How to Feel at Home with a Home Computer

This book answers the question: What are home computers for?

TEXAS INSTRUMENTS

How to Feel at Home with a Home Computer

By:

Gary G. Bitter
Roger S. Walker

With Contributions by:

Gerald Luecke, Managing Editor
Charles W. Battle, Editor
Douglas R. Luecke, Editor
Jacquelyn F. Quiram, Editor

TEXAS INSTRUMENTS

P.O. BOX 225012, MS-54 • DALLAS, TEXAS 75265

This book was developed by:
The Texas Instruments Information Publishing Center
P. O. Box 225012, MS-54
Dallas, Texas 75265

- Jacquelyn F. Quiram is Manager of Instructional Communications for Texas Instruments Consumer Products, Lubbock, TX
- Douglas R. Luecke is a Staff Member, Computer Science Dept., North Texas State University, Denton, TX
- Charles Battle is a member of the Information Publishing Center Staff

For marketing and distribution inquire to:
James B. Allen
Marketing Manager
P. O. Box 225012, MS-54
Dallas, Texas 75265

Acknowledgements:
Gene Sulek for his direction and support in establishing the book format. Ed Lawing, Liz Sanders, Shelley Evenson of Richardson/Smith for their fine work on the format. Roger L. Goodberlet, MA, for his consultations with Gary Bitter. Jon Nathan Walker for his valuable assistance in previewing and using home computer materials. Pete Sanborn for his assistance in checking programs and developing the glossary and index.

Word processing:
Betty Brown
Sharon Appel

Format design:
Richardson/Smith

Design and artwork:
Plunk Design

ISBN 0-89512-097-6
Library of Congress Catalog Number: 83-050901

Notice on Software copyrights:
Refer to the appendix for trademarks and license information for all software referenced in this book.

Table of Contents

Preface

Chapter *Page*

1 **Getting Acquainted** 1-16

Whether you have a home computer or not, this chapter answers questions you may have: What is a user-friendly computer? How does a home computer differ from a personal computer? How can a computer be useful in the home? What is a computer anyway?

2 **Easy-to-Use Software** 1-24

Maybe you've decided that a home computer could be useful for you, but you are wondering if you can set it up and use it. This chapter explains how easy it is to set up, then describes the easy-to-use software cartridges that you just plug into the computer to study math, learn reading skills or play games. Example programs are described in each of the categories of education, home entertainment and household information management.

3 **More About Software and Hardware** 1-16

This chapter has two parts. The section on software describes machine language, which the computer uses directly; talks about the languages that humans use to tell the computer what to do; and how the human languages must be converted to machine language inside the computer. The section on hardware explains the functions of a computer — CPU, memory, input/output, and peripherals that work with it — printer and modem.

4 **Beginning with BASIC** 1-40

Operating a home computer with preprogrammed software is certainly satisfactory, but to enhance its value to you, it should be doing tasks you want done. You must learn how to tell it what to do. This chapter begins the process by teaching you how to instruct the computer what to do using a minimum set of BASIC language instructions. Clear, easy-to-understand, step-by-step examples can be very beneficial to you even though you do not have a home computer. It is more beneficial if you have one.

5 **Steps to Successful Programs** 1-30

After learning the mechanics, this chapter continues the programming process by teaching you how to develop your own programs from an idea to a finished program in five steps. You learn about flowcharts, coding and running the programs. Actual working programs which you can follow step-by-step are presented to show you how.

6 **Creating Educational Programs** 1-36

Because the computer is patient and consistent, it is ideal for drill and practice programs to reinforce what a student has learned in the classroom. Using the instructions and techniques of the last two chapters, text discussion and example programs explain different uses of the home computer for educational purposes.

7 **Creating Information Management Programs** 1-24

Most adults have a lot of information to manage. Some examples are checkbooks, appointment calendars, medical records, and personal goods inventory. The computer, with the proper software, is good at keeping such information organized and sorted. This chapter demonstrates, again with clear, step-by-step examples using the techniques of Chapter 5, how the computer can be programmed to manage such information.

8 **Creating Entertainment Programs** 1-30

As in the previous chapters, the techniques of Chapter 5 are used to demonstrate how the home computer can be programmed to play games — games that are entertaining, or educational, or both. At the same time, an additional BASIC instruction is learned.

9 **Sound, Graphics and More** 1-38

This chapter encourages you to go further in programming. Speech, music and colorful graphics can make many computer presentations more interesting and exciting. Many new BASIC instructions are needed to use these features in your program. As before, these are introduced in examples that are actual working programs.

A **Appendix** A1-2

A partial list of available software for the TI-99/4A is presented.

G **Glossary** G1-2

I **Index** I-1

Preface

Computers seem to be a part of more and more of our daily activities. Now that home computers are readily available and affordable by most people, the need to know something about computers and what they are for increases each day. However, some people feel uncomfortable about computers — they just don't feel "at home."

This book was written especially for these people. It shows what home computers are for, what a user-friendly home computer is, and how easy it is to interconnect a home computer and begin using it. It was written for the person who's just thinking about buying a home computer, and also for the person who already has a home computer, but may not be using it to its fullest capability.

The book begins by getting you acquainted with a home computer. It continues by showing how easy-to-use software (programs) has been preprepared to accomplish specific tasks, whether it be for education, entertainment or information management for the home. Such software has been prepared by experts with much thought and is available in a wide variety of subjects. When you go shopping for a home computer, the salespeople may use words that are unfamiliar to you — computer jargon. A discussion of hardware will help you to learn this jargon and acquaint you with the different functions of the computer.

Many home computer users will be content to operate the home computer only using preprogrammed software. This certainly is satisfactory; however, to get even more benefit from the home computer, the second part of the book shows how you can tell the computer what you want it to do — how to program it. It explains how to get your ideas into a form that the computer understands in order to solve your problems. Clearly illustrated, easy-to-understand, step-by-step structured instructions and examples demonstrate how easy it is to program a home computer using the BASIC language. The examples are working programs in education, entertainment and home information management.

No other electronic system is likely to have as much impact on all levels of our society as the computer. This book will usher you into this exciting age.

G.G.B
R.S.W.

The user-friendly computer

2 **Purchasing a home computer**

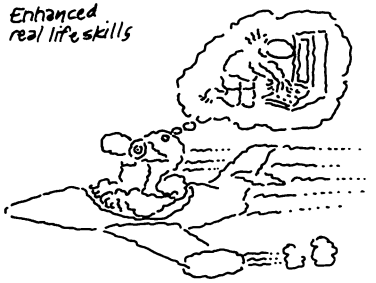
The user-friendly computer

The personal computer versus the home computer

5 **Uses for a home computer**

Entertainment

Enhanced real life skills

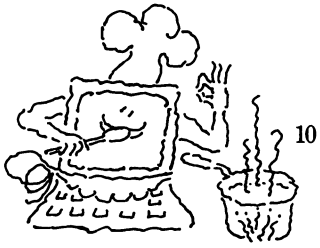


Suggested side benefits
Enhanced real life skills

Education

Drill and practice
Writing improvement
Cooperation and communication
New learning experiences
Adults too

Household needs



Administration

Household needs
Expanded applications

10 **Just what is a home computer anyway?**

A little history

The integrated circuit

13 **The functions of the computer**

Central processing unit

Memory

Input/output

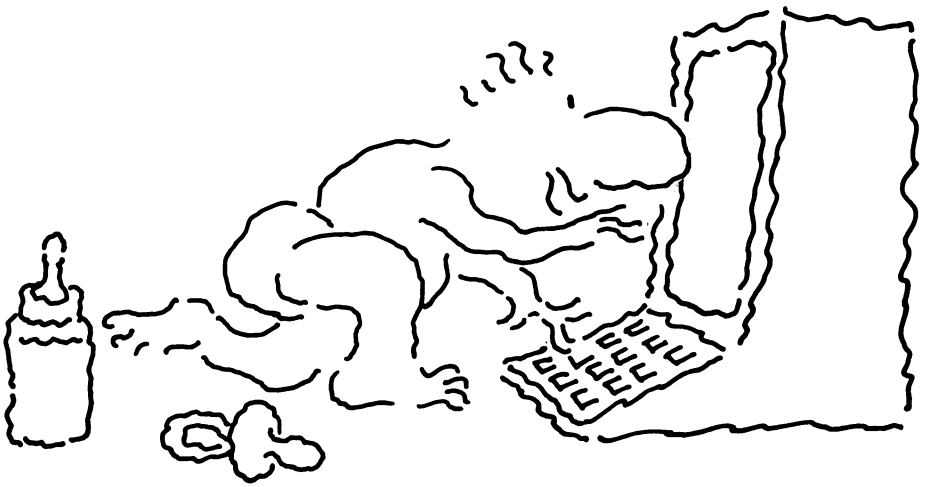
Software

16 **Summary map**

Getting Acquainted

Purchasing a home computer

Everyone seems to be using home computers — from kids to grandmothers.



What is a home computer for?

Why? For games? For maintaining home records? For helping their children in their school work?

How about all of the above!

The home computer is now bringing computer capability to the average consumer. Still, the question most prevalent in the minds of many who have not yet made the purchase is, "What can a home computer do for me?"

We hope to answer not only this question, but also to show you how to get the most from a home computer — and a Texas Instruments Home Computer in particular, as we will be using a Texas Instruments 99/4A Home Computer in our discussions.

First let's examine some common facts and myths about the computer in the home.

The user-friendly computer

A user-friendly computer is one that is easy to use.

Many advertisements talk about the "user-friendly" computer. Well, just what is a user-friendly computer? Very simply, a user-friendly computer is one that is easy to take home, set up and use.

As a person begins to investigate the many small computer systems, one finds just about all of them are described as user-friendly. But beware! What is user-friendly to some might not be user-friendly to you.

The easier it is for *you* to tell a computer what *you* would like for it to do, then the more user-friendly it is. And conversely, the easier it is for *you* to understand what a computer is asking of *you*, the more user-friendly it is.

The personal computer versus the home computer

You probably have heard the terms personal, professional, and home computer applied to the small computers. They all use the same technology and overlap in many of their capabilities, but they are different.

A personal computer usually is more powerful, costs more, and is not as easy to use as a home computer.

The personal or professional computer is often advertised as user-friendly, but typically it is more difficult to use than a home computer. However, it usually can handle larger amounts of data at a faster pace and more options are available to expand its use. Of course, this extra power and versatility will cost you more. The personal or professional computer often is used in small businesses for accounting, financial reporting and inventory functions. The personal computer also can be found in households, but usually one or more of the family members have an occupation closely associated with computers, or a profession, business or application that lends itself easily to computer use. In either location, the personal computer generally requires the user to learn more information in order to effectively operate the computer and get the most benefit from it.

Thus, we see that the personal computer is designed for the computer hobbyist, the small business person, or the professional — the person who is motivated from the beginning to understand more fully how to use a computer.

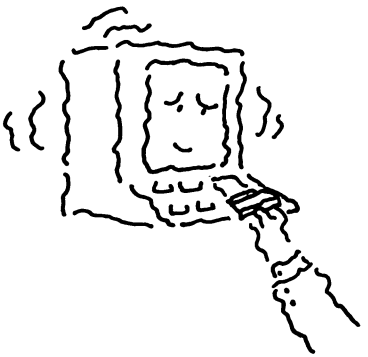
A home computer should be as easy to set up and operate as other home appliances.

The home computer, on the other hand, is designed for — well, the home. The home computer should always be very user-friendly. Whether it's to be used for games, educational purposes, or household administration, it should require very few instructions. A non-technical person should be able to use a good home computer system without extensive study or preparation. (However, a short course, readily available in local areas at learning centers, libraries, local colleges, taken prior to or right after obtaining a home computer may help a person get more benefits from the home computer.)

Anyone from teenager to grandparent should be able to unbox, plug in, and begin to use a well-designed home computer right away. Usually several connections must be made, but if the home computer system is well-designed, the connections can be made only one way. Each plug will go only in its respective socket. You can't make a mistake. Much like the television set, microwave oven, or clothes dryer, the home computer should be a useful and easy-to-operate home appliance.

A wide variety of prepared programs should be available for immediate use.

And what about the programs? The home computer should use preprogrammed application cartridges. These cartridges simply plug into the computer and let you immediately begin to use the program.



A good home computer should offer many application programs in home entertainment, education and household administration. For many of us, using programs that someone else has developed and tested is an efficient, time-saving way to get both immediate and long-term value from a home computer.

You don't have to learn how to write programs, but it's easy to do and it will make your home computer even more useful.

Writing your own programs isn't really necessary unless you have an interest in programming and want to learn how, but doing so will enhance the value of the computer because it expands the application of the computer to the user's needs. If you do want to write your own programs, this book will lead you through the steps, clearly, quickly and easily to show you it is not that difficult and worth a try.

Uses for a home computer

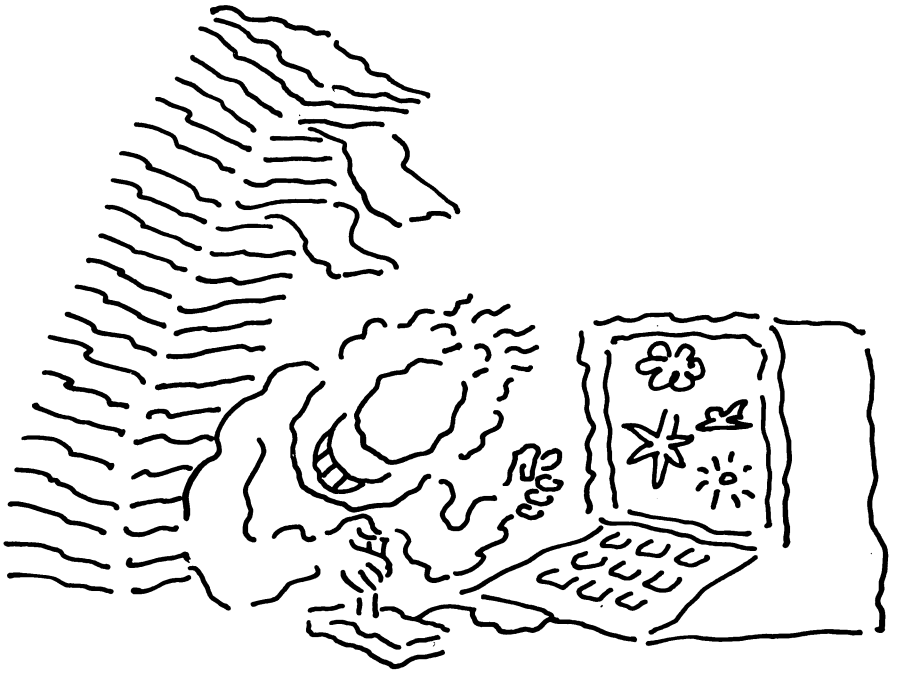
As we've indicated, the home computer can be used for three general functions in the home: entertainment, education and administration.

Entertainment

Many home computers have been sold just for game use. It has been, and still is a good use for the home computer. Most games that you play at an arcade, store, restaurant, etc., use a computer programmed for a specific task. As we shall discuss later, programs are instructions used to tell the computer what to do. These game computers are programmed and designed to play only one game type.

Many games can be played on a home computer while only one type of game is available on a particular arcade video game.

With the home (and also personal) computers you can play lots of different games. Each game will vary in the types of actions or inputs required of you and the types of game responses. Because of this variety and the use for other purposes as well, a more general approach is needed for input actions and the display of the games' responses.



Suggested side benefits

Those of us who have played and enjoyed the current video games recognize that they provide a high level of challenge and entertainment. However, many of these games also provide educational, physical and mental advantages. The use of games by children, for instance, has aided in developing eye-hand coordination. The ability to quickly scan an entire scene while making optimal movements of the playing components, and to recognize various patterns are greatly enhanced in participants. Players are introduced to mathematical and geometric concepts while having fun rather than by reading textbooks. The problem-solving ability of the player is continuously challenged as various strategies and decision-making actions are taken.

Computer games also have proven helpful for stroke victims and other handicapped individuals as they exercise the brain's thinking capability and its ability to cause the muscles to respond. Studies by medical personnel are being made to determine other benefits obtained from games which are played on these small computers.

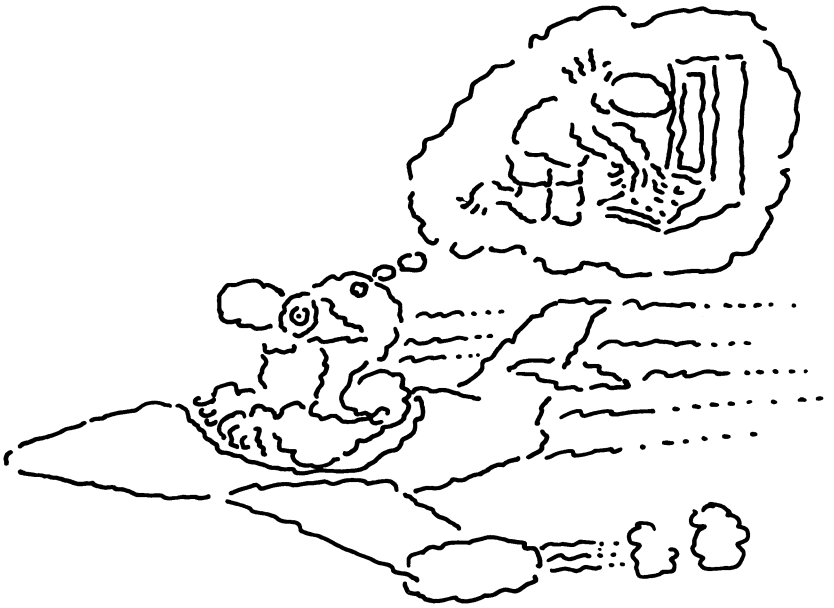
Enhancing real life skills

Video games provide several benefits besides entertainment. Some of these benefits are spatial awareness, planning and eye-hand coordination.

The video games and home computers being developed today are providing a laboratory for testing and developing real life skills — skills which in the past were difficult to develop or improve. Possibly, some of the skills which may result include:

1. The development of spatial awareness which is important for architects and pilots or just for finding your way in a city.
2. The planning of strategies or actions and seeing the effects of such planning.
3. Pattern recognition.
4. The testing of a hypothesis by simulations and role playing.
5. The development of computer operator interaction skills necessary for operating or using the increasing numbers of computer-controlled equipment.

So you see, games are not just for keeping the kids busy or for entertainment. Many hidden benefits may result to both you or your children as these programs are used.



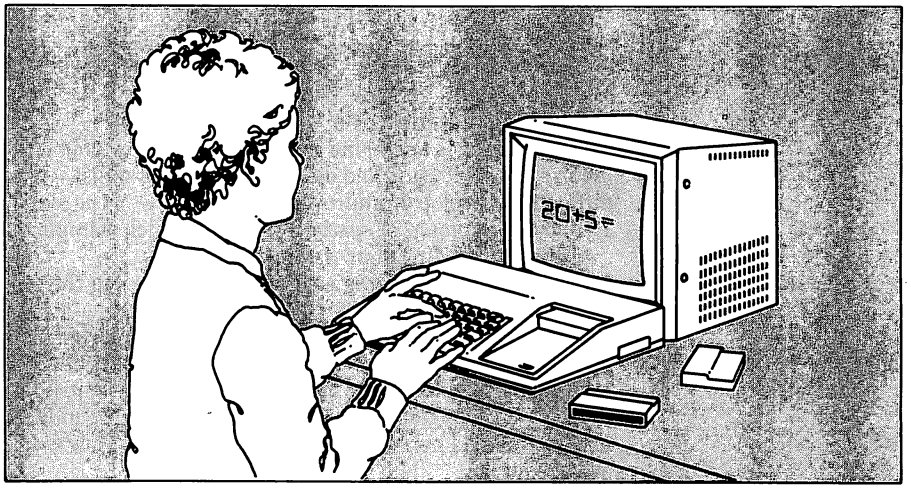
Education

Home computers are being used more and more for educational purposes. A computer never gets tired, or bored, or irritable.

Drill and practice

The computer is especially good for drill and practice because it is patient and never gets tired.

A child can use a home computer for addition, subtraction, multiplication, or division exercises. The computer can praise the child for right answers and give the correct answers when he or she is wrong. Such statistics as a running total of right and wrong answers can be displayed to the child at the end of a set of exercises. The response time of the child — how quickly he or she answers a question — can be used to adjust the score. The score can be given as "excellent", "good" or "you need more practice", rather than 95, 85, 65, etc. The computer provides a way for a person to compete against himself or herself to improve their skills.



The one-to-one interaction between computer and student helps remove the student's fear of failure.

The learning process can be a personal experience between the child and the computer. Although not nearly so important to a child as an adult, the number of times "you need more practice" is displayed will only be known to the individual using the computer. The computer can be turned off or a new program selected and it won't remember earlier performances. In other words, the computer doesn't record a permanent report card for the user. By removing the fear of failure, the computer encourages experimentation which is so important to the learning process.

When a speech synthesizer is added, educational programs can include reading and spelling skills, pronunciation and even learning a foreign language. So useful and economical is the computer for educational purposes that many primary schools are incorporating these machines into their curriculums.

Writing improvement

Studies have indicated significant improvements in the writing skills of students using small computers once the keyboard is mastered. The computer uses word-processing or text-processing programs. The programs are specially written for entering text into the computer's memory similar to what is done when typing it onto paper. The main difference, however, is that once the text is in the computer memory; characters, words, sentences or even paragraphs may be changed, deleted or moved by simple commands. The results of these changes can then be quickly reviewed on the screen or printed on paper. It doesn't have to be retyped by the user.

Word processing programs ease the task of writing and may even improve writing ability.

There are several theories about why improvements in writing ability have resulted from the use of word-processing programs. Some theorize that the improvements occur because we think much faster than we write. By using a simple key press, rather than having to write each character by hand, the expression of ideas is speeded up. Then, changes may be made quickly and simply and the results immediately made available with the TV monitor or printer. In other words, the user has more time to concentrate on what is to be said, rather than on how it's recorded.

Cooperation and communication

Peer teaching occurs many times when children use computers.

Other educational benefits which have resulted from children's using computers is their tendency to work more closely with one another. They have a common tool, the computer. They help each other in determining how to run the computer or use it to solve various exercises. In many cases the peer interaction can be a greater learning experience through discovery and trial and error than straightforward instruction. The computer also interacts, or "talks back" to them. Children enjoy an immediate reaction or response from a computer; a book doesn't talk back. This results in active rather than passive learning.

New learning experiences

Colorful graphics help accentuate the use of the computer for educational purposes. One relatively new programming language called LOGO, developed at Massachusetts Institute of Technology, initially for much larger computers, is now found on some small computers. This language, well suited for graphics as well as many other applications, creates an open-ended learning environment in which the child "teaches the computer" to perform complex graphics routines.

With such programs and a home computer, the children of today are beginning to get a far different type of educational learning experience. Soon all children will be exposed to computers in the primary and secondary schools. Meanwhile, however, children who use computers at home have a definite educational advantage.

Adults also can benefit from educational programs on the home computer.

Adults too

There are many educational programs for adults. Touch Typing Tutor™, a popular cartridge for the TI Home Computer, is used to either learn touch typing or improve on current typing skills. Also available is a BASIC tutorial program which teaches the user the fundamentals of the BASIC programming language. Other programs such as Video Chess™ challenge the adult user. These programs and others presently available, and many others to come, make the home computer a very valuable educational resource.

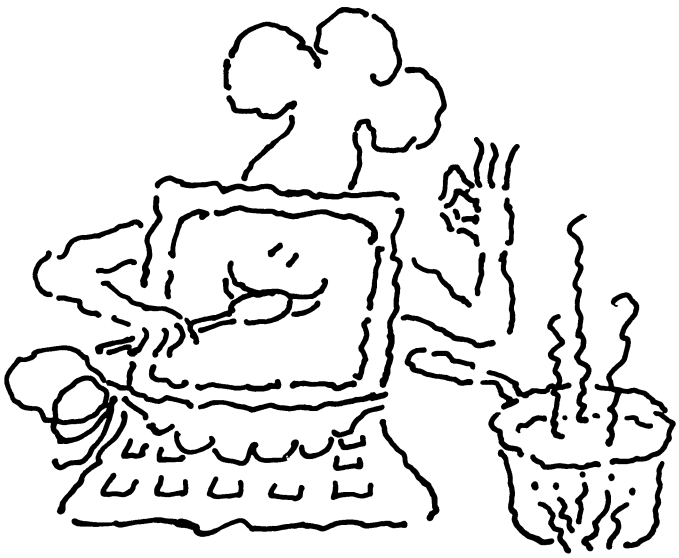
Administration

Probably one of the least developed applications of home computers today is for the household applications of home finance and household administration. Sure, you can use a computer to add or subtract numbers, but you can also use an inexpensive calculator for this. Many engineering, scientific and business application programs can be obtained for a home computer, but some of these likewise can be obtained on the more sophisticated calculators.

Record keeping and electronic filing on the home computer can help you with check records, budgets and tax records.

Household needs

But how about keeping records for medical or food expenses, balancing checkbooks, writing letters, or providing details of other various expenses? There are many household needs that a home computer can do. However, these applications have been developed somewhat more slowly than in the entertainment and educational areas. Probably one major reason has been in the unavailability until recently of any machines or programs which could be used economically for such purposes by the general population. You may not realize it but a home computer must have significant computer power and capability to handle a wide variety of these home application programs. It cannot be overemphasized to stress the computer power for the value that one gets with the Home Computer. Programs now are available for keeping track of your checks, maintaining files for income tax purposes, working out budgets, word-processing, determining specific ingredient amounts for food preparation for different numbers of people, and on and on.



In the future, a home computer will be considered a necessity.

Expanded applications

Eventually, the home computer will become an item we can't do without, much like air conditioners and washing machines. Future uses of the home computer may include such things as controlling home appliances and monitoring their energy usage so as to minimize costs. They also could provide centralized security or climate control. Wouldn't it be nice to have the computer automatically place orders for groceries or pay your bills? Maybe someday they will clean your house! One thing is certain — we can expect many more exciting uses of computers in our homes in the very near future.

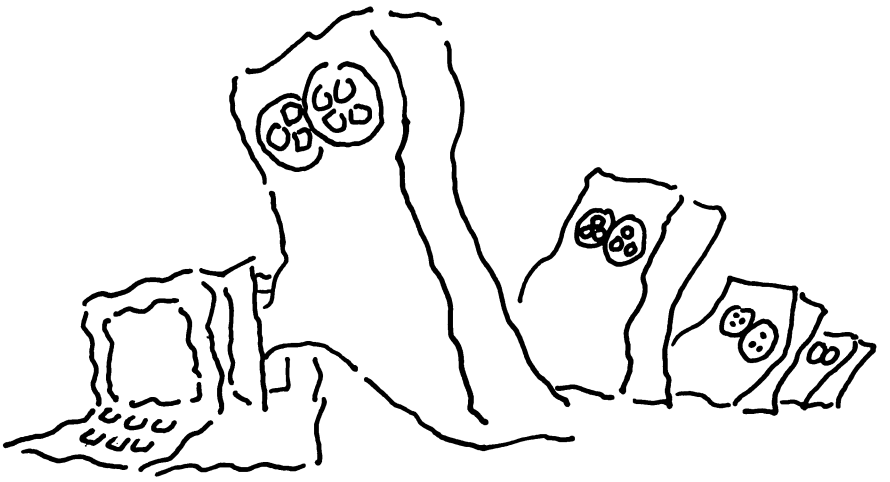
Just what is a home computer anyway?

Up to now we have talked about some of the things a home computer can do for you. It can provide entertainment which has many hidden educational and physical benefits for the whole family. It can help your children in their school work, and it can make many of your household responsibilities easier. You can use programs already written, or if you want, you can write your own programs for other application needs. With so many benefits how did such a thing come about? What made it possible? How did it evolve?

A little history

As you probably are aware, computers have been a major part of our lives since the 1950s. Our paychecks and our monthly bills often are obtained from computers. Banks use them for processing our checks. The IRS processes our taxes with computers. You may have never actually seen or used a computer, but they are used extensively in businesses, universities and governmental agencies. You may have a picture of a computer as filling a large room or even several rooms. And there are such computers today. What you may not know, however, is that the computers of the 1950s are not any more powerful than your home computer. That is, you can go to a local department store and purchase for about a hundred dollars a machine which thirty years ago would have cost in the millions. Furthermore, you can plug in a program cartridge and begin to use your home computer immediately. The multimillion-dollar systems of yesterday would have required a number of highly-trained and highly-paid technical personnel to set up and operate the equipment to provide the same capability.

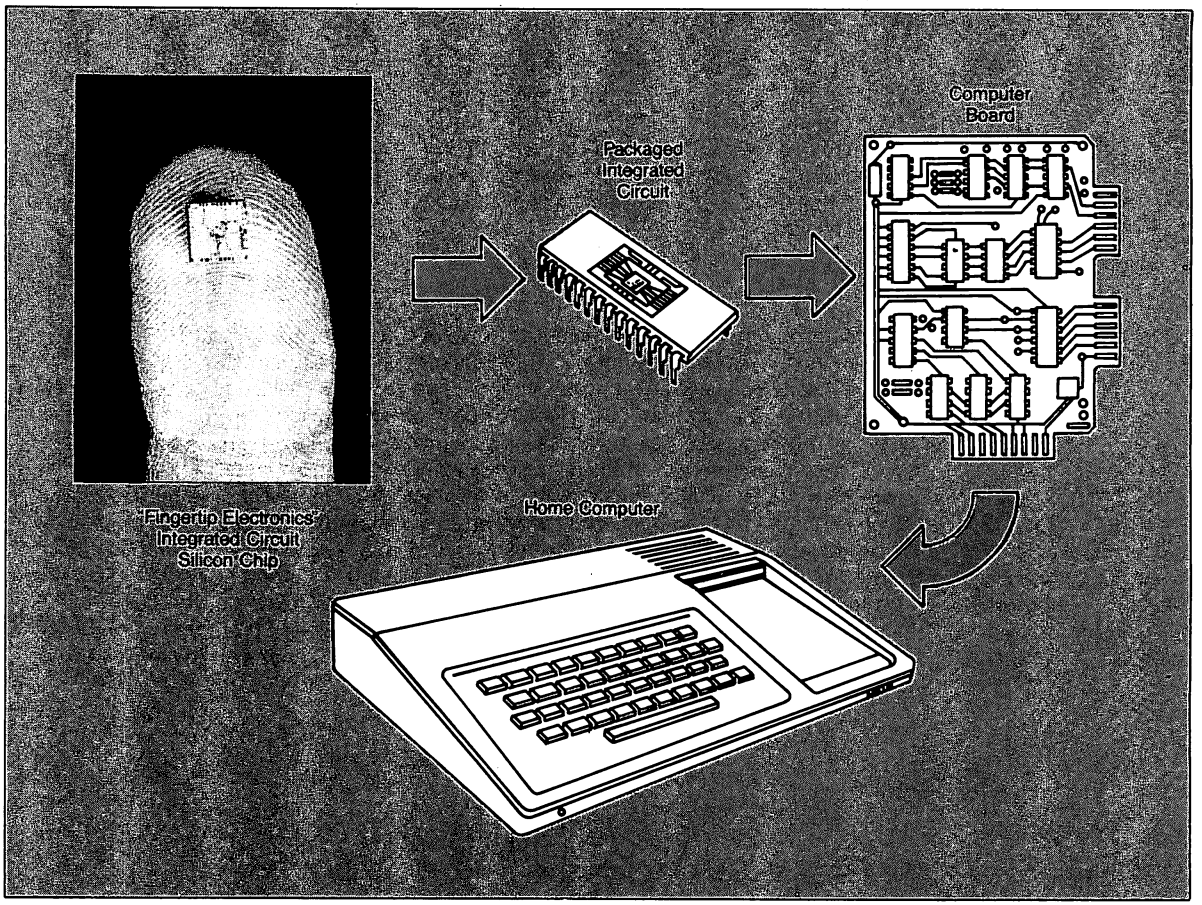
The computing power provided by a $100 home computer of today cost millions of dollars just 30 years ago.



The integrated circuit

The integrated circuit is the key element behind the development of today's small but powerful computers.

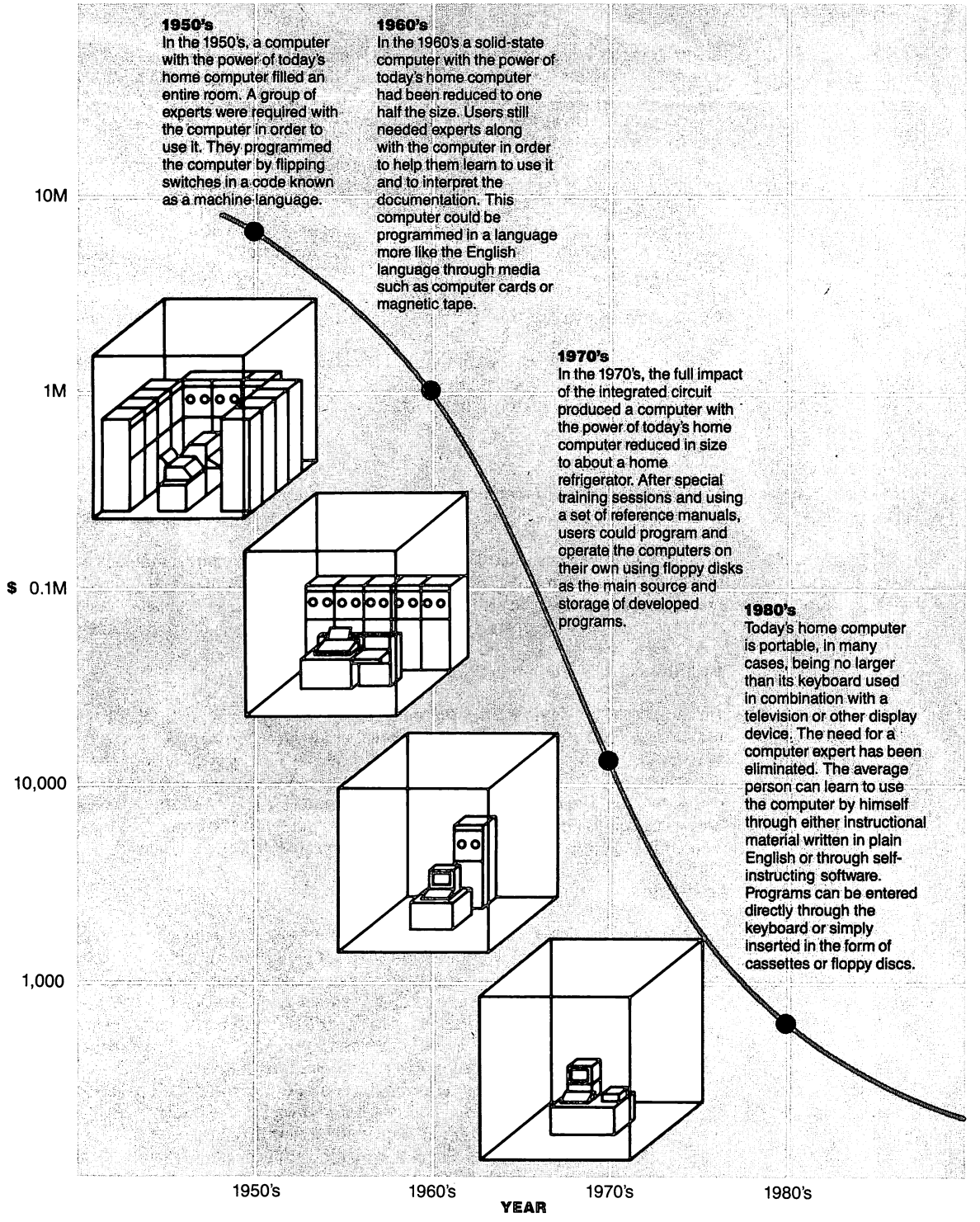
So what has happened in thirty years to make this possible? The answer is in the usage of a small piece of silicon material called the integrated circuit. By using highly sophisticated chemical processes, the large-scale integration of computer circuitry has been achieved to produce miniature computing modules or chips. These chips, when put together with others, have enabled the development of small powerful computers that are relatively inexpensive. And, when these small computers became "user-friendly" to the non-technical person, the home computer was the result.



"Finger tip electronics" — the IC chip — has made possible the size and price reduction in computers.

Thus, in thirty years, we have seen a transition from computers made of vacuum tubes and other separate components, to the current ones which use large-scale integration of computer circuitry on tiny silicon chips.

Along with size reduction, computers have become faster, more powerful, more reliable and less expensive. We are witnesses to a technological revolution which many have predicted will be of greater significance to our society than was the industrial revolution.



The reduction of size and cost of a computer with the same capability.

The functions of the computer

There are four major functions in any computer.

Even though the construction process has changed drastically since the first computers, the major functions have not. The four major functions within a computer are the central processing unit or CPU, the memory, the input/output devices and the programs. These are discussed in more detail in a later chapter, but we will briefly introduce you to them here.

A computer system depends on both hardware and software.

The actual electronic and other parts are referred to as "hardware." The programs which direct the computer to perform various tasks are referred to as "software". For the TI-99/4A Home Computer, much of the available software is contained in program cartridges that are plugged into the computer. Once plugged in, the programs in these cartridges direct the computer to perform application activities.

Central processing unit — CPU

The CPU controls the other computer components and performs calculation in accordance with a program.

The CPU is the brain of the computer. It decodes each instruction and directs the other components to perform the specified activities. It also contains an arithmetic unit similar to a calculator which performs arithmetic operations.

Memory

The program and data are kept in memory.

The computer must remember program instructions and intermediate results of calculations as well as other supporting data. That's why the computer must have a memory.

This memory is similar to ours. We can store things in our memories or recall things from our memories. So can the computer. The computer's memory can forget, however, everything it has stored if you turn off the power — unless it has stored this information in a more permanent storage medium such as a magnetic tape or disk.

Input/output — I/O

Input and output devices provide a way for humans to communicate with a computer.

Input and output devices (in computer jargon commonly called "I/O") provide a way for us to use the computer. These devices include keyboards, program cartridges, tape cassettes, disk drives, game paddles, and the TV set or monitor. The input devices provide us a means to enter commands and preprepared programs to the computer, and the output devices provide a means for the computer to respond. As noted earlier, the computer can actually talk to you with the Speech Synthesizer. This unit converts words and letters to the appropriate voice sounds.

Software

A computer can't do anything without the instructions provided by the program.

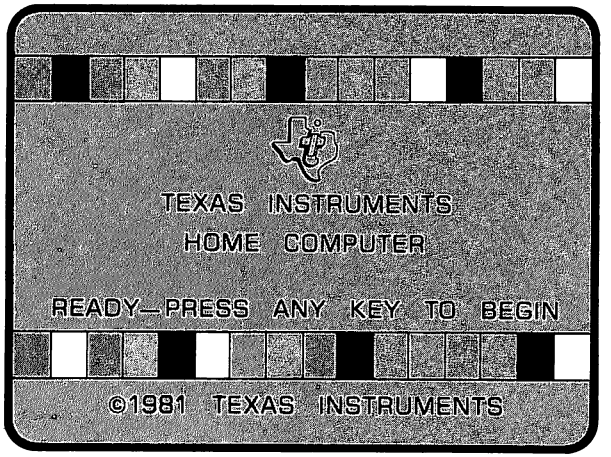
As discussed earlier, a program is a set of instructions directing the computer. The computer will do exactly what it is told to do. It cannot do anything on its own. It must be directed every step of the way by instructions set up by humans. It's often been said that the computer has about the same level of intelligence as a worm — that is, very little! However, once told what to do, the computer can do its tasks faster and more accurately than you or I could. It continues through its task, following each instruction diligently and in order, effortlessly and error-free.

BASIC is a computer programming language used on many home computers.

Developing a set of instructions for the computer is called "programming." Programming includes defining exactly what you want the computer to do and then writing the instructions for doing it. The instructions are written in a language that the computer can understand. There are many computer languages; one commonly used language is called BASIC, for Beginners All-purpose Symbolic Instruction Code.

ROM contains permanently stored programs. Some ROM is built into the computer.

Programs may be entered from the keyboard, or read from the computer memory or a more permanent storage device. Electronic memory that contains permanently stored programs is called read-only memory (ROM). The information stored in this type of memory is built in and is not to be changed. Also, it is not lost when power is turned off. If it wasn't for ROM inside the computer, nothing would happen when you first turned on your computer. For the TI-99/4A, for example, the screen is displayed when you turn the computer on because of instructions stored in ROM.



Initial screen on the TI-99/4A

The plug-in preprogrammed cartridges are made up of ROM.

Program cartridges are removable ROM. The ROM is manufactured on silicon integrated circuit chips just like the inner electronic parts of the computer. They simply plug in and direct the computer to perform a specific application. For the TI-99/4A, as we have stated, there are many program cartridges for educational programs, games, home administration and other applications.

Programs can be purchased that are stored on magnetic tape cassettes and magnetic disks.

Also, you will find that programs are available that are stored on magnetic tape cassettes that look exactly like the music cassettes, or on magnetic disks called "floppy disks" that are about the size of a 45 RPM phonograph record. The cassettes can be played on a standard cassette player in order to enter the program in the computer. A special disk drive must be used to enter the programs from the floppy disk into the computer. The computer can also store programs that you may write on the cassette tapes or on the floppy disks.

