

Formatting can make a simple program much more interesting. As you know, the PRINT TAB function allows the programmer to give an instruction which will place the output on any position on a line.

TRY THIS:

```
10 PRINT TAB (10);"THIS IS COLUMN 10"
RUN
```

Line 10 above shows a PRINT TAB statement followed by the number 10 in parentheses. The number in parentheses indicates the number of spaces to the right that you want THIS IS COLUMN 10 printed. The semicolon must separate the parentheses and the message in quotation marks. By combining PRINT TAB in a program with a FOR/NEXT loop some interesting effects can be created.

TRY THIS:

```
NEW
10 FOR A = 1 to 28 STEP 4
20 PRINT TAB (A);"PEACHES"
30 NEXT A
99 END
RUN
```

The FOR/NEXT loop has already been used in a variety of ways. In this program, if A is equal to 1 in line 10, the first time through the program PEACHES will be printed at space (TAB) number 1. Step 30 sends it back to line 10 where A is now equal to 1 plus 4 (5). The STEP 4 instruction tells the computer to add 4 each time it goes through the loop. (It automatically adds only one if you do not specify STEP with a number.) When the computer encounters line 20 for the second time it will print "PEACHES" beginning at space (TAB) 5. On what TAB will it print "PEACHES" the third time around? Add another 4 to the previous value 5 and you get TAB 9.

Run the program again. Try to follow the logic of the computer to determine how the PRINT TAB works.

TRY THIS:

```
10 FOR A = 28 TO 1 STEP -2
RUN
```

Why does PEACHES appear in the places it does? Experiment with the FOR/NEXT loop and TAB statement with some original ideas of your own.

EXERCISES

1. Write a program to flash your name in different spots on the screen.
2. Melissa has three dogs and wants to write a program to print their names, ages, and weights. Write a program using READ/DATA, FOR/NEXT and PRINT TAB statements that will produce the following output:

```
NAME          AGE          WEIGHT
PIP            5             20
COOKIE         3             40
FLUFF          2             10
```

SUPPLEMENT 12
Character Strings

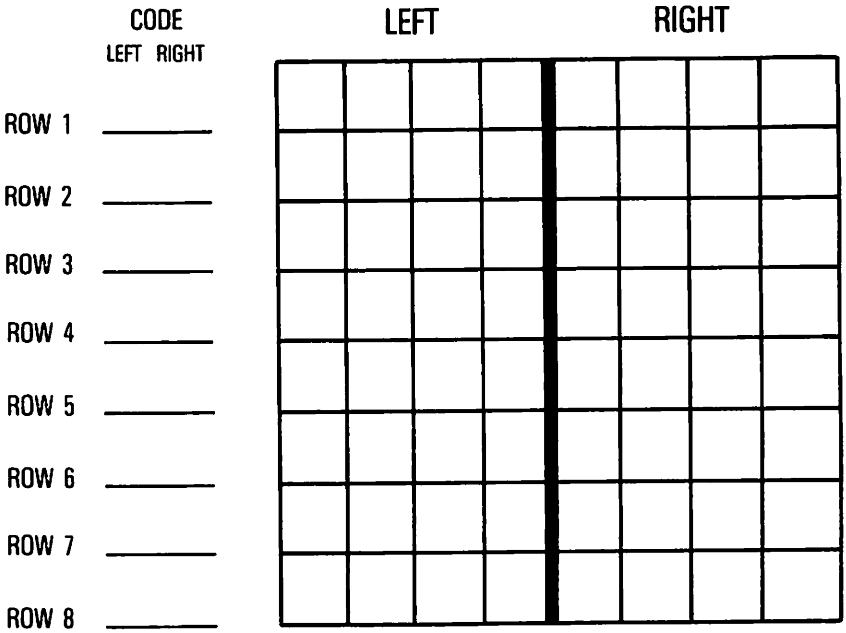
In this chapter you will learn how to make your own characters to print on the screen. For example, press the letter K and look at it. You can make a letter, or anything the same size of a letter, and then print it out on the screen. It is also possible to make several small characters and put them together so that they make a large design of some kind.

This is how it works. Look at Figure S12-1 on page 120. It has a total of 64 boxes. See the dark line drawn down the middle? This separates the left group of boxes from the right group of boxes. Each of the eight rows has four left boxes and four right boxes.

These 64 boxes look large but when put altogether they are actually just the size of the letter K or any other letter you press. Our plan is to make a design the size of a letter that looks like a happy face. To do this, we must darken certain boxes in the diagram in Figure S12-1. The diagram will look like Figure S12-2 on page 120.

There is a special code to tell the computer how to darken in the boxes, done in groups of four, beginning with Row 1 Left, and Row 1 Right; then Row 2 Left, and Row 2 Right, and so on. The code for darkening each set of four boxes is given a number or letter.

There are a total of 16 ways each set of four boxes can be filled in. They can all be empty, or all dark—that is two ways. Only the first box or last box can be dark—that makes four ways. There are 12 other combinations of filling in four boxes.