Floppy disk and cassette tape

In summary, we have found that a home computer, to be a home computer, should be user-friendly — easy to put together, easy to get started and easy to use. It is suggested that it has the possibilities of helping with the children's education — and be fun. It is suggested that it provides entertainment — and possibly additional side benefits can be gained. It is suggested that it will be an aid to many of the financial decisions and home information and administration tasks that each household requires. It is suggested that its value will be enhanced if you will learn just the elementary steps of how to instruct it to do what you need to do.

These suggestions will be investigated in more detail as the chapters proceed. First a look at the preprepared software that helps to make the Home Computer so easy to use, then a little more detail about the actual hardware so you can get some of the terms and jargon down pat, and then to show you how easy it is for you to get started giving instructions to the computer yourself — using step-by-step examples. You'll see what hardware looks like. How to connect it together. How to get it started. You'll be into the excitement of the computer age before you know it. So let's get going.

Purchasing a home computer

The personal computer may be more powerful, but it costs more and is not as easy to use.

The user-friendly computer

One that's easy for *you* to use.

The personal computer versus the home computer

Uses for a home computer

Develop eye-hand coordination, decision-making ability and spatial awareness while having fun playing games.

Entertainment

Suggested side benefits
Enhanced real life skills

Education

In most areas of learning, the computer can provide routine drill and practice, or whole new ways of learning that are exciting and fun.

Drill and practice
Writing improvement
Cooperation and communication
New learning experiences
Adults too

Administration

Taxes, budgets, recipe files, check records, and just about any kind of records can be organized and maintained on the home computer. In the future, it may automate your whole house.

Household needs
Expanded applications

Just what is a home computer anyway?

A little history

The integrated circuit

Just 30 years ago, it took millions of dollars and a room full of equipment to do what the desk top computer can do. The IC is the invention that made it possible.

The functions of the computer

Central processing unit

Memory

Input/output

Software

Both hardware and software are required for a working computer.

Summary map

2 **Can you use a home computer?**

Setting it up

Getting started

Available software

6 **Educational software**

Programs for reading

Programs for mathematics

Other programs

PLATO® programs

Program descriptions

Reading Flight
Alligator Mix
Earling Learning Fun
TI LOGI II

Home entertainment

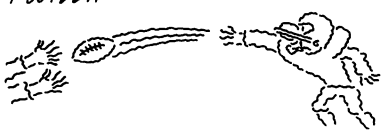
12 **Home entertainment**

Game programs

Program descriptions

Parsec
A-MAZE-ING
Football

Football

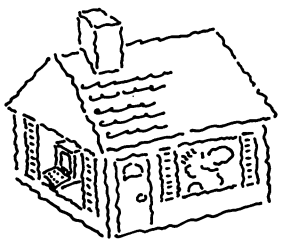


16 **Information management for household administration**

Program descriptions

Home Financial Decisions
Touch Typing Tutor™
Teach Yourself BASIC
Household Budget Management
Microsoft™ Multiplan™
TI Writer
Other programs for information management

Household Budget Management



23 **Summary**

24 **Summary map**

® ™Refer to the appendix for trademarks and license information for all software referenced in this book.

Easy-to-Use Software

Can you use a home computer?

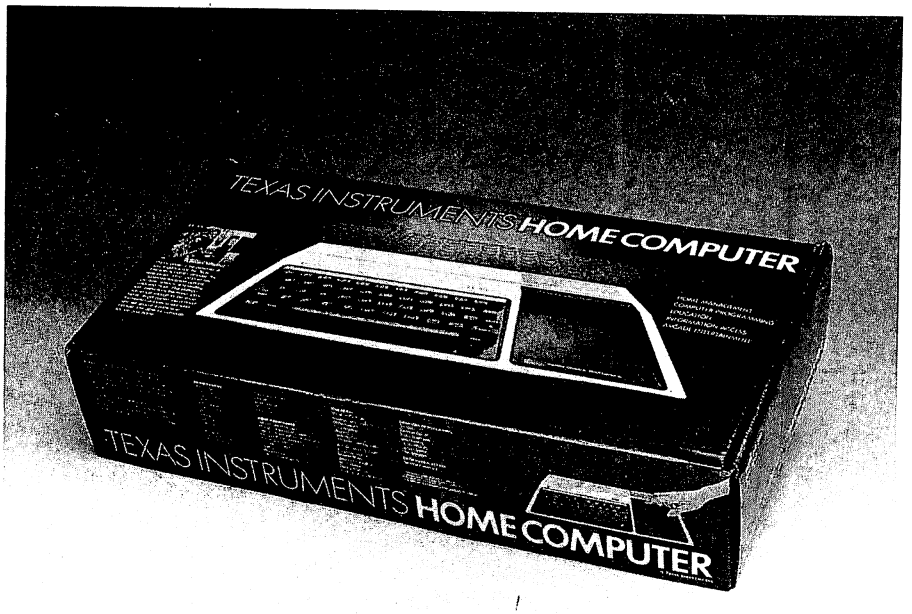
You can do it!

So maybe you are convinced that a home computer can do things for you, but you're wondering if it is possible for *you* to connect it together and use the available programs. The answer is yes. Let's look at how simple it is.

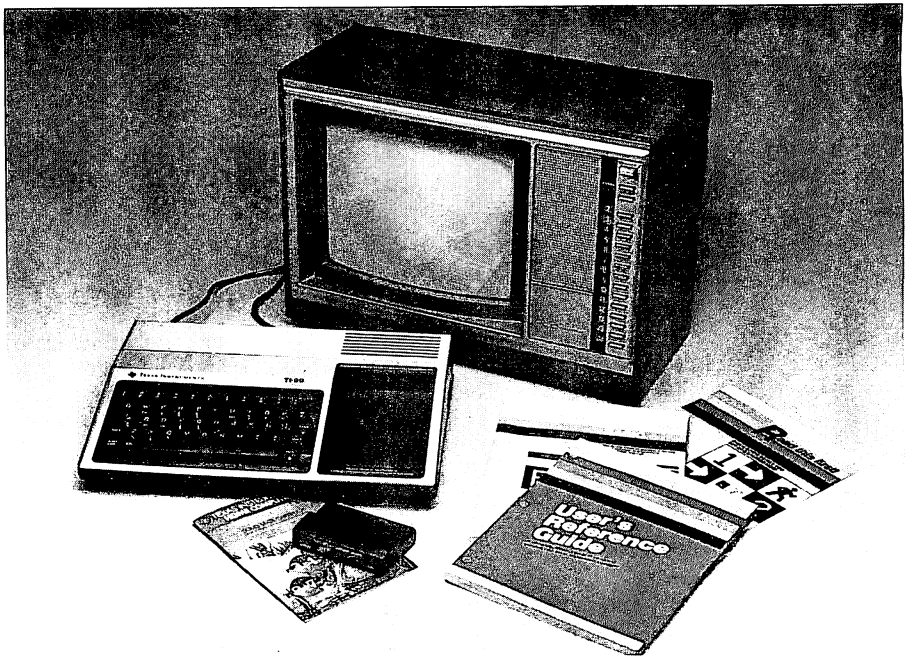
Setting it up

It's as easy as 1-2-3.

Let's suppose that you have purchased a TI-99/4A Home Computer. The basic TI-99/4A is quite easy to set up. It consists of only a keyboard and computer console, a power transformer with attached cables, and a RF Modulator. These parts come in one carton and all you need to do is unpack them, connect them together, and plug them into a standard 120 volt ac wall outlet. Instruction manuals also are included in the carton.

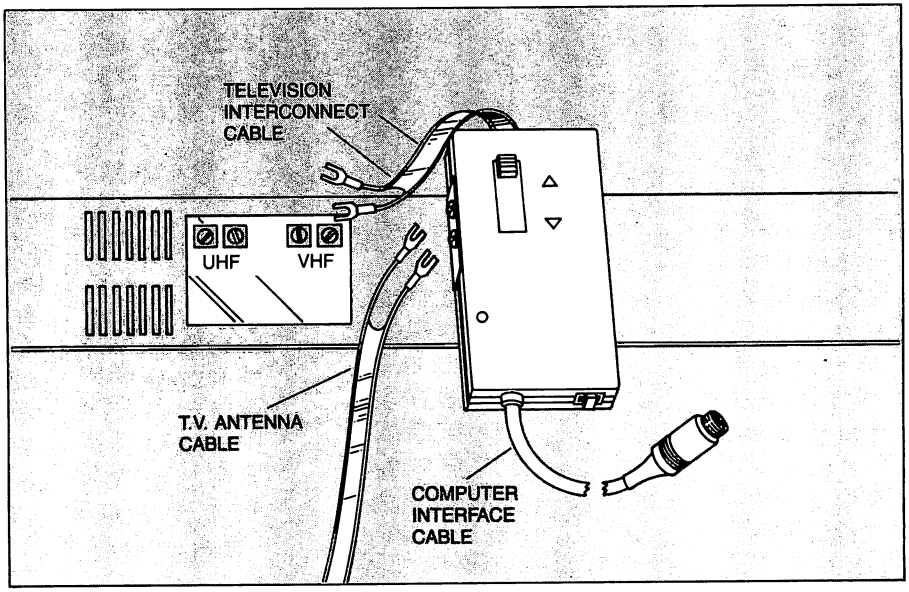


TI-99/4A in carton



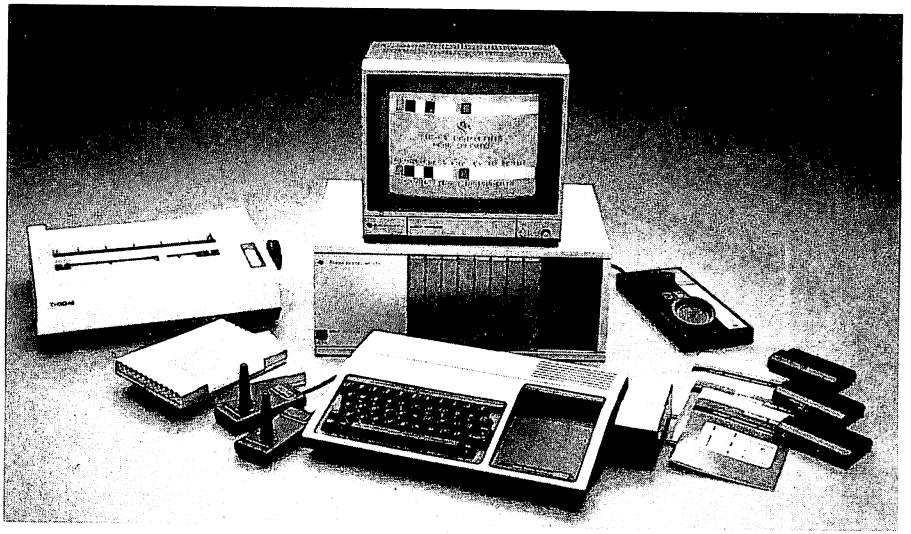
TI-99/4A home computer, manuals and TV

The RF Modulator connects to the antenna terminals of the TV set and to the back of the computer as shown in the figure. The RF Modulator is needed to convert the computer signals to signals that are compatible with your TV so the TV can be used for the output display.



Connecting the RF modulator between computer and TV

This system is enough for you to start using preprogrammed software in self-contained cartridges. However, as you grow in the use of the computer, it can be expanded from this basic system to meet your requirements by adding a cassette tape recorder or disk drive for program and data storage, a peripheral expansion unit with more memory, and a speech synthesizer.



An expanded home computer system

Getting started

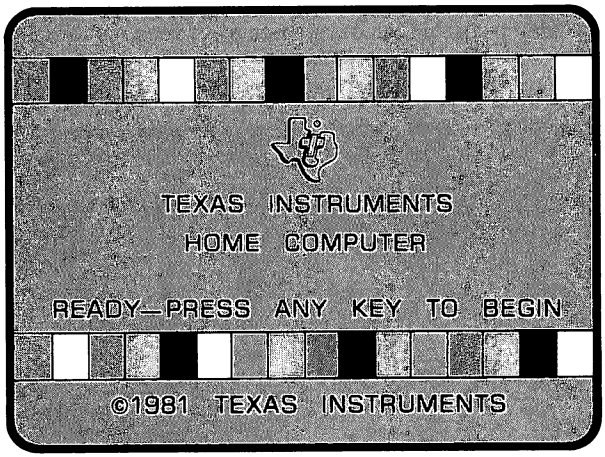
Just plug in a cartridge and you're ready to play.

One of the first things that many people want to do with a home computer is to play a game. Suppose you have selected Parsec[1], a popular computer game. All that is required is to insert the game cartridge into the slot of the computer.

Inserting a solid-state cartridge

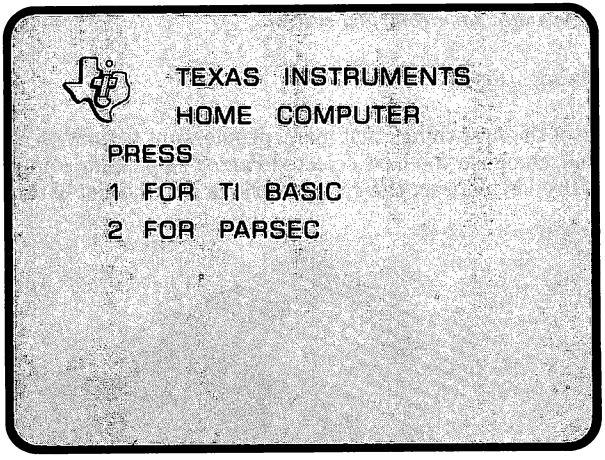
[1]Refer to the appendix for trademarks and license information for all software referenced in this book.

Now turn on the TI-99/4A and the title screen is displayed.



Title screen

The screen instruction tells you to press any key. After you press one of the keys, a selection screen called a "menu" appears. By following the instructions on the screen, you can choose to play Parsec or you can choose to use BASIC. Since you want to play Parsec, press the 2 key. The program selected is then initiated.



Menu screen

Available software

Much software is available in solid-state cartridges.

One of the things that makes the home computer immediately useful to you is the preprogrammed software such as the Parsec game. Preprogrammed software is easy to use. It provides useful and user-friendly programs that have already been written so your home computer can be of immediate benefit to you and your family.

You may have bought only one or two cartridges when you purchased your home computer, but if you're like most people, you'll soon want more. The store has a list of all the available programs and a partial list is in the appendix of this book. For the TI-99/4A, much of the preprogrammed software comes in the form of Solid State Software™ cartridges. Each one will be just as easy to plug in and use; and as shown previously, you will be directed by screen instructions through the steps required to use it. Each time you use a cartridge, check the information printed on the cartridge box to see if you have the equipment necessary for that program. For example, some cartridges require more memory.

The preprogrammed software can be classified into the three areas; education, home entertainment and information management. Let's discuss examples of software in each of these areas, and some of the benefits this software can provide to you and your family.

Educational software

CAI is now available for the home.

The computer has been actively used for education for sometime. Control Data Corporation was one of the first major computer companies to design and sell computer-based educational courses on a large scale. Computer Aided Instruction (CAI) is the popular term used for such instruction. The series of courses, named PLATO®, was originally designed for large mainframe computers and covers a wide range of topics for students in grade school, high school and college. Now, with the home computer, this form of computerized instruction can be brought into the typical household at a very nominal price. To meet set learning objectives, such computerized instruction is designed with the aid of leading educators and in cooperation with major educational software companies. A wide range of subjects are covered. Let's look at some examples and their learning objectives. For the TI-99/4A, the following are available in Solid State Software cartridges.

Programs for reading

Learning to read is more fun with these programs.

Some of the widely used programs for reading include:

1. Early Learning Fun - a program designed for ages 3 through 6 to help teach shape, letter and number recognition, as well as counting, sorting and the alphabet.
2. Beginning Grammar - introduces the basic parts of speech and how they are used to build sentences. It is developed for grade levels 2 through 5.
3. Early Reading - uses the combination of color graphics and speech via the optional Solid State Speech™ Synthesizer to introduce and reinforce important reading skills.
4. Reading Flight - teaches how to classify, summarize and outline information at a sixth grade level.
5. Other available cartridges for reading development include Reading Fun, Reading On, Reading Roundup and Reading Rally.

Programs for mathematics

Some of these programs use colorful animated graphics to make the drill and practice a fun experience.

1. Number Magic — provides drill and practice in basic mathematics for ages 6 and up.
2. Addition/Subtraction 1 and 2 — provides drills for reinforcement of basic addition and subtraction skills. The program uses the optional Solid State Speech Synthesizer to provide a personal approach to learning.
3. Multiplication 1 — uses the optional Solid State Speech Synthesizer to teach the basics of multiplication for grade levels 3 and 4.
4. Division 1 — teaches the concepts and facts of division using speech, animation, color and graphics. Although designed for grade levels 3 through 5, it offers enrichment material for earlier grades and remedial material for later grades.
5. Alligator Mix — provides practice for addition and subtraction problems and discrimination skills in an animated game-like environment.
6. Other available programs for mathematics include Numeration I and II for grades 1 through 6, Computer Math Games I through VI for grade levels 1 through 9, Alien Addition, Minus Mission, Meteor Multiplication and the list goes on and on.

Other programs

Programs also are available for spelling, learning Spanish, learning and composing music, vocabulary improvement, physical fitness, bridge bidding, and learning how to type. TI LOGO II also is available on the TI-99/4A. This is a language which is perhaps easier than BASIC and is very useful in teaching programming and graphics.

PLATO programs

If an expanded system is used that has a disk drive, many of the PLATO program series are available for the TI-99/4A. The TI PLATO series offers 108 courses. Some are in basic skills for grade levels 3 through 8 in mathematics, reading, and grammar. Other courses are in high school skills for grade levels 9 through 12 and adults who want to brush-up on their skills. These high school level courses cover mathematics, practical reading, writing, science and social studies.

Program descriptions

Descriptions and demonstration of these and other programs can be obtained at one of the many department and computer stores that sell the TI-99/4A Home Computer. But before you go and look, and so you will understand how the prepared software meets the learning objectives, let's spend a few minutes discussing what you might expect from three very popular educational programs.

This program teaches how to classify, summarize and outline.

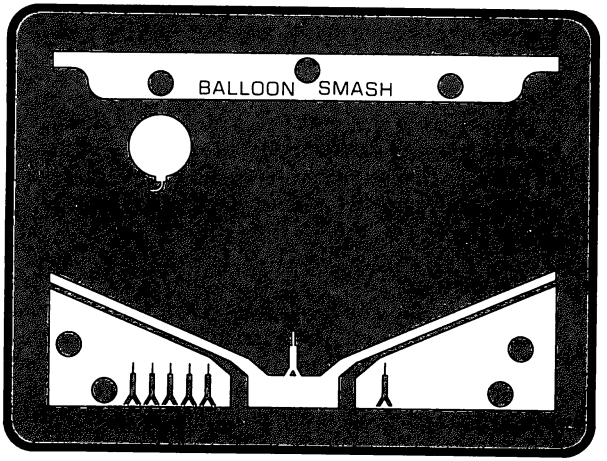
Reading Flight

Reading Flight was designed to teach the user how to classify, summarize and outline information. The user may select from seven activities. There are two activities for each of the three functions; classify, summarize and outline, and a separate seventh activity that includes all three. The first activity is always "study it", and the second one is to "try it out". Thus, for each function, you are given a chance to study what the function is, and then a chance to use it. The functions are applied to children's short stories.

In the study activities, the learning process is reinforced by a short test. For instance, in the case of the first function, classifying, the test consists of identifying related objects which will endanger a space adventure. The user must classify those objects which can be destroyed by the ship's ray gun, and those which must be deflected. Then the "try it out" activity provides a drill to see how well you have learned the function. If you are successful in the "try it out" activity, as indicated by correctly answering at least seven out of ten questions, then you are given the opportunity to play Balloon Smash. In this game, you shoot darts at balloons which move across the screen.

In the seventh activity, the user must apply the three functions of classifying, organizing and outlining to a story about an archeological dig on a remote island.

This popular program is very useful for teaching children the important concepts of classifying, organizing and outlining. The stories used and the reward of getting to play Balloon Smash are helpful in maintaining the child's interest.



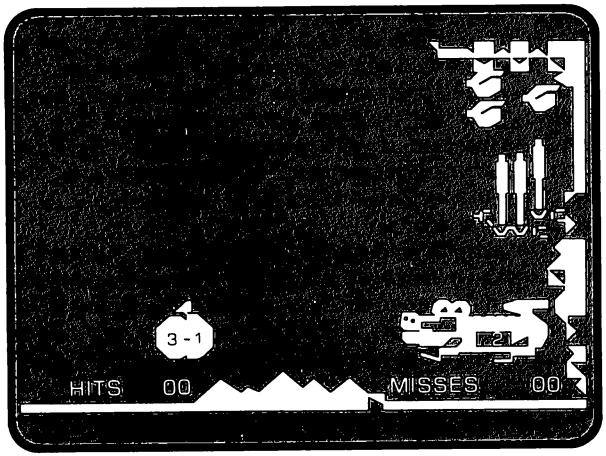
Balloon Smash activity

If the answer is correct, you feed the alligator an apple.

Alligator Mix

Another popular educational program is Alligator Mix. It teaches basic addition and subtraction at a level for children in the second and third grades. This program offers a unique way to improve math skills. The game starts with an apple that has an addition or subtraction problem placed on the apple and a number placed on an alligator. As the apple is moved toward the alligator's mouth, the user controls the opening and closing of the alligator's mouth with the space bar or the fire button on the joystick controller. If the answer to the problem on the apple is equal to the number on the alligator, then the user must open the alligator's mouth to eat the apple. If the solution is not equal to the number on the alligator, then the alligator's mouth must be kept closed. More alligators and apples appear as problems are answered correctly.

The user must decide whether or not to open the alligator's mouth in a short period of time; thus, the user must quickly determine the answer to play the game successfully. The user may change the skill level, the problem range, or the total run time of an exercise. The total number of hits and misses are displayed in the swamp at the bottom of the screen.



Alligator mix activity

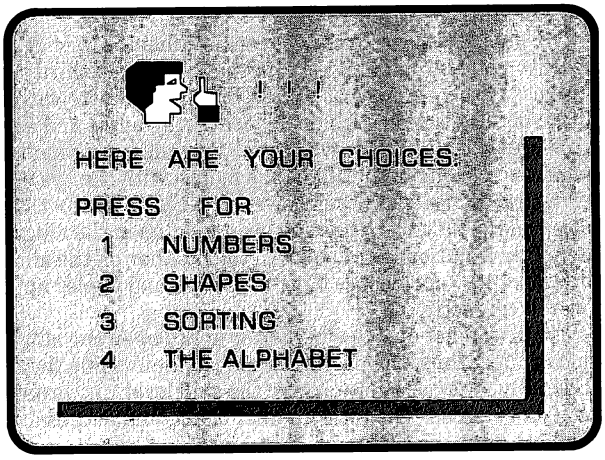
Alligator Mix is useful as a method for teaching addition and subtraction. It also requires the user to read the problem and answer and react quickly. Thus, the program is useful not only for improving arithmetic skills, but also it helps improve decision-making ability.

Early Learning Fun

A fun way to learn shape, number and letter recognition.

Yet another very popular educational program developed by TI for the TI-99/4A is Early Learning Fun. It was designed for the preschool child, ages 3 through 6. It specifically teaches the child shape, number and letter recognition, as well as counting, sorting and the alphabet. An additional benefit is that children at a very young age are introduced to a computer and learn how to interact with it. The bright colorful displays really fascinate children of this age which helps to keep their attention.

The program menu lets the user select activities about numbers, shapes, sorting and the alphabet.

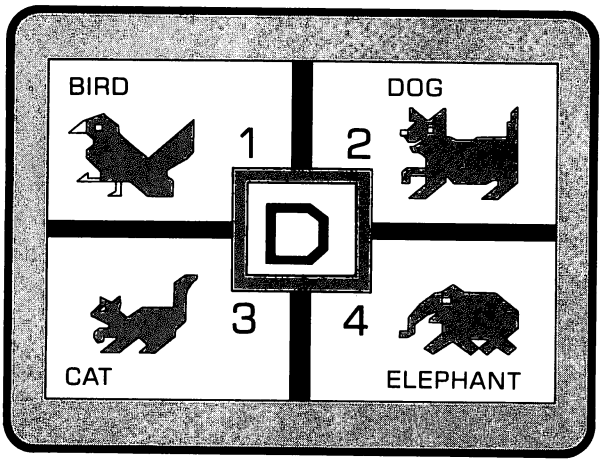


Early Learning Fun menu

When the shape activities are selected, the child is given a group of numbered shapes on one side of the screen. Displayed on the other side of the screen is a shape like one shape in the group. The child must then 'make a match'. The selected match is indicated by pressing a number key on the keyboard. A correct match occurs when the number of the shape in the group is the same as the shape by itself. A sound like 'uh-oh' is made when a wrong answer is selected, but a short tune is played when the answer is correct. When a shape is selected, the shape by itself moves over close to the selected shape so the child can make a better comparison. This is particularly useful if the wrong shape is selected.

In the sorting skills activities, the child is given four shapes of which three are identical and one is different. The child must identify the one that is different.

In the number activities, the child is asked to relate a number of shapes with the number. Another portion of the program permits the child to learn and identify objects that begin with a particular letter. This activity builds a vocabulary of sight words related to the letters of the alphabet.



Early Learning Fun activity

In all activities, the computer continues over and over again as long as the student wants to practice. It never gets tired, bored, or irritable. In summary, this educational program helps to teach basic skills useful in everyday life as well as building a base for further learning. A child who uses a computer in this manner at such an early age will not fear the computer.

TI LOGO II

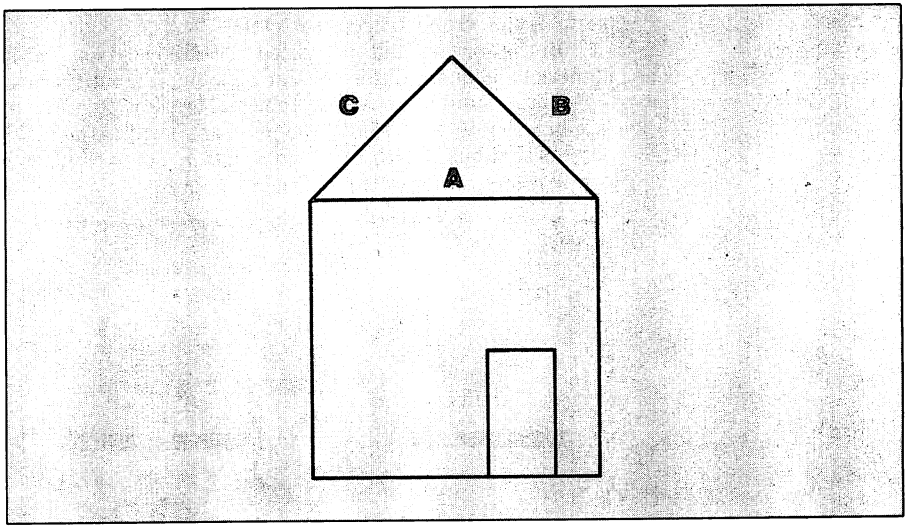
Logo is more than a language — it's a learning tool.

LOGO is a useful learning tool for programming and using the computer. It was initially developed about 15 years ago and now is available for the TI-99/4A under the title, TI LOGO II. Although many people have been involved with the development and improvements of this language, the major contributor was Seymour Papert, who for many years was director of the LOGO Group at the Massachusetts Institute of Technology (MIT).

Much different from the concept used in BASIC, in LOGO the computer is used as a control device. The LOGO concept uses a figure called a turtle or robot which the user commands. Commands include words like TELL TURTLE followed by other commands such as FORWARD 30, RIGHT 90, etc. The commands are used to move the turtle around the screen to draw lines on the screen along the path taken by the turtle.

Besides making graphics easy, probably one of the most useful aspects of the language is that it provides a powerful way to structure a program.

LOGO includes a set of simple commands called primatives. You define or teach the computer other commands by using combinations of these primatives to form what is called a procedure. Procedures also may contain other procedures as well as primatives. Usually procedures that are contained in other procedures are referred to as subprocedures. Suppose, for example, we want to draw a house like this:



House drawn with LOGO

The following procedure (HOUSE) and subprocedures (ROOF, BOX, and DOOR) could be used.

```
TO HOUSE
   ROOF
   BOX
   DOOR
   END
```

Each of the subprocedures must be defined. For instance, you could define the subprocedure ROOF as:

```
TO ROOF          Defines the roof procedure
   RIGHT 90      Rotates the turtle to position for moving toward right side of
                 screen
   FORWARD 50 Moves the turtle forward 50 units to draw line A in the figure
   LEFT 120      Rotates the turtle 120 degrees back to prepare to draw next line
   FORWARD 50 Moves the turtle forward 50 units to draw line B to form one side
                 of the roof.
   LEFT 120      Rotates the turtle 120 degrees to prepare to draw next line.
   FORWARD 50 Moves the turtle forward 50 units to draw line C to form other
                 side of roof
   END
```

The other subprocedures can be written easily in a similar fashion. LOGO has additional figures called sprites which can be moved about in a similar manner at the same time.

Some people have predicted that LOGO will replace BASIC as the language for the small computer. Besides doing graphics, it permits computation and works with music modules as well as almost any of the other devices that can be used with BASIC. BASIC, however, is very popular, and the more programs written in BASIC, the less likely it will be replaced anytime soon.

Home entertainment

Games develop useful real-life skills while having fun.

In Chapter 1, we discussed that many hidden benefits come from playing computer games. Many of the games offered by TI for the TI-99/4A were developed not only to provide interesting, challenging and enjoyable games, but also to specifically provide some of these hidden benefits. Let's look at some of the more popular games and take three examples and discuss some of the hidden benefits in more detail.

Game programs

Some of the games include:

1. Parsec — a popular space game that combines colorful graphics and voice; pits the player in a series of battles with alien ships.
2. Tombstone City — a twenty-first century old west ghost town where you must battle alien 'Mogs'.
3. TI Invaders — a popular game where you must use your wits and aim to destroy an armada of creatures from space.
4. Munch Man — an exciting maze game where you must out-maneuver Hoonos while connecting passages with a chain.
5. Car Wars — an enjoyable game involving a high speed race combined with the challenge of out-maneuvering a canny opponent.
6. Hunt the Wumpus — another very popular game involving an exciting hunt in a hidden maze of caverns and twisting tunnels.
7. A-MAZE-ING — a maze game where you are a mouse and must find your way out of various mazes before being eaten by a cat.
8. Football — a game where the computer uses actual football statistics to simulate a football game where you and your opponent call the plays.
9. Other available cartridges include Munch Mobile™, MoonMine, Sneggit, Alpiner, Othello™, Chisholm Trail, Indoor Soccer, Mind Challengers, Connect Four™, Yahtzee, and on and on. Also available are ten Milton Bradley Voice Command Video Games™ where you use your voice to give various commands to the games.

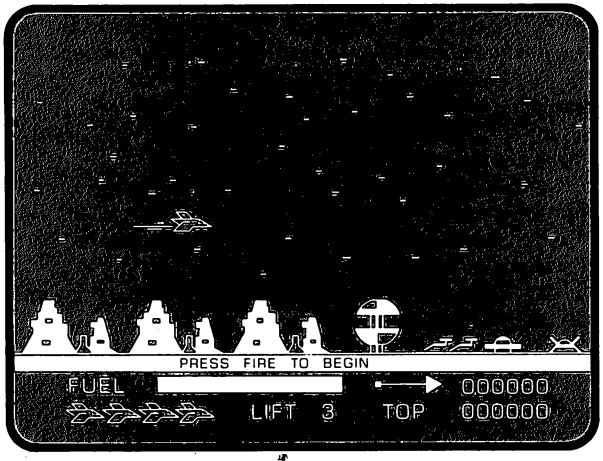
Program descriptions

Following is a more detailed description of three of the popular games which not only provide entertainment, but by playing them you may acquire some of the 'hidden benefits' we've discussed.

Parsec

Parsec is a fast action game that helps you recognize and react to patterns.

Parsec is an exciting, fast-paced, challenging space game. It offers colorful graphics, realistic sound, and even a talking computer (if you have the optional Solid State Speech Synthesizer) to help you along your journey into outer space.



Parsec activity

The game starts immediately with an invasion of attackers dubbed the "swoopers" who attack from the right side of the screen. They pose the danger of collision and the longer they fly, the faster they move.

The next attackers are the "urbites" who fire at you with a twin photon cannon. As soon as they change color from white, open fire!

After the "urbites", the "LTFs" appear. They are similar to the swoopers, but fly faster and become harder to hit.

As soon as you defeat the LTFs, be ready for the hardest ship you will face, the "dramites." These ships are similar to the urbites, but are much faster and fire at will. Be careful, these ships are really hard to destroy.

The next attackers have an unusual method of attack. The "saucers" sneak up from behind in a surprise attack. Study these ships carefully and watch for patterns.

The last group of fighters are the "bynites". These ships are slow and not hard to shoot down, but be careful of their photon clusters.

After defeating all of these ships, you will encounter the "asteroid belt." You must shoot a path through the many asteroids to get to the next level.

The next level pits you against your previous enemies, but you must shoot each ship an extra time to destroy it. In the fourth level, "killer satellites" will attack you at random.

During the game you must watch your fuel level. When you are nearly out of fuel, you must navigate your fighter through a "refueling tunnel" which takes excellent flying skill to negotiate.

Besides being great entertainment, Parsec is a useful game to improve both children's and adults' ability to recognize and react to patterns — patterns of shapes as well as patterns of events. Eye-hand coordination skills also are required, both to shoot down the various attackers as well as to navigate through the refueling tunnel.

A-MAZE-ING

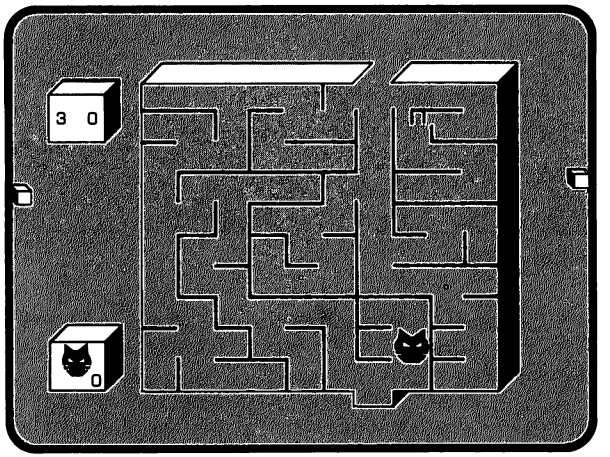
A cat and mouse game that can challenge the best game player.

A-MAZE-ING pits you, as a mouse, against the computer-controlled cats. You also can play with two people in either a competitive or cooperative mode. Strategy and wit must be combined in this challenging maze game.

A-MAZE-ING has many options from which to choose. The first set of options include the number of players and whether or not they are enemies. Then you must choose the type of game. Escape Maze requires the mouse to escape as quickly as possible. In Cheese Hunt, the mouse must gather ten pieces of cheese before escaping.

In the second set of options, you must choose a simple or complex maze, and a visible or invisible maze. Next, you must choose mouseholes or obstacles. Mouseholes are pathways through walls which the mouse can use, but the cats can't. If you choose 'obstacles', there are no mouseholes; instead, there are pillars throughout the maze which make travel very difficult. You can also choose a slow or fast mouse.

The final option list lets you pick how many cats will chase you. You can choose different speeds for the cats. You can make the cats stupid or smart. Stupid cats wander aimlessly through the maze, but smart cats sense the movements of the mouse. Cats also can be standard or pouncing cats. Pouncing cats can jump over the walls of the maze. If the cats are pouncing, you can choose the pouncing frequencies (low, medium, or high). Once you have finished all the option selections, get ready for a game the whole family will enjoy. At the high skill levels, it will challenge even an "arcade expert."



A-MAZE-ING activity

A-MAZE-ING is a game which develops and improves spatial awareness and planning. The player must be continually aware of the location of the mouse relative to the cheese, the cats and the exit. The game requires planning to successfully go through the maze, especially without being trapped by the cat. This planning is necessary both at the beginning as well as while you are in the maze. If the strategy is not well planned — goodbye mouse.

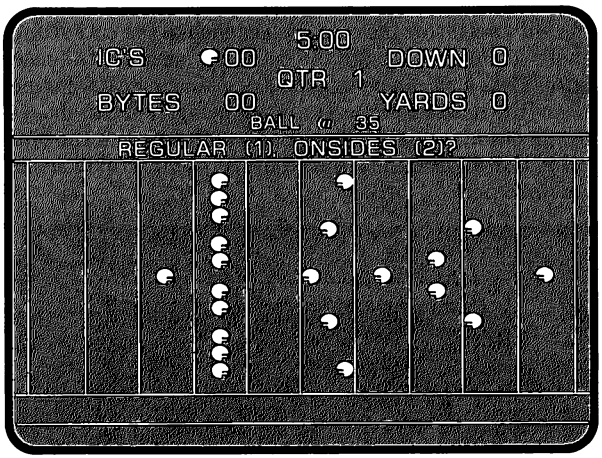
Football

This is realistic football!

Football pits you against your opponent in an all out struggle of wits and football knowledge. Most other video football games are won by the player who best controls his joystick. In TI Football, though, you must really know football to be successful in defeating your opponent. The players choose their plays and the computer actually simulates the game based on professional football statistics!

First of all, the computer asks for the length of time in each quarter (1-99 minutes). Each player then enters his team name. The computer randomly selects a player to call the toss and the winner can either kick or receive. Also, the player can select either an onsides or regular kickoff. As soon as the ball is in the air, the game is under way.

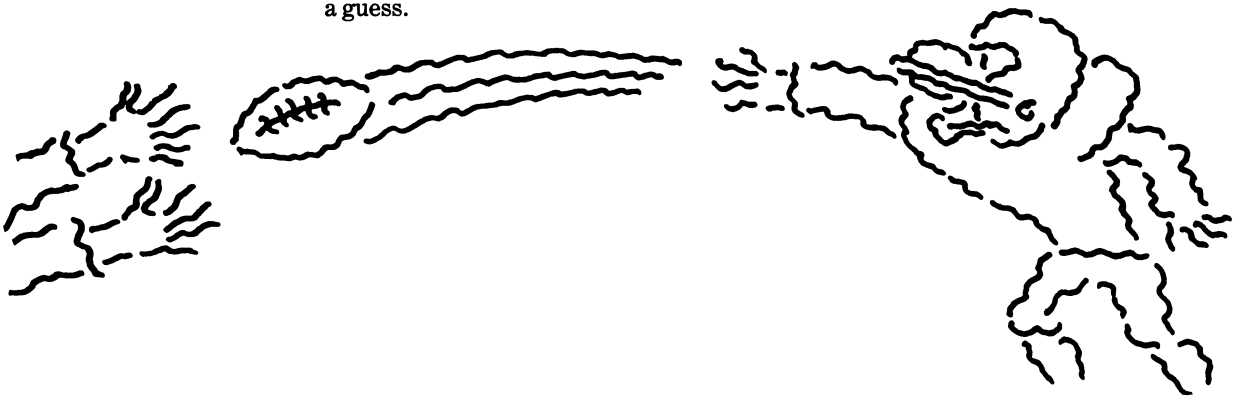
Both offensive and defensive players have a wide variety of plays from which to choose. The offense can choose pass plays, such as bombs, screens, and flares; and running plays, such as draws, sweeps and dive plays. The defense can choose plays to stop the run or pass, or even to block punts and field goals.



Football activity

Other features that add realism are penalties that range from five to fifteen yards, interceptions, and time-outs. There is no limit to the number of time-outs a team can take.

This game is for people who enjoy football and know the game. It combines the luck and strategy of a real football game. The players must think out each play carefully to successfully win the game. It promotes strategic thinking as well as simulates the real life experience of taking a risk by running a play on a hunch or a guess.



Information management for household administration

One of the least developed areas of programs for home computers has been in helping you perform the many functions required in and around the home. However, in just the last few years many different application programs have come on the market for the home and personal computer systems to meet these needs at an affordable price.

Here's where the computer gets put to really practical use around the home.

There are a number of areas in the everyday activities of the home in which a home computer can be helpful. For example, it can be used to help in planning and keeping track of a monthly budget, or keeping various kinds of records, or in making financial decisions such as buying a new home or car. We can expect to see many more helpful programs of this type in the future, because as the number of homes with these computers increases, a large market potential will provide a financial incentive to programmers. As stated previously, it does take extensive computing capability in your home computer to be able to execute some of the information management programs.

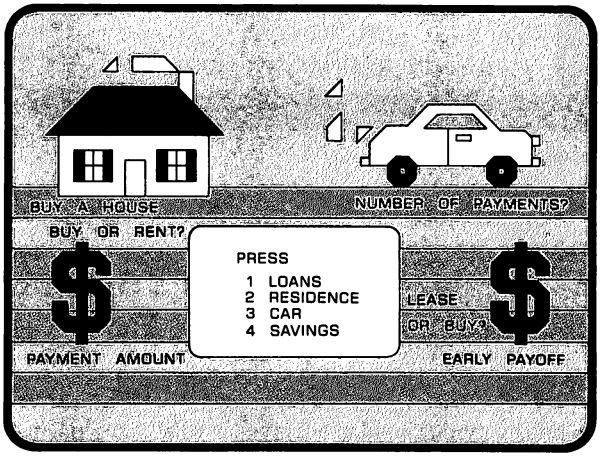
Program descriptions

Let's look at some specific examples of information management programs contained in solid-state cartridges used with the TI-99/4A.