CODE
LEFT RIGHT

ROW 1 ______

ROW 2 ______

ROW 3 ______

ROW 4 ______

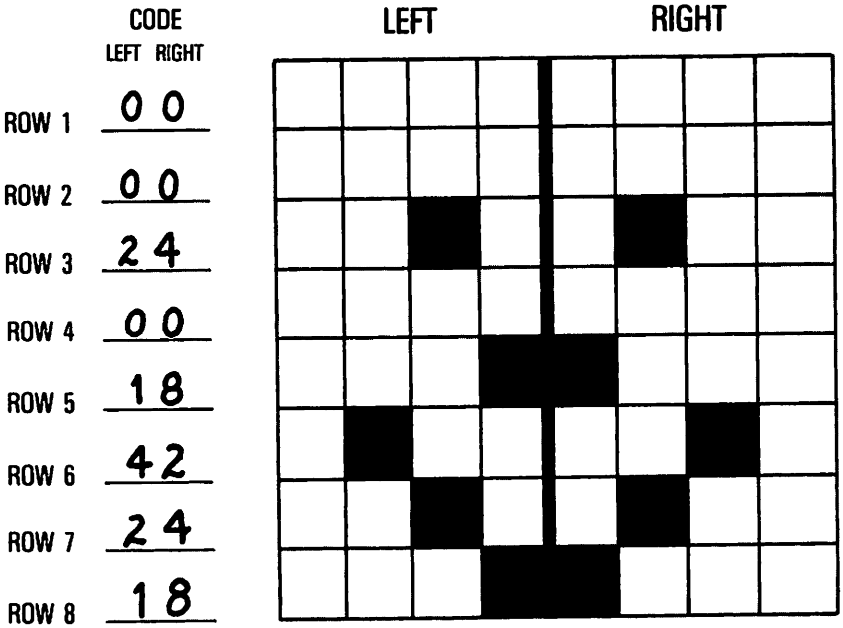
ROW 5 ______

ROW 6 ______

ROW 7 ______

ROW 8 ______

FIGURE S12-1



CODE
LEFT RIGHT

ROW 1 0 0

ROW 2 0 0

ROW 3 2 4

ROW 4 0 0

ROW 5 1 8

ROW 6 4 2

ROW 7 2 4

ROW 8 1 8

FIGURE S12-2

In the Pattern Chart below there is an X if a box is to be left empty, and a # if the box is to be darkened. Look at 0 and F before you look at the pattern codes in between.

If none of the four boxes in a row are dark, then the actual pattern for the 4 box set would be XXXX. It's pattern code is zero. If they all are darkened, the actual pattern would be ####, and the Pattern Code would be the letter F. The other codes are between 0 and F.

ACTUAL PATTERN	PATTERN CODE	ACTUAL PATTERN	PATTERN CODE
XXXX	0	#XXX	8
XXX#	1	#XX#	9
XX#X	2	#X#X	A
XX##	3	#X##	B
X#XX	4	##XX	C
X#X#	5	##X#	D
X##X	6	###X	E
X###	7	####	F

Since there are a total of 16 groups of four boxes, the code for a design the size of one letter will have 16 letters and/or numbers. If we wanted all the 64 boxes dark, the code would be

```
FFFFFFFFFFFFFFFF
```

If only the first box on the top left and last box on the bottom right were to be filled in, the code would be

```
8000000000000001
```

Now to the happy face. The diagram in Figure S12-2 shows that the first two complete rows of boxes will be empty. The code for that will be

```
0000
```

Remember that we first give the Pattern Code for Row 1 Left, then Row 1 Right, Row 2 Left, and Row 2 Right. Row 3 Left has its third box darkened, so the code will be 2 (XX#X). The code for Row 3 Right will be 4 (X#XX). The next entire row (4) is empty, adding two more zeros to the code:

```
00002400
```

We are half way done with the happy face code. This is what the completed code looks like:

```
0000240018422418
```

Check it out yourself with the diagram–is it right? It sure is. Now how do we get the computer to print it on the screen? There are two ways.

First we must give this little guy a name. The computer has already given every letter and number it knows a name (**Character Code**), but it left several undefined Character Codes available for you to use. These names are also numbers. For example, the letter A's number name is 65. The symbol $'s name/number is 36. The number names that we can use are 96-159. Let's choose the number name 100 for our happy face. Here is what the instruction looks like:

```
10 CALL CHAR(100,"0000240018422418")
```

In line 10, **CALL CHAR** is telling the computer "Call (or name) a new character by the first number inside the parentheses." Inside the parentheses 100 is the number/name it comes to. The pattern code inside the quotation marks is now known to the computer by the character code 100.

Now we want to see character 100. Line 10 is still in the memory defining character 100, but we haven't told the computer to put it on the screen yet. The character will be very small.

TRY THIS:

```
20 PRINT CHR$(100)
30 FOR T = 1 TO 1000
40 NEXT T
99 END
LIST
RUN
```

Line 20 literally means for the computer to "Print the CHARACTER STRING named 100 on the screen." CHARACTER STRING refers to a character that is defined by something inside quotation marks. Remember that CHR$ 100 is defined in line 10 with 16 separate numbers surrounded by quotation marks (see line 10).

Why is the timer loop necessary? Without it, the character would be flashed too quickly to see.

Another method of placing the new character on the screen is to make use of VCHAR.

TRY THIS:

```
5 CALL CLEAR
20 CALL VCHAR (10,16,100)
LIST
RUN
```

The difference in this line 20 and the first one is that CALL VCHAR allows exact placement on the screen using row and column number.

By designing characters that fit together, you can make a large design. For example, to darken all the spots in the 64 block grid you would use 16 F's in the CALL CHAR statement. The code would look like this:

```
  CALL CHAR (96,"FFFFFFFFFFFFFFFF")
```

Sketch out your design using several small grids carefully positioned together. The diagram in Figure S12-1 can be copied several times to fill in as you like and then taped together to design the characters. Then, using CALL VCHAR or CALL HCHAR and the correct row and column numbers, each character can be placed together to make a complete design.

EXERCISES

1. Use the CALL CHAR and VCHAR to place a design in several spots on the screen.
2. Design two figures using the CALL CHAR statement. Make one character with legs apart and the other with legs together. Design a program that will show one character briefly and then the other briefly so that it will appear that the figure is jumping.
3. Make a design using at least 9 new characters. Have it move across the screen.

SUPPLEMENT 13
LENGTH

String variables are a lot of fun. The **LEN statement** is an interesting concept. This short program gives an example of the LEN statement:

TRY THIS:

```
NEW
10 LET NAME$ = "MATT"
20 PRINT NAME$;LEN(NAME$)
99 END
RUN
```

Let's look at the program line by line:

Line 10 assigns MATT to a string variable called NAME$.

Line 20 first instructs that MATT be printed (NAME$) and then comes the new statement: LEN(NAME$). This simply tells the computer to print the number of characters in NAME$. Since NAME$ has MATT in its memory location, and MATT has four characters in it, a 4 is printed. The output is

```
MATT 4
```

Let's add a few more lines to the program.

TRY THIS:

```
30 LET PETNAME$ = "FLUFF"
40 PRINT PETNAME$;LEN(PETNAME$)
LIST
RUN
```

Line 30 defines a string variable named PETNAME$ as FLUFF.

Line 40 tells the computer to print the actual PETNAME$ (FLUFF) and the number of characters in it, so the output is

```
MATT 4
FLUFF 5
```

Here are two more lines to add.

Try this:

```
50 PRINT LEN(NAME$) + LEN(PETNAME$)
60 PRINT LEN("CAT")
LIST
RUN
```

Line 50 instructs that the LENgth of the two string variables, NAME$ and PETNAME$, be added together. The number 9 is the output since there are 4 letters in MATT and 5 in FLUFF. Notice it does not tell the computer to print the actual string variables.

Line 60 is an example of how to determine the LENgth of a string variable without first defining it in a LET statement. Specify LEN and inside the parentheses and quotation marks place the string variable you want to know the number of characters in.

There are many other statements used with string variables such as POS, SEG$, STR$, and VAL. They are somewhat complicated and involve more difficult concepts. Refer to the TI BASIC Manual if you wish to expand your learning of more advanced statements.

EXERCISES

1. Using LET or READ/DATA statements, enter several names and have the computer determine how long they are. Experiment by using long names, short names, and names with spaces or characters instead of letters.
2. Using LEN, write a program to produce this output:

```
A 1 BE 2 SEE 3 FOUR 4
```

SUPPLEMENT 14
Multiple Inputs

Expanding upon the simple INPUT statement, a shortcut is available. The program

```
NEW
10 PRINT "WHAT IS YOUR NAME"
20 INPUT NAME$
30 PRINT NAME$
99 END
```

can also be written

```
10   (omit this line)
20 INPUT "WHAT IS YOUR NAME":NAME$
30 PRINT NAME$
99 END
```

The PRINT statement can be included in the INPUT statement as long as a colon (:) is placed after the last quotation mark and the variable name is specified. (On some microcomputers a semicolon is used in this situation.) Enter and run the second 4 line program shown.

The colon (:) in line 20 below is necessary. It warns the computer that the variable names for your input follow.

TRY THIS:

```
NEW
10 INPUT "LIST YOUR NAME, AGE AND WEIGHT, SEPARATING EACH
         WITH A COMMA.":NAME$,AGE,WEIGHT
20 PRINT NAME$;AGE;WEIGHT
99 END
RUN
```

Follow the instructions on the screen. The name, age, and weight you input are printed on the screen.

The statement in line 10 is called a **MULTIPLE INPUT** statement since it asks for several pieces of information from the user. As long as each variable is named after the colon, and separated by commas, there is no limit to the amount of information that can be INPUT. The user must separate input by commas and use no spaces. Effective use of MULTIPLE INPUTS require that the question be stated properly in the INPUT statement. It is usually easier to input one item at a time.

EXERCISES

1. Using the INPUT and colon shortcuts, write a program that asks the user to enter three pieces of information about his/her family and print it out.
2. Write a program using MULTIPLE INPUT statements to get some information. Ask the user to input the name of one food and its price. Print it and then ask the user the same question three more times. You should be able to write it in 4 or 5 lines. The output will look something like this:

```
HAMBURGER    .99
FRIES        .59
ONION RINGS 1.19
DRINK        .79
```

3. Edit the TAKE OFF program so that you make use of colons to space the output more neatly.

SUPPLEMENT 15
Music

The **CALL SOUND** statement does not only produce sound, but it makes music! You can use READ/DATA and FOR/NEXT statements to make your own tunes.

The CALL SOUND statement is made up of several things.

TRY THIS:

```
NEW
60 CALL SOUND(1000,450,2)
RUN
```

This makes a sound with a duration of 1000, tone number 262, and a volume of 5.

TRY THIS:

```
60 CALL SOUND(D,T,V)
```

Don't run the line yet. Previously, you used numbers in the parentheses after the CALL SOUND statement. In line 60 there are now variable names inside the parentheses in place of numbers. The first variable (D) refers to the length of time the note is heard (duration). The T stands for the tone number and the V indicates the loudness of the sound (volume). Of course, values must be assigned to these three variables:

TRY THIS:

```
10 LET D = 1000
20 LET T = 450
30 LET V = 2
99 END
LIST
```

There should be 5 lines in the program. Run the program. One sound is produced.