Home Financial Decisions

Whether you're spending money or saving it, this program can help you do it wisely.

The Home Financial Decisions cartridge is a popular program used with the TI-99/4A by people that must make a major financial decision, such as buying a car or home, refinancing the mortgage on their home, or perhaps just borrowing some money for a new refrigerator. Many "what if" situations can be analyzed. It gives information such as expected monthly payments, total interest, etc. needed to consider various purchase strategies. The program provides evaluation data for loans, the purchase of a residence or car, and the monetary return from a savings account.



Home Financial Decisions menu

The loan option of this module provides five types of conventional loan analysis. For one analysis, it will specify the amount you can borrow based on the size of payment desired, the length of the loan and the loan interest rate that must be paid. The standard compound interest method is the computational procedure used.

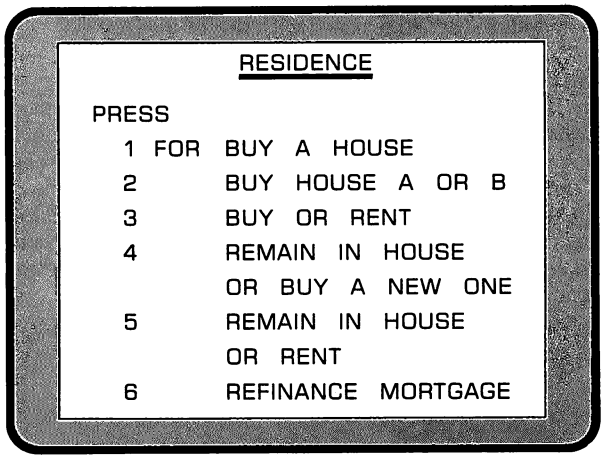
For example, if you could pay $100 a month for 24 months on money borrowed at an interest rate of 18%, what is the maximum amount you could borrow? The program will compute this after you enter the known information at the console keyboard. The maximum amount you could borrow, $2,003.04 for this example, is then displayed on the screen.

In another analysis, the program can determine the monthly payment when you input the amount borrowed, the interest rate and the length of the loan.

In still another analysis, the computer can determine the number of monthly payments that would be needed to pay off a loan of a given amount and interest rate when you specify a monthly payment amount.

Another option allows you to determine the size of the down payment, and still another option computes how much money would be needed for an early payoff.

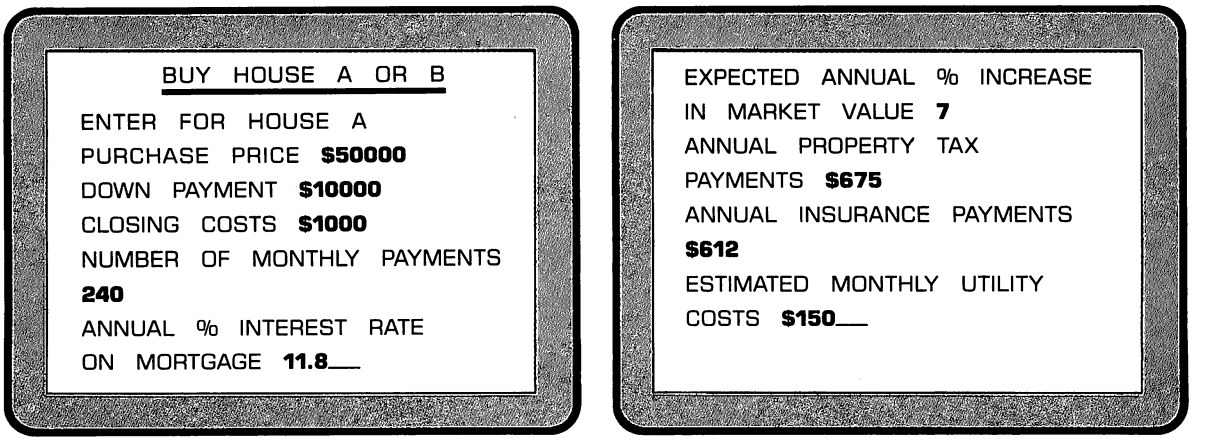
The residence program offers these types of residence analysis:



Residence menu

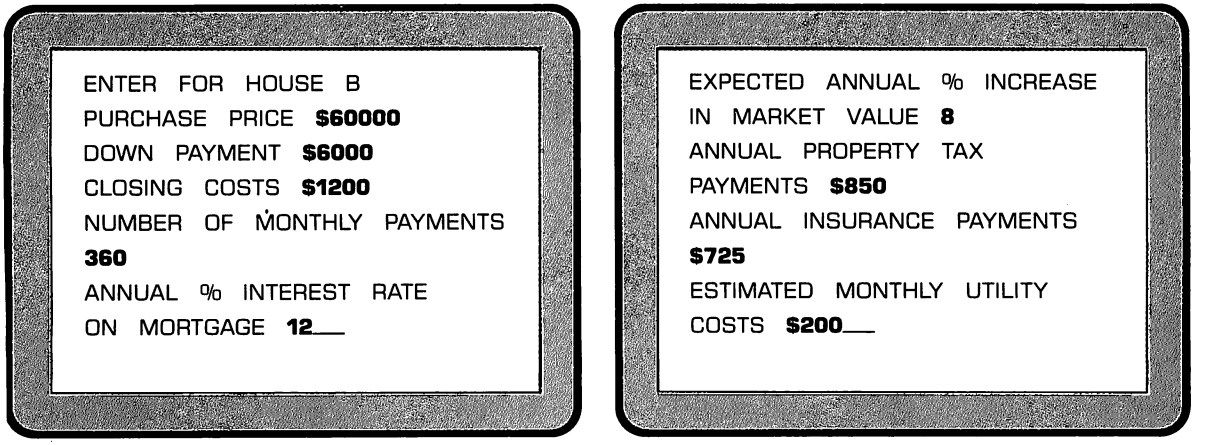
Let's look at an example of using this program. Suppose you are trying to make a decision between two houses where house A costs less than house B, but the down payment for B is less than that for A. On the other hand, the mortgage interest for B is greater than that for A. If you don't spend as much on the down payment, you can keep the extra money in a savings account and earn interest. Also entering into the decision are the cost of taxes and insurance. The program will analyze the costs of both houses and give the difference.

By selecting 2 on the main menu, the program will begin asking for information. The information printed in bold type is what you type on the keyboard. Here is the information input on house A:



Entering information for house A

Here is the information input on house B:



Entering information for house B

After entering this information about the houses, the computer asks for some more information.

```
YOUR FEDERAL INCOME TAX
BRACKET IN % 30
ANNUAL % INTEREST RATE
ON YOUR SAVINGS ACCOUNT
6.25
NUMBER OF TIMES PER YEAR
SAVINGS ACCOUNT INTEREST
IS COMPOUNDED 12__
```

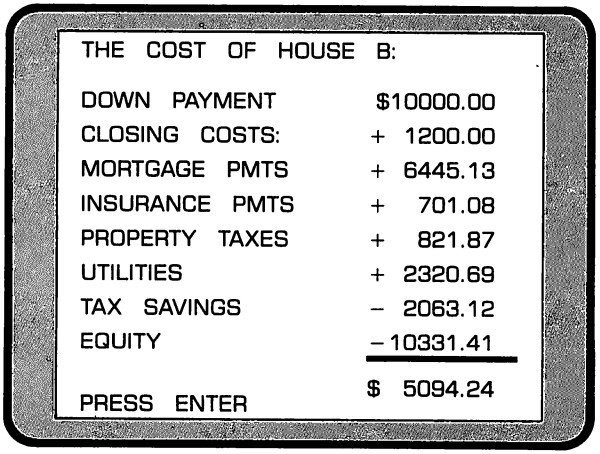
```
NUMBER OF YEARS IN THIS
ANALYSIS 1__
```

Entering financial information

The following information is then displayed on the screen by the computer:

```
THE COST OF HOUSE A:

DOWN PAYMENT          $10000.00
CLOSING COSTS:      +  1000.00
MORTGAGE PMTS       +  5045.99
INSURANCE PMTS      +   591.78
PROPERTY TAXES      +   652.69
UTILITIES           +  1740.52
TAX SAVINGS         -  1512.83
EQUITY              - 13178.63
                    ----------
                    $  4339.52
PRESS ENTER
```

Results for house A



Results for house B

On a final screen the computer displays "THE COST OF BUYING HOUSE A IS $754.72 LESS THAN THE COST OF BUYING HOUSE B". As a result, the difference of $754.72 in the first year is the savings of purchasing house A over house B.

As this example shows, the information that can be quickly and easily obtained from this program can be very helpful in determining important financial decisions about your residence.

Two other types of financial analysis are possible with the program. The *Car* section can help you evaluate a new car loan, the cost of your present car versus buying a new one, and whether you should lease or buy a car. The *Savings* section lets you evaluate and schedule your savings plans.

Thus, with a program such as this, you can have help for solutions to a majority of the financial decisions that face many families today. This user-friendly program prompts you for any data you must input in an easy-to-understand way.

Touch Typing Tutor™

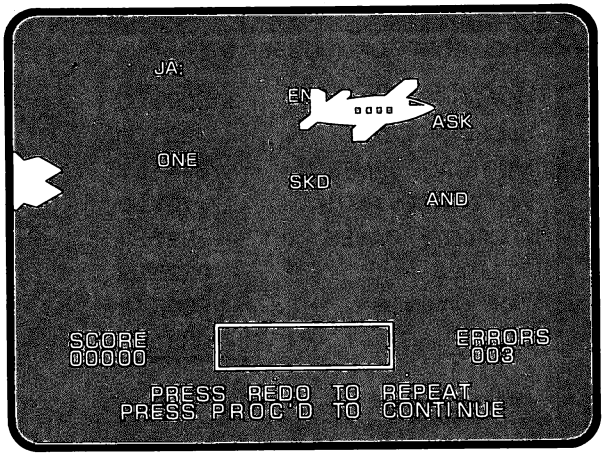
Whether you're learning to type or just want to practice to improve your speed and accuracy, this program can help you while you play a game.

This program will teach you how to touch type; that is, to type using all fingers without looking at the keys. Even if you already know how to type, Touch Typing Tutor will help you develop speed and accuracy.

When you first select the program, you have three program choices; lessons, diagnostic or game.

If you choose 'lessons', the computer will ask you what you want to try. Levels 1 through 4 teach you to type the alphabetic characters by touch, and levels 5 through 8 teach you to type numbers, symbols and punctuation. You can choose from two lessons and a review. There are four exercises in each lesson.

When you choose 'diagnostic,' you have further choices of either wpm timing, analysis, or practice. The first choice will time you by how many words per minute you type. Then the computer will recommend a wpm goal for you. (You can also choose your own goal.) The second choice, 'analysis', will test your skill on certain keys and recommend practice if needed. 'Practice', the third choice in diagnostic, will display either letters, numbers, or symbols (as you choose) one at a time inside a box. You must type that key as quickly as you can.



Touch Typing Tutor activity

The final selection is 'game.' You can choose from 1 through 8 levels. At the start of the game, an airplane flies across the screen and places words which you must type. This method offers a fun way to learn touch typing.

As you can see, with Touch Typing Tutor, you can learn to touch type and improve your skills. Learning to touch type, like so many things, requires practice, practice, practice. However, with a program such as Touch Typing Tutor, the drill and practice can be fun, challenging and motivating.

Teach Yourself BASIC

By learning to program in BASIC, you can get even more benefit from your home computer.

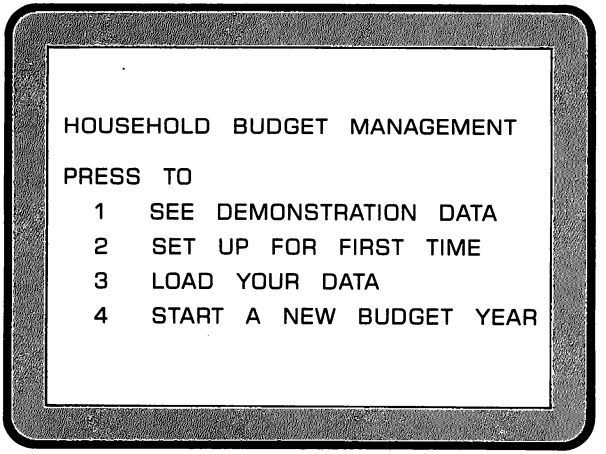
In the later chapters of this book, you will see what a home computer is for by learning how to make the computer do what you want it to do by programming it yourself. You will be introduced to the BASIC programming language. If after reading these chapters, you decide you want to learn more about how to write your own programs, the Teach Yourself BASIC tutorial will be very helpful. This program offers a way to learn the BASIC programming language more thoroughly in ten easy lessons. Learning BASIC this way often can be easier than from reading manuals that sometimes are difficult to follow. Combining the chapters in this book with Teach Yourself BASIC will give you a good start in BASIC programming.

The Teach Yourself BASIC program is designed to introduce you to constants, variables and commands. Once you have completed all ten lessons, you should have much more specific programming knowledge to allow you to program your own detailed games, maintain specific tailored financial records, obtain help for your math homework, or many other tasks you may want the computer to do. Teach Yourself BASIC is for people who are seriously interested in programming.

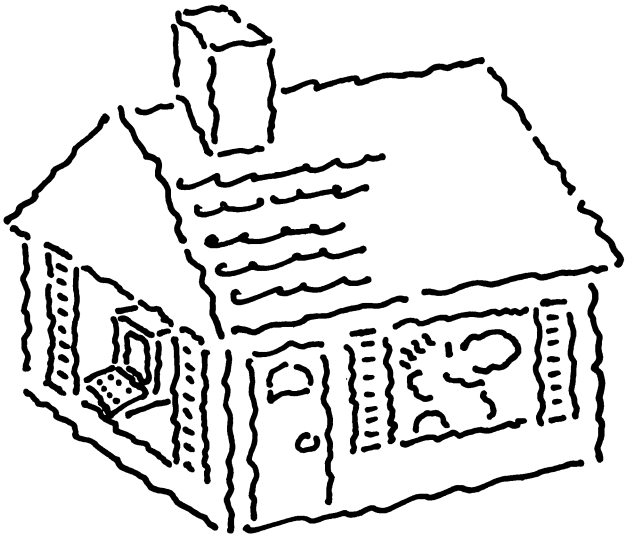
Don't know where the money's going? This program is for you!

Household Budget Management

The Household Budget Management program helps you set up the various monthly expenses and income that you think will occur during the year. Then each month, you put in the actual expenses and income as they occur. The program compares the predicted versus actual expenses and displays this information in your choice of either tabular or bar chart form. It is a valuable program for helping you set budget guidelines, track income and expenses, spot financial problem areas and keep easily accessible records.



Household Budget Management activity



If you need help in business planning, consider this powerful program.

Microsoft™ Multiplan™

Microsoft™ Multiplan™ is a second-generation worksheet program with many advanced features that at one time were found only on very expensive computer systems used by large corporations. This is one of the most powerful modeling and planning tools ever invented for the small computer. It can be used for almost any spreadsheet type application. It enhances the generation of financial reports for managing and planning a business and simulates selected business conditions by analyzing "what if" situations.

Hello, word processor. Goodbye, typewriter.

TI Writer

TI Writer is a word processing program which you will greatly enjoy if you write many letters, reports, or documentation of any kind. You will not want to go back to using a typewriter once you use a word processor. Rather than text that you type appearing on paper, it appears on the screen and is stored in memory at the same time. Then revisions can be made to the material on the screen by typing over, deleting, inserting, moving, respacing, etc. After all changes are made, you can print the text on paper as many times as you want and each copy will be identical.

And there's lots more.

Other programs for information management

Other software offered for the TI-99/4A for information management include Securities Analysis, Personal Record Keeping, Tax/Investment Record Keeping, Personal Real Estate, Personal Report Generator, and on and on. Just as in the other areas, there is a wide variety of currently available and useful software packages.

Summary

When you purchase a home computer system, it should be easy to set up and you should not have to write programs to use it effectively. Whether it is to be used for entertainment, education or information management, much preprogrammed software in plug-in solid-state cartidges is currently available to make the home computer a useful product for your household. What's even more encouraging is that there will be an ever expanding amount of software available in the future.

Can you use a home computer?

Setting it up ← A home computer is easy to set-up, get started, and use.

Getting started

Available software

Educational software

Reading programs introduce and reinforce reading skills. → **Programs for reading**

Programs for mathematics ← Math programs develop skills such as counting, sorting, addition, subtraction.

Other programs

PLATO® programs

Program descriptions → These programs provide education and fun experiences at the same time.

Reading Flight
Alligator Mix
Earling Learning Fun
TI LOGI II

Home entertainment

Game programs

Program descriptions → A fast action space game that helps to recognize and react to patterns and develops eye-hand coordination.

A cat and mouse game that can challenge the best game player. →

Parsec
A-MAZE-ING
Football

Information management for household administration

Program descriptions

Programs to make wiser financial decisions when loaning or saving money. →

Home Financial Decisions
Touch Typing Tutor™
Teach Yourself BASIC
Household Budget Management
Microsoft™ Multiplan™
TI Writer
Other programs for information management

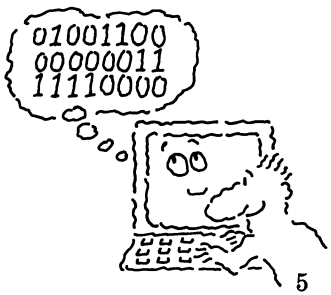
Don't know where the money is going? This program's for you! →

Summary

Summary map → Hello, word processor. Goodbye, typewriter.

2 **More about software**

BASIC interpreter



Firmware

Documentation

Inside solid-state cartridges

Programs you write

BASIC interpreter

5 **Computer hardware**

Central processing unit

CPU word size
CPU differences

Memory capacity

Memory

Semiconductor memory
Memory capacity
Expansion memory
Magnetic tape and disk

Peripheral expansion system

Speech synthesizer

Going further

Input/output devices

Printers
RS-232 Interface
Modems for telephone communication

15 **Summary**

16 **Going further**

16 **Summary map**

More About Software and Hardware

More about software

In the last chapter you learned about easy-to-use software; that is, the programs that instruct the computer to perform specific tasks, preprogrammed for you and packaged in a convenient plug-in cartridge that make it easy for you to use the program and the computer.

Firmware

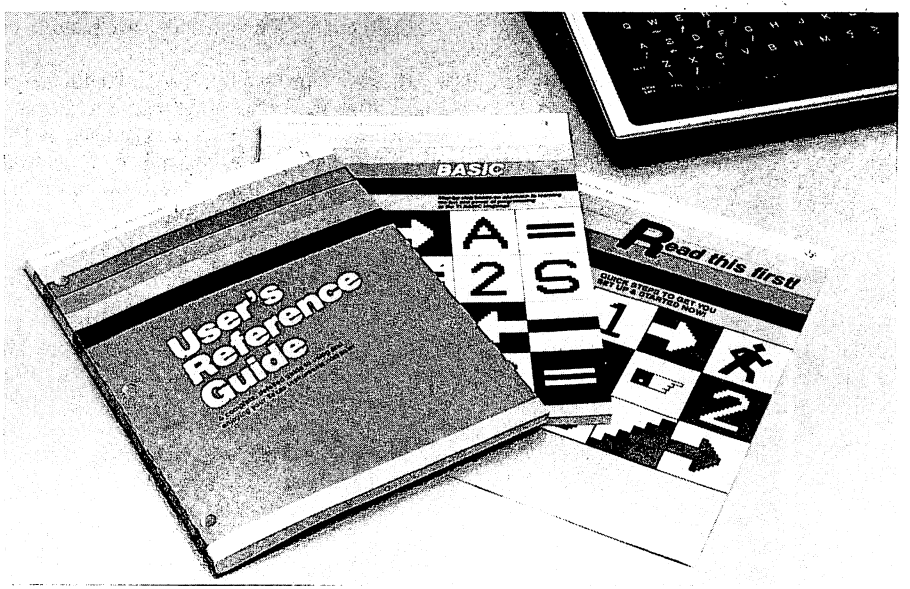
Firmware is software manufactured in a form that is not changed easily.

When you look at these cartridges, you see that they are made of plastic and are "hard." However, they aren't called hardware because they contain the program that the computer follows step-by-step. The program is stored electronically in ROM which is manufactured using integrated circuit silicon chips. Therefore, the cartridges, even though they are "hard," are called software because they contain stored programs. Actually, a more proper name is "firmware" because they are manufactured in a permanent form that is not changed easily.

Documentation

Printed instructions for people are called documentation.

Some people also use the term software to refer to operating manuals, instruction manuals and other types of printed information that tell people how to operate hardware or to use software. However, to eliminate confusion, a better term for this printed information is documentation.



Documentation

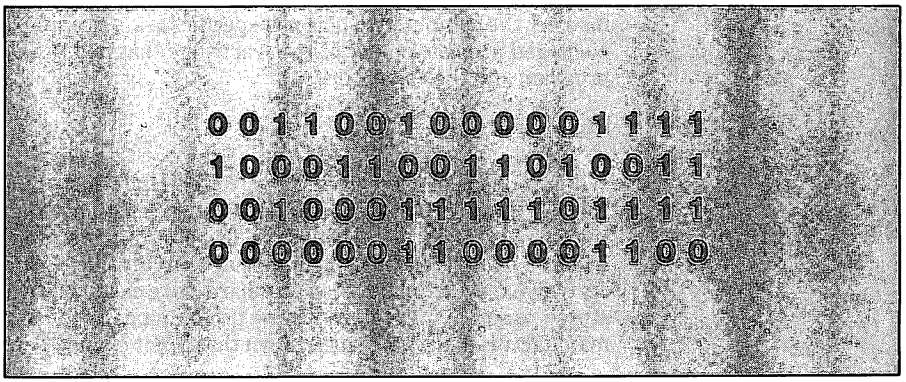
Inside solid-state cartridges

A solid-state cartridge contains a program stored in machine language - the language of the computer.

In the Solid State Software™ cartridges, the program is stored electronically in the ROM integrated circuits at the time that it is manufactured. The electronic signals are in machine language, the language the computer can understand.

Machine language consists of electrical signals that have two possible values; zero and one.

Although the computer can easily execute or follow the instructions in machine language, it would be hard for you or anyone to understand. Machine language is the lowest level computer language; it consists of electrical signals that are identified by humans as zeros or ones. Zero and one are the two possible values of a binary digit (bit). You probably have heard of bits and bytes. A bit is the smallest piece of information that an electronic digital computer can handle. Bytes will be covered a little later.



Example of machine language

Humans can't work efficiently with zeros and ones so programs in the computer convert our keyboard input letters and numbers to machine language.

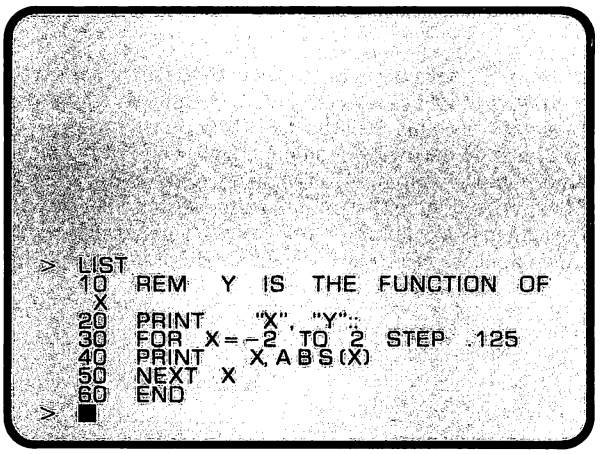
The computer can work only with these zeros and ones. Since we humans work with many other numbers as well as alphabetic characters and symbols, they must be changed to an appropriate binary number made up of only zeros and ones. When bits are used in combination to represent numbers, characters and symbols, these combinations are called codes. Various codes are used to convert human inputs to machine language.

For example, there is a program in ROM within the computer console that converts the alphabetic characters, numbers, or symbols that you type on the keyboard to the appropriate binary codes. By means of this built-in program, the computer does the conversion for us. Therefore, you actually don't have to be concerned about machine language and the binary numbers. If you did, you probably would never buy the computer in the first place.

Programs you write

Humans write programs in a high-level language such as BASIC.

If you want to write a program with instructions for the computer to follow, you don't have to write programs in machine language. You write the program in a high-level language that you understand, then another program in the computer converts this high-level language to the machine language of the computer. One high-level language which is used with many home computers is BASIC. A few lines of a program written in BASIC are shown below. As you will see later in this book, BASIC is easy to learn and use.

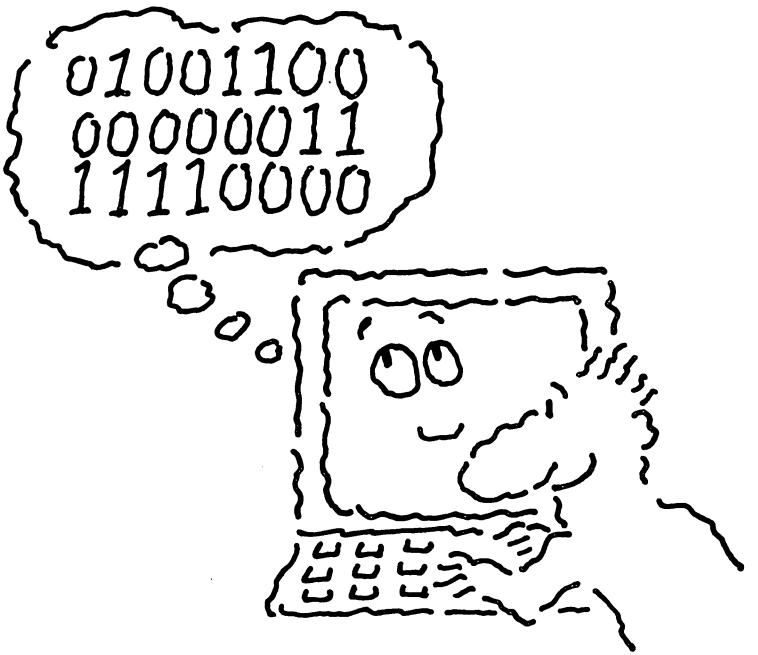


Example of BASIC instructions

BASIC interpreter

The BASIC interpreter translates the BASIC language instructions to machine language instructions.

Just as the keyboard inputs by humans must be converted to machine language by an internal program, so also must the BASIC language instructions be converted to machine language. The BASIC interpreter is the program inside the computer which interprets each BASIC instruction you write and generates the appropriate set of machine language instructions. It is the program inside the computer that changes the high-level input instructions that humans understand to machine language instructions that the computer understands. After generating a set of machine instructions for a given BASIC instruction, the computer is directed to execute this set of instructions before interpreting the next BASIC instruction. Step-by-step, the BASIC program is interpreted to complete the task.

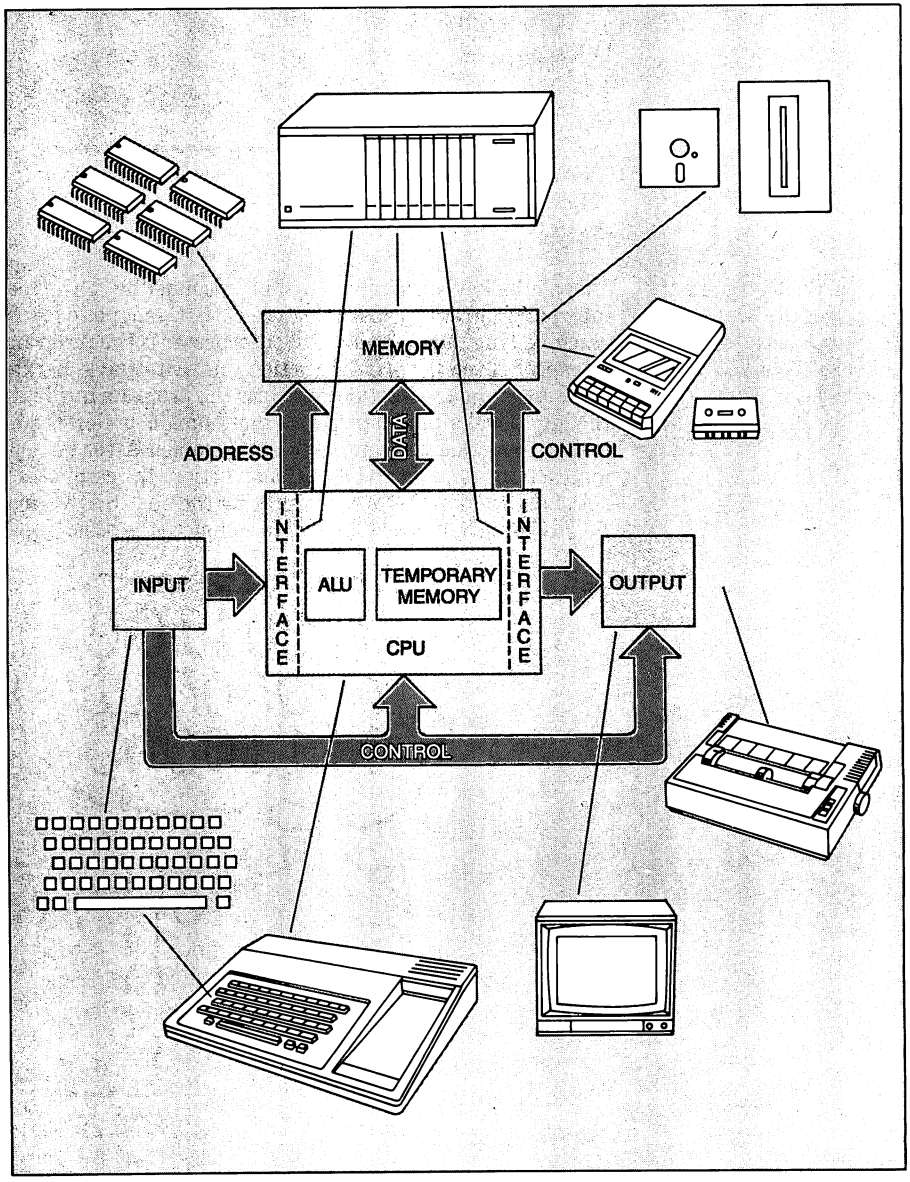


Computer hardware

Hardware refers to the nuts and bolts, the cabinet, the cables and the electronic circuits of a computer and its peripherals.

Hardware is the general name given to the actual electronic, metal and plastic parts of a computer, its accessories and its peripheral equipment. It also includes the cables used to connect all the parts together.

The main hardware components of a computer; the central processing unit, the memory and the input/output devices, were defined and briefly described in Chapter 1. Now let's examine these components in more detail. The diagram below relates the physical parts as you can see them with a general block diagram. A block diagram is a convenient way to show how parts or functions of a system are connected together.



Block diagram

Central Processing Unit

The CPU is the brain of the computer.

The Central Processing Unit (CPU) has been defined as the 'brain' of the computer. It controls all the other hardware components as specified by the program. The CPU contains a decoder to decode program instructions, an arithmetic and logic unit (ALU) which performs the computational functions for the computer, and registers which are used mainly for temporary storage of instructions and results from computations. Registers also are used to keep status information and to keep track of where in memory the next instruction is.

When the computer is turned on, the CPU instructs the memory to obtain (fetch) the first instruction of a standard program stored in memory at a set place. For the TI-99/4A this program, called an initialization program, is stored in ROM. The first instruction is decoded and executed by the CPU. For example, it might be an instruction that tells the computer to clear all of the user memory. The CPU then continues to execute instructions in the initialization program until it is finished. At that point, the computer is ready for you to do something.

The CPU functions like a business manager to be sure a task is completed by following instructions.

The CPU is similar to a business manager. The manager obtains an instruction from his/her memory to begin the accomplishment of a specific task. The manager then directs the people that report to him/her (the other computer components) to do their functions to carry out (execute) the task. Some of the functions may have to be done by the manager, while others are done by people in the manager's group. Similarly, the CPU directs the memory to fetch a number so that the ALU (in the CPU), for example, can add it to a number already in the CPU. The manager (CPU) is in control and is responsible for the successful completion of the task. At the next step, a new task is obtained (next instruction is fetched) and the process is repeated continually until all of the planned tasks in the program are completed.

CPU word size

A computer word may be one, two, or more bytes. A byte is a group of eight bits.

The CPU usually handles more than one bit at a time. Most often, it operates on a code which is a group of bits called a byte or word. The CPU in most home computers operates on a byte. A byte is made up of eight bits. Usually a computer word will be some whole multiple of a byte; that is, 8 bits, 16 bits, 32 bits, etc. Generally, the larger the word size, the more complex the circuitry and the more powerful the CPU. The TI-99/4A uses a 16-bit microprocessor, the TMS9900, for its CPU. For this reason, it has very good computing capability. Most other home computers and many personal computers use only an 8-bit microprocessor for their CPU.

CPU differences

All CPUs are similar in their main function, which is to control the operation of the computer and perform the arithmetic and logic operations. CPUs primarily differ in the amount of internal storage, the size of word they work with in the ALU, and the amount of control necessary to manage the other components. The bigger or more powerful the computer, the more complex is the CPU. The CPU in most home and personal computers is contained on one integrated circuit or chip. The large computers used by banks, companies, and other such organizations have CPUs made with many integrated circuits and other components. Of course, they cost a lot more!

Of all the units that the CPU controls, the one that is used most often is the memory. Let's next examine the functions and types of memory.

Memory

Both short-term and long-term memory is needed in a computer system.

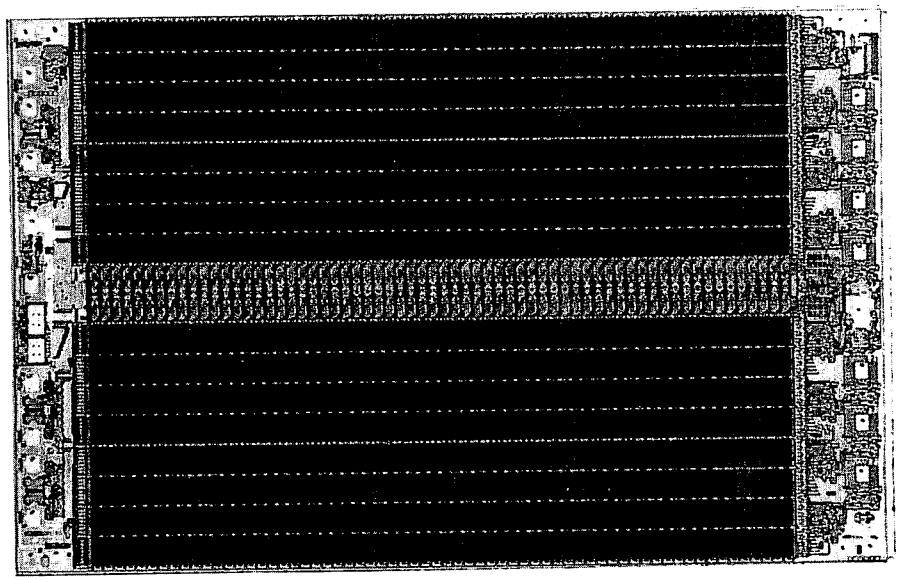
Computer memory is generally classified as either short-term or long-term. Short-term memory loses whatever is stored in it when the power to the computer is turned off. If you have used one of the small hand-held calculators, you probably have noticed that the stored numbers are lost when the power is turned off. (Some of the newer calculators maintain power on the memory even when the power switch is off.) We humans also have short-term memory and we tend to forget quickly things that are not considered important.

Long-term computer memory is more permanent. Information stored here is always available unless something else is stored in its place or if there is a failure in the storage medium. Information in our own long-term memory would be something which we considered too important to forget, like our name. However, we could forget information in our long-term memory if we likewise had a 'failure' as, for example, in the case of amnesia. The memory components commonly used with small computers today are semiconductor memory, magnetic tape cassette and magnetic disk. Semiconductor read/write memory usually is used for short-term memory and semiconductor read-only memory is used for long-term memory inside the computer. Both magnetic tape cassette and magnetic disk memory are used for external long-term memory. Magnetic tape and magnetic disks, of course, do not require any power to keep their contents; however, they must be kept away from strong magnetic fields which can destroy their contents.

Special types of integrated circuits used for semiconductor memory will be able to store up to 256K bits.

Semiconductor memory

Semiconductor memory is usually a set of integrated circuits especially designed with storage capability. Solid-state semiconductor technology has advanced to the point where over 64,000 bits are commonly stored on one silicon chip as shown in the figure. This technology continues to advance so that very soon, over 256,000 bits will be delivered in a similar package.



Semiconductor memory chip

Semiconductor memory is of two types, read-only memory and read/write memory. Read-only memory often is referred to as permanent or non-volatile semiconductor memory because the stored information is not changed readily and it does not lose its stored information if power is removed. This is the reason it is used for long-term storage. Read/write memory can have its information changed readily by writing in new information. As stated before, it is used for short-term storage because it is volatile memory; that is, it loses its contents when power is removed.

Read/write semiconductor memory often is referred to as RAM.

As its name suggests, information can be written into as well as read from read/write memory. After data is written into memory and used, new data can be written into the same locations. Thus, the data stored in read/write memory may change very often. Such memory is also called data memory and, more often, random access memory (RAM). Random access means that the contents of any memory location can be obtained by the CPU in the same amount of time as that in any other location.

Memory capacity

Memory capacity usually is given in bytes where each byte is 8 bits. The maximum internal memory capacity is limited by the microprocessor type and the memory addressing scheme.

K stands for 1,024 in computer jargon.

The TI-99/4A is designed to work with 72K bytes of storage. The character K signifies a multiplier of 1,024, thus 72K bytes is 72 x 1,024 or 73,728 bytes. The basic system includes 16K bytes of RAM for user programs and data, 26K bytes of ROM which contains the system monitor and the BASIC interpreter, and 30K bytes for the plug-in solid-state cartridges. The system monitor is not the same as the video monitor. The system monitor is software in the console which contains the initialization program and other programs that tell the computer what to do when input comes from the keyboard or output is required to the TV or video monitor screen.

Memory can be added with the memory expansion unit.

Expansion Memory

When you write a BASIC program, it is placed in the 16K user RAM inside the TI-99/4A console. Most programs you write will fit in this memory, but if any programs take more than 16K bytes, the optional Memory Expansion Card with another 32K bytes of semiconductor RAM will have to be added to the system.

Also, some of the solid-state cartridges such as TI LOGO II need the additional 32K of RAM.

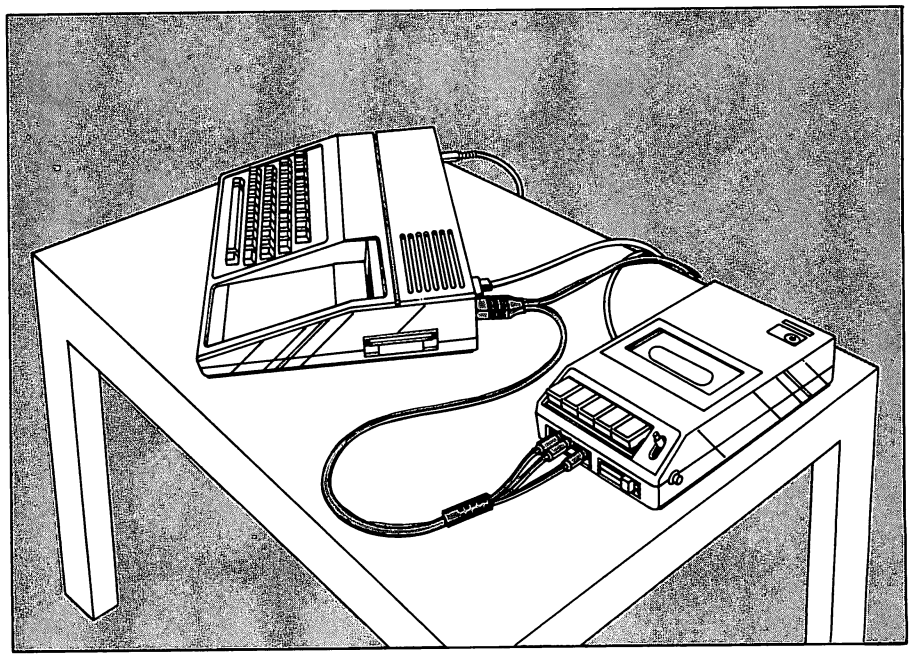
Magnetic tape and disk

The two common non-volatile memory devices, magnetic tape and magnetic disk (this is the floppy disk referred to in Chapter 1), do not use a random access technique for addressing data as is used for the internal solid-state semiconductor memory. As a result, the time it takes to read one data word or byte will be different than for another. However, they are very useful because the storage is long-term and the information does not take up valuable memory space in the computer until it is needed and loaded. Also, cassette tapes and diskettes provide essentially an unlimited amount of storage. Of course, each disk or tape can store only a specific amount of information, but when more storage is needed, just use another tape or disk.

Magnetic tape and disk provide almost unlimited long-term storage.

Cassette tape provides an inexpensive form of long-term storage.

The magnetic tape cassette is used by many home computer users for long-term storage, especially at first. A blank cassette tape is less expensive than a blank disk, and the tape recorder/player is much less expensive than a disk drive. Many of the commonly available audio cassette tape recorder/players work satisfactorily, but TI has one made especially to use with the TI-99/4A. Many programs are available on cassette tape and these offer an inexpensive way to add to your collection of programs. However, programs on cassette tape are a little harder to load and take longer to load than programs in solid-state cartridges and programs on floppy disks.



Cassette recorder/player connected to computer

Computer programs and data must be converted to analog tones before they can be recorded on cassette tape. One frequency or tone is used for a binary zero and a second for a binary one. A circuit in the computer performs the conversion so the analog tones are available at the cassette connector on the console. To use a cassette recorder with the TI-99/4A, you need the optional cable shown in the figure to connect between the computer console and the cassette recorder.

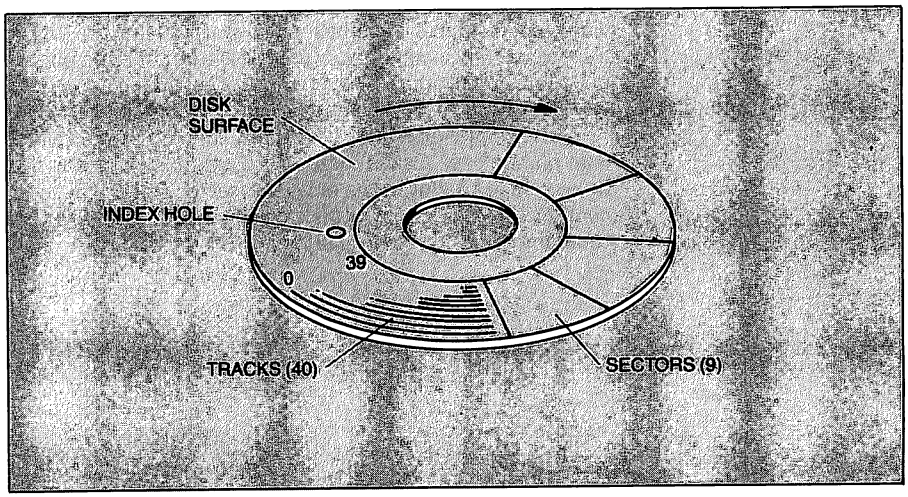
Floppy disk provides long term storage with faster access and better reliability than cassette tape. The disadvantage is that it costs more.

The magnetic disks used in home or personal computers can either be a firm fixed disk (often referred to as a "hard disk") or a removable flexible thin plastic disk. The hard disk provides much more storage, but the disk and its drive are much more expensive. Because of lower costs, the thin flexible plastic disk is the one commonly used for home computer systems. It's called a "floppy" disk because it can be flexed or bent, however, you should not intentionally flex it.

The floppy disk used with many home computers is 5-1/4 inches in diameter. The thin plastic base disk is coated with a thin layer of iron oxide and the disk is sealed in an envelope. A slot in the envelope permits the read/write head of the disk drive to contact the surface of the disk. You should keep this exposed part of the disk from touching anything, especially your fingers.

Floppy disk

The disk must be inserted into a disk drive unit which rotates the disk and moves the read/write head. A disk drive controller is required to control the operation of the disk drive. Information is read or written by the read/write head after it is physically moved to one of the invisible concentric tracks as shown in the figure. As the disk rotates at a constant speed, information is written on the disk when electric signals in the write head magnetize very small areas in the iron oxide coating. When the disk is read, the small magnetized areas reproduce the original signals in the read head.



Tracks and sectors on a disk

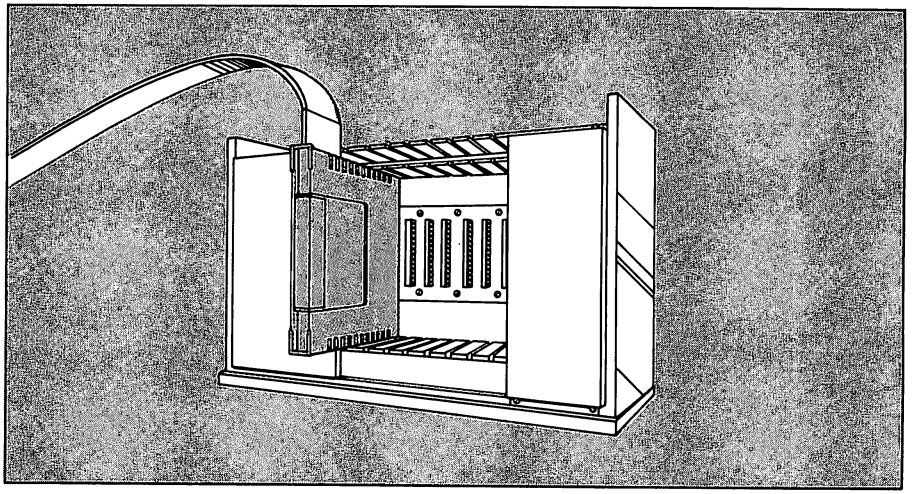
The disk system for the TI-99/4A is optional. When you begin to write your own programs or use programs that require permanent storage of data, you may want to consider purchasing a disk system.

The disk capability offered with the TI-99/4A provides a system that competes with most personal computer systems in disk capability. In fact, if you were to purchase all of the optional equipment for the TI-99/4A, your system would essentially be upgraded to a personal computer. To accommodate some of this additional equipment, TI offers the Peripheral Expansion System as an option.

Peripheral Expansion System

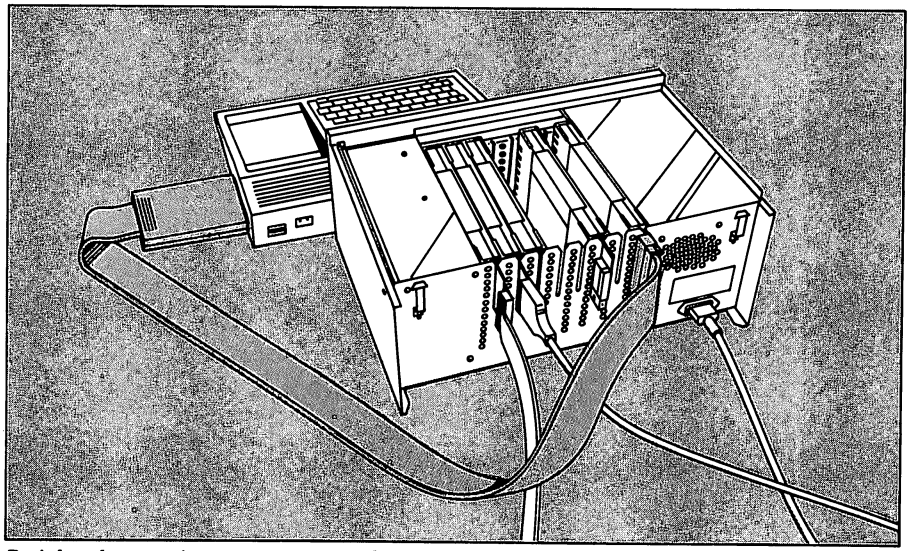
The Peripheral Expansion System allows the TI Home Computer to grow to a system equivalent to a personal computer.

The Peripheral Expansion System is a metal cabinet that provides operating power and plug-in connections for up to eight peripheral interface cards and a disk drive. The cards that may be plugged in may be memory, interface for input/output devices, and special high-level language decoders. This system is designed so that you can install these expansion cards yourself just as easily as you connect the basic system together.



Installing a peripheral expansion system card

Each card plugs into one of the connectors in the expansion unit. Each connector is connected to a collection of wires called a bus which is connected to the computer console by a large flat (ribbon) cable.



Peripheral expansion system connected to console

A Disk Memory Drive can be installed into the right-hand side of the unit and the Disk Drive Controller Card fits into the slot next to the drive.

The Memory Expansion Card that adds 32K of RAM as discussed earlier also fits into one of the slots in the expansion unit. Using the Peripheral Expansion System, as your interest grows and your finances permit, you can build up from the basic system to whatever level system you require.

Speech synthesizer

Programs that have computer-generated speech are more interesting and more useful.

The TI-99/4A has the capability of actually speaking to you by means of the Solid State Speech™ Synthesizer. Most previous systems with this capability used actual recordings of a person's voice on tapes or records. However, the speech synthesizer builds words from basic sounds that are stored in code in a ROM. This ROM contains the special codes to direct sounds for up to 373 words and phrases. Each code converts digital information to analog voltages which cause the speaker in the console to make human-sounding speech. The vocabulary even provides for words such as "read" which can be pronounced more than one way.

The speech synthesizer permits the computer to speak directions and comments as directed by your programs or programs in the Solid State Software cartridges that use this feature. For example, the game Parsec and the educational program Early Reading are greatly enhanced when the speech synthesizer is used.

Input/output devices

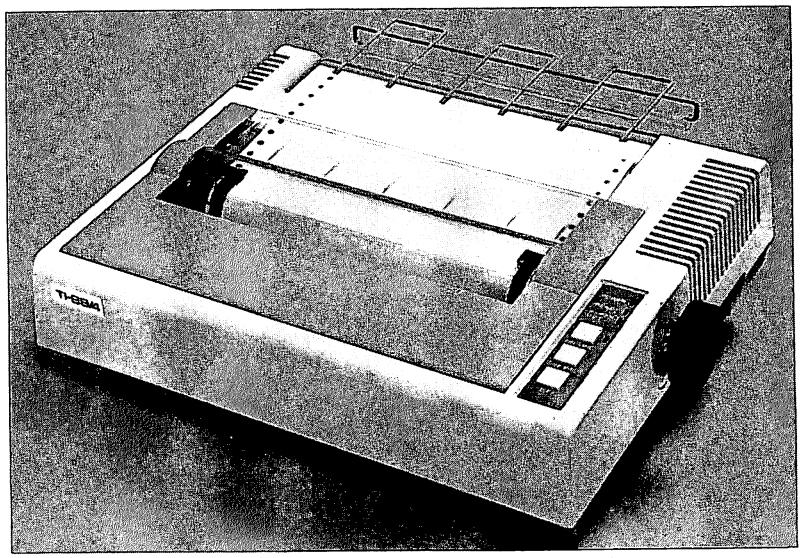
Printers

A printer provides hard copy output from your computer.

A printer provides a means of making a copy on paper of your program, text or data so that you can use it away from your computer, or so you can keep it for one reason or another.

Printers can be classified as either impact or nonimpact. Impact printers are those which use a print head which presses an inked ribbon against the printer paper. It works much like a standard typewriter, in fact, the so called "letter quality" printer forms characters that look like they were made on a typewriter. Another type of impact printer, called the dot matrix printer, forms letters by striking very small pins. Often you can see the small dots that form the characters on the paper. The TI Impact Printer shown below is a dot matrix printer. It requires an RS-232 Interface Card and the Peripheral Expansion System.

Most nonimpact printers are thermal printers. These use a print head that forms each character by heating chemically treated paper. The character appears as a change in color. The thermal printer operates quietly, but the impact printer is noisy. On the other hand, special paper is required for the thermal printer, but standard paper is used with the impact printer.



TI impact printer

The RS-232 standard interface provides compatibility and a means of communication between computer equipment.

RS-232 interface

Because peripheral devices must have a method of communication with computers, a standard interface was set up by the Electronic Industries Association (EIA) and assigned the designation RS-232. This interface helps to ensure compatability between the various manufacturers of computers and computer peripheral equipment. Essentially, the RS-232 standard defines signal conditions needed for communication, electrical signal characteristics, the connector size and shape, and the connector pin assignment for each signal. The TI RS-232 Interface Card is compatible with the current EIA RS-232 standard.

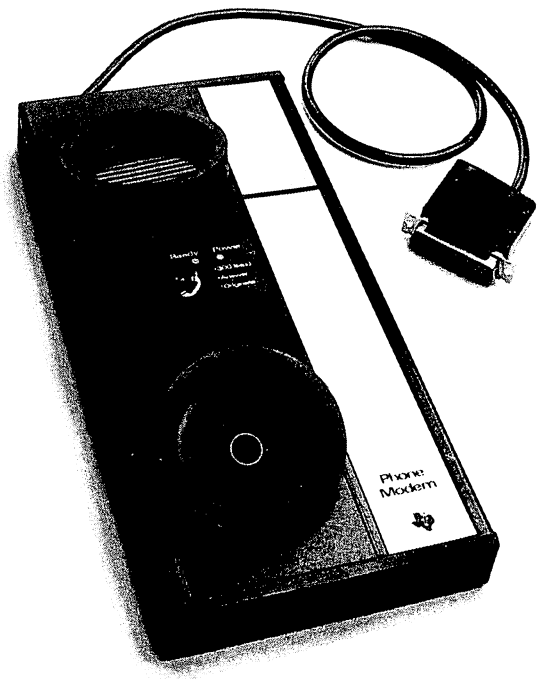
A modem permits the use of the telephone lines to connect computers together.

Modems for telephone communication

Your home computer can be connected by telephone lines to another home computer or even a larger computer system by using a telephone coupler modem. The modem converts the binary 0's and 1's used by your computer to tones of two frequencies, one frequency for a 0 and another for a 1. These tones are within the voice frequency range of telephones so the tones will pass through the telephone system as easily as your voice. The computer at the other end must have a similar modem to convert these tones back to the original binary signals so the computer can understand them.

The modems are made in two types; direct connect or acoustically coupled, and both types are available for use with the TI-99/4A. The direct connect modem is wired directly to the phone lines and is more reliable. To use the acoustically coupled modem shown in the figure, the handset of the telephone is placed in the rubber cups of the modem.

In addition to the modem, special software must be loaded into the computers at each end. This software for the TI-99/4A is available in the Terminal Emulator II Solid State Software cartridge. The program has instructions to guide you so that you can easily communicate directly with other computers from your keyboard.



TI telephone coupler (modem)

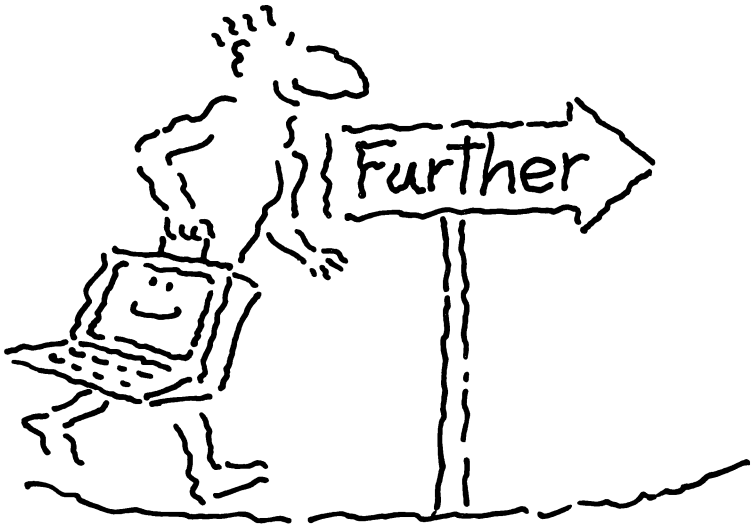
Summary

Thus far in the book, you have discovered that a home computer can add new dimensions to your household. You have seen that the TI-99/4A Home Computer has been designed on a firm foundation, using the necessary hardware to keep it in date for years to come. You have been given reasons why you should want to buy a home computer. You know what it looks like, how to set it up, and how to use it with preprogrammed software. You have learned about hardware, software, CPU, memory, input/output, ROM, RAM, bit, byte and many other terms that can help make you a knowledgeable shopper for *your* home computer. What remains is for you to step into the computer era if you haven't already done so.

Going further

It is entirely satisfactory for you to use preprogrammed software to apply a home computer. You will get excellent results and be able to use the computer for many applications. However, if you really want to enhance the value of a home computer and, in addition, feel some real excitement because you have truly accomplished something, try your hand at telling the computer what you want it to do.

In the remaining chapters of the book, we hope you will realize how really versatile and useful the home computer is when you learn to program it yourself. The chapters are designed to lead you, step-by-step, through a learning process, with fully illustrated working examples, so that you will be able to get the ideas from your head into a form that the computer can use to perform the task you desire. You may think you can't do it, but we think you will find it quite easy if you give it a fair try. So if you are ready, let's proceed.



More about software

Firmware ← Firmware is software manufactured in a form that is not changed easily.

Documentation

Inside solid-state cartridges ← Programs stored in solid-state cartridges are in machine language.

Programs you write

Human inputs must be converted to machine language by programs in the computer. → **BASIC interpreter**

Computer hardware

Central processing unit ← The brain of the computer.

CPU word size
CPU differences ← Eight bits equal one byte. 1K is equivalent to 1,024 in memory capacity.

Memory ← Memory holds the program and data for the computer to use.

Semiconductor memory
Memory capacity
Expansion memory
Magnetic tape and disk

Peripheral expansion system

Speech synthesizer ← Allows computer to generate speech from program instructions.

Input/output devices

Printers ← Provides hard copy output from computer.
RS-232 Interface
Modems for telephone communication

Summary

Going further

Summary map

4 **Introduction**

4 **Modes**

Paths of information flow

The BASIC interpreter

The immediate mode

Processing BASIC commands

The program mode

Typing in BASIC statements
Executing BASIC statements

7 **Getting started**

8 **Instructions in the immediate mode**

Using PRINT in a command

The CALL CLEAR command

10 **Instructions in the program mode**

Line numbers

The RUN Command

How to end program statements

The RUN command

Stopping and restarting a program

An exercise in the program mode

The CALL CLEAR statement

Using PRINT in a statement

Adding a program line

The LIST command

14 **Correcting errors**

Retyping

Editing

If one character is incorrect in the line
If one or more characters have been left out
If one or more characters need to be removed

15 **More commands and programming exercises**

The NEW command

Another exercise in the program mode

15 **Naming variables**

16 **The LET statement**

Examples of assigning numeric values to variable names

Examples of assigning strings to variable names

17 **The REM statement**

17 **The INPUT statement**

Example program using an INPUT statement

18 **INPUT with a prompting message**

Example program with an INPUT prompt

19 **Punctuation in INPUT statements**

Colons in INPUT statements

Commas in INPUT statements

19 **Formatting computer output**

Semicolons in PRINT statements

Example program using semicolons

Commas in PRINT statements

Colons in PRINT statements

The TAB function

The PRINT TAB statement

Example program using PRINT TAB

22 **The GOTO statement**

Example program using GOTO

23 **The IF-THEN statement**

Relational operator symbols

Example program with relational operator in condition

Example program with computation in condition

Example program with computation in condition

Example program with string variables

26 **The FOR-NEXT statement**

Example program to print out all numbers from 2 to 20

Example program to print out only the even numbers from 2 to 20

Nesting FOR-NEXT loops

Example program with nested FOR-NEXT loop

Using FOR-NEXT to create a time delay loop

Example program with a time delay loop

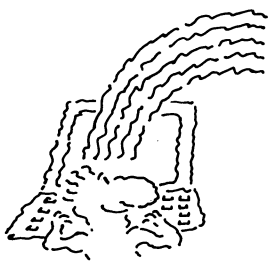
Using a FOR-NEXT loop to accumulate a sum

Example program to accumulate a sum

30 **The GOSUB and RETURN statements**

Example program using GOSUB

Example Color Program One



31 **Color graphics**

The CALL SCREEN statement

The CALL COLOR statement

The CALL HCHAR and CALL VCHAR statements

Example color program one
Example color program two

34 **The RND function**

Example program using the RND function

35 **The RANDOMIZE statement**

Example program using the RANDOMIZE statement

35 **The INT function**

Example program using the INT function

36 **Evaluation of equations**

Example program for evaluating an equation

38 **Summary map**

Beginning with BASIC

Introduction

The first three chapters have described what a home computer is, how it works, and how to use it with preprogrammed software. The remaining chapters describe how you can program your home computer.

To program your home computer, you must learn to communicate with it in a language called BASIC which your computer understands. This chapter teaches you the essential BASIC instructions. Chapter 5 tells you how to combine BASIC instructions to create programs. The remaining chapters describe programs for specific applications and teach you more advanced BASIC instructions.

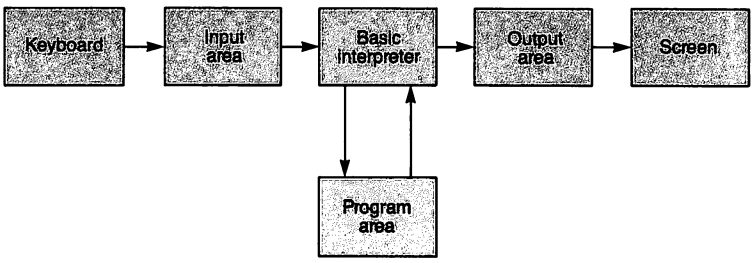
Modes

The computer has two modes of operation. In the first mode, called the immediate mode, the computer responds immediately to each of your requests, but doesn't remember the request. If you want it to do the same thing again, you have to request it again. In this mode, you can get the computer to print requested information on the screen or perform calculations and print the answers on the screen.

In the second mode, called the program mode, the computer stores each of your requests in its memory. You indicate the order in which the requests are to be carried out by numbering each request or statement. A set of numbered statements is called a program. A program tells the computer how to ask for data from you, how to process the data, and in what form to output the data. Since the program is stored in memory, the program can be recalled and used again and again.

Paths of information flow

Information entered at the keyboard in the BASIC language flows through the input area before being interpreted. The BASIC interpreter translates instructions written in BASIC into a code the computer can understand. If the information is a program statement, it is sent to the program area for storage. Information coming from the interpreter then flows through the output area before being displayed on the screen.



The BASIC interpreter translates BASIC instructions into machine language instructions.

The BASIC interpreter

The BASIC interpreter is a permanently stored program that reads the instructions in the BASIC language and interprets or translates them to the language of the computer. Without it, we would have to learn the complex machine language that the computer understands. Therefore, the interpreter saves us time and trouble and allows us to communicate with the computer in a more natural, English-like language.

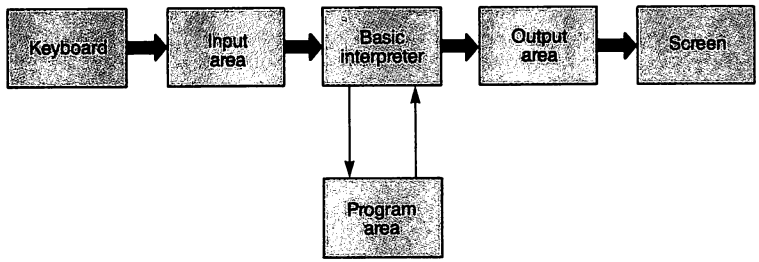
The immediate mode

In the immediate mode, the instructions you enter are performed immediately after you press the **ENTER** key. In computer jargon, we say "The computer executes these instructions in the immediate mode."

Instructions that are given to the computer in this mode are called *commands*, and the BASIC interpreter executes commands in the Immediate Mode.

Processing BASIC commands

Information is not sent into the program area in this mode. The commands are processed directly from the input area, and the results are sent directly to the screen. The command itself is also sent to the screen so that you can see what you are typing on the keyboard. Because the commands are not stored in program memory, they are simply executed and then "forgotten". To execute a command more than once, you must type the entire command each time.



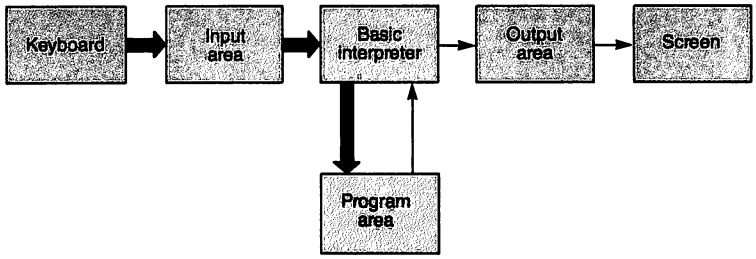
The program mode

Instructions given to the computer in the program mode must have a number, called the line number, before each instruction. These instructions are not performed immediately, but are stored in a part of the computer's memory that is set aside for this purpose. This allows you to build groups of instructions, called programs, that cause the computer to perform certain tasks when they are "executed" or "run." When you run the program, the instructions are executed one instruction after the other in the line number order.

Instructions that are given to the computer in the program mode are called *statements*. The BASIC interpreter executes program statements in the program mode.

Typing in BASIC statements

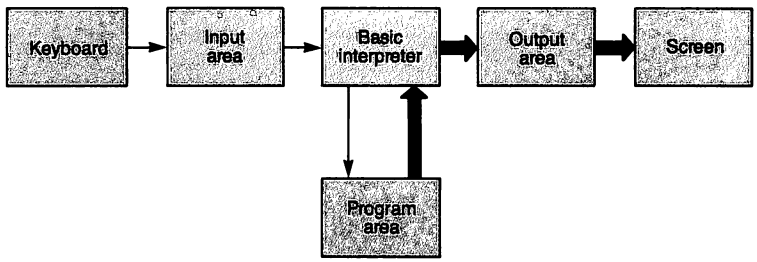
When BASIC program statements are typed and the **ENTER** key is pressed, the statements are stored in the program area for execution at a later time. The only output to the screen is the statement itself so that you can see it as you are typing and constructing your program.



Executing BASIC statements

As a BASIC program is executed, the interpreter follows these steps:

1. Reads one statement from the program area.
2. Performs the specified instruction, sending output to the screen or getting input from the keyboard if necessary.
3. Reads the next statement from the program area and performs the required functions.
4. Continues to execute one statement at a time until the end of the program is reached or one of the statements tells the interpreter to stop.

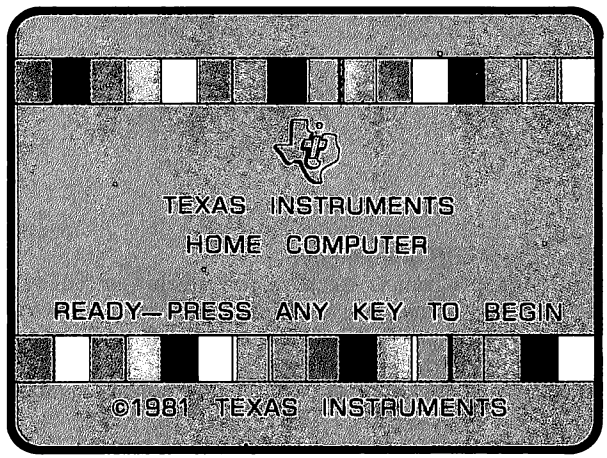


Getting started

A good way to learn how to program the computer is to give it instructions and watch how it responds to each of the instructions it receives. This is the method we will use in this chapter. If you have a TI Home Computer, we encourage you to enter the examples into your computer as you read through them. This not only makes the function of each instruction clearer and easier to follow, but also it is exciting to see the computer respond to the instructions you have given it.

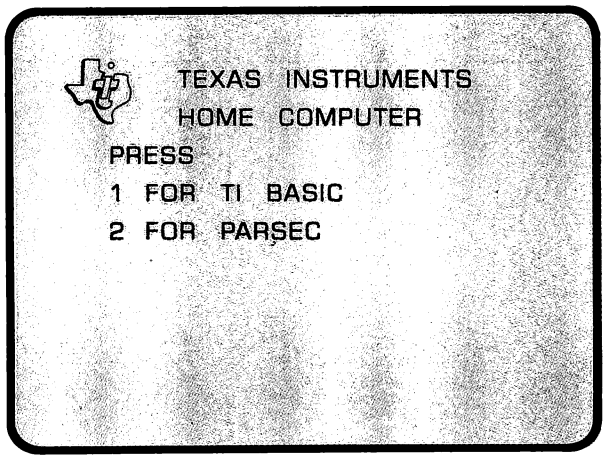
Before you can give the computer instructions in the BASIC language, you must follow three simple steps. These steps are much like those you did in Chapter 2 when you used preprogrammed cartridges.

1. Turn on the computer and television set.



2. Press any key to get the Master Selection Menu.

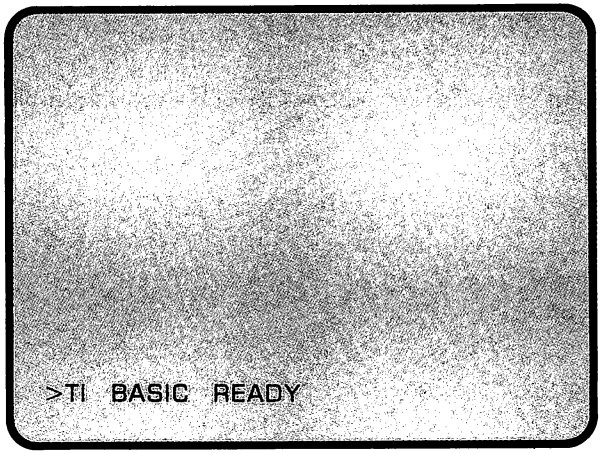
This type of screen is called a "menu" screen because it allows you to select a choice much as you would order from a restaurant menu.



If you still have a cartridge plugged into the slot on the console, its title will be included as the second, and possibly third or fourth, option on the list, just as it was before. If there is no cartridge in the slot, the only item on the list will be TI BASIC.

3. Press the **1** key to select TI BASIC.

The computer is now ready to receive instructions in the BASIC programming language.



Instructions in the immediate mode

Because each command entered in the immediate mode is executed as soon as the **ENTER** key is pressed, you can see the results of the command instantly. This is an excellent way to learn the functions of the individual BASIC instructions.

The PRINT instruction is a good place to start. PRINT is to the computer what speaking, sign language, and writing are to us.

Using PRINT in a command

PRINT is the BASIC language instruction that tells the computer to print information on the screen. Let's try this instruction as a command in the immediate mode to see how it works.

Type PRINT "HELLO"
Press **ENTER**

Note
Notice that the letters you want printed on the screen must be enclosed in quotation marks.

```
PRINT "HELLO"
HELLO
```

Note
Throughout the remainder of the book, actual screen output is indicated by the curved line representing the upper left-hand corner of the screen boundary.

The PRINT instruction can also be combined with an expression to perform a mathematical computation. Certain computer keys perform mathematical operations.

Key	Operation
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponentiation (raising a number to a power)

Try these examples to see how PRINT performs mathematical computations.

Type PRINT 36 + 49
Press **ENTER**

```
PRINT 36 + 49
 85
```

Type PRINT 36/49
Press **ENTER**

```
PRINT 36/49
 .7346938776
```

In these examples, notice that the computer prints only the answer as output. If you want the problem printed also, you must enclose in quotation marks what you want printed.

Note
The use of the semicolon is very important. It will be explained later.

Type PRINT "36 + 49 =";36 + 49
Press **ENTER**

```
PRINT "36 + 49 =";36 + 49
 36 + 49 = 85
```

The CALL CLEAR command

CALL CLEAR is actually a subprogram, but it can be used like a command in the immediate mode or as a statement in the program mode. As you might have guessed, it tells the computer to clear the screen.

Type CALL CLEAR
Press **ENTER**

Note
CALL CLEAR doesn't erase program instructions from memory.

All the information that you typed and the computer printed is erased off the screen. If you had placed any instructions in the computer's memory, however, they would not be erased. CALL CLEAR erases only the screen.

Instructions in the program mode

A set of instructions arranged in a particular order to direct the computer to perform a task is called a computer program.

Just as a sentence in the English language must have its parts arranged in a particular order, so it is with computer instructions. A computer program can be organized to do computations, make a table of names, print a message on the screen, and many other things. Without careful attention to the order of your program statements, however, you will not get the result you want from your program.

Line numbers

A line number at the beginning of each program statement tells the computer the order in which you want that instruction performed. It also tells the computer to store the instruction in the program area of its memory, rather than performing the instruction immediately.

Line numbers can be any numbers from 1 through 32767. Each statement in a program must have a different line number. If you use the same number twice in the same program, the last statement will replace the first.

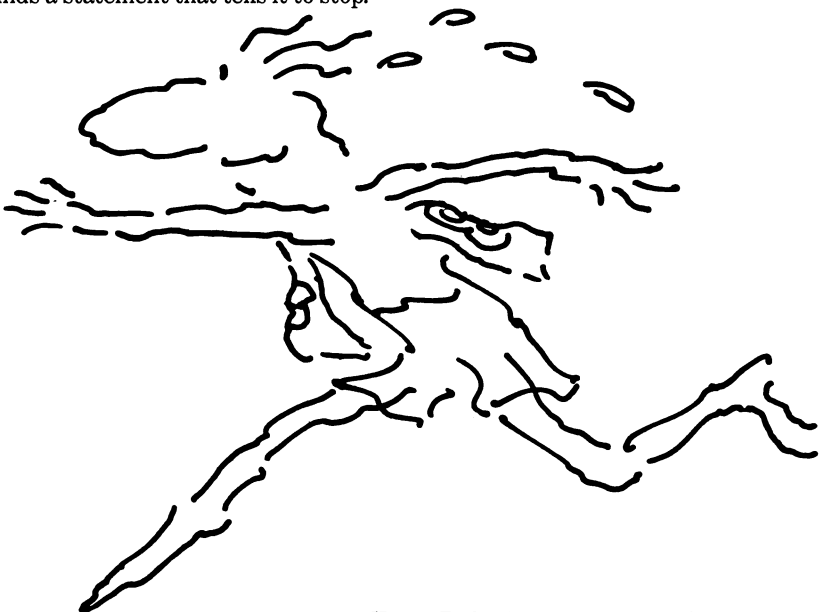
How to end program statements

You must also tell the computer where each statement ends. Pressing **ENTER** is your signal to the computer that you have finished one statement and are ready to go to another.

When you press **ENTER** at the end of a statement, the computer stores the line number and statement in the program area of its memory. You are then ready to enter another statement or to run the program if you have finished entering all your statements for the program.

The RUN command

The RUN command instructs the computer to execute the program in its memory. When you type RUN and press **ENTER** the computer executes each instruction in sequence, beginning with the lowest numbered statement and proceeding on with the next higher numbered statement until it reaches the last line of the program or until it finds a statement that tells it to stop.



Stopping and restarting a program

There are two ways to stop a program while it is running. Turning the computer off will stop it, but you also lose the program. A better way to stop a program is to enter FCTN 4. To do this:

Hold down the **FCTN** key
Type 4

If you wish to restart the program:

Type CONTINUE
Press **ENTER**

An exercise in the program mode

The best way to learn how a program works is to enter some statements, run the program, and watch the results. If you have turned the computer off, turn it on again and select TI BASIC.

The CALL CLEAR statement

CALL CLEAR can be used as a statement in a program if you give it a line number. It is good programming practice to include a CALL CLEAR statement as the first line in a program. It then automatically clears the screen every time you run the program. You can add a CALL CLEAR statement in a program anywhere you want the screen to be erased and you can use as many CALL CLEAR statements as you need. A CALL CLEAR statement will be used in the next program.

Using PRINT in a statement

PRINT can be used both as a command in the immediate mode and as a statement in the program mode. You have seen how it performs in the immediate mode; now you can see how it works in the program mode.

Type
10 CALL CLEAR
Press **ENTER**
20 PRINT "HELLO"
Press **ENTER**

When you press the **ENTER** key, the computer stores the statement in memory and waits for you to type the next statement or run the program. So let's run the program.

Type RUN
Press **ENTER**

```
HELLO

** DONE **
```

Note
If you made an error when you entered the program, the computer will stop executing the program at the error and print an error message on the screen to help you find it. When you find an error, just retype the line correctly using the same line number (remember, the new line replaces the old one), and press ENTER. Then run the program again.

Now run the program again by typing RUN and pressing **ENTER.**The computer does exactly the same thing without your having to retype the instructions.

Let's add some more statements to the program. Remember to press **ENTER** at the end of each statement. Special attention is required in line 40 and line 60. The statement does not end at "PLUS", but continues on the next screen line.

Note
Don't mistake the 47 in lines 40 and 60 for a line number.

Type
```
30 PRINT "LET'S ADD"
40 PRINT "THE SUM OF 53 PLUS
 47 IS"
50 PRINT 53+47
60 PRINT "THE SUM OF 53 PLUS
 47 IS";
70 PRINT 53+47
80 END
```

This is a simple computer program. You told the computer what you want it to do by entering the instructions through the keyboard. When you run the program, the computer will read the statements and execute the program in the order of the line numbers, from lowest to highest. Try it.

Type RUN
Press **ENTER**

Note
Notice that the semicolon at the end of line 60 told the computer not to move down to the next screen line, but to print the result (line 70) on the same screen line. You'll see more examples of this later.

```
HELLO
LET'S ADD
THE SUM OF 53 PLUS 47 IS
 100
THE SUM OF 53 PLUS 47 IS 100

** DONE **
```

Adding a program line

Did you notice in the previous exercise that the program lines were numbered 10, 20, 30, and so on rather than 1, 2, 3, and so on? Programmers do this to allow a program to be changed without having to retype and renumber all the lines. Line numbers were purposely left out between the statements so that new lines can be added. To see how this works, add new line 31.

Type 31 PRINT "OK"
Press **ENTER**

Run the program again.

Type RUN
Press **ENTER**

```
HELLO
LET'S ADD
OK
THE SUM OF 53 PLUS 47 IS
 100
THE SUM OF 53 PLUS 47 IS 100

** DONE **
```

The LIST command

The LIST command without any numbers following it tells the computer to print on the screen all the program statements in its memory. If you only want one line listed, type after LIST a space and the line number; for example,

LIST 20

If you want only a particular group of lines, type after LIST a space, the first line number, a hyphen, and the last line number; for example,

LIST 20-50

Clear the screen and let's list the entire program.

Note
From here on, we won't tell you when to press ENTER. Remember that you have to do it after typing each statement and each command, such as RUN, LIST and CALL CLEAR.

Type CALL CLEAR
Type LIST

```
LIST
10 CALL CLEAR
20 PRINT "HELLO"
30 PRINT "LET'S ADD"
31 PRINT "OK"
40 PRINT "THE SUM OF 53 PLUS
47 IS"
50 PRINT 53+47
60 PRINT "THE SUM OF 53 PLUS
47 IS;"
70 PRINT 53+47
80 END
```

When you enter the LIST command, the computer lists all the statements in sequence from smallest to largest line number. Therefore, you see that line 31 has been put in the correct order, right after line 30, even though you entered it last.

Correcting errors

Unfortunately, we all make errors, but fortunately, the computer checks for errors. When it finds one, it will print an error message to help you find and correct it. You probably will need to list all or part of the program to find the error.

The most common errors for beginners are:

1. Misspelling.
2. Leaving off quotation marks in PRINT statements.
3. Using the wrong punctuation.
4. Mixing up zero and the letter O.

Retyping

One obvious way to correct an error is to retype the entire line including the line number. Of course, this can take quite a bit of time if the line is very long. What's worse, you may make a new error while retyping.

Editing

A faster way to correct errors is to use the edit mode. For example, let's assume that you had made an error in line 10 of the previous program. You can edit the line without retyping the whole line.

Type EDIT 10

```
EDIT 10
10 CALL CLEAR
```

Note
To input any of the FCTN actions, hold down the FCTN key and type the letter or number.

If one character is incorrect in the line

1. Use the right-arrow key (**FCTN D**) or the left-arrow key (**FCTN S**) to position the cursor over the character to be changed.
2. Type the correct character. The incorrect character is replaced by what you type.
3. Press **ENTER.**

If one or more characters have been left out

1. Use the right-arrow key (**FCTN D**) or the left-arrow key (**FCTN S**) to position the cursor over the character to the right of the missing character or characters.
2. Press INSert (**FCTN 2**).
3. Type the correct character or characters, including spaces.
4. Press **ENTER.**

If one or more characters need to be removed

1. Use the right-arrow key (**FCTN D**) or the left-arrow key (**FCTN S**) to position the cursor over the character to be deleted.
2. Press DELete (**FCTN 1**). If you need to delete more characters, including spaces, press DELete for each character you want to remove.
3. Press **ENTER.**

More commands and programming exercises

The NEW command

Note
The CALL CLEAR command clears only the display screen. It does not erase the program from memory. The NEW command does both.

This command erases the present program from the computer's memory and clears the screen. If you want to erase the current program from the computer's memory:

Type NEW
Press **ENTER**

Another exercise in the program mode

The more you practice, the easier it will be for you to program. Try entering and running this new program.

```
NEW
100 CALL CLEAR
120 PRINT "I AM PROGRAMMING THIS COMPUTER."
130 PRINT "IT DOES EVERYTHING I TELL IT TO DO,"
140 PRINT "INCLUDING MISSPELLING WORDS I MISSSSPELL!"
150 PRINT "ADDITION AND SUBTRACTION ARE EASY. WATCH THIS!"
160 PRINT "5 + 8 =";5+8,"13 - 6 =";13 - 6
170 END
RUN
```

```
I AM PROGRAMMING THIS COMPUT
 ER.
IT DOES EVERYTHING I TELL IT
 TO DO,
INCLUDING MISSPELLING WORDS
 I MISSSSPELL!
ADDITION AND SUBTRACTION ARE
 EASY. WATCH THIS!
5 + 8 = 13   13 - 6 = 7

** DONE **
```

Naming variables

A variable in a program can have different values assigned to it at different times. Each different variable must have a different name. Variable names must always start with a letter. They may be only one character or up to 15 characters long on the TI Home Computer. Since each character takes up memory space, it is good practice to use short variable names to save memory.