Let's add more notes. Since the tone number now has a variable name, we can change it by reading different values for it from a DATA statement. In fact, we'll give it 8 different values to play.

TRY THIS:

```
20
40 FOR S = 1 TO 8
50 READ T
70 NEXT S
80 DATA 523,659,587,698,659,784,698,880
LIST
RUN
```

Do you know why the FOR/NEXT loop is from one to eight? It's because there are eight pieces of data (eight different tone numbers) to read from the DATA statement. Each time it reads a new tone number it plays it before going through the FOR/NEXT loop again. It does this eight times. Four scales of notes are shown in Appendix B on page 149 with their actual tone names and the code numbers for each.

Find code number 523 in Appendix B (523 is the first value in the DATA statement). The tone name is C. On the piano it would be called High C. The second number in the DATA statement is E. Find the tone names for the other six code numbers in the DATA statement.

Guess what? The CALL SOUND statement can be written so that more than one sound is played at a time. The first variable inside the parenthesis, duration, applies to all of the tones to be played. However, the actual notes and their loudness (volume) must be specified for each note. After the duration number, the tone code and volume number are listed for the first note. The same is done for the next two notes.

TRY THIS:

```
NEW
10 CALL SOUND(1000,110,2,175,1,117,3)
```

Notice that the first tone, 110, has a volume (or loudness) of 2; tone 175 has a volume of 1, and tone 117 has a volume of 3. Each number must be separated by a comma, and no spaces are allowed. Run the one-line program and see if you can hear the three different tones.

When you use the CALL SOUND statement you can also produce special noises. There are eight noises on the TI, identified with the code numbers −1 to −8.

TRY THIS:

```
10 CALL SOUND(2000,-1,5)
RUN
```

Again the three numbers stand for duration, tone code, and volume. Write seven more lines so that you can hear all eight noises. You can even add a sound to as many as three musical tones just by adding the noise code and volume at the end.

TRY THIS:

```
10 CALL SOUND (1000,440,2,880,10,659,5,-6,17)
RUN
```

If you would like to write tunes on the computer, this can also be done. You have already learned how to produce one, two, or three sounds at one time. You also know that the length of the sound can be changed. Think of the tune to Happy Birthday To You. There are a total of six tones when you say the first line. Say it to yourself and notice the different lengths of the note for "Hap" and "py" and "To' "You". A program for these four words require six lines—one for each note.

TRY THIS:

```
NEW
10 CALL SOUND (500,262,2)
20 CALL SOUND (500,262,2)
30 CALL SOUND (1000,277,2)
40 CALL SOUND (1000,262,2)
50 CALL SOUND (1000,349,2)
60 CALL SOUND (1000,330,2)
99 END
```

The shorter sounds have a duration of 500 and the longer ones have a duration of 1000. Run the program. Does it sound OK?

With written music, notes that are not filled in are called whole notes and look like this:

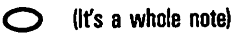


FIGURE S15-1

Their duration is about 1000. Notes that are not darkened in and have tails are called half notes and look like this:

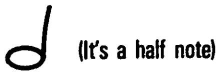


FIGURE S15-2

Their duration is about 500. In the Happy Birthday tune there are 2 half notes and 4 whole notes. A quarter note has a duration of about 250 and looks like this:

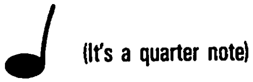


FIGURE S15-3

EXERCISES

1. Examine the musical scale Tone Chart below and write a program to produce this short piece of music. Notice that there are some whole notes and some half notes. The first program line will look like this:

   ```
   10 CALL SOUND (250,262,2)
   ```

   Choose any volume you like.

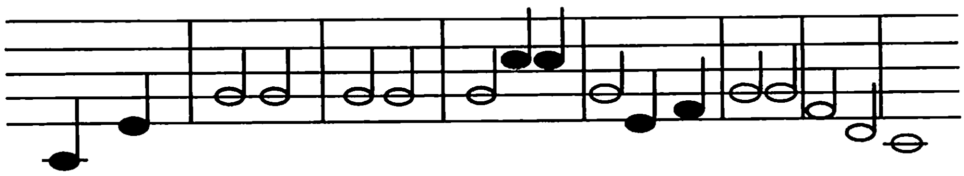


FIGURE S15-4

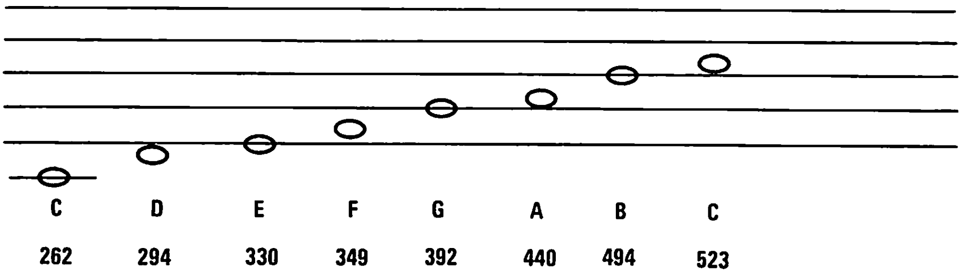


FIGURE S15-5

2. Write a program to play this song. A quarter note with an extra tail,

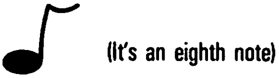


FIGURE S15-6

is an eighth note. It receives 1/2 of a quarter note duration or about 125.