Variables can be either numeric or string.

There are two types of variables; numeric and string. Numeric variables are numbers. For example, you can assign the numeric variable AZ to mean 18. Then, when the computer performs an instruction using variable AZ, it knows that AZ equals 18.

String variables are different. Like numeric variables, they must be designated by a letter, or a combination of letters and numbers, but unlike numeric variables the string variable must be followed by a dollar sign ($). For example, you could designate string variable AZ$ to mean the name, Tom. When the computer performs

an instruction using AZ$, it knows that AZ$ is equal to Tom. Numbers can be assigned to string variables, but string variables cannot be used in arithmetic calculations; only numeric variables can be used in arithmetic calculations.

The LET statement

The LET statement assigns a value to a variable name. The LET statement requires a line number, the variable name, an equal sign and finally the value assigned to the variable name.

Examples of assigning numeric values to variable names

Example statement	Numeric variable name	Numeric value assigned
10 LET A = 5	A	5
20 LET NUMBER = 25	NUMBER	25
30 LET R2 = 100	R2	100

The PRINT and LET statements make a powerful combination. Try the following program to see how they work together.

Instruction	Explanation
NEW	Clears memory and gets ready for new program
10 CALL CLEAR	Clears screen before beginning output
20 LET A = 2	Assigns A the value of 2
30 LET B = 3	Assigns B the value of 3
40 LET ANSWER = A + B	Assigns ANSWER the sum of A and B
50 PRINT A;"+";B"=";ANSWER	Prints the equation and the sum of A and B
60 END	Stops the program
RUN	Tells the computer to run the program

```
 2 + 3 = 5

** DONE **
```

Note
When assigning a string of alphanumeric characters to a string variable, the characters in the string must be enclosed within quotation marks.

Examples of assigning strings to variable names

Example statements	String variable name	String assigned
10 LET A$ = "INNING"	A$	INNING
20 LET A2$ = "TODAY IS MONDAY"	A2$	TODAY IS MONDAY
30 NAME$ = "SUSAN"	NAME$	SUSAN

Note
Notice the last statement. TI BASIC allows you to omit the word LET in any LET statement. The choice is yours.

Now try a program that includes string variables.

Instruction	Explanation
NEW	Clears memory
10 CALL CLEAR	Clears screen
20 LET CITY$ = "PHOENIX,"	Assigns CITY$ the value of PHOENIX,
30 LET STATE$ = "ARIZONA"	Assigns STATE$ the value of ARIZONA
40 PRINT CITY$;" ";STATE$	Prints the variables
50 END	Stops the program
RUN	Run the program

```
PHOENIX,  ARIZONA

** DONE **
```

The REM statement

The REMark statement provides a means to insert comments into the program. REM statements can appear anywhere in a program. REM statements are not executed in the program or printed as program output, but they are shown in a program LISTing. During program execution, the computer simply ignores a REM statement and goes on to the next instruction. However, a REM statement does take up space in the computer's memory, so that must be kept in mind.

Simplicity and clarity are the key to good REM statements. If they are used to help explain how the program works, they can be very useful to someone trying to modify a program at some later time. The format for a remark statement is a line number followed by REM followed by the comment. This example shows how REM statements can be used to title a program:

```
10 REM  THIS PROGRAM COMPUTES
20 REM  COST OF A NEW CAR LOAN
```

The INPUT statement

The INPUT statement provides a method for entering information from the keyboard as called for by a program that is running.

When the computer encounters an INPUT statement in a program, it prints a question mark on the screen and waits for the information to be entered from the keyboard. You type the correct value, either string or numeric, and press ENTER to instruct the computer to accept the value and continue program execution.

An INPUT statement consists of a line number followed by the word INPUT and a variable name. The variables can be either numeric or string (alphanumeric). You must enter a numeric value for a numeric variable when you run the program. If you don't, the computer will give you an error message and ask you to try again.

Example statements	Variable name
10 INPUT A$	A$
20 INPUT A2	A2
30 INPUT NAME$	NAME$

Example program using an INPUT statement

Instruction	Explanation
NEW	Clears memory
10 CALL CLEAR	Clears screen
20 PRINT "GIVE ME A NUMBER"	Prints the sentence in quotation marks
30 INPUT K	Prints ?, waits for you to type a number and press ENTER, assigns the value to the variable name K, and continues the program
40 PRINT "YOUR NUMBER IS";K	Prints the sentence in quotation marks and the number you entered
50 END	Stops the program
RUN	Run the program

Note
When the question mark appears on the screen, type 3 and press ENTER. Input that you type is indicated here and in the remainder of the book by the heavy bold type.

```
GIVE ME A NUMBER
? 3
YOUR NUMBER IS 3

** DONE **
```

INPUT with a prompting message

In the previous example, line 20 printed a message on the screen to prompt you for the kind of information to be entered. However, you can include a prompt message in the INPUT statement so that a separate PRINT statement is not required for the prompt.

Example program with an INPUT prompt

Instruction	Explanation
NEW	Clears memory
10 CALL CLEAR	Clears screen
20 INPUT "WHAT IS YOUR NAME ":NAME$	Prints the sentence in quotation marks, waits for you to type a name and press ENTER, assigns the value to the variable name NAME$, and continues the program
30 PRINT "MY NAME IS ";NAME$	Prints the sentence in quotation marks and the name you entered
40 END	Stops the program
RUN	Run the program

Note
Notice that you had to include a space at the end of each sentence in the quotation marks. This will be explained later.

Note
Notice that the question mark does not appear on the screen when you include a prompt message in the INPUT statement. If a question mark is needed (as in this example), you should include it in the prompt message.

```
WHAT IS YOUR NAME JILL
MY NAME IS JILL

** DONE **
```

Punctuation in INPUT statements

The colon and the comma are two punctuation marks used with the INPUT statement.

Colons in INPUT statements

In an INPUT statement that includes a prompt message, the colon must be used and it suppresses the question mark prompt. For example, if you enter:

```
10 INPUT K
```

the computer stops the program, displays a question mark and waits for you to enter a value. However, if you enter:

```
10 INPUT "WHAT IS THE NUMBER":K
```

the computer stops the program, displays the prompt without a question mark, and waits for you to enter a value. If a question mark is desirable with a prompt, include it in the prompt; for example:

```
10 INPUT "WHAT IS THE NUMBER?":K
```

Commas in INPUT statements

Commas are used in an INPUT statement if more than one variable is to be assigned; for example:

```
10 INPUT "VARIABLES A, B, AND C? ":A,B,C
```

When this line is executed, the computer prints the prompt and waits as usual. But now, after you type a number for A, you must type a comma, then a number for B, a comma and a number for C before pressing **ENTER.** When you run a program containing this line, you would see the prompt and type in the numbers as shown below.

```
.
.
.
VARIABLES A, B, AND C? 5,12,35
.
.
.
```

If you press **ENTER** before typing in all variables, an error message will be printed.

Formatting computer output

The format of your computer output refers to how the output is positioned on the screen. You can format your computer output in many ways using different punctuation in the PRINT statement and by using the TAB function.

Semicolons in PRINT statements

As you may have noticed from some of the previous program examples, the variable following the semicolon in a PRINT statement is placed next to the previous printed information. With string variables, the string is printed immediately after the prior information with no space (unless a space is included in the string). With numeric variables, however, a positive numeric value has one space between it and the prior information.

The space between the prior information and the first positive number occurs because TI BASIC allows a leading space for the sign of a number in front of the number. The + sign for a positive number is not displayed; therefore, a blank space appears. However, the − sign is displayed before a negative number so there is no extra space. A trailing space is printed after all numeric variable values to guarantee that different values always will be separated. If they weren't, the printed values of two variables would look like one value.

Example program using semicolons

Instruction	Explanation
NEW	Clears memory
10 CALL CLEAR	Clears screen
20 LET A = 1	Assigns value of 1 to A
30 LET B = 2	Assigns value of 2 to B
40 LET A$ = "ANN"	Assigns ANN to A$
50 LET B$ = "BOB"	Assigns BOB to B$
60 PRINT A;B;A$;B$	Prints the values of A, B, A$, and B$ on the same line
70 END	Stops the program
RUN	Run the program

```
 1  2  ANNBOB

** DONE **
```

Note
Notice that two spaces separate the two numbers because of the leading and trailing space of the numbers, but that ANN and BOB are printed together with no separating space.

Commas in PRINT statements

A comma after the first variable in a PRINT STATEMENT causes the second variable in the statement to begin printing in column 15, which is at the approximate center of the screen.

Change the semicolons in line 60 of the previous program to commas and run the program to see the effect of commas in a PRINT statement.

```
 1             2
ANN           BOB

** DONE **
```

Note
Notice the arrangement of the output on the screen. This happens because there are only two print zones on the screen, one starting at column 1 and the other starting at column 15. When the PRINT statement runs out of print zones on one line, but still has more variables in print, the computer goes to the next line to continue printing.

Colons in PRINT statements

A colon between variables in a print statement causes the printing of each variable to begin on a new line of the screen. Change the commas in line 60 of the previous program to colons and run the program to see the effect of colons in a PRINT statement.

Note
The program now prints the value of each variable on a different line. You could have achieved the same results by including a separate PRINT statement for each variable, but think of all the typing you saved by using colons.

```
 1
 2
ANN
BOB

** DONE **
```

The TAB function

A function is a short subroutine made up of a few instructions to accomplish a task that is needed often in programs. Several are built into the computer and most computers allow the programmer to define other functions. A function may be called in a program by just typing in one word.

The TAB function is used often in PRINT statements to format output. It does the same job as the mechanical tab key on a typewriter; that is, it determines the starting position on the print line for the next print item. The format is:

TAB(numeric expression)

where the numeric expression may be a constant integer number, a numeric variable, or a mathematical expression. A print separator (colon, comma, or semicolon) must be used before a tab function unless it is the first item in the PRINT list. A print separator also must be used after a TAB function except when it is the last item in the PRINT list. A semicolon usually is used in both places. More than one TAB may be used in a PRINT list.

The PRINT TAB statement

The PRINT TAB statement permits more flexibility in the formatting of printed output than the various punctuation marks used with the PRINT statement. The format for the PRINT TAB statement is:

PRINT TAB(n)

where n specifies any column position on the screen. For example, PRINT TAB (12) causes printing to begin in the twelfth column on the screen. The (n) may be an integer from 1 to 28 since there are 28 columns (horizontal positions) on the screen.

Example program using PRINT TAB

Instruction	Explanation
NEW	Clears memory
5 CALL CLEAR	Clears screen
10 A = 25	Assigns value of 25 to A
20 B = 15	Assigns value of 15 to B
30 PRINT TAB (10); A	Begins printing the value of A in column 10 (sign space is in column 10)
40 PRINT TAB (B); "TAB"	Begins printing TAB in column 15 because B = 15
50 PRINT TAB (B-5); 34	Begins printing the constant 34 in column 10 because B = 15 and 15-5 = 10 (sign space is in column 10)
60 END	Stops the program
RUN	Runs the program

Note
A semicolon must separate the PRINT TAB(n) statement from the variable that is to be printed.

```
 25
    TAB
 34

** DONE **
```

The GOTO statement

The statements we've discussed so far are important for input, output and doing computations. Now let's look at statements to make computer programs more useful and efficient.

The GOTO statement tells the computer to quit following the normal sequence of line numbered instructions and to go to the specified line number in the program. The format for the GOTO statement is:

GOTO (line number)

Note
GOTO also may be written as GO TO (with a space).

The line number may be lower or higher than the current one. This is called an unconditional branch because the computer obeys the command without questioning any conditions. Here's a simple program that illustrates a GOTO branch to a lower number line to create a loop.

Example program using GOTO

Instruction	Explanation
NEW	
5 CALL CLEAR	Clears screen
10 LET K = 1	Assigns 1 to K
20 PRINT K	Prints value of K
30 LET K = K + 1	Adds 1 to K each time through this statement
40 GOTO 20	Unconditional branch to line 20
50 END	
RUN	

Note
In line 40, the GOTO statement tells the computer to execute line 20 next.

Note
This program is in an endless loop and will continue counting by one and printing the numbers until you enter FCTN 4.

```
 1
 2
 3
 4
 .
 .
 .
 .
* BREAKPOINT AT 30
```

The IF-THEN statement

The IF-THEN statement permits conditions to be tested and a decision made based on the results of the test. It is called a conditional branch statement. In other words, IF a certain condition exists (is true), the THEN part of the statement is followed. IF the condition is not true, the THEN part is ignored and the next line of the program is executed. The format is:

IF (condition) THEN (line number)

The conditional part of the statement (IF) can use the relational operator symbols with either numeric or string expressions.

Relational operator symbols

=	equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
<>	not equal to

Example program with relational operator in condition

If the previous program is still in memory, just change line 40 to obtain this program which prints the numbers 1 to 10 and stops. The value of K is checked each time in line 40 to see if K is less than 11. IF that condition is true, THEN the program loops back to line 20. IF the condition is not true (K is 11 or more), execution goes to the next line, line 50.

Instruction	Explanation
5 CALL CLEAR	Clears screen
10 LET K = 1	Assigns 1 to K
20 PRINT K	Prints value of K
30 LET K = K + 1	Increases K by 1 each time through this statement
40 IF K < 11 THEN 20	Branches to line 20 IF K is less than 11, otherwise the program continues at line 50
50 END	Terminates execution of the program
RUN	

Note
The IF-THEN statement can be used to avoid the problem of an endless running program loop.

Try experimenting with this program by changing the value (11) in line 40 to various other numbers and watch how the output changes.

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10

** DONE **
```

Example program with computation in condition

This program executes the loop in lines 30 to 50 until A = 5 since 3*3-4 = 5. When A = 5, execution proceeds with line 60. Thus, this example shows branching to a statement with a higher statement number than the current one. Such branching is referred to as forward transfer.

Instruction	Explanation
NEW	
5 CALL CLEAR	Clears screen
10 A = 1	Assigns 1 to A
20 B = 3	Assigns 3 to B
30 IF A = 3 * B - 4 THEN 60	Exit loop if A = 5
40 A = A + 1	Increase A by 1 each time through
50 GOTO 30	Unconditional branch to 30 for loop
60 PRINT "A =";A	Print output
70 END	
RUN	

```
A  =  5

** DONE **
```

Example program with string variables

The next program example shows the use of relational operators with string variables to test the alphabetical order of two words. If the words are already in alphabetical order, they are printed as entered. If not, the order is changed before printing.

Instruction	Explanation
NEW	
5 CALL CLEAR	Clears screen
10 INPUT "EXPRESSION FOR A$? ": A$	Prints the prompt, waits for you to type a word and press ENTER, assigns the value to the variable A$ and continues the program
20 INPUT "EXPRESSION FOR B$? ": B$	Prints the prompt, waits for you to type a word and press ENTER, assigns the value to the variable B$ and continues the program
30 IF A$ <= B$ THEN 70	Check present alphabetical order. If in order, go to line 70
40 T$ = A$	Move A$ string to temporary variable T$
50 A$ = B$	Move B$ string to A$
60 B$ = T$	Move T$ (former A$) string to B$
70 PRINT:A$: B$	Print strings in alphabetical order
80 END	End program
RUN	

Note
Lines 40, 50 and 60 swap the order of the A$ and B$ expressions.

```
EXPRESSION FOR A$? SMITH
EXPRESSION FOR B$? BAKER

 BAKER
 SMITH

** DONE **
```

Note
The blank line in the output is caused by the first colon in line 70.

The FOR-NEXT statement

The FOR-NEXT statement is a looping instruction. It can be used to execute a statement or group of statements a specified number of times. Thus, the FOR-NEXT statement can be used in the place of loops programmed by the GOTO and IF-THEN statements as used in the examples above.

The FOR-NEXT is a paired statement. This means that both the FOR and the NEXT are required each time the statement is used. The format is:

```
FOR V = a TO b STEP c
NEXT V
```

where:

1. V is the control variable name. It can be any valid variable name.
2. "a" specifies the starting value for V.
3. "b" specifies the ending value for V.
4. STEP c specifies the step size or how much the loop counter is changed for each pass through the loop. If the STEP size is not indicated, it is automatically set to one.
5. NEXT changes the control variable V by the STEP size and tests whether or not the loop has been performed the specified number of times. If not, the loop is repeated. If so, the looping ends and the program line following the NEXT statement is executed.

Example program to print out all numbers from 2 to 20

Instruction	Explanation
NEW	
10 FOR X = 2 TO 20	Assigns X as control variable, specifies 2 as starting value for X, and specifies 20 as ending value of X
20 PRINT X	Prints value of X
30 NEXT X	Increases value of X by 1, checks if X is greater than (>) 20; if not, loops to line 10, if so, goes to next line
40 END	Program ends

Example program to print out only the even numbers from 2 to 20

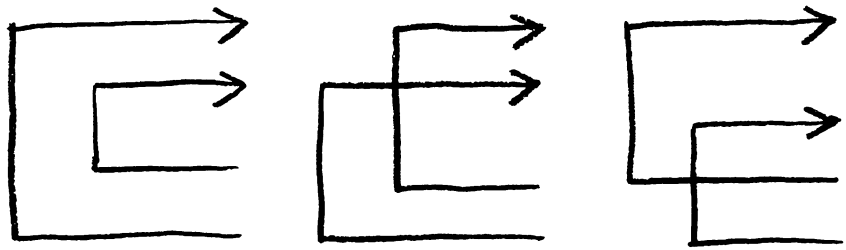
Instruction	Explanation
10 FOR X = 2 TO 20 STEP 2	Same as above except step size is set to 2
20 PRINT X	
30 NEXT X	Increases value of X by 2, remainder as above
40 END	

You can produce this program by using the EDIT mode to change line 10.

Nesting FOR-NEXT loops

A FOR-NEXT loop can be placed within another FOR-NEXT loop. The inner loop is called a nested loop.

There are two important rules for nested loops. One is that a different control variable must be used in each loop. The second is that nested loops cannot cross each other.



Correct nesting *Incorrect nesting* *Incorrect nesting*

Example program with nested FOR-NEXT loop

This program will print out the first value of X (2), then all values of Y (2 through 20), then the second value of X (4), then all values of Y (2 through 20), then the third value of X (6), and so on until X is greater than 20.

Instruction	Explanation
NEW	
10 FOR X = 2 TO 20 STEP 2	Assigns X as control variable, 2 as starting value, 20 as ending value, 2 as step size
20 PRINT X	Prints value of X
30 FOR Y = 2 TO 20	Assigns Y as control variable for inner loop, 2 as starting value, 20 as ending value
40 PRINT Y	Prints value of Y
50 NEXT Y	Increases value of Y by 2, checks if Y > 20; if not, loops to line 30, if so, goes to next line
60 NEXT X	Increases value of X by 2, checks if X > 20; if not, loops to line 10, if so, goes to next line
70 END	Ends program

Note
The FOR X-NEXT X loop is the outer loop and the FOR Y-NEXT Y is the nested loop.

Using FOR-NEXT to create a time delay loop

The following program replaces the GOTO and IF-THEN statements in a previous example program with a FOR-NEXT statement. The STEP is omitted since a STEP size of 1 is desired. A nested FOR-NEXT loop provides a time delay between the printing of numbers.

Example program with a time delay loop.

Instruction	Explanation
NEW	
10 CALL CLEAR	Clears screen
20 K = 1	Assign the value of 1 to K
30 FOR A = 1 TO 10	Causes execution of lines 30 - 80 ten times
40 PRINT K	Prints the value of K each time through loop
50 K = K + 1	Increases the value of K by one each time through loop
60 FOR B = 1 TO 500	Causes execution of lines 60 and 70 to be done 500 times for time delay
70 NEXT B	End of loop B
80 NEXT A	Increases value of A by 1, checks if A > 10; if not, loops to line 30, if so, goes to next line
90 PRINT "LOOPS COMPLETED"	Prints message
100 PRINT A	Prints value of A
110 END	Ends program
RUN	

Note
Note that K has nothing to do with the loop control.

Note
Observe that for this program, the same output can be obtained by changing line 40 to PRINT A and deleting lines 20 and 50.

Note
Time delay loops are often used to hold the computer display for a few seconds to allow time to read information on the display.

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
LOOPS  COMPLETED
 11

** DONE **
```

Using a FOR-NEXT loop to accumulate a sum

The need to accumulate a sum is required in many programs. For example, to find the average monthly income for a year, the sum of the 12 monthly incomes is divided by 12 to get the average monthly income.

Example program to accumulate a sum

Instruction	Explanation
NEW	
20 CALL CLEAR	Clears screen
30 TOTAL = 0	Set sum accumulator variable to zero
40 FOR I = 1 TO 12	Begin FOR-NEXT loop
50 INPUT "MONTHLY INCOME? ":MONTHLY	Prints prompt and assigns value typed to variable MONTHLY
60 TOTAL = TOTAL + MONTHLY	Adds value last input to sum accumulator variable
70 PRINT TOTAL	Prints value of TOTAL which is sum accumulated so far
80 NEXT I	End of FOR-NEXT loop; when I > 12, goes to next line
90 PRINT:"AVERAGE MONTHLY INCOME ="	Prints output heading
100 PRINT TOTAL/12	Prints average monthly income obtained by dividing accumulated sum by 12
110 END	Ends program
RUN	

Note
When a sum accumulator variable is used, the variable must be set to zero at the beginning of the program.

```
MONTHLY INCOME? 1000
  1000
MONTHLY INCOME? 2000
  3000
MONTHLY INCOME? 6000
  6000
MONTHLY INCOME? 1000
  7000
MONTHLY INCOME? 2500
  9500
MONTHLY INCOME? 500
  10000
MONTHLY INCOME? 1500
  11500
MONTHLY INCOME? 3000
  14500
MONTHLY INCOME? 4100
  18600
MONTHLY INCOME? 5200
  23800
MONTHLY INCOME? 6100
  29900
MONTHLY INCOME? 3300
  33200

AVERAGE MONTHLY INCOME =
  2766.666667

** DONE **
```

The GOSUB and RETURN statements

The GOSUB statement allows a program to be divided into several small programs called subroutines. Long complicated programs can be written in several subroutines and the subroutines put together into one program. Each subroutine is called when needed with the GOSUB statement. The format is:

GOSUB (line number)

Note
GOSUB also may be written as GO SUB (with a space).

A subroutine can be located anywhere before or after the line from which it is called. The subroutine must always end with a RETURN statement. This statement returns program execution to the main program line following the GOSUB which called the subroutine.

Note
RETURN is used alone, without a line number after it.

The GOSUB and RETURN statements permit a subroutine to be used again and again without rewriting it each time. This pair of statements can save much programming effort and should be part of any serious programmer's approach to programming. We'll discuss GOSUB and RETURN further in the next chapter, but the following simple example shows how the pair is used.

Example program using GOSUB

Instruction	Explanation
NEW	
90 CALL CLEAR	Clears screen
100 H = 24	Assigns 24 to H
110 B = 50	Assigns 50 to B
120 GOSUB 200	Branches program to subroutine at line 200
130 PRINT "THE SUM EQUALS";SUM	Prints the output heading and the sum
140 PRINT:"THE PRODUCT EQUALS"; PRODUCT	Prints the output heading and the product
150 END	Ends program
200 SUM = H + B	Computes sum of H and B
210 PRODUCT = H * B	Computes product of H and B
220 RETURN	Returns control to program line following GOSUB (line 130)
RUN	

The GOSUB statement causes the computer to "remember" the line to which it should return.

```
THE SUM EQUALS 74

THE PRODUCT EQUALS 1200

** DONE **
```

Color graphics

The TI Home Computer can be programmed to display patterns in various colors. This is referred to as color graphics. Specific subprograms are used to control the graphics display colors. These subprograms are called by a two-word statement in your program.

The standard display colors are cyan (greenish blue) for the screen and black for the letters when a program is being typed into the computer. When a program is running, the screen color is light green unless statements in the program change the color. When the program stops running, the screen color returns to cyan.

On the TI Home Computer, any letter, number, or symbol in the character sets can be printed in any of the available colors on any color background with any available screen color.

The CALL SCREEN statement

The CALL SCREEN statement can be used in a program to change the screen color. The format is:

CALL SCREEN (n)

where (n) is a code number which specifies the desired color. The 16 available colors in TI BASIC are identified in Chapter 9.

The CALL COLOR statement

The CALL COLOR statement can be used in a program to change the color of characters and the character background. The format is:

CALL COLOR (a, b, c)

where:

1. a refers to the character set number
2. b refers to the character color or foreground color
3. c refers to the background color or color of the block upon which the characters are printed

Note
Background color and screen color do not refer to the same thing.

The following is an example instruction:

Instruction	Explanation
10 CALL COLOR (2, 7, 12)	The 2 is for the character set that contains the asterisk, 7 is for the dark red color of the asterisk, and 12 is for a light yellow background

Characters for the TI Home Computer are identified by character codes. Character codes are grouped into sets of eight characters each. The character codes and sets are identified in the Appendix. The CALL COLOR statement requires only that the set number for the character be specified. The actual character code within the set is specified in the statement that controls the location of the character on the screen (CALL HCHAR or CALL VCHAR discussed below).

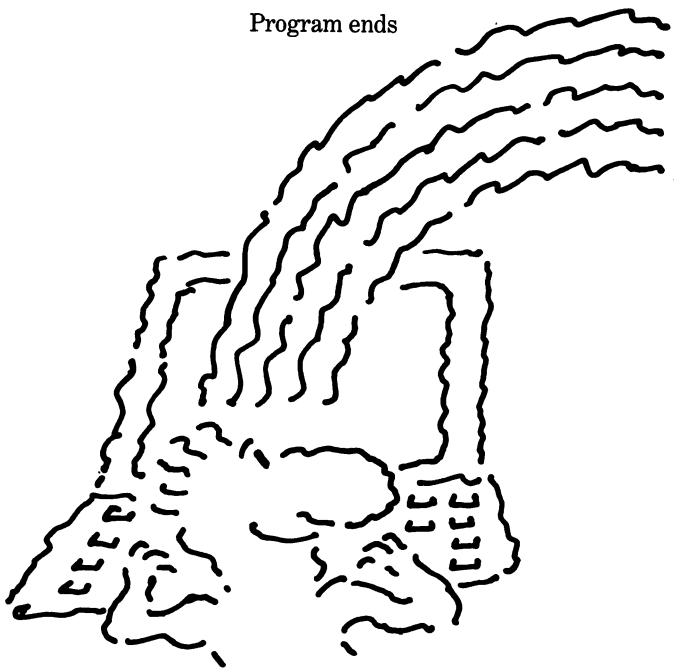
If the foreground and background colors are the same number, then the character is indistinguishable from the background. Also, if the foreground, background, and screen colors are all the same number, the display will appear to be a blank screen of all one color even though characters and background colors are being printed.

The CALL HCHAR and CALL VCHAR statements

The CALL COLOR statement is useless without statements like CALL HCHAR and CALL VCHAR that control the location of the character(s) on the screen. CALL HCHAR prints characters horizontally (across) the screen while CALL VCHAR prints characters vertically (up and down) the screen. These statements will be discussed in more detail in a later chapter, but for now we'll just use them in order to illustrate the color operation. The following program illustrates what happens when these statements are used.

Example color program one
This program displays 28 dark red asterisks on a light yellow background with the standard green screen until the FOR-NEXT delay loop reaches a count of 2000.

Instruction	Explanation
NEW	
10 CALL CLEAR	Clears screen
20 CALL COLOR (2,7,12)	Sets color of asterisk to dark red with a light yellow background
30 CALL HCHAR (12,3,42,28)	Starts printing in row 12, column 3 of the screen; 42 is the code value for the asterisk, and 28 is for how many asterisks are printed
40 FOR DELAY = 1 TO 2000	Delay loop to hold display for several seconds
50 NEXT DELAY	
60 END	Program ends



Example color program two
This program allows you to input color selections at the keyboard. It goes through ten cycles before ending.

Instruction	Explanation
NEW	
5 I = 1	Assign I the value 1
10 REM CHANGE COLORS	Remark to tell what this part of program does
15 CALL CLEAR	Clears screen
17 REM S = SCREEN COLOR	Reminder that S sets the screen color
20 REM F = FOREGROUND COLOR	Reminder that F sets the foreground color
30 REM B = BACKGROUND COLOR	Reminder that B sets the background color
35 INPUT "SCREEN COLOR? ":S	Has user type in desired screen color number
40 INPUT "FOREGROUND COLOR? ":F	Has user type in desired foreground color number
50 INPUT "BACKGROUND COLOR? ":B	Has user type in desired background color number
60 CALL CLEAR	Clears screen
65 CALL SCREEN (S)	Assigns screen color on basis of S input
70 CALL COLOR (2,F,B)	Assigns colors on basis of F and B inputs
80 CALL HCHAR (12,3,42,28)	Row 12 column 3 is the beginning point to print 28 asterisks
90 FOR DELAY = 1 TO 2000	Holds display on screen
100 NEXT DELAY	
130 I = I + 1	Increments I by one
140 IF I < 11 THEN 15	As long as I is less than 11, the program branches to line 15 to loop
150 END	
RUN	

Try changing this program to use a FOR-NEXT loop instead of the IF-THEN loop.

```
SCREEN COLOR? 10
FOREGROUND COLOR? 6
BACKGROUND COLOR? 11
.
.
.
```

The RND function

Note
Random numbers are often used in computer games.

The RND function is a built-in function as was described for the TAB function. The RND function produces a random number between 0 and 1. It can be used with PRINT as a command in the immediate mode or as a statement in the program mode.

For example, if you simply type PRINT RND, the result might be .4211325812. The second time might produce .3332842123.

Example program using the RND function

Instruction	Explanation
NEW	
10 CALL CLEAR	Clears screen
20 FOR COUNTER = 1 TO 3	Repeat loop 3 times
30 PRINT: RND	Prints random number
40 NEXT COUNTER	Returns to line 20 if COUNTER is less than 4
50 END	Program ends
RUN	

Note
Your numbers probably will be different.

```
 .1209835232

 .0125325301

 .1235329098

** DONE **
```

The RANDOMIZE statement

Run the program again and notice that it produces the same series of random numbers. This can be useful for some applications; however, it's often necessary to have unpredictable numbers. This can be accomplished with a RANDOMIZE statement which reseeds the RND function generator so that different numbers are produced. Just add line 15 to the program above. Run the program several times and observe that it now produces a different series of random numbers each time.

Example program using the RANDOMIZE statement

Instruction	Explanation
10 CALL CLEAR	Clears the screen
15 RANDOMIZE	Tells computer to generate a new series of random numbers
20 FOR COUNTER = 1 TO 3	Repeats loop 3 times
30 PRINT: RND	Prints the random number
40 NEXT COUNTER	Returns to line 20 if COUNTER is less than 4
50 END	Program ends
RUN	

Note
Again, your numbers probably will be different.

```
 .7035323537

 .0932343212

 .1235323353

** DONE **
```

The INT function

The INT function is still another built-in function. This one truncates a positive number to its integer value; that is, the whole part of the number is kept, but the decimal fractional part is dropped without rounding. The INT function can be used in combination with the RND function to generate a random series of integers.

To generate a random series of integers, the program must do two things. First, the generated random number must be multiplied by some number to increase the value so it is greater than 1. For example, 10 * RND produces a number range of 0 through 9.9999... Second, to get rid of the fractional value (the part after the decimal point), the INTeger statement can be used.

Here are the results of various program instructions using RND and INT.

Instruction	Range of numbers generated
RND	0 through .9999...
10 * RND	0 through 9.999...
INT (10 * RND)	0 through 9 (integers only)
100 * RND	0 through 99.999...
INT (100 * RND)	0 through 99 (integers only)

Example program using the INT function

The following program produces random integer numbers from 0 through 5. Just change line 30 in the present program.

Instruction	Explanation
10 CALL CLEAR	Clears screen
15 RANDOMIZE	Tells computer to generate a new series of random numbers
20 FOR COUNTER = 1 TO 3	Repeat loop 3 times
30 PRINT: INT (6 * RND)	Prints random integer from 0 through 5
40 NEXT COUNTER	Return to line 20 if COUNTER is less than 4
50 END	Program ends
RUN	

Note
To simulate the roll of a single die where the outcome is 1 to 6, change line 30 to: PRINT:INT (6*RND) + 1

```
 3

 1

 4

** DONE **
```

Evaluation of equations

The following arithmetic operators are used for expressing mathematical equations or formulas.

Key	Operation
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponentiation

Note
When typing in a complex equation with several sets of parentheses, be sure you close every opening parenthesis. A quick check is to simply count all the opening parentheses and all the closing parentheses. The quantity of each must be the same.

The computer follows a specific order, called the operator hierarchy, when evaluating equations. The operator hierarchy must be observed when programming equations to ensure that you get the correct answer.

Here's how the computer works:

1. The computer reads the statement from left to right just as you do, but it reads the whole equation before doing any operations.
2. It computes all operations within parentheses according to the operator rules. If there are nested parentheses, the computations within the innermost parentheses are completed first, then the next innermost and so on.
3. Next, it performs exponentiation from left to right.
4. Then multiplications and divisions are completed from left to right
5. Finally, additions and subtractions are completed from left to right.

Example program for evaluating an equation

To convert temperature in degrees Fahrenheit to degrees Celsius, the following equation is used:

C = 5/9*(F-32)

The computer first performs the subtraction F − 32 because it is enclosed in parentheses. If F = 40, then F − 32 = 8. The computer next divides 5 by 9 which equals .555... Finally the computer multiplies 0.555... by 8 with a final answer of 4.444..; therefore, 4.4 degrees Celsius is about the same as 40 degrees Fahrenheit.

Instruction	Explanation
NEW	
10 REM FAHRENHEIT TO CELSIUS	Remarks
20 REM CONVERSION PROGRAM	
100 CALL CLEAR	Clears screen
110 INPUT "FAHRENHEIT TEMPERATURE ": F	Asks you to type in Fahrenheit temperature
120 PRINT: 5/9*(F-32);"DEGREES CELSIUS =";F;"DEGREES FAHRENHEIT"	Performs calculations and prints results
130 END	Program ends
RUN	

```
FAHRENHEIT TEMPERATURE 40

 4.4444444
DEGREES CELSIUS = 40
DEGREES FAHRENHEIT

** DONE **
```

Introduction

Modes ← The two modes of operation are the immediate mode and the program mode.

Paths of information flow

Commands are used in this mode. → **The immediate mode**

The BASIC interpreter ← Translates BASIC instructions into machine instructions.

Processing BASIC commands

Statements are used in this mode. → **The program mode**

Typing in BASIC statements
Executing BASIC statements

Getting started

Instructions in the immediate mode

Prints information on the screen. → **Using PRINT in a command**

The CALL CLEAR command ← Clears the screen.

Instructions in the program mode

Line numbers ← Each must be different.

How to end program statements ← By pressing ENTER.

Tells the computer to execute the program. → **The RUN command**

Stopping and restarting a program ← FCTN 4 and CONTINUE

An exercise in the program mode

The CALL CLEAR statement

Using PRINT in a statement

Adding a program line ← It's automatically put in the right order.

Prints program lines. → **The LIST command**

Correcting errors

Retyping ← The hard way.

The easy way. → **Editing**

If one character is incorrect in the line
If one or more characters have been left out
If one or more characters need to be removed

More commands and programming exercises

The NEW command ← Clears memory.

Another exercise in the program mode

Assigns values to variables. → **Naming variables** ← The two types are numeric and string.

The LET statement

Examples of assigning numeric values to variable names

Examples of assigning strings to variable names

The REM statement ← For program comments.

The INPUT statement

Use this when the program needs input from the keyboard. →

Example program using an INPUT statement

INPUT with a prompting message

Example program with an INPUT prompt

Punctuation in INPUT statements ← Supresses the question mark.

Colons in INPUT statements

Commas in INPUT statements ← Use for multiple variable inputs.

Formatting computer output

Supresses line feed for same line printing. → **Semicolons in PRINT statements**

Example program using semicolons

Fixed automatic tabbing. → **Commas in PRINT statements**

Colons in PRINT statements ← Forces separate lines.

Works like the typewriter tab. → **The TAB function**

The PRINT TAB statement

Example program using PRINT TAB

The GOTO statement ← For unconditional branching.

Example program using GOTO

The IF-THEN statement ← For conditional branching.

Relational operator symbols

Example program with relational operator in condition

Example program with computation in condition

Example program with string variables

The FOR-NEXT statement

For looping to perform the same or similar operation several times.

Example program to print out all numbers from 2 to 20

Example program to print out only the even numbers from 2 to 20

Nesting FOR-NEXT loops

Example program with nested FOR-NEXT loop

Using FOR-NEXT to create a time delay loop

For structured programming or when the same subroutine is used several times.

Example program with a time delay loop

Using a FOR-NEXT loop to accumulate a sum

Example program to accumulate a sum

The GOSUB and RETURN statements

Example program using GOSUB

Color graphics

All of these are required to produce color graphics.

The CALL SCREEN statement

Generates random numbers.

The CALL COLOR statement

The CALL HCHAR and CALL VCHAR statements

Example color program one
Example color program two

The RND function

Reseeds the random number generator to produce a different sequence of numbers.

Example program using the RND function

The RANDOMIZE statement

Example program using the RANDOMIZE statement

The INT function

Obtains the integer from a decimal number.

Example program using the INT function

Evaluation of equations

The operator heirarchy of the computer must be kept in mind when programming equations.

Example program for evaluating an equation

Summary map

5 Steps to Successful Programs

2 What kind of problems can the computer solve?

2 How do you develop a computer program?

Identify the problem

Outline the solution

Prepare a flowchart

Start and end
Input and output
Direction
Process
Decisions
Connectors

Write the code

Run the program and debug it

Example one: Make a cup of hot tea

Example two: Print names and addresses

Example three: Compute total car cost if bought on time payments

Example four: Expanding the auto cost program

16 Structured programming

Change in presentaton of flowcharts

Example five: Structured programming of the auto cost program

19 Arrays

Name and subscripts

DIM statement

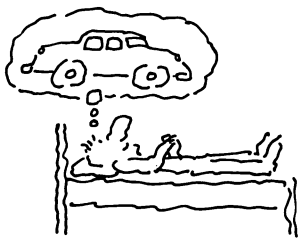
Example six: Calculate grade averages

Example seven: Procedure for diagnosing automobile failure

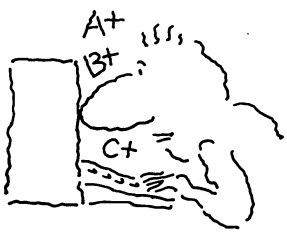
30 Summary map

Example two

Example three



Example six



Example seven

Steps to Successful Programs

What kinds of problems can the computer solve?

A computer can solve many kinds of problems, but first you must decide what you want it to do for you.

The first question to ask is: What do you want the computer to do? It may be that you require the solution to a problem that uses a simple mathematical equation over and over. Calculating the take home pay for 50 employees is a good example of a repeated calculation that is an ideal application for a computer. Another test may require the solution of a complicated equation that the computer can solve easily and quickly, but would be very difficult if done by hand. Calculating and printing an amortization (payback) schedule for a home mortgage is an example of the use of a complicated equation.

The computer is good at drilling children in math and grammar.

Drill and practice is a technique for helping people learn basic educational concepts. The technique requires repeated use of the same or similar material. The computer is well suited for this task so it may be used as a teaching aid to help children master a new subject. Drilling a third grader in beginning grammar or math is such a teaching example.

There are many other tasks that a home computer can perform if it has the proper program. You can develop a computer program to solve a problem.

How do you develop a computer program?

You can program the computer!

To develop a computer program that solves a problem, you should follow these steps:

1. Identify the problem
2. Outline the solution
3. Prepare a flowchart
4. Write the code
5. Run the program and debug it

This may sound very technical and difficult, but before you throw up your hands and say, "I can't do it.", try it. It's really simple and easy if you just follow the steps.

Step 1
Identify the problem

The first step consists of answering the question: What do you want the computer to do?

Step 2
Outline the solution

You must determine the inputs the computer needs, what processing it is to do, and what outputs it is to provide.

Outlining the solution to the problem includes determining the program inputs, processes and outputs as well as determining the sequence of steps the program is to follow. The program inputs are those items or pieces of information that must be put into the computer so it can do the requested task. The program outputs are the desired answers, actions or other results. The processes that take place between the inputs and the outputs are the work (such as calculating or sorting) done by the computer.

The outline for the solution to the problem consists of ordinary English sentences that describe the inputs, processes, and outputs. A very general flowchart (described below) also can help you organize your thoughts and determine the sequence of steps.

Variable names must be assigned.

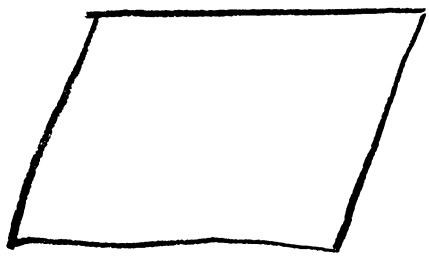
Once this is done, the second part consists of assigning names for the variables needed in the program instructions. Some variables are used for all three functions (input, process, and output) and others are used only for one function. Variables are quantities in the program that can assume different values as the program executes.