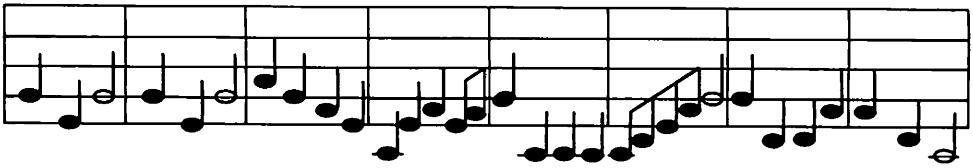


FIGURE S15-7

3. Choose three notes and read them into memory. Write a program using a FOR/NEXT statement to play the notes forward and backward.
4. Use some of your own music ideas, favorite songs, or just experiment with some different sounds.
5. Experiment with the lower frequencies and the noise command to make a rocket blastoff sound. If you want, put it in your TAKE OFF program.

SUPPLEMENT 16
ON/GOTO and RANDOM

There is a statement called ON/GOTO which is similar to GOSUB/RETURN statements.

TRY THIS:

```
NEW
10 INPUT "PICK A NUMBER FROM 1 TO 3": N
20 ON N GOTO 50,70,90
30 PRINT "THOUGHT YOU COULD FOOL ME, DIDN'T YOU?"
40 GOTO 10
50 PRINT "YOU CHOSE THE NUMBER 1"
60 GOTO 10
70 PRINT "YOU CHOSE THE NUMBER 2"
80 GOTO 10
90 PRINT "YOU CHOSE THE NUMBER 3"
95 GOTO 10
99 END
```

Run the program. Play it several times to get the idea of what it is doing. Enter numbers other than 1, 2 and 3 once in a while to see what happens. Try to figure out what line 20 is doing. Stop the program using FCTN 4.

Line 20 instructs the computer "If N (the name given to the input number of the user) is equal to 1 then go to line 50; if N is equal to the number 2, go to line 70; if the input number is 3, go to line 90."

Lines 50, 70 and 90 tell it what to print.

What if the user chooses the number 0 or 4? Then the computer ignores line 20 after determining that the input is not 1, 2 or 3 and continues to line 30 and prints a message. Line 40 then returns the program to line 10 to allow the user to pick another number.

There must be one line number specified in the ON/GOTO statement for each acceptable input number. The input numbers must begin with 1.

Here is another application of the ON/GOTO statement:

TRY THIS:

```
NEW
10 RANDOMIZE
20 LET N = INT(6 * RND) + 1
30 ON N GOTO 40,50,60,70,80,90
40 PRINT "DICE THROW = 1"
45 GOTO 20
50 PRINT "DICE THROW = 2"
55 GOTO 20
60 PRINT "DICE THROW = 3"
65 GOTO 20
70 PRINT "DICE THROW = 4"
75 GOTO 20
80 PRINT "DICE THROW = 5"
85 GOTO 20
90 PRINT "DICE THROW = 6"
95 GOTO 20
99 END
RUN
```

Lines 10 and 20 have new concepts in them. In this program the computer will continuously simulate throwing a die by selecting a random number from 1 to 6 and tell you each time what it chose. (Don't forget FCTN 4 stops the execution of the program.) A random number is one that is chosen off the top of your head. If you asked a friend to choose a number from 1 to 6, the choice would be a random number.

The computer can be instructed to choose a random number too. In line 10 the word **RANDOMIZE** is necessary to warn the computer than it will be expected to choose a different number each time. (Sometimes it does choose the same number twice, but this is just a coincidence.) Line 20 tells the computer to choose a number from 1 to 6 (like rolling one die). It is not important to know what everything else means in line 20 except to know that the 6 in it refers to choosing a number from 1 to 6. If you changed the 6 to a 10, the computer would choose numbers from 1 to 10.

After it chooses a number from 1 to 6, line 30 tells it that if it chose the number 1, it should go to line 40 and print DICE THROW = 1. If it chose the number 6, for example, it goes to line 90, and so on. Notice that there are 6 line numbers in the ON/GOTO statement, one for each of the six possible die throws.

EXERCISES

1. Write a program using ON/GOTO to ask a multiple choice question. Print the question and list four answers (1, 2, 3, and 4). The user chooses an answer. If the answer is correct the user receives a message that says "RIGHT ON!". If incorrect the question will be repeated and the user gets to choose another answer.
2. Write a program to have the computer choose random numbers from 1 to 4. Each time it chooses one, it goes to a line and adds one to a variable. For example, if the number chosen is 1 then you might have a LET statement that looks like this: LET ONE = ONE + 1. Stop it after 20 random numbers have been choosen and print out the number of times it chose each number from 1 to 4.
3. Make a game for the user to pick a number from 1 to 100. Tell the user that the computer has chosen a number to guess. If the guess is too low, give a message that says YOUR NUMBER IS TOO LOW. If the guess is too high, give an appropriate message. Let the user continue until the correct number is guessed and then give the message CONGRATULATIONS! YOU GUESSED THE NUMBER!
4. There is a much more efficient way to write the die throwing program without using an ON/GOTO statement. Can you do it?

SUPPLEMENT 17
More Color

In Chapter 12 you learned how to change the color of the screen. It requires a color code number, in parentheses, after the CALL SCREEN statement. Another statement dealing with color is CALL COLOR. It uses the same color codes as CALL SCREEN, which are listed for you here:

COLOR CODES

TRANSPARENT	1	DARK RED	7	DARK GREEN	13
BLACK	2	CYAN	8	MAGENTA	14
MEDIUM GREEN	3	MEDIUM RED	9	GRAY	15
LIGHT GREEN	4	LIGHT RED	10	WHITE	16
DARK BLUE	5	DARK YELLOW	11		
LIGHT BLUE	6	LIGHT YELLOW	12		

The CALL COLOR statement does not change the color of the screen; it changes only the color behind a letter or symbol. For example, you could have a black asterisk on the screen with red behind just that one asterisk. The CALL COLOR statement must also have a **CALL HCHAR** or **CALL VCHAR** statement following which also includes important information.

TRY THIS:

```
 1 CALL CLEAR
10 CALL COLOR(2,7,5)
20 CALL HCHAR(5,10,42,15)
30 GOTO 30
```

Let's discuss this three line program before you run it.

Line 10 is the new statement with three numbers in parentheses, separated by commas.

The first number (2 in the example) is the SET to which the asterisk belongs in the Character Code Chart. Take a look at page 151 in Appendix C. Notice that in addition to each different number and symbol having its own code, each is also assigned to a SET. For example, code 42 is an asterisk in SET 2; code 72 is the letter H in SET 6. The SET number of the letter or symbol you choose must be specified in the CALL COLOR statement.

The second number (7 in the example) is the color the character will be. Color code 7 is dark red, so the asterisks will be dark red.

The third number (5 in the example) is the color that will appear behind the dark red asterisks–dark blue.

Line 20 is the HCHAR statement you are familiar with. The numbers represent the row and column numbers, the character code, and the number of repetitions.

It is the HCHAR statement that gives the character code, but the CALL COLOR statement that gives the SET number.

Line 30 is a new way to keep the program on the screen without using a FOR/NEXT loop. It is actually sending itself to itself–an infinite loop. To stop it, use FCTN 4 again, and keep this GOTO trick in mind for use in debugging programs.

Run the program now and 15 dark red asterisks with a dark blue background should appear in a horizontal line across row 5 on the screen beginning in column 10.

Edit line 20 using VCHAR.

TRY THIS:

```
20 CALL VCHAR(5,10,42,15)
RUN
```

This time the same color scheme is used, but the asterisks are in a vertical line (downward).

You can also use a character you defined yourself in the VCHAR or HCHAR statements, as long as you identify the SET number in the CALL COLOR statement. Appendix C also lists SET numbers for these on page 151.

EXERCISES

1. Use CALL CHAR to make a design combining at least 6 blocks and then use CALL COLOR to color it.

2. Make a border of #'s around the screen. Use different colors for the screen, the #'s and their background.
3. Design a fancy title page for the TAKE OFF program using color, a special border, and possibly special characters for your name.

Appendix A

CHECKPOINT ANSWERS

CHECKPOINT 1

1. The television connected to the TI-99/4A is also called a **MONITOR** or **CRT**.
2. Any piece of equipment used with microcomputers is called **HARDWARE.**
3. The ON/OFF switch of the computer is on the front of the **KEYBOARD.**
4. We usually call the Texas Instruments microcomputer the **TI** for short.
5. Information that you give to the computer is called **INPUT.**
6. Information that the computer gives to you is called **OUTPUT.**
7. Sentences in English are similar to **STATEMENTS** in BASIC.
8. Writing instructions for the computer in BASIC is called **PROGRAMMING.**

CHECKPOINT 2

1. You should number your lines by **TENS** when writing programs.
2. Every program must have an **END** statement.
3. If you want to write yourself messages within a program, use **REM** statements.
4. When you first decide to write a program follow the four steps for a good **PLAN.**

CHECKPOINT 3

1. When you type the command **NEW**, the computer erases everything in its memory.
2. Type the command **LIST** to see the program you have written so far.
3. When are commands executed by the computer? **IMMEDIATELY AFTER YOU TYPE THEM IN AND PRESS ENTER.**
4. If you want the computer to follow instructions in a program, type the command **RUN.**
5. To erase everything on the screen, but not from the computer's memory, use the command **CALL CLEAR.**
6. Each time you type in a command, you must press **ENTER** to let the computer know you are finished.

7. Which key must you hold down first if you want to use the left or right arrow? **FCTN**
8. Use the left and right arrows when you want to **MOVE THE CURSOR BACK AND FORTH TO CORRECT ERRORS.**

CHECKPOINT 4

1. If you want the computer to show your name or any words on the screen as part of a program, you must write a **PRINT** statement.
2. You can write yourself messages in **REM** statements, but they won't show up on the screen when the program is run.
3. A design or picture on the computer is called **CHARACTER GRAPHICS.**
4. Before each statement you must have a **LINE NUMBER.**
5. To stop the execution of a program hold down the **FCTN** key and press **4.**

CHECKPOINT 5

1. To make changes in program lines you must be in the **EDIT** mode.
2. When you want to change a line, you call for it by typing the **LINE NUMBER,** holding down the **FCTN** key, and typing **E.**
3. To make room for additions in a line in the EDIT mode, hold down the **FCTN** key, type a **2,** and then enter the information.
4. If you want to remove something from a line in the EDIT mode, hold down the **FCTN** key and type a **1.**
5. Once you make the changes to the line, you must press **ENTER** for the computer to put the changes into its memory.

CHECKPOINT 6

1. When you want the computer to print numbers, it is not necessary to use **QUOTATION MARKS** as you do when you want it to print words.
2. If you want the computer to run only part of a program, you can make the program stop anywhere as long as you type a **STOP** statement with a line number.
3. To get the computer to list only lines 100 through 200 of a program, you would type in the command **LIST 100-200.**

CHECKPOINT 7

1. You can put a number into the computer's memory but first you must give the number a **VARIABLE** name.
2. To tell the computer what value you want a variable to have, you must use a **LET** statement.