Step 3
Prepare a flowchart

A flowchart is like a roadmap. It shows logical direction. It has forks (branch points) that require a decision. It uses special symbols.

Flowcharting is the act of "writing" the program in a symbolic form. Flowcharting helps to develop the step-by-step actions of the program as well as to determine decision points and branching. The problem solution flow follows a logical path; therefore, errors in logic often are found in this step of program development because they are more apparent in the flowchart than in the outline. The flowchart symbols and their meanings are shown below.

Start and end
The oval symbol is used at the beginning and end of the program. The beginning symbol shows the starting point of the flowchart and is labeled START. At the end of the program, the oval is labeled END.



Input and output
The parallelogram is used to indicate the INPUT and OUTPUT in a flowchart.

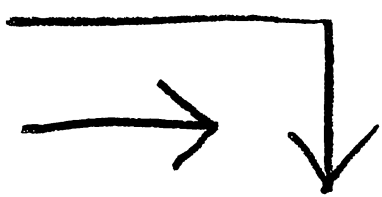
Examples are:

General Applications: what, write, say, etc.

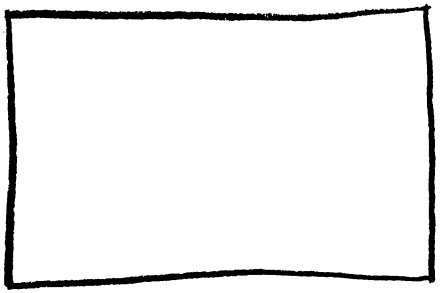
What is your name?
Write today's date
Say "good work"
Write the answer to the problem

BASIC Language Applications: INPUT and PRINT

INPUT your name
INPUT today's date
PRINT "good work"
PRINT the answer



Direction
Flow lines with arrowheads are used in the flowchart to indicate the direction of the logic flow.



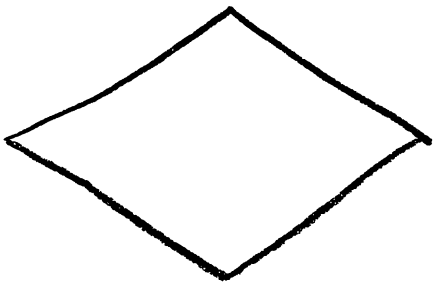
Process
The process symbol has the shape of a rectangle.

General Applications: Find, add, subtract, look up, etc.

Find the square of a number
Add the result
Look up the dictionary definition

BASIC Language Applications: LET

LET X = 5
LET AMOUNT = AMOUNT + MONTHLY
LET LOOP = LOOP + 1



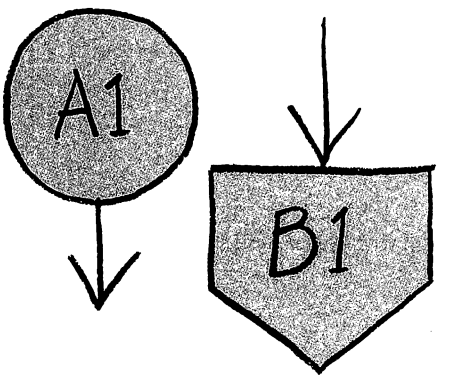
Decisions
The diamond shape is the decision symbol. This symbol is used when conditions are tested for a conditional branch in the program. Only two branches are possible; they are usually labeled YES and NO, but may be labeled TRUE and FALSE.

General Applications: Is, what if, will, etc.

Is it raining?
(Yes) Take umbrella
(No) Leave umbrella

BASIC Language Applications: IF-THEN.

IF A = 1
(Yes) THEN PRINT A
(No) ELSE PRINT B



Connectors
The circle is used as a flow line connector to connect the flow on the same page when drawing a line between the two points would create clutter and confusion. The same number, letter, or combination is placed in the mating circles. The broad "arrow" symbol is used to connect the flow from one page to another. The same number, letter, or combination is placed in the mating connectors.

Step 4
Write the code

Writing the code is the fun part of programming. It is easier if the first three steps are done properly.

It is not uncommon for the first three steps of program development to overlap one another. It is important, however, that all three steps be completed before beginning to write the code. This will save time in the long run. The amount of work that needs to be done in the first three steps depends on the complexity of the program and the experience of the programmer. Even programmers with a lot of experience need to do some work before writing the code, especially in longer and/or more complicated programs. Once the first three steps have been completed, the program instructions can be written in the code of the BASIC computer language.

Step 5
Run the program and debug it

Running the program is the real test. Watch out for the bugs.

The final step in developing a computer program involves running and testing the program to be sure it is correct. It is a rare occasion when a program runs successfully the first time because a few errors usually get in. Finding errors is called debugging the program. Errors in programs are called bugs because they can be hard to find. Remember that to run the program:

Type RUN
Press **ENTER**

Example one:
Make a cup of hot tea

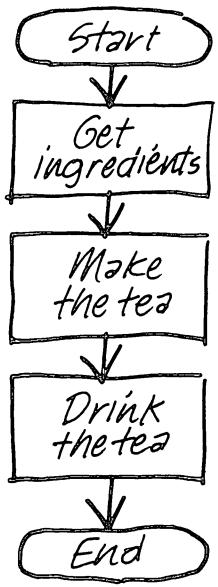
Step 1
Identify the problem
Make a cup of tea

Step 2
Outline the solution

Inputs
1. Get a teapot.
2. Get a teabag.
3. Get a cup.
4. Get water.

Process
1. Pour water into teapot.
2. Put teapot on stove.
3. Turn stove on high heat.
4. Check water temperature.
5. When water boils, pour water into cup.
6. Turn off stove.
7. Put teabag in cup with water for 2 minutes.
8. Remove teabag.

Outputs
1. Drink tea.



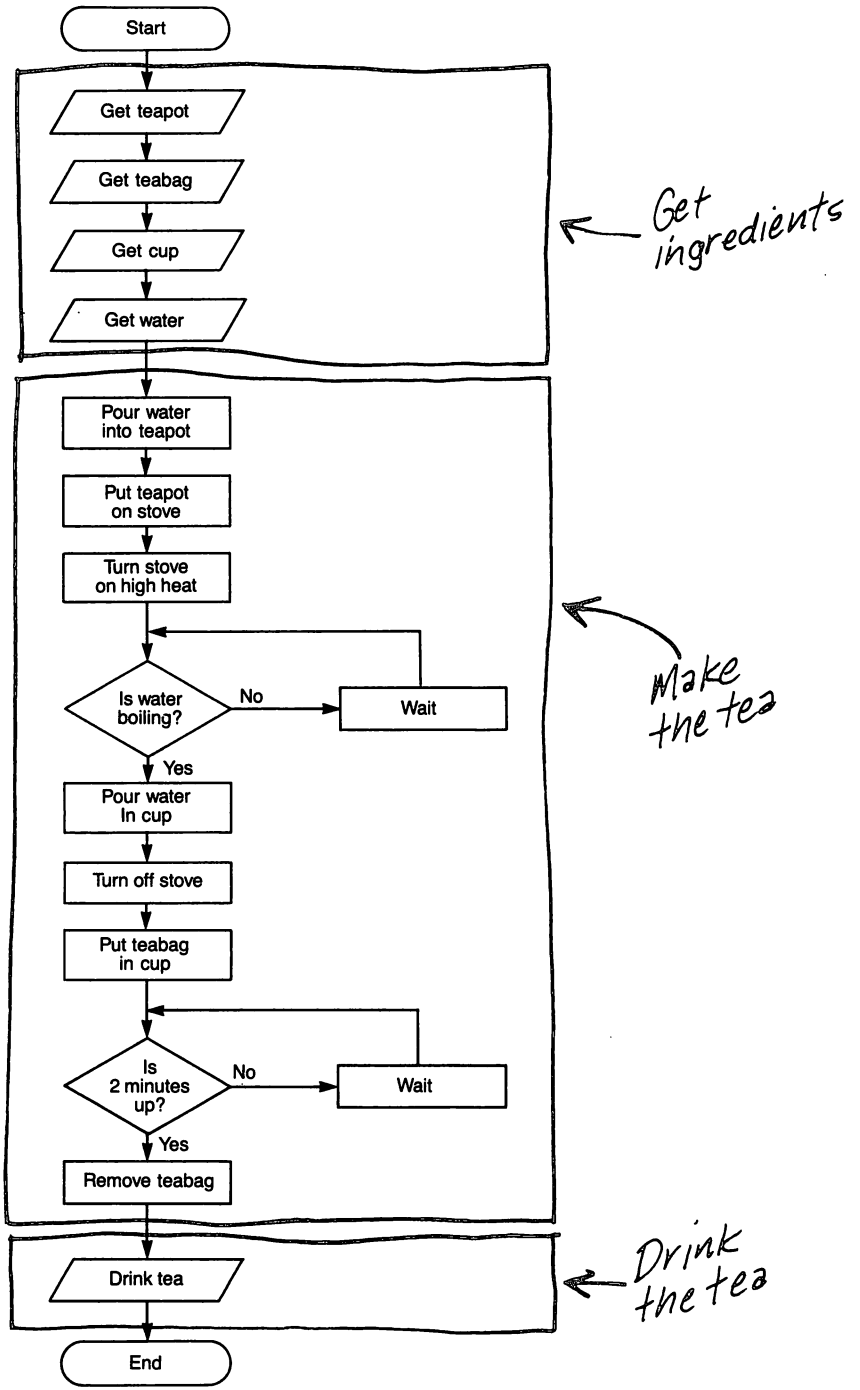
Step 3
Prepare a flowchart

Assign variable names

Inputs
CUP
TEABAG
TEAPOT
WATER

Process
CUP
WATER
TEAPOT
STOVE
HIGH HEAT
BOILING
TEABAG
TIME

Outputs
YOU



Step 4
Write the code

It would be very difficult to write the code for a home computer to actually make a cup of tea because of the extensive amount of control equipment that would be required.

However, in order to illustrate what some typical BASIC statements might look like to write such a program, the following program is used as an example:

Instruction	Explanation
100 INPUT CUP	Get a cup
110 INPUT TEABAG	Get 1 teabag
120 INPUT TEAPOT	Get a teapot
130 INPUT WATER	Get a cup of water
140 TEAPOT = TEAPOT + WATER	Pour water into teapot
150 TEAPOT = TEAPOT + STOVE	Put teapot on stove
160 STOVE = HIGH HEAT	Turn stove on high heat
170 IF WATER = BOILING THEN 190	Check teapot for boiling water
180 GO TO 170	If not boiling, wait
190 CUP = CUP + WATER	Pour boiling water into cup
200 STOVE = OFF	Turn off stove
210 CUP = CUP + TEABAG	Put teabag in cup with water
220 IF TIME = 2 THEN 240	Leave teabag in cup for 2 min.
230 GO TO 220	If not 2 min., wait
240 CUP = CUP − TEABAG	Remove teabag
250 YOU = YOU + CUP	Drink tea
260 END	

To save time, you might get the cup and teabag while you are waiting for the water to boil.

If you should discover that the teabag must be left in the water 3 minutes instead of 2, line 220 would change accordingly.

This program has a bug because you would burn your tongue if you drank the tea immediately. To correct this problem, you could add a delay loop between lines 240 and 250 to allow time for cooling.

Step 5
Run the program and debug it

This first example is only to show you what's involved in going through the five steps. All following examples are valid BASIC programs that will run on the computer.

Example two:
Print names and addresses

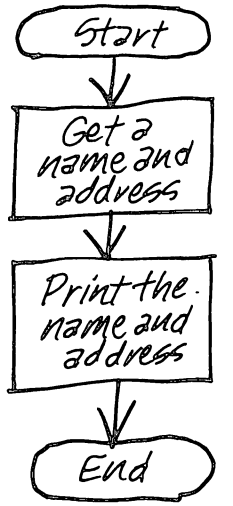
Step 1
Identify the problem
Input and print the name and address of a friend.

Step 2
Outline the solution

Inputs
1. Get name.
2. Get street address.
3. Get city.
4. Get state.

Process
None

Outputs
1. Print name.
2. Print street address.
3. Print city.
4. Print state.



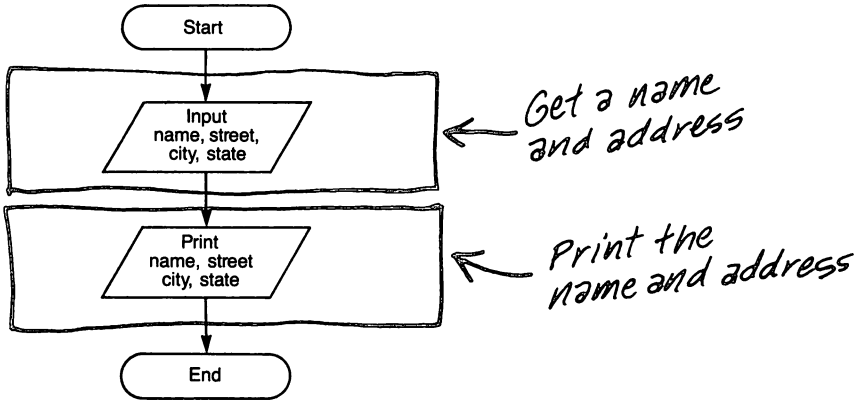
Step 3
Prepare a flowchart

Assign variable names

Inputs
N$ = Name
S$ = Street
C$ = City
T$ = State

Process
None

Outputs
N$ = Name
S$ = Street
C$ = City
T$ = State



Step 4
Write the code

With TI BASIC™, the program will end and the screen will change color as soon as printing is finished.

To prevent the program from ending, you could add: 95 INPUT E$. This will hold the display until you press ENTER.

A way to delay the program ending is to use a FOR-NEXT delay loop between lines 90 and 100.

Instruction	Explanation
10 CALL CLEAR	Clears the screen
20 INPUT "NAME ": N$	Prints NAME and waits for name to be entered
30 INPUT "STREET ": S$	Prints STREET and waits for street address to be entered
40 INPUT "CITY ": C$	Prints CITY and waits for the city to be entered
50 INPUT "STATE ": T$	Prints STATE and waits for state to be entered
60 CALL CLEAR	Clears the screen
70 PRINT N$	Prints name of person
80 PRINT S$	Prints street address of person
90 PRINT C$;", ";T$	Prints city and state of person
100 END	Ends the program

Step 5
Run the program and debug it

Remember that the words in dark type are what you type.

```
NAME BASIC PROGRAMMER
STREET GOTO STREET
CITY BASIC
STATE TEXAS
```

The computer prints the name and address exactly as you typed it. Try experimenting with others including your own.

```
BASIC PROGRAMMER
GOTO STREET
BASIC, TEXAS

** DONE **
```

Example three:
Compute total car cost if bought on time payments.

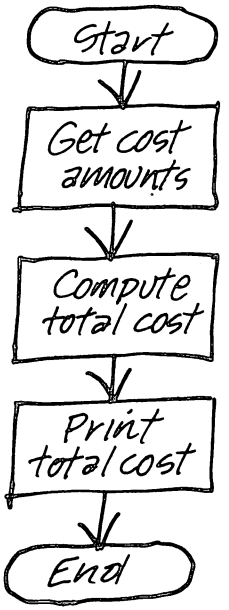
Step 1
Identify the problem
Compute the total cost of a car when purchased on a time payment basis.

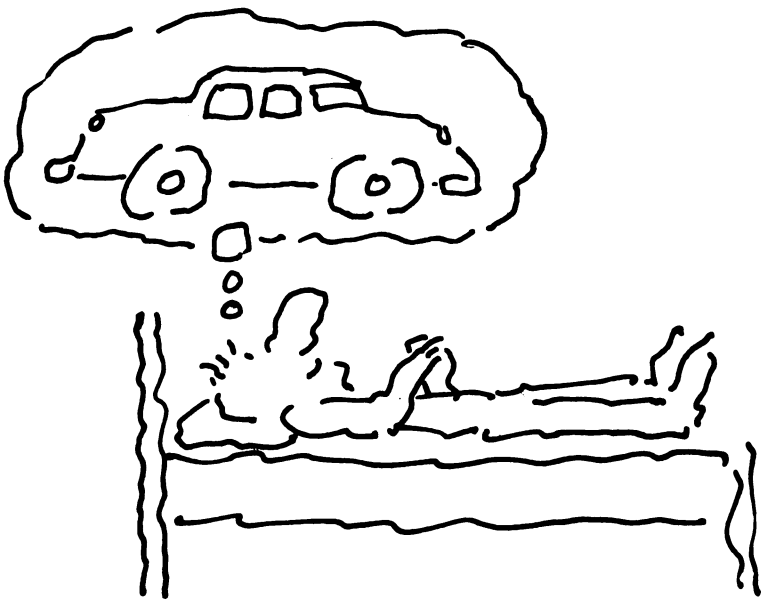
Step 2
Outline the solution

Inputs
1. Input down payment.
2. Input loan amount.
3. Input interest rate.
4. Input loan period.

Process
1. Compute interest.
2. Compute total cost.

Outputs
1. Print total cost of automobile including interest.





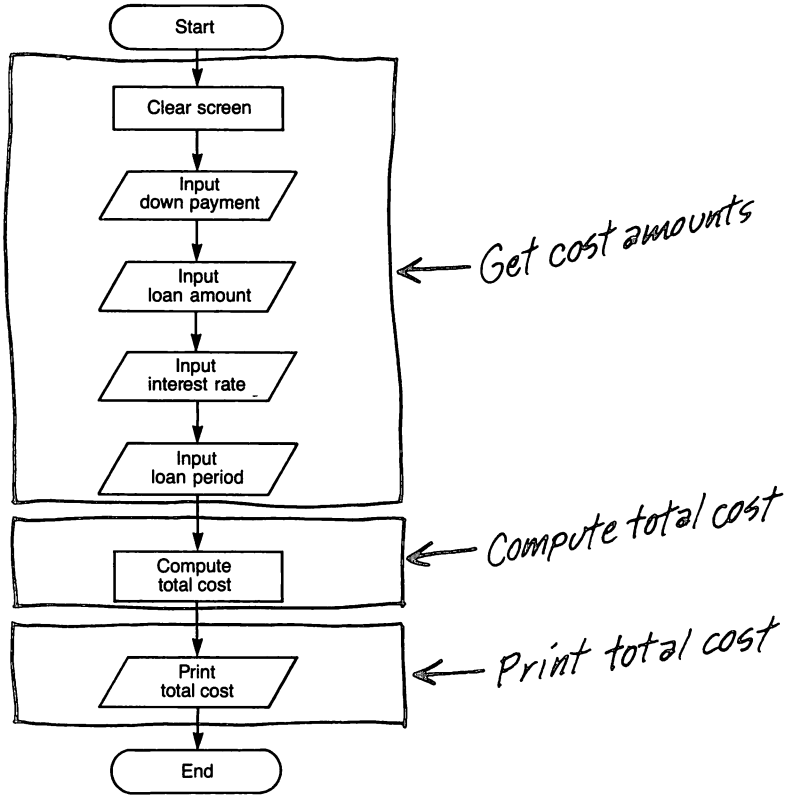
Step 3
Prepare a flowchart

Assign variable names

Inputs
DPAYMENT = Down payment
LOAN = Loan amount
RATE = Interest rate
TIME = Loan period

Process
TOTCOST = Total cost

Outputs
TOTCOST = Total cost



Step 4
Write the code

Instruction	Explanation
5 REM AUTO COST PROGRAM	
6 REM VERSION 1	
10 CALL CLEAR	Clears screen
20 INPUT "DOWN PAYMENT ": DPAYMENT	The computer waits for the down payment amount to be entered
30 INPUT "AMOUNT OF LOAN ": LOAN	Computer waits for the loan amount to be entered
40 INPUT "RATE IN DECIMAL FORM ": RATE	Computer waits for interest rate to be entered
50 INPUT "LOAN LENGTH IN YEARS ": TIME	Computer waits for loan period in years to be entered
60 TOTCOST = DPAYMENT + LOAN + (LOAN * RATE * TIME)	Compute the total cost of the car
70 PRINT "TOTAL COST OF AUTO"; TOTCOST	Computer prints out the cost
90 END	

Step 5
Run the program and debug it

```
DOWN PAYMENT 5000
AMOUNT OF LOAN 1000
RATE IN DECIMAL FORM .10
LOAN LENGTH IN YEARS 5
TOTAL COST OF AUTO 6500

** DONE **
```

Example four:
Expanding the auto cost program

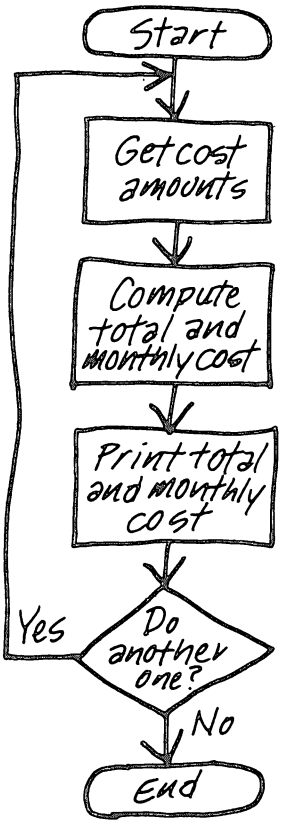
Step 1
Identify the problem
To make the auto cost program more useful, let's make the program give the buyer the amount of the monthly payment as well as the total cost of the automobile. Also, we'll input the purchase price as cost, but not input the down payment. The program will compute the required down payment.

Step 2
Outline the solution

Inputs
1. Input cost (purchase price).
2. Input loan amount.
3. Input interest rate.
4. Input loan period.

Process
1. Compute down payment.
2. Compute total cost.
3. Compute monthly payment.

Outputs
1. Print cost and down payment
2. Print loan amount and loan period
3. Print total cost.
4. Print monthly payment.



Step 3
Prepare a flowchart

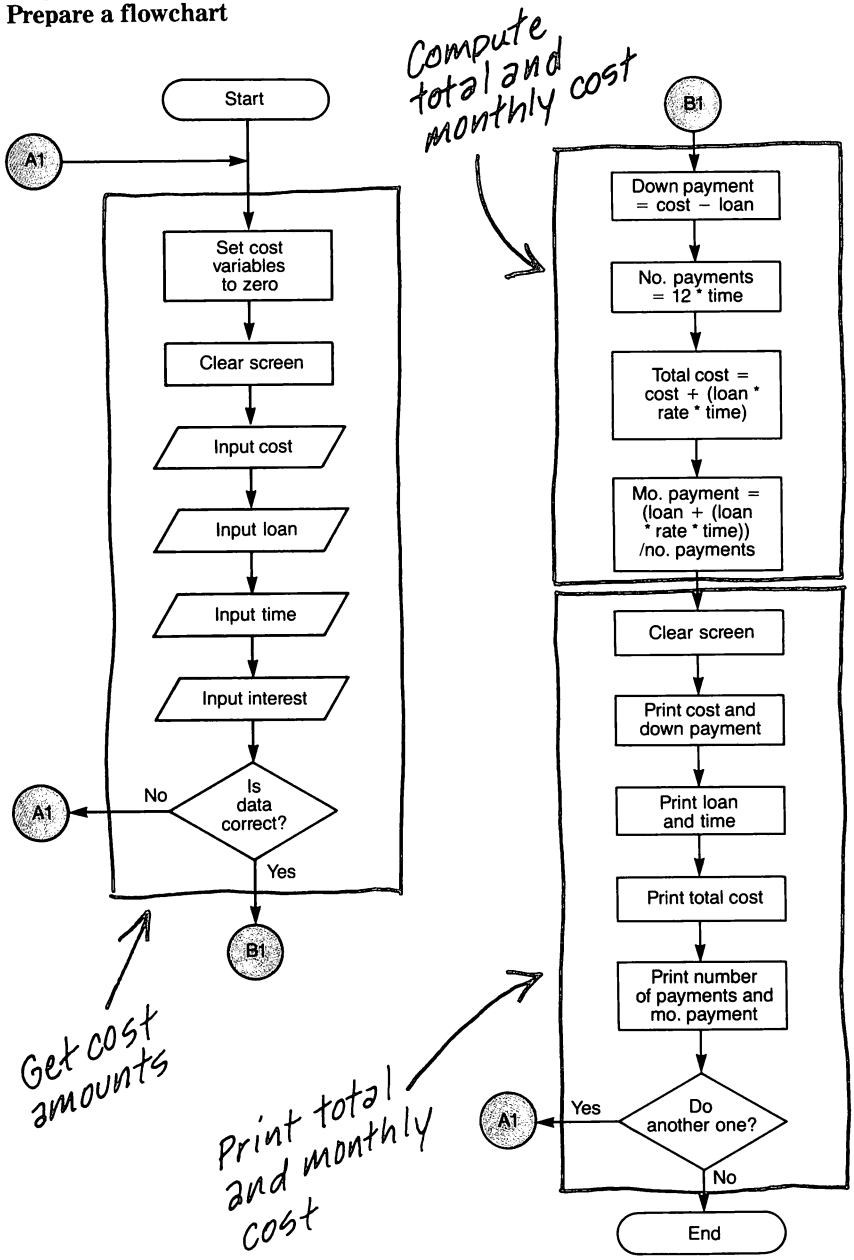
Assign variable names

Inputs
CST = Cost
RTE = Rate
LOAN = Loan amount
TIME = Loan period
A$ = User's response

Process
DPAYMENT = Down payment
TCST = Total cost
MOPAYMENT = Monthly payment
NPAYMENT = No. of payments

Outputs
CST = Cost
DPAYMENT = Down payment
TCST = Total cost
MOPAYMENT = Amount of monthly payment
LOAN = Loan amount
TIME = Loan period
NPAYMENT = No. of payments

Note
The use of connectors eliminates several lines that would clutter the flowchart.



Step 4
Write the code

Instruction	Explanation
10 REM AUTO COST PROGRAM	Program identification
20 REM VERSION 2	
30 TCST = 0 40 RTE = 0 50 LOAN = 0 60 CST = 0 70 DPAYMENT = 0 80 TIME = 0 90 NPAYMENT = 0 100 MOPAYMENT = 0	Initialize variables to zero
110 CALL CLEAR	Clear screen
120 PRINT "COST"; 130 INPUT CST 140 PRINT "LOAN AMOUNT"; 150 INPUT LOAN 160 PRINT "TIME IN YEARS"; 170 INPUT TIME 180 PRINT "INT. RATE IN %"; 190 INPUT RTE	Print prompts and get input values
200 PRINT "IS DATA CORRECT"; 210 INPUT A$ 220 IF A$ = "NO" THEN 30	If user types NO and presses ENTER, program will start over
230 DPAYMENT = CST − LOAN	Compute down payment = cost of car − amount of loan
240 NPAYMENT = 12 * TIME	The number of months = 12 times the number of years
250 RTE = RTE/100	Converts percentage to a decimal fraction
260 TCST = CST + (LOAN*RTE*TIME)	Compute the total cost by adding interest amount to purchase price
270 MOPAYMENT = (LOAN + (LOAN * RTE * TIME))/ NPAYMENT	Computes the monthly payments
280 MOPAYMENT = INT (100 * MOPAYMENT + .5)/100	Round the monthly payment to the nearest cent by using the INTeger function. Adding .5 ensures correct rounding
290 CALL CLEAR	Clears screen
300 PRINT "COST", "DOWN PAYMENT" 310 PRINT CST, DPAYMENT	Prints headings Print variables for cost and down payment
320 PRINT "LOAN", "TIME" 330 PRINT LOAN, TIME; "YRS"	Print second set of headings Print the variables for loan amount and time. Also prints YRS after the time
340 PRINT "TOTAL COST"; TCST 350 PRINT: "NO. PAYMENTS", "MO. PAYMENTS"	Prints header and total cost amount Prints the headers for the next group

The logic is similar to the original problem; however, the program is now designed so that more than one set of computations can be made without typing RUN. This makes it necessary to initialize all values to zero at the beginning of the program.

Different INPUT statements are used because different information is needed.

After all data has been entered, you are asked if the entered data is correct. If not, the variables are reset to zero, and the input questions are asked again.

The number of monthly payments is calculated by multiplying 12 times the number of years input for the loan period.

Dollar amounts are rounded to two decimal places as we normally see them.

Instruction	Explanation
360 PRINT NPAYMENT, MOPAYMENT	Prints values for variables; No. of payments and amount of monthly payment
370 PRINT	Prints a blank line
380 INPUT "DO YOU WANT TO DO ANOTHER? ":A$	Also serves to hold the values on the screen
390 IF A$ = "YES" THEN 30	If user types YES and presses ENTER, program starts over at line 30
400 END	If any response other than "YES" is encountered in line 390, the program terminates

Here you are asked if another set of calculations is wanted for comparison. For example, you might compare the data for the same car with different loan amounts, interest rates, and/or a different number of time payments. By using this program, you can determine the best method of financing the car.

Step 5
Run the program and debug it

```
COST? 5000
LOAN AMOUNT? 4000
TIME IN YEARS? 5
INT. RATE IN %? 10
IS DATA CORRECT? YES
```

```
COST                    DOWN PAYMENT
 5000                    1000
LOAN                    TIME
 4000                    5 YRS
TOTAL COST 7000

NO. PAYMENTS            MO. PAYMENTS
 60                      100

DO YOU WANT TO DO ANOTHER? NO

** DONE **
```

Structured programming

A subroutine is a subprogram that accomplishes a specific task that fits into a main program.

Another way to write the auto cost program is to use the technique called structured programming. Structured programming is based on the principle that only a few step-by-step structures should be used to solve a problem and these few structures should be organized in stair-step relationships. A main program, sometimes called the driver routine, serves as a "backbone" for the structure. The main program doesn't do much except call the subroutines which actually do the work.

GOSUB and RETURN are the statements that are used with structured programming.

To call a subroutine, the programmer uses the instruction pair GOSUB and RETURN, which was introduced in the previous chapter. The GOSUB statement allows branching to a particular subroutine from anywhere in the program to perform the subroutine and then RETURN to the main program. The RETURN statement always must be used at the end of the subroutine. The same subroutine may be called by the GOSUB again and again which can save a lot of writing effort and computer memory.

A subroutine can be called from within another subroutine.

A GOSUB may be used within a subroutine called by a GOSUB to call another subroutine, thus, you can see how the stair-step relation can be developed. This relation can continue for several levels of steps (the number depends on the particular computer), but the programmer must be very careful in developing the structure so the computer doesn't get confused.

Change in presentation of flowcharts

Flowcharts always show logic flow which is not necessarily in the order of inputs, process and outputs.

Until now, the example programs have been neatly separated into inputs, process, and outputs in a straight-line flow. With the introduction of structured programming and as programs become more complex, this straight-line flow does not always exist. The main routine in a structured program does not fit into any of these categories since it does not do any of these functions.

Even if the structured programming technique is not used, all of the inputs to a program aren't necessarily input at the very beginning. Some may be input after some processing has occurred, or maybe even after some output has occurred. Therefore, from this point on in the book, the flowcharts will show the actual flow of the program, but the outline and variables list will remain grouped by inputs, process and outputs regardless of their order in the program.

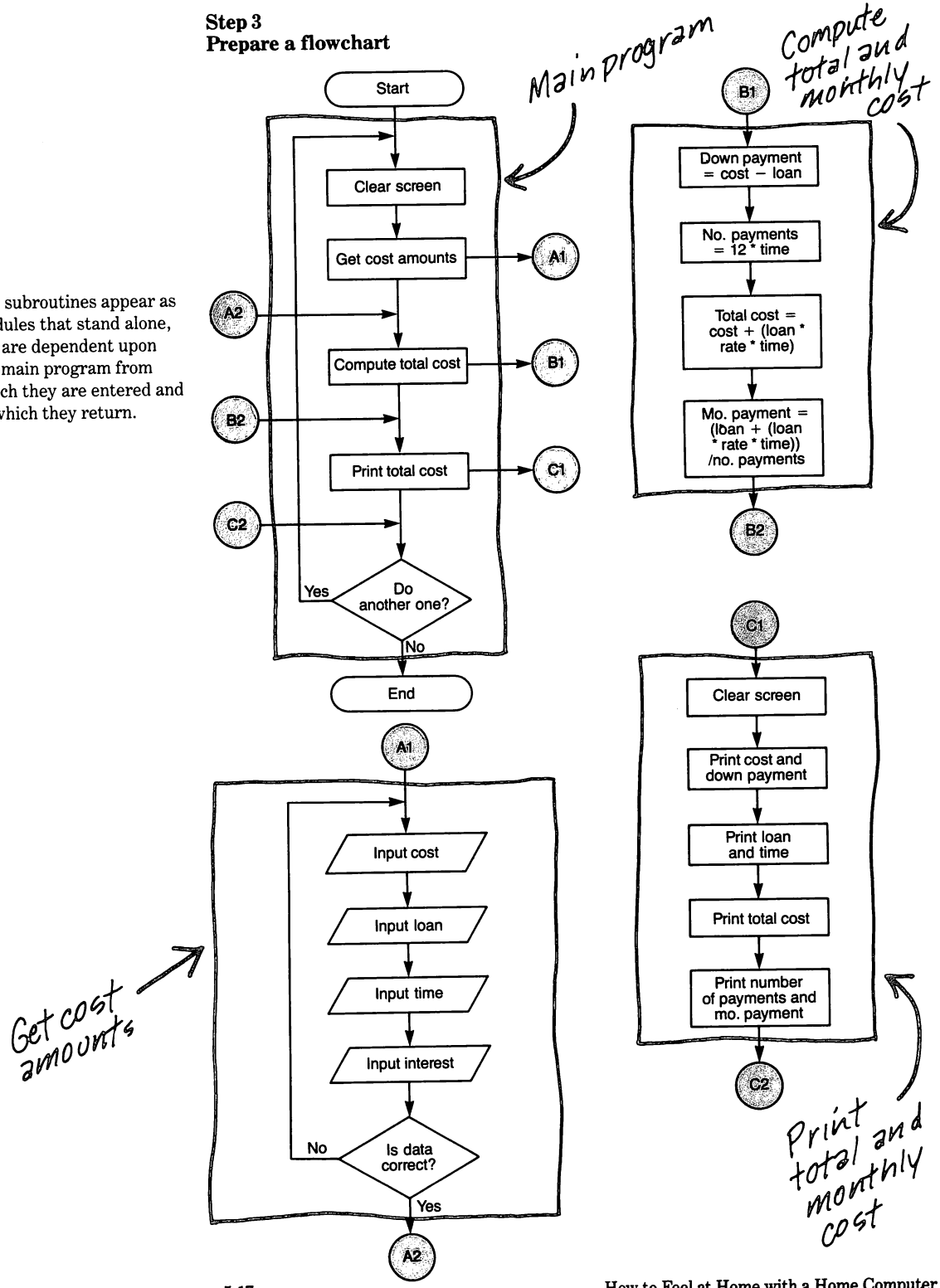
Example five:
Structured programming of the auto cost program

Step 1
Identify the problem
Same as for example 4 except use structured programming.

Step 2
Outline the solution
Same as for example 4

Step 3
Prepare a flowchart

The subroutines appear as modules that stand alone, but are dependent upon the main program from which they are entered and to which they return.



Step 4
Write the code

The advantages of the structured program technique are more apparent for longer complicated programs, but this example illustrates the principle.

Note
Remember, experimentation is one of the best teachers for programming, but don't be afraid to ask help from others. Sharing ideas helps you and others learn programming techniques more quickly.

Instruction	Explanation
5 REM AUTO COST PROGRAM	Identifies the program
6 REM VERSION 3	
10 CALL CLEAR	Clears the screen
20 GOSUB 200	Goes to subroutine starting at line 200
30 GOSUB 300	Goes to subroutine starting at line 300
40 GOSUB 400	Goes to subroutine starting at line 400
50 PRINT	Prints one line space
60 INPUT "DO YOU WANT TO DO ANOTHER? ":A$	Waits for user to respond to question
70 IF A$ = "YES" THEN 10	Returns to beginning of program if user wants to perform another calculation
80 END	Ends program
200 REM INPUT VALUES	Name of subroutine
210 INPUT "COST? ": CST	Waits for cost to be input
220 INPUT "LOAN AMOUNT? ": LOAN	Waits for loan amount to be input
230 INPUT "TIME IN YEARS? ": TIME	Waits for length of loan to be input
240 INPUT "INT. RATE IN %? ": RTE	Waits for interest rate to be input
250 INPUT "IS DATA CORRECT? ":A$	Waits for user to check that all inputs are correct
260 IF A$ = "NO" THEN 210	If inputs not correct, program returns to beginning
270 RETURN	Ends subroutine and returns to main program for next instruction
300 REM COMPUTE TOTAL COST	Name of subroutine
310 DPAYMENT = CST – LOAN	Computes amount of down payment
320 NPAYMENT = 12*TIME	Computes number of payments
330 RTE = RTE/100	Converts interest rate to decimal
340 TCST = CST + (LOAN*RTE*TIME)	Computes total cost of car
350 MOPAYMENT = (LOAN + (LOAN*RTE*TIME))/NPAYMENT	Computes amount of monthly payment
360 MOPAYMENT = INT(100*MOPAYMENT + .5)/100	Rounds monthly payment amount to nearest whole dollar
370 RETURN	Ends subroutine and returns to main program for next instruction
400 REM PRINT TOTAL COST	Name of subroutine
410 CALL CLEAR	Clears the screen
420 PRINT "COST", "DOWN PAYMENT"	Prints variable names on screen
430 PRINT CST, DPAYMENT	Prints variables
440 PRINT "LOAN", "TIME"	Prints variable names on screen
450 PRINT LOAN, TIME;"YRS"	Prints variables and word YRS on screen
460 PRINT "TOTAL COST";TCST	Prints variable name and variable on screen
470 PRINT:"NO. PAYMENTS", "MO. PAYMENTS"	Prints variable names on screen
480 PRINT NPAYMENT, MOPAYMENT	Prints variables on screen
490 RETURN	Ends subroutine and returns to main program for next instruction

Step 5
Run the program and debug it

```
COST? 5000
LOAN AMOUNT? 4000
TIME IN YEARS? 5
INT. RATE IN %? 10
IS DATA CORRECT? YES
```

```
COST                DOWN PAYMENT
 5000                1000
LOAN                TIME
 4000                5 YRS
TOTAL COST 7000

NO. PAYMENTS        MO. PAYMENTS
 60                  100

DO YOU WANT TO DO ANOTHER? NO

** DONE **
```

Arrays

Arrays store grouped information.

Sometimes it is desirable to have a collection of information pertaining to the same topic identified and stored as one group. For computer programs, this can be done with an array.

Arrays can have one, two or three dimensions.

A one-dimension array is like a list of items. The items may be either numeric or alphanumeric in nature. Examples of lists are: the supermarket shopping list; the list of things you need to do; the list of a person's grades for one subject in a semester; and the months of the year.

Arrays also can be two-dimensional with rows and columns like a mathematics multiplication table; a freight cost chart for packages of different weights and different zones; and a teacher's gradebook showing all students and grades by subject.

Computer arrays also can have three dimensions like the height, weight and sex of people. Three-dimensional arrays are used by experienced programmers, but we won't cover them because their use is beyond the scope of this book.

Name and subscripts

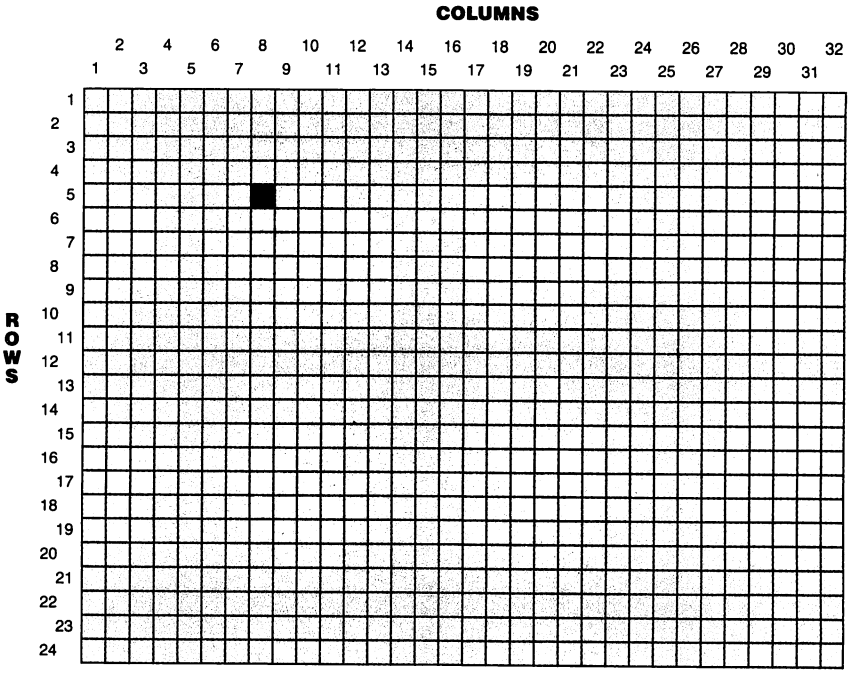
Arrays are named like other variables.

Arrays have an array name. With TI BASIC, the array can be named by the same rules as other variables. The elements stored in an array are referred to by subscripts which are enclosed in parentheses. One-dimension arrays require one subscript to identify an array element. The subscript may be a specific integer; for example A (5) refers to the fifth element of the array named A. The subscript may be a variable; for example AX(T) refers to the fifth element of AX if T equals 5, while AX(T) refers to the twentieth element of AX if T equals 20.

Subscripts locate a particular element in an array.

Two-dimension arrays are thought of as rows and columns, as shown on page 5-20, and require two subscripts. For A (R,C), R is the row subscript which is always first. The column subscript C follows the row subscript and is separated from it by a comma. The rows and columns each have their own subscript values. A particular set of subscripts refer to the element where the horizontal (row) and vertical (column)

paths meet. For example, the shaded element in the illustration is identified by row 5, column 8.



Two dimensional array structure

In summary, arrays can be referenced as follows:

A(5) — A specific subscript for a one-dimension array A which refers to element 5 of array A.

A(T) — A variable subscript for a one-dimension array A which refers to element T of array A.

A(2,3) — Specific subscripts for two-dimension array A, which refer to the element located in row 2 and column 3.

A(R,C) — Variable subscripts for a two-dimension array A which refer to the element located in row R and column C.

DIM statement

The DIM statement reserves memory for an array.

When an array is used in a program, space must be allocated (reserved) in the computer memory for the size of the array. This is done with a DIM (dimension) statement and it must appear in the program before the array is referenced. The size of the array is limited only by the available computer memory. If that memory is exceeded, the program will not run and you will get a MEMORY FULL message.

TI BASIC automatically dimensions a 10-element, one-dimension array if an array variable is used in the program and it has not been dimensioned. However, if the array needs to be dimensioned for less than ten elements, the automatic dimension wastes memory space. Therefore, it is good programming practice to dimension all arrays at the beginning of the program regardless of their size. The format is:

10 DIM A(6) — One-dimension array with six elements

20 DIM B(5,3) — Two-dimension array with five rows and three columns

Example six:
Calculate grade averages

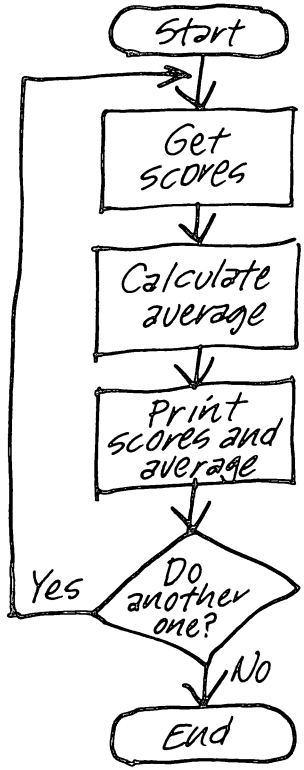
Step 1
Identify the problem
Calculate the average of five scores provided as input. Print the five scores and the average. Use structured programming techniques.

Step 2
Outline the solution

Inputs
1. Input each score.
2. Answer "Do another?"

Process
1. Dimension the score array.
2. Set the variables to zero.
3. Calculate the average.

Output
1. Print the average.
2. Ask "Do another?"



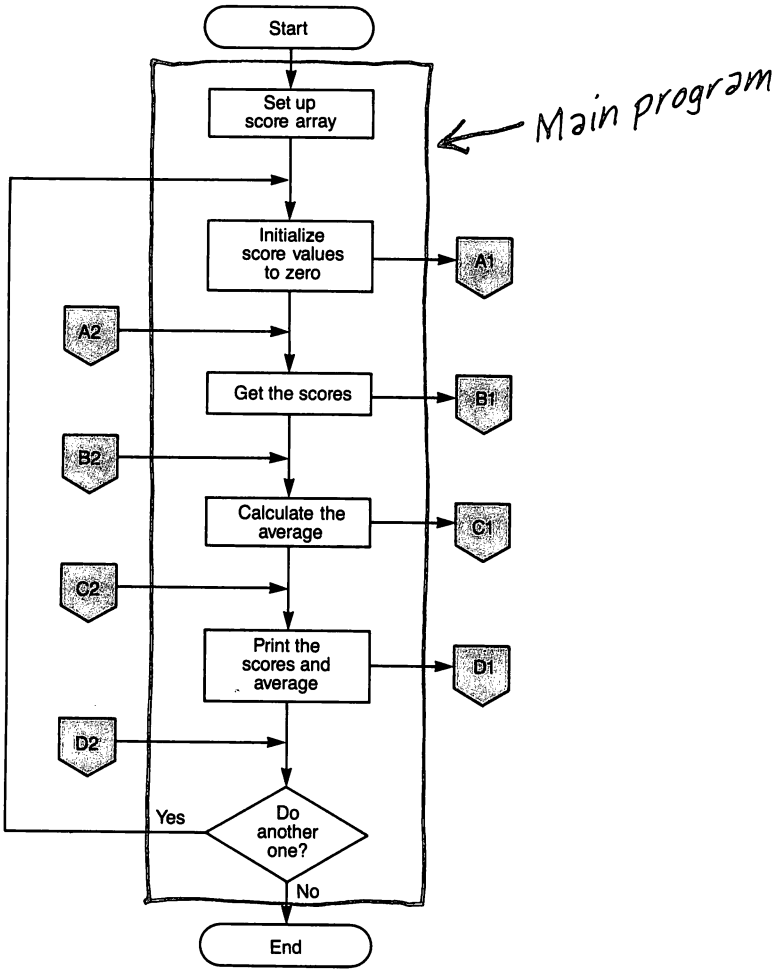
Step 3
Prepare a flowchart

Assign variable names

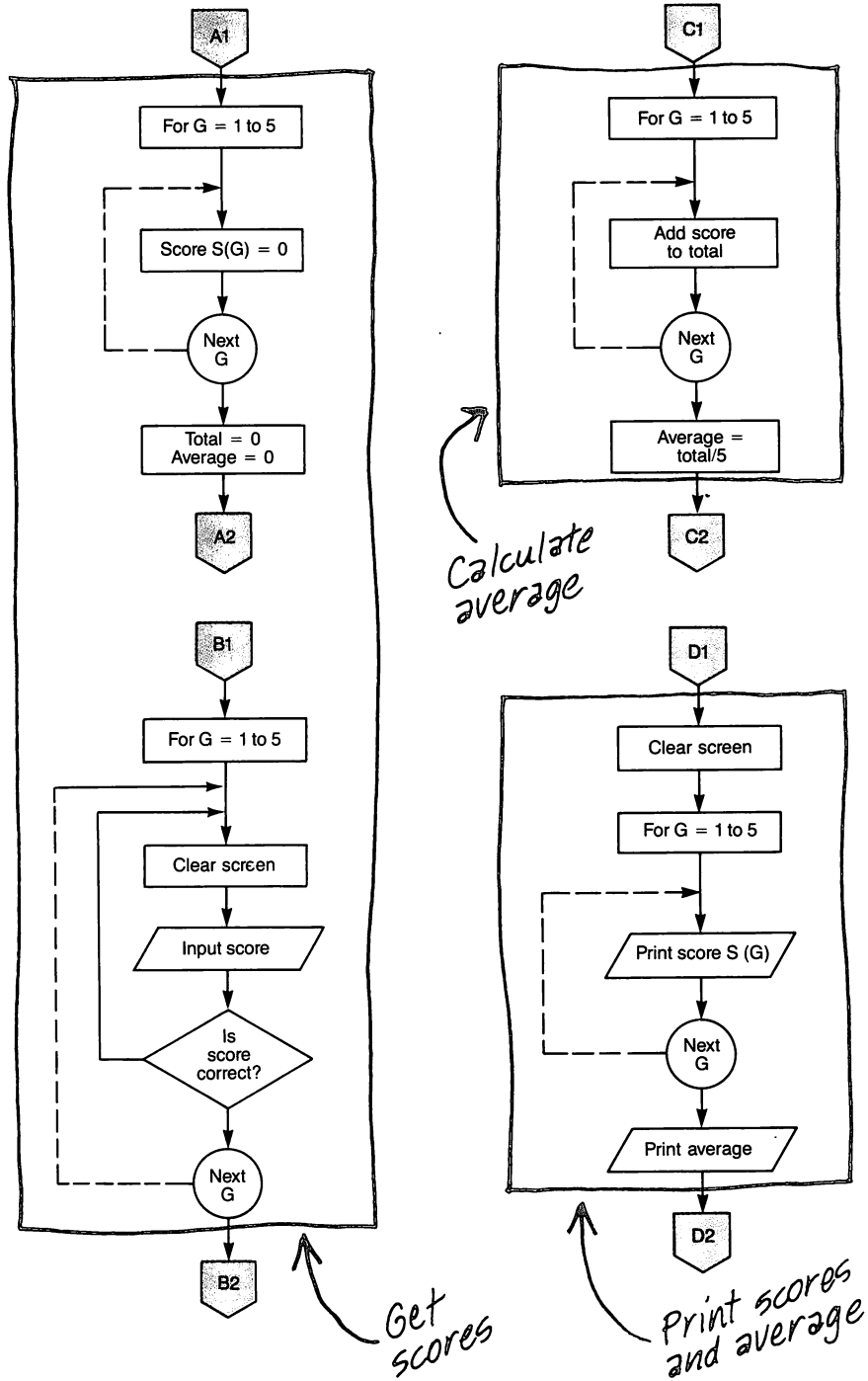
Inputs
S(G) = Scores
A$ = User's response

Process
TOTAL = Sum of scores
AVE = Average score
G = Grade counter

Outputs
S(G) = Scores
AVE = Average



Flowchart continued on next page



Step 4
Write the code

Instruction	Explanation
10 REM GRADE AVE PROGRAM	
15 REM DRIVER ROUTINE	
20 DIM S(5)	Dimension 5 elements for scores
30 GOSUB 200	Initialize all values to zero (subroutine)
40 GOSUB 300	Input the data (subroutine)
50 GOSUB 400	Compute the average (subroutine)
60 GOSUB 500	Print the scores and average (subroutine)
70 PRINT: "CALCULATE ANOTHER AVERAGE";	Determines if more averages are to be computed
75 INPUT A$	Requests Y or N answer
80 IF A$ = "Y" THEN 30	If yes, go to line 30
90 END	Terminate program
200 REM SET VARIABLES TO 0	Subroutine comment
210 FOR G = 1 TO 5	Beginning of FOR-NEXT loop to initialize score array
220 S(G) = 0	Sets score in S(G) to zero
230 NEXT G	End FOR-NEXT loop
240 AVE = 0	Set value to zero
250 TOTAL = 0	Set value to zero
260 RETURN	Exit subroutine, return to line 40
300 REM INPUT THE DATA	Subroutine heading
310 FOR G = 1 TO 5	Begin FOR-NEXT loop for entering scores
320 CALL CLEAR	Clears screen
325 PRINT TAB(8); "GRADE";G	Tab sets literal eight positions in from left margin. Then the particular element to be entered is listed.
330 PRINT: TAB (8); "SCORE";	Tab sets literal eight positions in from left margin
335 INPUT S(G)	Requests the score to be entered for that particular element.
340 PRINT "IS DATA CORRECT";	Requests confirmation of data accuracy
345 INPUT A$	
350 IF A$ = "N" THEN 320	If data is not correct, reinput
355 NEXT G	End FOR-NEXT loop
360 RETURN	Exit subroutine, return to line 50
400 REM CALCULATE AVERAGE	Header comment
410 FOR G = 1 TO 5	Begin FOR-NEXT loop for calculation of total score
415 TOTAL = TOTAL + S(G)	Add the element to the total score
420 NEXT G	End loop
430 AVE = TOTAL/5	Compute average of scores
450 RETURN	Exit subroutine, return to line 60
500 REM PRINTOUT ROUTINE	Header comment
510 CALL CLEAR	Clears the screen
520 PRINT "SCORES";	Print heading on screen
530 FOR G = 1 TO 5	Begin FOR-NEXT loop for printing output
540 PRINT S(G);	Print score
545 NEXT G	End FOR-NEXT loop
547 PRINT::	Print a blank line
550 PRINT TAB (2); "AVERAGE = "; AVE	Prints average score
560 RETURN	Exit subroutine, return to line 70

Step 5
Run the program and debug it

```
        GRADE 1

        SCORE? 10
IS DATA CORRECT? Y

        GRADE 2

        SCORE? 20
IS DATA CORRECT? Y

        GRADE 3

        SCORE? 15
IS DATA CORRECT? N

        GRADE 3

        SCORE? 30
IS DATA CORRECT? Y

        GRADE 4

        SCORE? 40
IS DATA CORRECT? Y

        GRADE 5

        SCORE? 50
IS DATA CORRECT? Y

SCORES 10  20  30  40  50
   AVERAGE = 30
CALCULATE ANOTHER AVERAGE? N
** DONE **
```

Example seven:
Procedure for diagnosing automobile failure

Step 1
Identify the problem
Develop a procedure to determine why a car won't start.

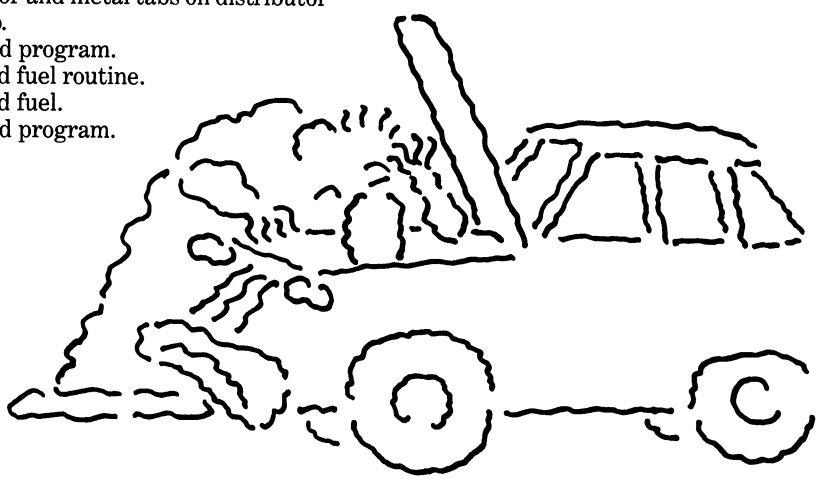
Step 2
Outline the solution

Inputs
1. Answer questions yes or no.

Process
1. Compare answer to determine branching to next test.

Outputs
1. Print statements of problem and check fuel.
2. If fuel empty, go to add fuel routine.
3. Have someone try to start car while you watch for spark to jump from wire to block.
4. If spark jumps go to check points routine.
5. Check other cylinders.
6. If cylinders don't have spark go to inspect rotor routine.
7. Problem may be in spark plugs.
8. End program.
9. Check points routine.
10. Remove distributor cap and check points.
11. End program.
12. Inspect rotor routine.
13. Remove distributor cap and inspect rotor and metal tabs on distributor cap.
14. End program.
15. Add fuel routine.
16. Add fuel.
17. End program.

The program is a series of instructions with branching based on the answer to each question.



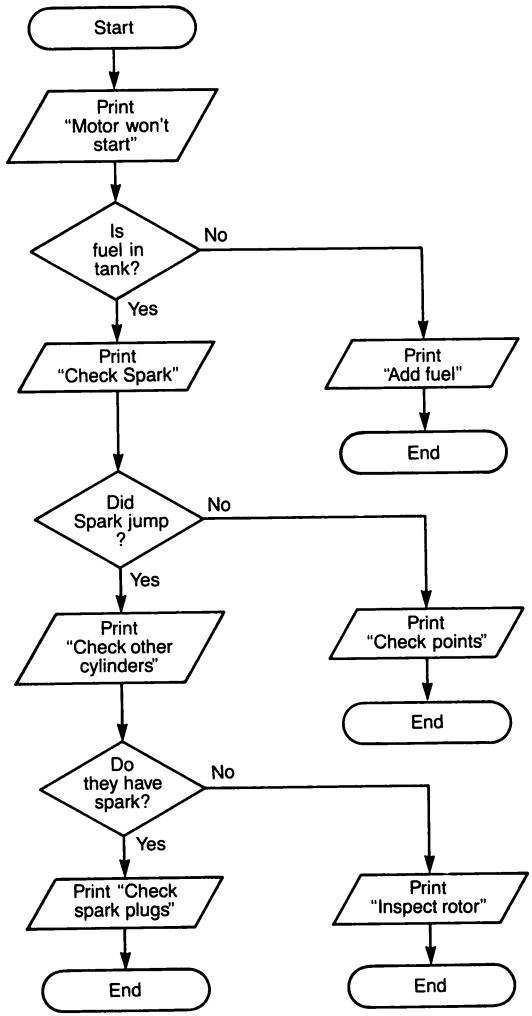
Step 3
Prepare a flowchart

Assign variable names

Inputs
A$ = User's YES or NO answer.

Process
B$ = "YES"

Outputs
C$ = Motor won't start.
T$ = Is fuel in tank?
D$ = Check ignition system.
E$ = Remove one spark plug wire at plug.
F$ = Hold wire about one-quarter inch from motor.
G$ = Have another person try starting the car.
H$ = Did spark jump from wire to motor, Yes or No?
I$ = Check all other cylinders for spark in same way.
J$ = Did all cylinders have spark, Yes or No?
K$ = Problem may be in spark plugs.
L$ = Remove and check spark plugs.
O$ = Remove distributor cap. Check to see if points are opening and closing as motor is turned.
R$ = If opening and closing, replace points. If not, adjust point gap.
X$ = Remove distributor cap.
Y$ = Inspect rotor and metal tabs on distributor cap.
U$ = Add fuel.



Step 4
Write the program

Instruction

The coding is an alternating sequence of LET statements and print statements. Each LET has a matching PRINT.

The input variable is always A$ and is compared to the value "YES" assigned to B$. If A$ precedes B$; that is, if A$ = NO, the program branches to a different routine. If the answer (A$) does not precede "YES", then the program continues at the next line.

The program continues dependent upon your answers at each point. As you can see, a computer program can guide you in a step-by-step procedure that includes decision points.

Note
This program has four terminating points with the END statement.

```
5 LET B$ = "YES"
10 LET C$ = "MOTOR WON'T START"
20 PRINT C$
22 LET T$ = "IS FUEL IN TANK?"
23 PRINT T$
24 INPUT A$
25 IF A$ < B$ THEN 340
30 LET D$ = "CHECK IGNITION
SYSTEM"
40 PRINT D$
50 LET E$ = "REMOVE ONE SPARK
PLUG WIRE AT PLUG"
60 PRINT E$
70 LET F$ = "HOLD WIRE ABOUT
ONE-QUARTER INCH FROM MOTOR"
80 PRINT F$
90 LET G$ = "HAVE ANOTHER
PERSON TRY STARTING THE CAR"
100 PRINT G$
110 LET H$ = "DID SPARK JUMP
FROM WIRE TO MOTOR, YES OR
NO?"
115 PRINT H$
120 INPUT A$
130 IF A$ < B$ THEN 250
140 LET I$ = "CHECK ALL OTHER
CYLINDERS FOR SPARK IN SAME
WAY"
150 PRINT I$
160 LET J$ = "DID ALL CYLINDERS
HAVE SPARK, YES OR NO?"
165 PRINT J$
170 INPUT A$
180 IF A$ < B$ THEN 300
190 LET K$ = "PROBLEM MAY BE IN
SPARK PLUGS"
200 PRINT K$
210 LET L$ = "REMOVE AND CHECK
SPARK PLUGS"
215 PRINT L$
220 END
250 LET O$ = "REMOVE
DISTRIBUTOR CAP. CHECK TO SEE IF
POINTS ARE OPENING AND CLOSING
AS MOTOR IS TURNED."
260 PRINT O$
270 LET R$ = "IF OPENING AND
CLOSING, REPLACE POINTS. IF NOT,
ADJUST POINT GAP"
```

Instruction

```
275 PRINT R$
280 END
300 LET X$ = "REMOVE
DISTRIBUTOR CAP"
310 PRINT X$
320 LET Y$ = "INSPECT ROTOR AND
METAL TABS ON DISTRIBUTOR CAP."
325 PRINT Y$
330 END
340 LET U$ = "ADD FUEL"
350 PRINT U$
360 END
```

Step 5
Run the program and debug it

```
MOTOR WON'T START
IS FUEL IN TANK?
? YES
CHECK IGNITION SYSTEM
REMOVE ONE SPARK PLUG WIRE A
T PLUG
HOLD WIRE ABOUT ONE-QUARTER
INCH FROM MOTOR
HAVE ANOTHER PERSON TRY STAR
TING THE CAR
DID SPARK JUMP FROM WIRE TO
MOTOR, YES OR NO?
 .
 .
 .
** DONE **
```

And so on.

5 Steps to Successful Programs

What kind of problems can the computer solve?

The computer can solve any task that can be programmed.

How do you develop a computer program?

By diligently following five basic steps, a computer program can be developed.

Identify the problem

Outline the solution

Prepare a flowchart

A flowchart is similar to a road map.

Start and end
Input and output
Direction
Process
Decisions
Connectors

Write the code

This is "the program."

The real test of a program — will it run? If not, you have to get the "bugs" out.

Run the program and debug it

Example one:	Make a cup of hot tea
Example two:	Print names and addresses
Example three:	Compute total car cost if bought on time payments
Example four:	Expanding the auto cost program

Structured programming

A more efficient and orderly way to program. Makes use of the GOSUB and RETURN statements.

Change in presentaton of flowcharts

Example five:	Structured programming of the auto cost program

Arrays

Permit similar information to be identified and stored as a group, yet each item is individually accessible.

Name and subscripts

DIM statement

Reserves memory to store array information.

Example six:	Calculate grade averages
Example seven:	Procedure for diagnosing automobile failure

Summary map

2 **Introduction**

3 **Computer assisted instruction**

Drill

Concepts

Eye-hand coordination

Simulation

4 **Computer literacy**

5 **Administrative applications**

Grade records

Special education applications

Word processing

Attendance records

7 **Educational program examples**

Example one:	Math drill of addition facts.

10 **RES command**

Example one:	Renumbered
Example two:	Math drill of addition, subtraction and multiplication facts
Example three:	Multiple choice tests
Example four:	Compute the chronological age of a person
Example five:	A gradebook program

33 **ABS function**

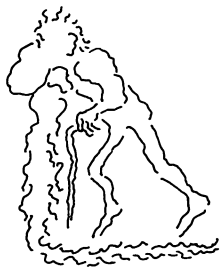
Example six:	Solving an equation

36 **Summary map**

Example one

Example two

Example four



Example five

Creating Educational Programs

Introduction

Microcomputers are showing up everywhere in the educational community. They can be seen in the elementary classroom as well as in graduate school. They are being used in administrative offices in increasing numbers. Large numbers of children and adults are learning about computers, how to program them, and how to use them effectively.

The use of the small computer is increasing rapidly for both classroom use and administrative use.

The value of computers in the educational community is increasing daily. During the 60's, the microcomputer had not yet arrived and the computer had limited impact. Money from the federal government made it possible for some high schools to teach programming to their students, but only a few administrative offices could afford the luxury of a computer. In the early 70's, even less money was available for computers and the future looked bleak for the use of computers in the educational community. The minicomputer had reached the marketplace, but the schools didn't have the money to buy them.

With the introduction of the microcomputer in the late 70's, and their acceptance by a large percentage of the general public, it became imperative that schools once again become involved with the use of computers. Schools have begun to use microcomputers to teach their students about what computers are and what they can do, as well as using them for assistance in instruction of other subjects. Also, microcomputers are being linked to a large main computer (called the host computer) so educational programs written on a microcomputer can be stored on the host computer. In return, all the programs on the host computer can be accessed by the microcomputers at the individual schools. This method of sharing results in increased efficiency.

The three primary uses for computers in education are:

a. Computer assisted instruction
b. Computer literacy
c. Administrative

Computer assisted instruction

Drill

The computer can be a valuable helper to the busy teacher. It doesn't replace the teacher, but rather reinforces what the teacher has already taught. Drill programs can help a student master the basics. The computer is effective with drill because it is patient and consistent.

Teachers have busy schedules and large classes; therefore, they have few opportunities to work with the children on a one to one or one to two basis. The microcomputer can help fill this need because a major emphasis of computer assisted instruction (CAI) is to provide drill work to reinforce basic skills already introduced and taught by the teacher. If a student has not mastered the material and is forced to go on to the next lesson, large and sometimes crucial gaps can develop in the student's knowledge. For example, it is difficult to multiply two-digit numbers that involve carrying when one hasn't even mastered the basic addition facts. If a properly programmed microcomputer is available, the teacher need not wait until it's time for correction of these fundamental weaknesses, but can let the student use CAI drill programs to practice addition, subtraction, multiplication, and division as needed.

The better programs have color, sound and animation to help hold the student's attention.

The computer is a very patient teacher and is consistent in whatever it teaches. Good CAI programs not only aid children (and adults for that matter) in the learning process, but also are structured to motivate the students and to hold their interest. Programs written for preschool and early elementary grade children usually include color and sound for stimulation, motivation, and emphasis. Older children and adults also like color and may stay with a program longer when color graphics are used. The TI Home Computer has both color and sound capabilities which will be discussed in Chapter 9.

Computers can be very useful for developing vocabulary, spelling, and mathematical drills. They aid in memory drills that call for facts such as state capitals, United States presidents, countries of the world, and oceans of the world. Reading skills are tested to determine comprehension by a variable set of questions randomly selected by the computer. Testing also may be done with multiple choice questions or with essay questions if the library of questions is large enough so that random selection can occur.

Concepts

Computers also can be used to teach new concepts. The advantage is that the material can be presented and advanced at the pace of the student. TI has several educational packages to teach basic concepts that are colorful, fun and motivating.

Eye-hand coordination

Computer games can improve a person's eye-hand coordination and self-confidence

Although not mentioned often, the use of CAI can have benefits other than education. The child that has poor eye-hand coordination usually can gain increased coordination through the use of appropriate computer games. The children may play a game which includes trying to hit moving targets on the screen, or they may guide a plane through a group of missiles to avoid being hit. In addition to the eye-hand coordination benefit, the attention span of a hyperactive child is increased because the child becomes involved in the fast-paced action of the game. A child also can gain self-confidence as he/she masters the game without the embarrassment of the other children watching and criticizing their mistakes.

Simulation

Simulation of real events on the computer allows students to observe and participate in situations that otherwise are not possible.

Computers can be used for simulations. Computer simulation is the attempt to duplicate on a computer the events that occur in the real world so the student can learn about them without direct experience. The simulation may be of events that cannot be observed in the normal operating environment. An example is the simulation of the activity within an internal combustion engine. Motion picture films have been used for this for many years, but films are difficult to rerun quickly so the student can review a scene several times. By using a computer, the simulation can be done over and over at the press of a button until the student has mastered the concept.

Computer simulation provides a quick and easy means to change variables to answer "what if" questions.

Another purpose of simulation is to use it as a predictive tool for "what if" situations. What if farmers had severe crop failures for two consecutive years? What if an automobile body were designed and shaped in a different way? By using a computer with the proper simulation programs, the student can do a "what if" analysis again and again with a slight modification of the variables each time. For example, if the birth rate increased by two percent per year with the food supply remaining at the current level, how many people would starve over the next ten years? What if the birth rate remained at the same level? What if the birth rate were to decline by 2%? By 5%? By 10%? This kind of data can be obtained by computer simulation. In the real world, you could only obtain it by actual events, and then it's too late to do anything about it.

Computer literacy

Introduction of computers to children at an early age prevents the development of fear of computers experienced by some adults.

Another major development in the educational community is the teaching of computer literacy, particularly in the fourth through twelfth grades. These students are learning what computers are, what they can and cannot do, and their impact on society. Even preschool and first grade students are being introduced to computers. As a result, the present generation is growing up with computers and does not fear them as many adults do. Adults developed this fear because they thought computers would take their jobs or take control of their lives. Some have felt a loss of personal identity because of the computer. But now, computers are gradually becoming a part of all schools and classrooms and many parents have already bought computers for their family to use at home.

Computer-type equipment is used in many home appliances and in the family car.

People are also learning that electronic microprocessors similar to those which control computers are in their games, toys, color tv, food processor, microwave oven, and even the family car. The youngsters of today are expected to understand and use the technology in today's world; therefore, computer literacy is a necessary part of their education.

The children of today are learning about computers and programming so they will be able to function in the computerized world of tomorrow.

Another part of computer literacy is learning computer programming. BASIC is the computer language used by most microcomputers and many large high school districts have classes in programming in the BASIC language. Since it is a relatively easy computer language to learn, children, teachers and parents are learning to program computers to do some intricate things. Even children as young as six years old are learning how to program. Some youngsters are assisting adults in teaching programming and some are being employed as part-time programmers for major businesses.

Once people learn to program, they can apply the capabilities of the computer to solve problems. Recall that the steps for developing a program to solve problems were presented in Chapter 5.

Administrative applications

The third major area of computer use in the educational community is that of record keeping. Both teachers and administrators have to keep many kinds of records on students and the school system. Some examples of records kept by teachers are:

1. Assignments completed and grades given
2. Schedules
3. Individualized education plans
4. Diagnostic report analyses of test results
5. Student information files

Some administrative recordkeeping uses are:

1. Attendance records
2. Discipline reports
3. Teacher and class schedules
4. Monitoring costs
5. Up-to-date audio/visual library usage
6. Budget forecasting
7. Maintenance reports

Grade records

The computer can help the teacher and administrators with grade recording, averaging and reporting.

To illustrate the use of record keeping, let's consider grade records. As assignments are returned to the teacher, the grades are logged into the grade book. The teacher keeps grades for each student and must post them periodically. Most teachers today are still doing this task by hand, but it can be done with a home computer (if the school district permits it). Programs can be written so that all classroom information can be recorded. Whenever requested, even between grading periods, the grade averages can be obtained quickly and easily by running the grade average program. (Refer to Chapter 5 for a simplified grade averaging program.) Many options can be programmed to fit the needs of the particular teacher using the program. A job that has taken hours can take only minutes.

With the cooperation of the local school and/or school district, the report cards also can be computer generated, thus eliminating the need to write the reports by hand. (Of course, the teacher still must provide the inputs for the computer generated reports.) If properly designed, programs that utilize student files also can generate on request the required administrative reports on pupil progress for the district. Thus, by allowing the computer to do some of the routine chores, a teacher's productivity can be increased.

Special education applications

In many cases, public law requires that every child enrolled in a special education program have an individualized education plan (IEP). This document requires:

1. Personal data such as name, age, sex and birthdate
2. Test scores for reading, math, spelling, writing, perceptual/motor skills, psychological scores
3. Data about any physical impairments
4. Goals and objectives to be achieved in both the long and short term in all areas of the child's education in the academic and non-academic areas
5. The personnel responsible for carrying out the goals and objectives
6. A list of other services needed such as psychological and physical therapy, glasses and hearing aids
7. Review date of progress

8. Signatures of parents as well as the professionals in the child's educational plan
9. Type of placement, whether resource room or self-contained classroom
10. Any other documentation.

Education plans and similar reports that use the same resources again and again can be done easier with the aid of a computer.

As you can see, writing the IEP requires a lot of data and time. The crucial part is writing the goals and objectives. By having computerized files that contain the goals, objectives, and resources available, these items will not have to be typed in repeatedly for each student. Many sequences are the same for various students, but not all the same resources are appropriate for different students. By having the names of all of the available resources stored in the computer file, the teacher can select the appropriate ones and have the IEP computer program write the lengthy reports. The process often does not end here, as revisions are required before everyone agrees on the plan. Any changes would require that the report be redone manually if it were not computerized. By using the computer, revisions can be made quickly and with little human effort.

Word processing

The use of the computer as a word processor is a valuable asset to anyone who ordinarily would use a typewriter to prepare letters, reports, announcements, and so on. Text can be quickly corrected, changed and rearranged without ever printing it on paper. By merging the text with a computer file of names and addresses, individualized originals can be sent to many people with only one "typing".

The use of the microcomputer as a word processor can be valuable to a school administrator as well as to teachers. A word processor is a computer program that allows one to type in text much as one would type a report or letter on a typewriter. The differences are:

1. The text is stored in the computer memory or on an external storage device.
2. The text can be corrected without retyping the entire letter which saves valuable time and the possibility of new errors in the retype.
3. Commonly used passages and paragraphs can be stored in a file and used in different reports, letters, etc. without the need for retyping each one.
4. Multiple "originals" of entire reports can be made without typing each one.
5. Personalized letters can be generated without doing any typing other than the initial list of names, addresses, and other variable information.

For example, a word processor can be used to write a letter to all parents of children in a class to inform them of a planned field trip and to obtain their permission for their child to participate. Instead of having the secretary type the same letter 20 times with the appropriate names, addresses, salutations, etc., the letter is entered into the computer, edited, and corrected by using the word processor. The individual information, such as name, address and salutation, that varies with each letter is contained in a data base (a computer file). This data is automatically accessed when each letter is printed and the word processor program places the correct name, address, and salutation on each letter. If the data base has not already been created through registration procedures, the secretary can use the word processor to create the data base and save it for future use. The data base can also be used to address the envelopes. By using this method, each parent receives an original personalized letter with minimum human effort.

Attendance records

Attendance records are very important to school administrators. This routine task can be handled easily by a computer.

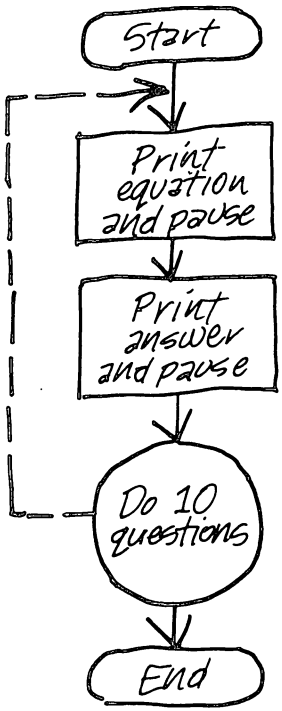
Programs can be written that calculate the attendance figures, the absentee rates, and all other information required for school records. Much of this information is required to receive the funding that is based on attendance records. This type of work is time consuming and prone to human error when done manually. However, by just entering the requested information, the computer does all the calculations and prints the reports quickly and easily. If errors in the input data are found, they can be easily corrected and the report program rerun.

Educational program examples

Six programs will be written to demonstrate the educational uses of microcomputers and to help you learn to program.

Example one:
Math drill of addition facts

Step 1
Identify the problem
Generate ten addition problems using random numbers between −50 and +50. Allow time for the student to think of the answer, then print the answer.



Step 2
Outline the solution

Inputs
None

Process
1. Clear the screen.
2. Generate two random numbers.
3. Compute the answer.
4. Repeat for 10 problems

Outputs
1. Print the two random numbers with addition indicated.
2. Print the answer after a time delay which allows the student to think of the answer (student does not type in the answer).

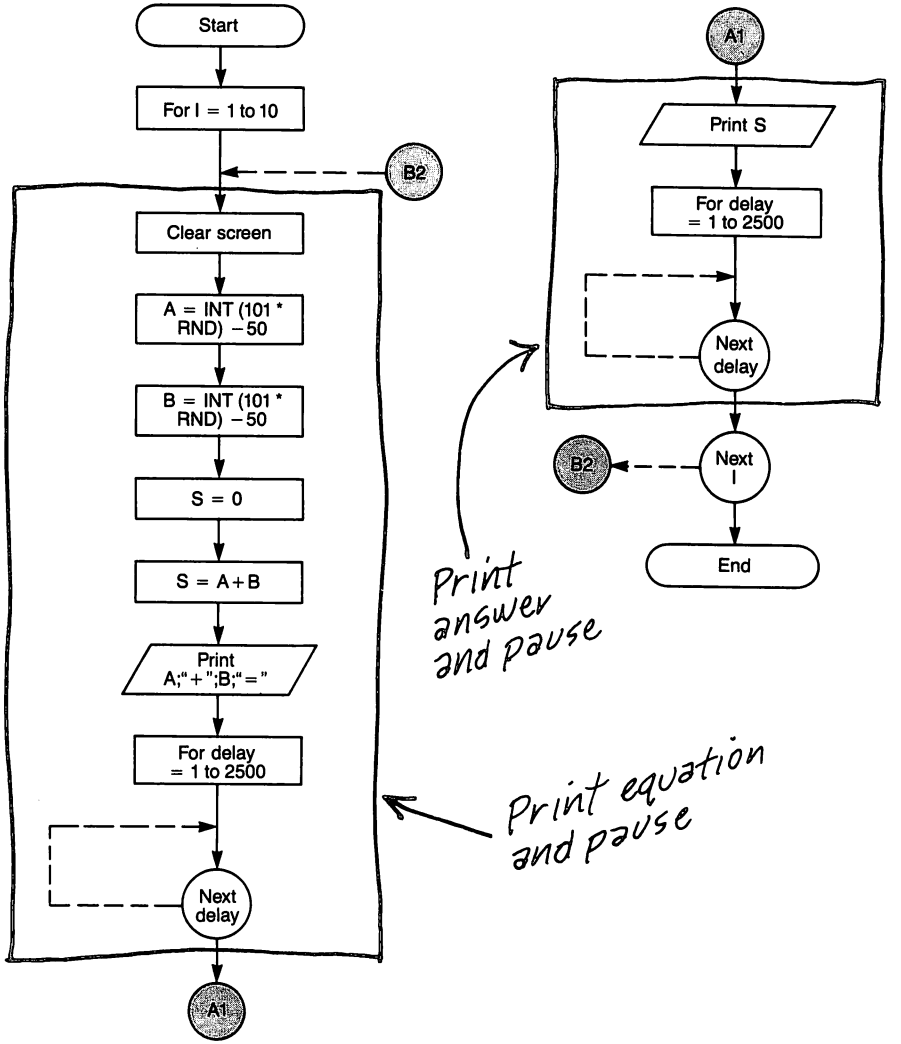
Step 3
Prepare a flowchart

Assign variable names

Inputs
None

Process
A = Random number
B = Random number
S = Answer
DELAY = Delay loop index for timing
I = Main loop index

Outputs
A + B = S



Step 4
Write the code

	Instruction	Explanation
	10 REM MATH DRILL 1	
	20 RANDOMIZE	Sets random generator for new series
The FOR-NEXT loop causes 10 different problems to be presented.	100 FOR I = 1 TO 10	Begin loop to present 10 problems
	110 CALL CLEAR	Clears the screen
	120 A = INT(101*RND) – 50	Produces first random number between – 50 and + 50 for variable A
	130 B = INT(101*RND) – 50	Produces second random number between – 50 and + 50 for variable B
	135 S = 0	Sets answer variables to zero as a safety measure
	140 S = A + B	Computer calculates the answer
	150 PRINT A; " + "; B;" = ";	Prints the addition problem on the screen
	170 FOR DELAY = 1 TO 2500	Begin delay loop to allow student time to think of answer before printing answer
	172 NEXT DELAY	End delay loop
	175 PRINT S	The correct answer is printed on the screen for the student to compare his/her answer
	180 FOR DELAY = 1 TO 2500	Delay loop to hold printed answer on screen for a few seconds before next problem is presented
	190 NEXT DELAY	
	200 NEXT I	End of control loop to present the 10 problems where control is returned to line 100. After the last problem, line 210 is executed
	210 END	

Step 5
Run the program and debug it

This program can be expanded to allow the student to choose between addition, subtraction and multiplication.

```
20  + – 5  =
```

```
20  + – 5  =  15
```

```
42  +  37  =
```

```
42  +  37  =  79
```

Note
And so on for 8 more problems

```
.
.
.
** DONE **
```

RES command

The above program appears as originally constructed with some of the intial statements removed and new statements added as needed between initial statements. This results in irregularly spaced line numbers. While this causes no problem for the computer, a program with line numbers spaced in equal increments is easier to type and read. The RESequence command on the TI Home Computer will resequence (renumber) all the line numbers in equal increments and change the line number references in statements. If only RES is typed, the program will be renumbered in increments of 10 beginning with line number 100. Let's renumber the example one program now.

Type RES
Press **ENTER**

Example one:
Renumbered

List the program and it should be numbered as follows:

Instruction	Explanation
100 REM MATH DRILL 1	
110 RANDOMIZE	Sets random generator for new series
120 FOR I = 1 TO 10	Begin loop to present 10 problems
130 CALL CLEAR	Clears the screen
140 A = INT(101*RND) – 50	Produces first random number between – 50 and + 50 for variable A
150 B = INT(101*RND) – 50	Produces second random number between – 50 and + 50 for variable B
160 S = 0	Sets answer variable to zero as a safety measure
170 S = A + B	Computer calculates the answer
180 PRINT A; "+"; B;"=";	Prints the addition problem on the screen
190 FOR DELAY = 1 TO 2500	Begin delay loop to allow student time to think of answer before printing answer
200 NEXT DELAY	End delay loop
210 PRINT S	The correct answer is printed on the screen for the student to compare his/her answer
220 FOR DELAY = 1 TO 2500	Delay loop to hold printed answer on screen for a few seconds before next problem is presented
230 NEXT DELAY	
240 NEXT I	End of control loop to present the 10 problems where control is returned to line 120. After the last problem, line 250 is executed
250 END	

Example two:
Math drill of addition, subtraction and multiplication facts

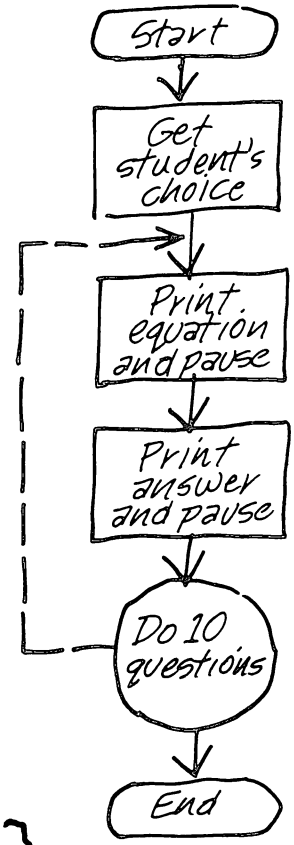
Step 1
Identify the problem
Expand on example 1 to allow student to choose addition, subtraction, or multiplication. Generate ten problems using random numbers.