3. If you want a variable to increase by one each time it is printed, you would write a LET statement with a +1.
4. When you cause the computer to do something forever, this is called an **INFINITE** or **ENDLESS** loop.

CHECKPOINT 8

1. The symbol that stands for EQUAL TO is =.
2. The symbol that stands for NOT EQUAL TO is <>.
3. The symbol that stands for LESS THAN is <.
4. The symbol that stands for GREATER THAN OR EQUAL TO is >=.
5. The symbol that stands for LESS THAN OR EQUAL TO is <=.
6. The symbol that stands for GREATER THAN is >.
7. The symbols referred to in this chapter are called **RELATIONS.**

CHECKPOINT 9

1. Every time a FOR statement appears in a program, there must also be a **NEXT** statement to go with it.
2. FOR/NEXT loops can be used as **TIMERS** to slow down the program.
3. FOR/NEXT loops slow down the computer because it makes the computer **COUNT** to itself.

CHECKPOINT 10

1. In timer loops the NEXT must come right after the FOR statement. Is this true when used for other purposes besides timers? **NO.**
2. What does the computer do when it has finished a FOR/NEXT loop? **IT CONTINUES ON TO THE VERY NEXT LINE AND EXECUTES IT.**
3. In a FOR/NEXT loop as follows:

   ```
   FOR N = 5 TO 60
   PRINT N
   NEXT N
   ```

   What will the value of N be on the fourth time around the loop? **THE VALUE WILL BE 8.**
4. Write a four line program (including an END statement) to make the computer count and print the numbers 150 to 300 by threes:

   ```
   10 FOR C = 150 TO 300 STEP 3
   20 PRINT C
   30 NEXT C
   99 END
   ```

5. Write a four line program (including an END statement) to make the computer count and print the numbers -50 to -150 by sevens.

```
10 FOR C = -50 TO -150 STEP -7
20 PRINT C
30 NEXT C
99 END
```

6. Unless you tell it to do otherwise, the computer will count **BY ONES** in FOR/NEXT loops.
7. Explain why the FOR/NEXT statement is referred to as a "loop". **A LOOP IS SOMETHING THAT KEEPS GOING AROUND, LIKE A CIRCLE. A FOR/NEXT LOOP CONTINUES DOING WHAT IT IS INSTRUCTED TO DO UNTIL IT COMES TO THE LAST NUMBER IN THE FOR STATEMENT.**

CHECKPOINT 11

1. What command can you insert into the program, along with a line number, that will produce a new, clean screen each time it is encountered by the computer? **CALL CLEAR**
2. By using the word **TAB**, along with a number in parentheses, you can alter the placement of information on the screen.
3. Write a statement that puts the word BANANA 15 spaces to the right on the screen:

```
10 PRINT TAB(15);"BANANA"
```

CHECKPOINT 12

1. How many different colors are available on the TI? **16**
2. What is the statement used in programs to give the screen color? **CALL SCREEN**
3. How do you specify what color you want the screen to be in the color statement? **YOU PUT THE NUMBER OF THE COLOR INSIDE PARENTHESES AFTER THE CALL SCREEN STATEMENT.**
4. Where can you put color statements in the program? **ANYWHERE**

CHECKPOINT 13

1. What is a STRING VARIABLE? **A VARIABLE THAT IS MADE UP OF LETTERS OR WORDS.**
2. Write a LET statement using a STRING VARIABLE.

```
10 LET PET$ = "FLUFF"
```

(Your variable name can be anything as long as you use the $ and quotation marks correctly.)

3. What is the opposite of a STRING VARIABLE? **NUMERIC VARIABLE**
4. How would you say this variable in English? NAME$ **NAME STRING**
5. If you want to have a variable to represent a street name, what might you call it? **STREET$** or **STNAME$**
6. What is wrong with each of the following statements?

`10 LET H = "HEIGHT"`	**HEIGHT IS STRING DATA SO THE VARIABLE NAME SHOULD BE HEIGHT$**
`20 LET TIME$ = 3`	**3 IS A NUMERIC VALUE SO THE $ IN THE VARIABLE NAME TIME$ SHOULD NOT BE THERE. THE CORRECT LINE SHOULD BE:** **20 LET TIME=3**
`30 LET NAME$ = MATT`	**MATT IS STRING DATA AND MUST BE INSIDE QUOTATION MARKS. THE CORRECT LINE SHOULD BE:** **30 LET NAME$ = "MATT"**
`40 LET A$ = "1962"`	**THIS ACTUALLY IS OK BUT FOR 1962 TO BE TREATED AS A NUMBER INSTEAD OF A WORD IT WOULD HAVE TO BE WRITTEN** **40 LET A = 1962**
`50 LET PHONE = "5558155"`	**THE NUMBER SHOULD NOT BE IN QUOTATION MARKS SINCE IT IS A NUMERIC VALUE. THE CORRECT LINE SHOULD BE:** **50 LET PHONE = 5558155**

CHECKPOINT 14

1. What does the computer do when it comes to an INPUT statement in a program? **IT PRINTS A ? AND WAITS FOR A RESPONSE FROM THE USER.**
2. What kind of statement should come before each INPUT statement? **A PRINT STATEMENT WITH A QUESTION TO ANSWER.**
3. What does it mean when we say that the computer INTERACTS with the user? **IT ALLOWS THE USER TO PUT INFORMATION INTO THE COMPUTER TO BE USED IN THE PROGRAM.**
4. If you ask a question that will have a numeric answer, the INPUT statement must have a **NUMERIC** variable name.

CHECKPOINT 15

1. What are the two words necessary in the statement that instructs the computer to play a musical note? **CALL SOUND.**
2. What does DURATION mean? **THAT IS HOW LONG YOU WANT THE SOUND TO PLAY.**
3. What does TONE mean? **TONE REFERS TO THE NOTE YOU WANT PLAYED.**
4. What does VOLUME mean? **VOLUME IS HOW LOUD OR SOFT THE NOTE WILL BE PLAYED.**
5. What are the numeric limits for VOLUME? **0 TO 30**
6. What are the numeric limits for DURATION? **1 TO 4250**
7. How do you indicate what TONE you want played? **YOU MUST INCLUDE THE TONE NUMBER IN THE CALL SOUND STATEMENT.**
8. Inside the parentheses, what number comes first? **DURATION** What number comes second? **TONE** Third? **VOLUME**

CHECKPOINT 16

1. What is a SUBROUTINE? **A SUBROUTINE IS A SET OF INSTRUCTIONS THAT CAN BE USED ANYWHERE AND ANYTIME IN A PROGRAM.**
2. What two statements are necessary as part of the instructions to use a subroutine? **GOSUB and RETURN**
3. What must come after the GOSUB statement on the same line? **A LINE NUMBER INDICATING WHERE THE SUBROUTINE IS LOCATED IN THE PROGRAM.**
4. How many times can you use one subroutine in a program? **AS MANY TIMES AS YOU WANT.**

APPENDIX B

MUSICAL TONE CHART

CODE	TONE	CODE	TONE
NUMBER	NAME	NUMBER	NAME
110	A	440	A
117	A#, B♭	466	A#, B♭
123	B	494	B
131	C (Low C)	523	C (High C)
139	C#, D♭	554	C#, D♭
147	D	587	D
156	D#, E♭	622	D#, E♭
165	E	659	E
175	F	698	F
185	F#, G♭	740	F#, G♭
196	G	784	G
208	G#, A♭	831	G#, A♭
220	A (Below middle C)	880	A (Above high C)
233	A#, B♭	932	A#, B♭
247	B	988	B
262	C (Middle C)	1047	C
277	C#, D♭	1109	C#, D♭
294	D	1175	D
311	D#, E♭	1245	D#, E♭
330	E	1319	E
349	F	1397	F
370	F#, G♭	1480	F#, G♭
392	G	1568	G
415	G#, A♭	1661	G#, A♭
440	A	1760	A

Appendix C

CHARACTER CODES

STANDARD CHARACTER CODES

SET #1		SET #2	
CODE #	CHARACTER	CODE #	CHARACTER
32	(SPACE)	40	(
33	!	41	)
34	‘	42	*
35	#	43	+
36	$	44	,
37	%	45	-
38	&	46	.
39	’	47	/

SET #3		SET #4	
CODE #	CHARACTER	CODE #	CHARACTER
48	0	56	8
49	1	57	9
50	2	58	:
51	3	59	;
52	4	60	<
53	5	61	=
54	6	62	>
55	7	63	?

SET #5		SET #6	
CODE #	CHARACTER	CODE #	CHARACTER
64		72	H
65	A	73	I
66	B	74	J
67	C	75	K
68	D	76	L
69	E	77	M
70	F	78	N
71	G	79	O

SET #7		SET #8	
CODE #	CHARACTER	CODE #	CHARACTER
80	P	88	X
81	Q	89	Y
82	R	90	Z
83	S	91	]
84	T	92	\
85	U	93	[
86	V	94	^
87	W	95	-

UNDEFINED CHARACTER CODES

These are the codes which can be defined by the programmer using CHR$ (see Supplement 12).

SET # 9	SET #10	SET #11	SET #12
96	104	112	120
97	105	113	121
98	106	114	122
99	107	115	123
100	108	116	124
101	109	117	125
102	110	118	126
103	111	119	127

SET #13	SET #14	SET #15	SET #16
128	136	144	152
129	137	145	153
130	138	146	154
131	139	147	155
132	140	148	156
133	141	149	157
134	142	150	158
135	143	151	159

INDEX

Prepublication reviewers say:

"Style and level of presentation are excellent . . . it's easy-to-read and very direct, great continuity throughout . . ."

". . . of all the TI kids' books, this one is superior!"

Taking Off with BASIC on the Texas Instruments Home Computer

Nancy Ralph Watson

Now—a comprehensive, self-paced guide to BASIC programming on the Texas Instruments Home Computer, uniquely designed for children and young adults! This book starts with the most simple BASIC statements and commands, then proceeds to more complex ideas through the "building" effect of using just one program theme throughout—the "rocket take-off" program. With this program, children learn all the necessary elements of BASIC as well as the specific character of the TI 99/4A itself. What's more—they'll have fun while learning, too, as they experience the countdown, sound effects, and color along the way—all designed to enhance their "take off" of BASIC knowledge!

Actually two books in one, this guide also contains 17 supplemental chapters that expand the BASIC concepts introduced in the first 17 corresponding chapters. Also includes:

- A thorough listing of musical tones for use of sound capabilities on the TI Home Computer
- A list of the most frequently-used statements and commands with page numbers for quick reference
- A comprehensive character code chart!

CONTENTS

ISBN 0-89303-870-9