Step 2
Outline the solution

Inputs
1. User selects either addition, subtraction or multiplication from menu.

Process
1. Clear the screen.
2. Generate two random numbers.
3. Compute the answer.
4. Repeat for 10 problems.

Outputs
1. Print the two random numbers with the chosen mathematical operation indicated.
2. Print the answer after a time delay which allows the student to think of the answer (student does not type in the answer).



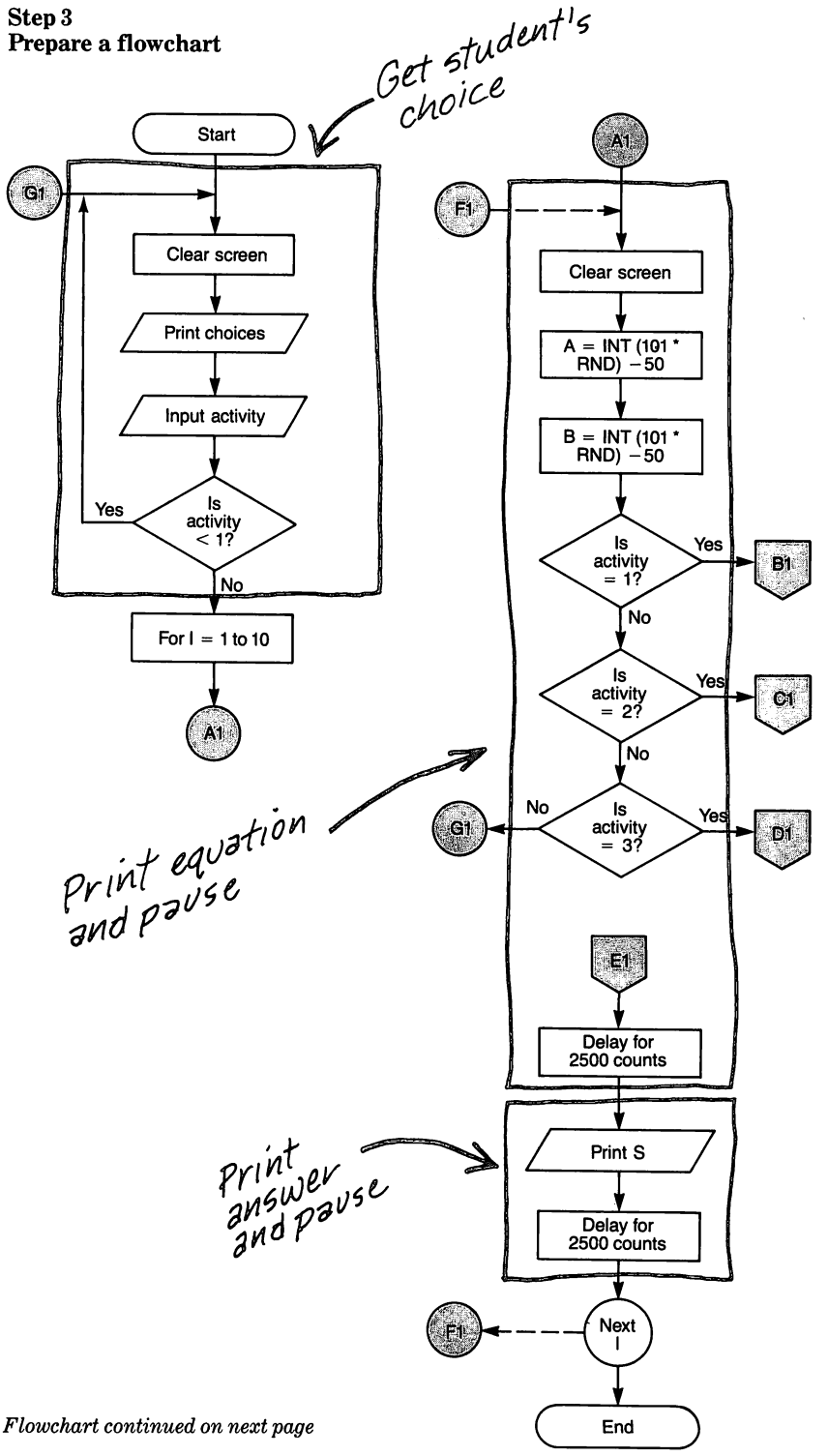
Step 3
Prepare a flowchart

Assign variable names

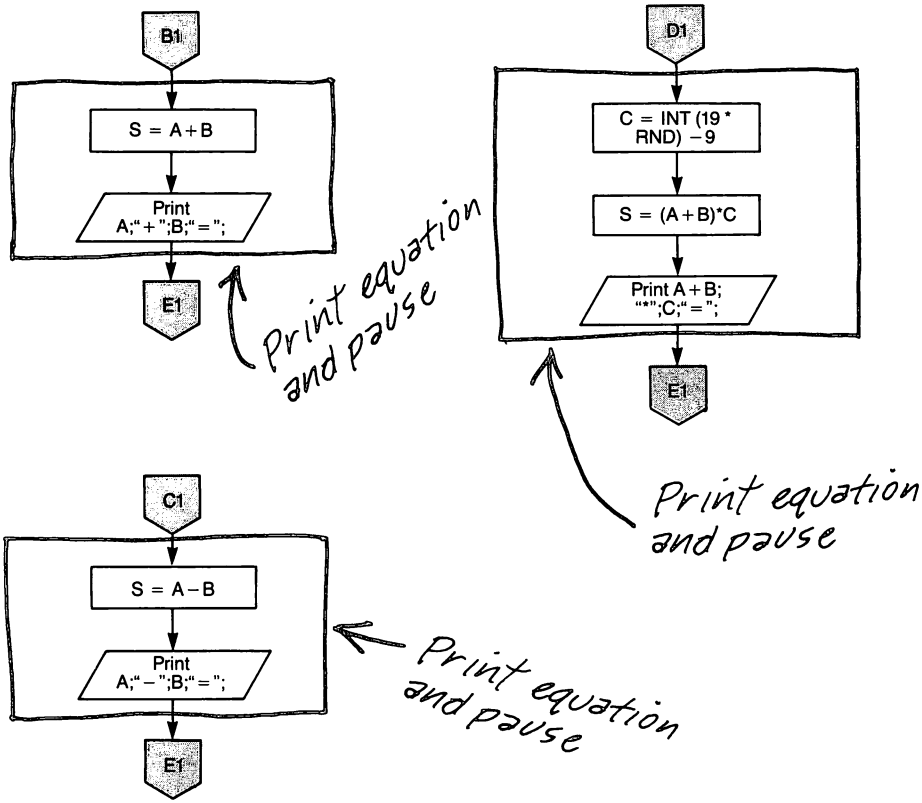
Inputs
ACTIVITY = Type of mathematical operation desired

Process
A = Random number
B = Random number
C = Random number
S = Answer
DELAY = Delay loop index for timing
I = Main loop index

Outputs
A + B = S
A − B = S
(A + B) * C = S



Flowchart continued on next page



Step 4
Write the code

This program is slightly more complicated since the selection routine and subroutines for the three operations have been added.

Instruction	Explanation
10 REM MATH DRILL EXERCISE	
20 RANDOMIZE	Set random generator
30 CALL CLEAR	Clears screen
40 PRINT TAB(8);"MATH DRILL"::::	Prints heading followed by 2 blank lines. Indentation is 8 spaces from left margin
50 PRINT TAB(8);"1 ADDITION"	Lines 50 through 70 print choices indented 8 spaces from left margin
60 PRINT TAB(8);"2 SUBTRACTION"	
70 PRINT TAB(8);"3 MULTIPLICATION"	
80 PRINT::	Prints two blank lines
90 PRINT TAB(8);"TYPE 1, 2, OR 3";	Prints instruction for selecting choice
100 INPUT ACTIVITY	Requests the user to select one of the three operations
110 IF ACTIVITY < 1 THEN 30	Line 110 is an error monitor to prevent numbers lower than 1 from being accepted

	Instruction	Explanation
	120 FOR I = 1 TO 10	Begin loop that generates the 10 problems
	130 CALL CLEAR	Clears the screen
	140 A = INT(101*RND) – 50	Generates a random number from – 50 to + 50 and assigns it to A
	150 B = INT(101*RND) – 50	Generates a random number from – 50 to + 50 and assigns it to B
	160 S = 0	Sets the answer variable to zero as a safety precaution
Be sure to keep the loops straight.	170 IF ACTIVITY = 1 THEN 280	If operation chosen is addition, execution is transferred to line 280
	180 IF ACTIVITY = 2 THEN 320	If operation chosen is a subtraction, execution is transferred to line 320
	190 IF ACTIVITY = 3 THEN 360	If operation chosen is multiplication, execution is transferred to line 360
	200 GO TO 30	If the choice is not 1, 2 or 3, then the program is returned to line 30 to again display the options and allow another chance at selection
	210 FOR DELAY = 1 TO 2500	Begin delay loop to allow student time to think of answer
	220 NEXT DELAY	End delay loop
	230 PRINT S	The correct answer is printed on the screen
	240 FOR DELAY = 1 TO 2500	Delay loop holds the printed answer on the screen for a few seconds before the screen is cleared for the next problem
	250 NEXT DELAY	
	260 NEXT I	Program execution returns to line 120 until the last problem has been presented, then line 270 is executed
You will probably think of other ways to code the program, and you are encouraged to modify the program to fit your own needs.	270 END	
	280 REM ADDITION SUBROUTINE	
	290 S = A + B	The solution is calculated by the computer
	300 PRINT TAB(5);A;" + "; B;" = ";	Prints the values for the variables A and B and the literals " + " and " = "
	310 GOTO 210	Goes to the delay routine in line 210
	320 REM SUBTRACTION SUBROUTINE	
	330 S = A – B	Subtraction calculated by computer
	340 PRINT TAB(5);A;" – "; B;" = ";	Prints the values for the variables A and B and the literals " – " and " = "
	350 GOTO 210	Goes to the delay routine before printing the answer
	360 REM MULTIPLICATION SUBROUTINE	
	370 C = INT(19*RND) – 9	Obtains random value of – 10 and + 10 for multiplication variable C
	380 S = (A + B)*C	Computes A + B and multiplies that result times C. The answer is stored in variable S

Instruction	Explanation
390 PRINT TAB(5);A+B; "*";C;"==";	Prints sum of variables (A + B) as one number, variable C, and literals "*" and "="
400 GOTO 210	Returns execution to line 210 for the time delay before printing the answer

Step 5
Run the program and debug it

```
MATH DRILL

1 ADDITION
2 SUBTRACTION
3 MULTIPLICATION

TYPE 1, 2, or 3? 3
```

```
20 *-5 =
```

```
20 *-5 =-100
```

```
42 * 3 =
```

```
42 * 3 = 126
```

```
.
.
.
** DONE **
```

Note
And so on for 8 more problems.

Example three:
Multiple choice test

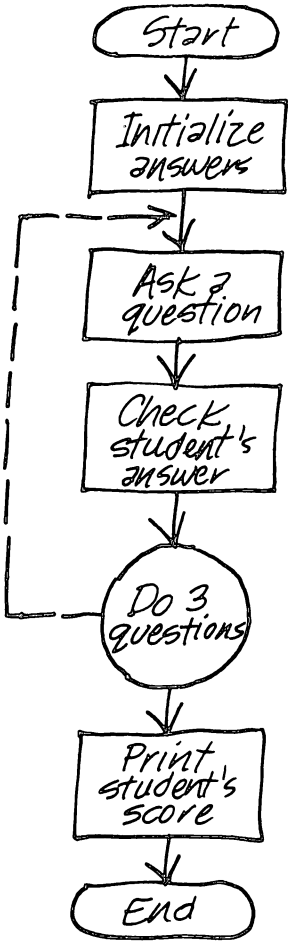
Step 1
Identify the problem
Give three questions one at a time with each question having multiple choice answers. The student selects an answer from the multiple choice list by typing the corresponding number. The computer evaluates the student's choice and keeps score. If the answer is correct, 1 point is added to the correct answer counter; if wrong, 1 point is added to the wrong answer counter. Print the number of right answers, the number of wrong answers, and the score in percent.

Step 2
Outline the solution

Inputs
1. Student selects answer by typing the choice number.
2. End program by pressing ENTER.

Process
1. Dimension the array for correct answers.
2. Clear screen for each question.
3. Computer presents the question.
4. Computer evaluates the answer choice and keeps track of response by adding to appropriate score counter.
5. Repeat for three questions.

Outputs
1. Print scores.



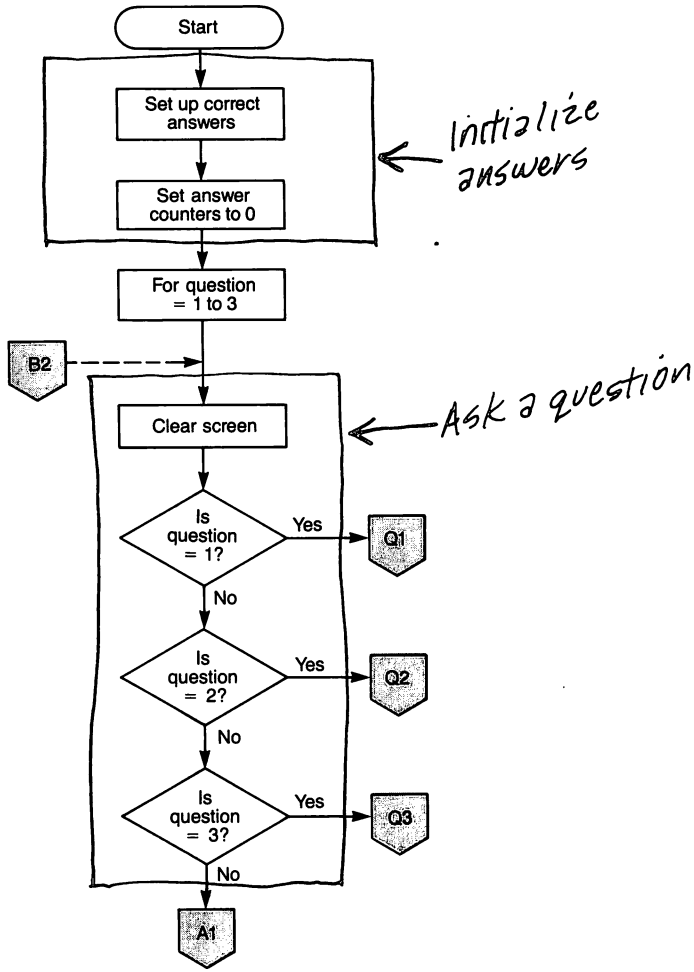
Step 3
Prepare a flowchart

Assign variable names

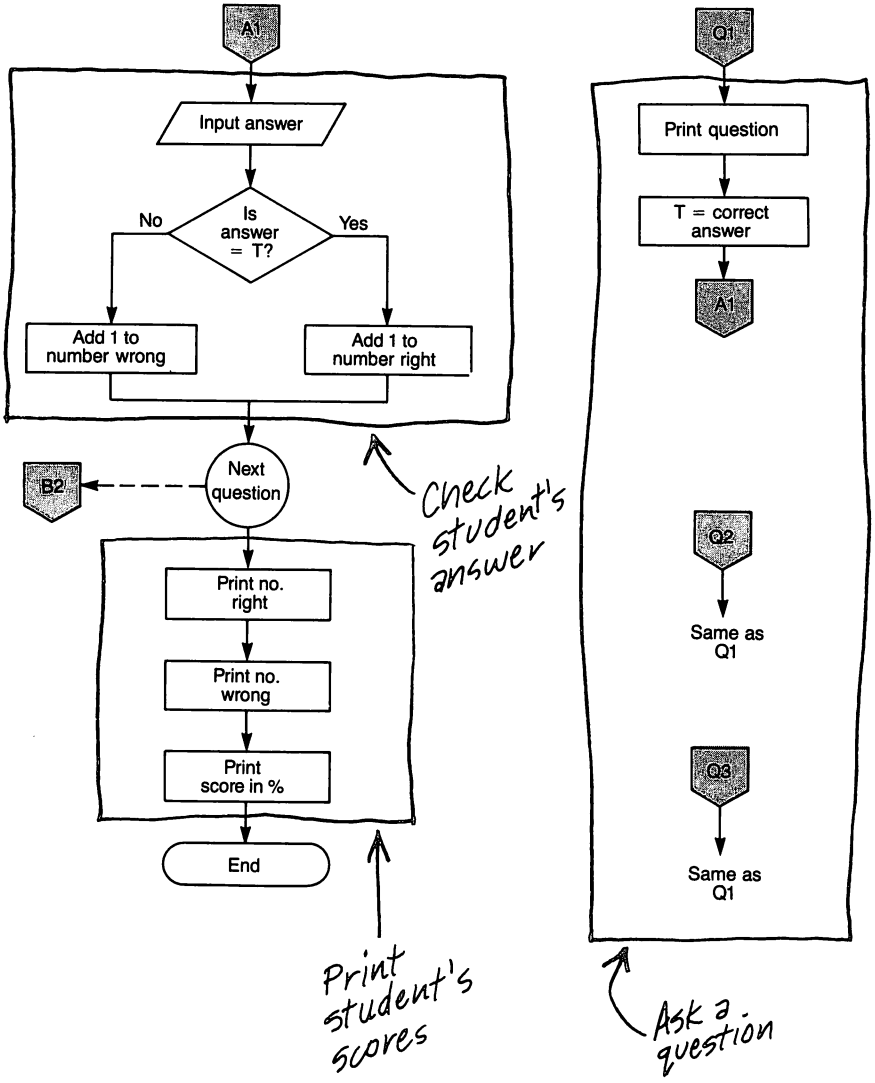
Inputs
C = User answer to question
A$ = Response to terminate program

Process
T = Correct answer for comparison to C
QUE( ) = array for answers to questions
RIGHT = Right answer counter
WRONG = Wrong answer counter
C = User answer
Q = Loop index

Outputs
RIGHT = number of answers correct
WRONG = number of answers wrong
SCORE = Score in percent



Flowchart continued on next page


A1
Input answer
No
Is answer = T?
Yes
Add 1 to number wrong
Add 1 to number right
B2
Next question
Print no. right
Print no. wrong
Print score in %
End
Check student's answer
Print student's scores
Q1
Print question
T = correct answer
A1
Q2
Same as Q1
Q3
Same as Q1
Ask a question

Step 4
Write the code

Instruction	Explanation
10 REM GEOGRAPHY TEST	
20 DIM QUE(3)	This array stores the correct answer for each question
30 REM SET VARIABLES TO ZERO	Initialize variables
40 C = 0	
50 T = 0	
60 RIGHT = 0	
70 WRONG = 0	
80 REM SET THE CORRECT ANSWER VALUES	Correct choices for the three questions are stored
90 QUE(1) = 3	
100 QUE(2) = 2	
110 QUE(3) = 4	
120 REM MAIN PROGRAM	
130 FOR Q = 1 TO 3	Loop control for program
140 CALL CLEAR	Clears screen
150 IF Q = 1 THEN 330	If this is first question, go to line 330
160 IF Q = 2 THEN 410	If this is second question, go to line 410
170 IF Q = 3 THEN 500	If this is third question, go to line 500
180 INPUT "TYPE 1, 2, 3 OR 4?":C	Answer the question with one of 4 choices
190 IF T = C THEN 220	If the correct answer in T is equal to the student's answer in C, then go to line 220
200 WRONG = WRONG + 1	Add 1 to number wrong counter
210 GOTO 230	Unconditional branch to line 230
220 RIGHT = RIGHT + 1	Add 1 to number right counter
230 T = 0	Set correct answer variable to zero
240 C = 0	Set student answer variable to zero
250 NEXT Q	Repeats main loop until 3 questions have been presented
260 CALL CLEAR	Clears screen
270 PRINT "NO. RIGHT";RIGHT	Prints the heading and value for the number answered correctly
280 PRINT:"NO. WRONG";WRONG	Prints the heading and value for the number answered incorrectly
290 PRINT ::"SCORE";INT((RIGHT/3)*100)	Prints two blank lines and computes and prints score on percentage basis
300 PRINT :::	Print 3 blank lines
310 INPUT "PRESS ENTER KEY TO END":A$	Press ENTER key to terminate program
320 END	
330 REM QUESTION 1	
340 PRINT "THE CAPITAL OF ARIZONA IS"	Prints question 1 and answer choices
350 PRINT:"1 TUCSON"	
360 PRINT:"2 LOS ANGELES"	
370 PRINT:"3 PHOENIX"	
380 PRINT:"4 PRESCOTT"::	
390 T = QUE(1)	Set answer variable to value for correct answer
400 GOTO 180	Unconditional branch

If external storage files for the questions and answers were used, a much larger library could be established. Another advantage of an external file is that a student couldn't LIST the program to find the correct answers.

Each question and its corresponding answers were put into separate subroutines so that the questions and answers are contained within the program.

Instruction	Explanation
410 REM QUESTION 2	
420 PRINT "WHAT DIRECTION IS THE TOP"	Question 2 and choices
430 PRINT "OF A MAP?"	
440 PRINT:"1 SOUTH"	
450 PRINT:"2 NORTH"	
460 PRINT:"3 EAST"	
470 PRINT:"4 WEST"::	
480 T = QUE(2)	Set answer variable to value for correct answer
490 GOTO 180	Unconditional branch
500 REM QUESTION 3	
510 PRINT "ARID CLIMATE IS USUALLY"	Question 3 and choices
520 PRINT:"1 WET"	
530 PRINT:"2 MOIST"	
540 PRINT:"3 WINDY"	
550 PRINT:"4 DRY"::	
560 T = QUE(3)	Set answer variable to value for correct answer
570 GOTO 180	Unconditional branch

You probably have some ideas to improve this program. Try them. Learning by doing and experimenting is the best teacher.

Step 5
Run the program and debug it

```
THE CAPITAL OF ARIZONA IS

1 TUCSON

2 LOS ANGELES

3 PHOENIX

4 PRESCOTT

TYPE 1, 2, 3 or 4? 3
```

```
WHAT DIRECTION IS THE TOP
OF A MAP?

1 SOUTH

2 NORTH

3 EAST

4 WEST

TYPE 1, 2, 3 or 4? 1
```

```
ARID CLIMATE IS USUALLY

1 WET

2 MOIST

3 WINDY

4 DRY

TYPE 1, 2, 3 or 4? 4

NO. RIGHT 2

NO. WRONG 1

SCORE 66


PRESS ENTER KEY TO END

** DONE **
```

Example four:
Compute the chronological age of a person

Step 1
Identify the problem
Determine a person's age to the nearest year and month when given the person's birth date and the current date.

Step 2
Outline the solution
The basic equation to determine age is

CURRENT DATE:		YEAR	MONTH	DAY
– BIRTH DATE	:	YEAR	MONTH	DAY
= AGE	:	YEARS	MONTHS	DAYS

The following rules will be used:

1. 30 days to every month.
2. When borrowing from months, add 30 to the days and subtract 1 from months.
3. When borrowing from years, add 12 to months and subtract 1 from years.
4. When determining age in years and months, if the number of days is greater than 15, add 1 to number of months.
5. If the number of days is 15 or less, no change in months.

With these rules established, let's work some examples by hand to determine what a computer program will have to do.

		Yr.	Mo.	Da.
Current date:		83	09	19
Birthdate	:	75	05	12
Age	:	8	4	7
or		8 yr.	4 mo.	

That was easy and straightforward since no borrowing or conversions were required. The next one requires borrowing from the months.

		Yr.	Mo.	Da.		Yr.	Mo.	Da.
Current date:		83	09	19	=	83	08	49
Birthdate	:	75	07	30	=	75	07	30
Age	:					8	01	19
or						8 yr.	2 mo.	

The second example requires borrowing because the day of the current date is less than the day of the birth date. Following rule 2, borrow 1 from the months (which changes months from 9 to 8) and add 30 to the days (19 + 30 = 49). Subtract dates. Then using rule 4, add 1 to months because days are greater than 15. Therefore, the age is 8 years and 2 months.

In the next example, borrowing is required from both the years and months. Follow rule 2 first, then rule 3. Use rule 4 for rounding.

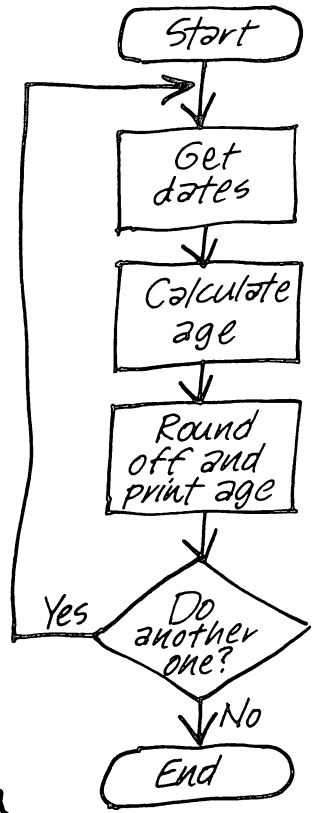
		Yr.	Mo.	Da.		Yr.	Mo.	Da.
Current date:		83	09	19	=	82	20	49
Birthdate	:	75	09	30	=	75	09	30
Age	:					7	11	19
or						8 yr.	0 mo.	

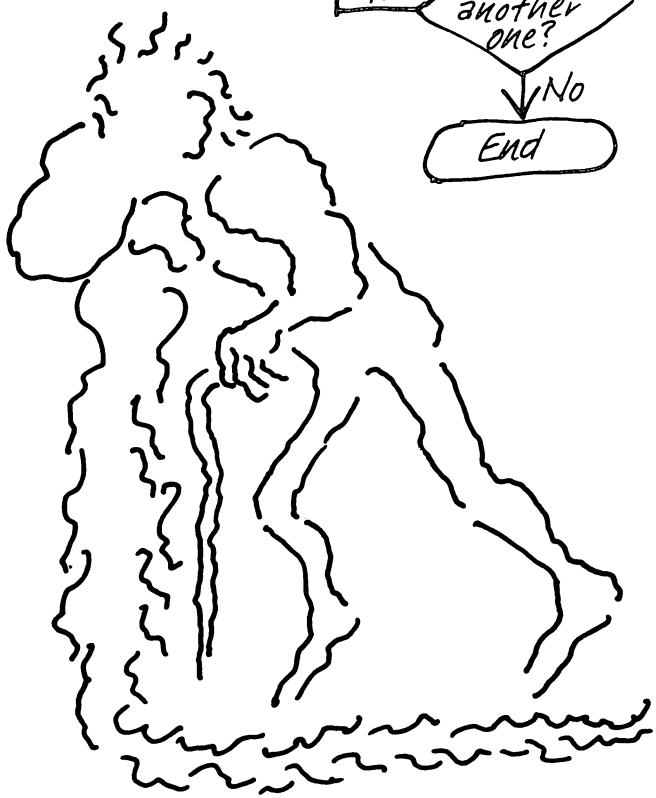
Now we can develop the outline.

Inputs
1. Input the current date.
2. Input the birthdate.
3. Answer response.

Process
1. Provide the rules for adjusting dates.
2. Compute the age.

Outputs
1. Print the age.
2. Ask if another calculation is desired.





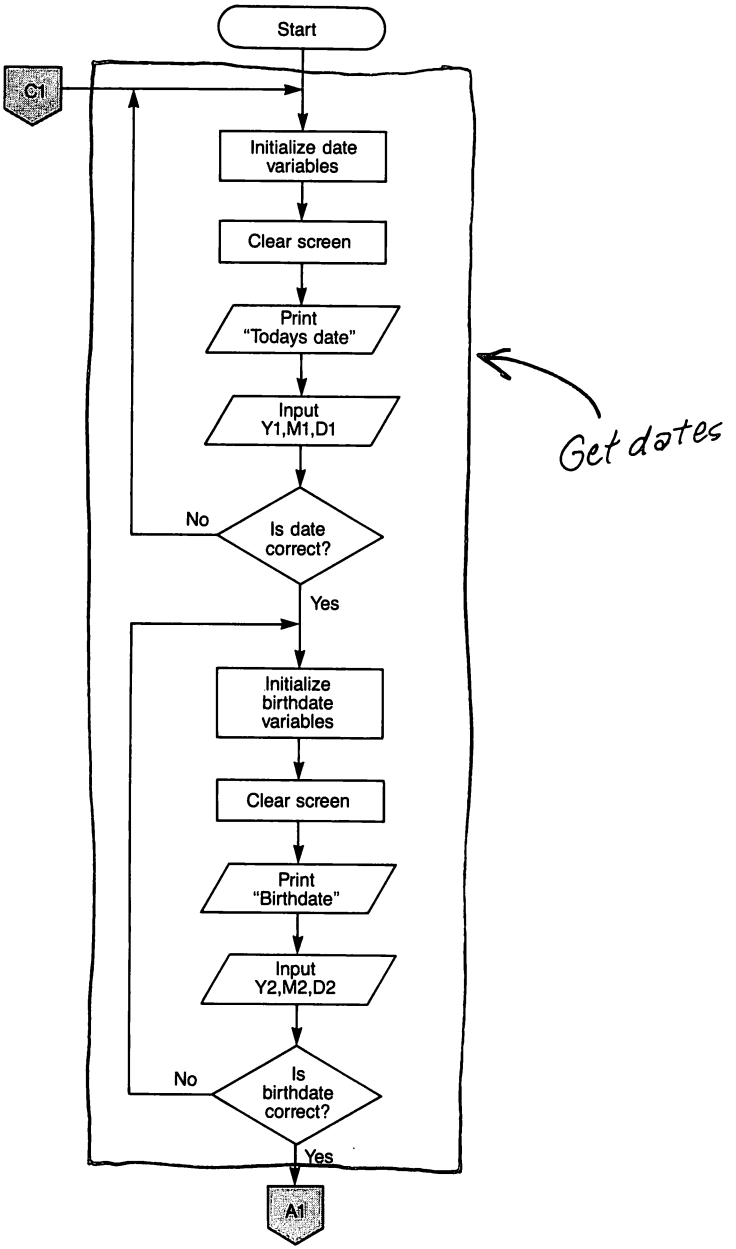
Step 3
Prepare a flowchart

Assign variable names

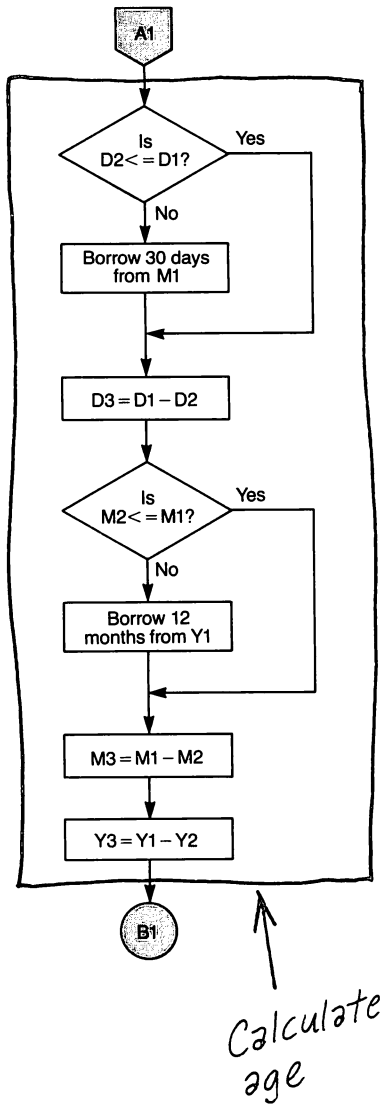
Inputs
Y1 = Current year
M1 = Current month
D1 = Current day
Y2 = Birth year
M2 = Birth month
D2 = Birth day
A$ = Answer input

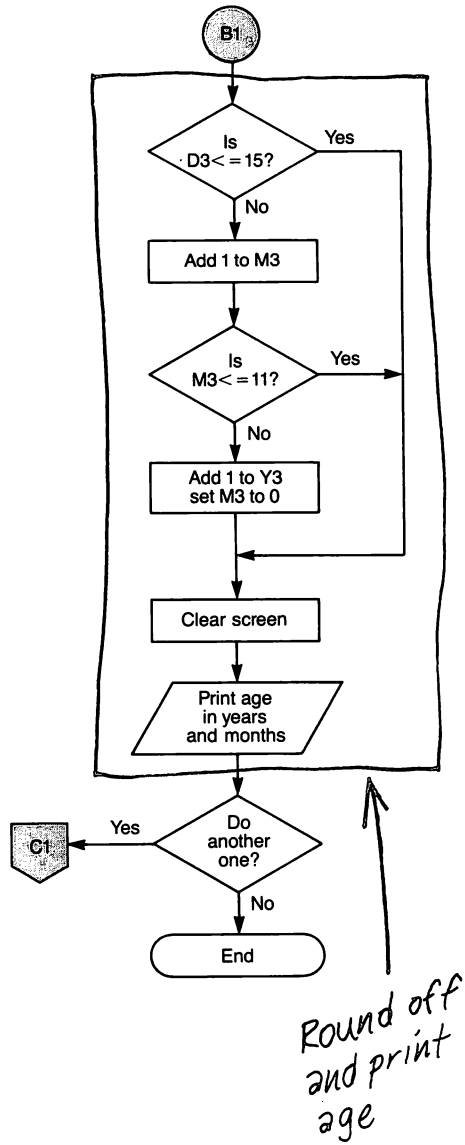
Process
Y1,M1,D1 = Current date
Y2,M2,D2 = Birth date
Y3 = Calculated years
M3 = Calculated months
D3 = Calculated days

Output
Y3 = Calculated years of age
M3 = Calculated months of age



Flowchart continued on next page


A1
Is D2< = D1?
Yes
No
Borrow 30 days from M1
D3 = D1 – D2
Is M2< = M1?
Yes
No
Borrow 12 months from Y1
M3 = M1 – M2
Y3 = Y1 – Y2
B1
Calculate age


B1
Is D3< = 15?
Yes
No
Add 1 to M3
Is M3< = 11?
Yes
No
Add 1 to Y3 set M3 to 0
Clear screen
Print age in years and months
Do another one?
Yes
C1
No
End
Round off and print age

Step 4
Write the code

Instruction	Explanation
10 REM CALCULATE AGE IN MONTHS AND YEARS	
20 Y3=0	Set variables to zero
30 M3=0	
40 D3=0	
50 Y1=0	
60 M1=0	
70 D1=0	
80 REM GET TODAY'S DATE	
90 CALL CLEAR	Clears the screen
100 PRINT "TODAY'S DATE"::::	Print heading literal
110 INPUT "MONTH ":M1	Enter the current month
120 INPUT "DAY ":D1	Enter the current day
130 INPUT "YEAR ":Y1	Enter the current year
140 INPUT "IS DATE CORRECT? " :A$	Answer Y or N
150 IF A$<>"Y" THEN 50	If not correct, re-enter
160 REM GET BIRTH DATE	
170 Y2=0	Set birth date variables to zero
180 M2=0	Clears the screen
190 D2=0	
200 CALL CLEAR	
210 PRINT "BIRTHDATE"::::	Print heading
220 INPUT "MONTH ":M2	Enter birth month
230 INPUT "DAY ":D2	Enter birth day
240 INPUT "YEAR ":Y2	Enter birth year
250 INPUT "IS DATE CORRECT? " :A$	Check to see if birth date is correct; answer Y or N
260 IF A$<>"Y" THEN 170	If not correct, return to line 170 and re-enter date
270 REM CALCULATE THE AGE	
280 IF D2<=D1 THEN 310	If birth day is less than or equal to current day, go to line 310
290 M1=M1-1	Otherwise, subtract one from current month
300 D1=D1+30	Add 30 to current day
310 D3=D1-D2	Compute calculated days = current day − birth day
320 IF M2<=M1 THEN 350	If birth month is less than or equal to current month, go to line 350
330 Y1=Y1-1	Otherwise, subtract one from current year
340 M1=M1+12	Add 12 to current month
350 M3=M1-M2	Compute calculated months = current month − birth month
360 Y3=Y1-Y2	Compute calculated years = current year − birth year
370 IF D3<=15 THEN 420	If calculated days is less than or equal to 15, go to line 420

Note
The variables for computations are set to zero initially to avoid possible errors.

Current and birth dates are entered month, day and year. This order can be changed, if required.

Using this program as a subroutine in placement reports and tests that require age interpretation can save time.

Instruction	Explanation
380 M3 = M3 + 1	Otherwise add 1 to calculated months
390 IF M3 < = 11 THEN 420	If calculated months are less than or equal to 11, go to line 420
400 Y3 = Y3 + 1	Otherwise, add 1 to calculated year
410 M3 = 0	Set calculated months to zero
420 REM PRINT THE AGE	
430 CALL CLEAR	Clears the screen
440 PRINT "AGE =";Y3;"YRS"; M3;"MOS"	Print headings and values for age in years and months
450 PRINT ::	Print two blank lines
460 INPUT "CALCULATE ANOTHER AGE? ":A$	Question to do another calculation
470 IF A$ = "Y" THEN 20	If so, go to line 20; otherwise terminate at line 480
480 END	

Step 5
Run the program and debug it

```
TODAY'S DATE

MONTH 7
DAY 19
YEAR 83
IS DATE CORRECT? Y

BIRTHDATE

MONTH 9
DAY 30
YEAR 75
IS DATE CORRECT? Y

AGE = 7 YRS 10 MOS

CALCULATE ANOTHER AGE? N

** DONE **
```

Example five:
A gradebook program

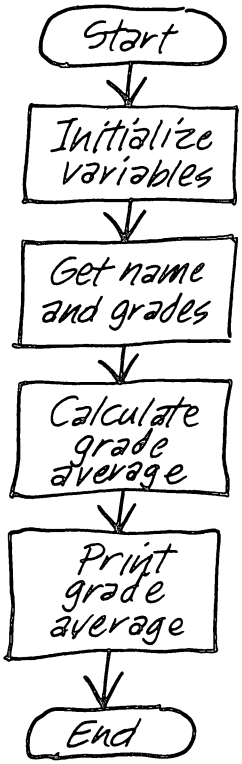
Step 1
Identify the problem
Permit input of five grades in letter form. Input student's name and five grades. Check that an input grade is either an A, B, C, D, or F. If not, reject the entry and allow re-entry. Print student's name, the five input letter grades, and the grade point average based on A = 4.0.

Step 2
Outline the solution

Inputs
1. Get student name.
2. Get letter grades.

Process
1. Dimension the array.
2. Clear the screen.
3. Convert letter grades to points and compute the average.

Outputs
1. Print the student's name, letter grades and grade point average.
2. Print "stop" message.



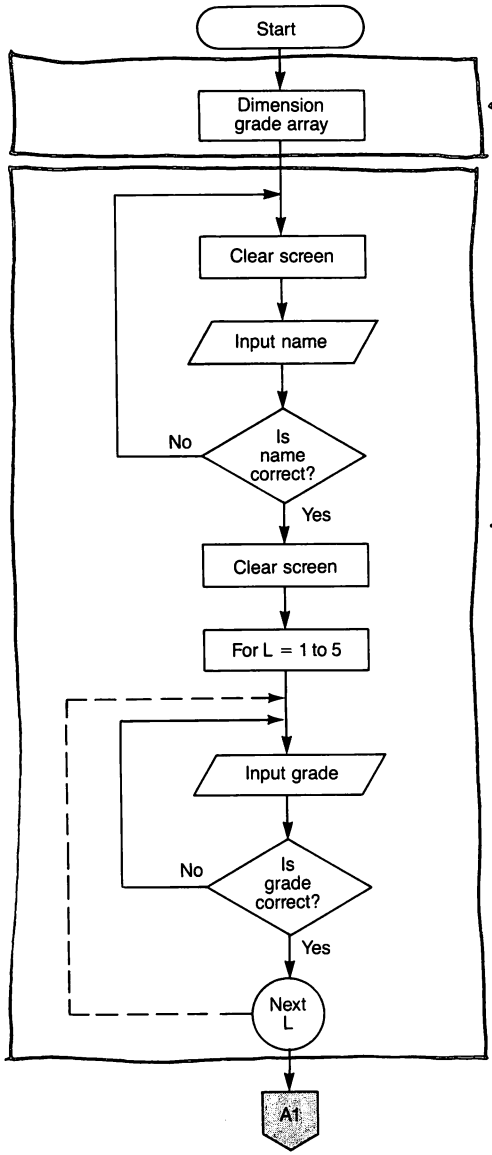
Step 3
Prepare a flowchart

Assign variable names

Inputs
A$ = Answer
NAME$ = Student name
GRADE$( ) = Array for grades

Process
L = Loop index
T = Total score accumulator

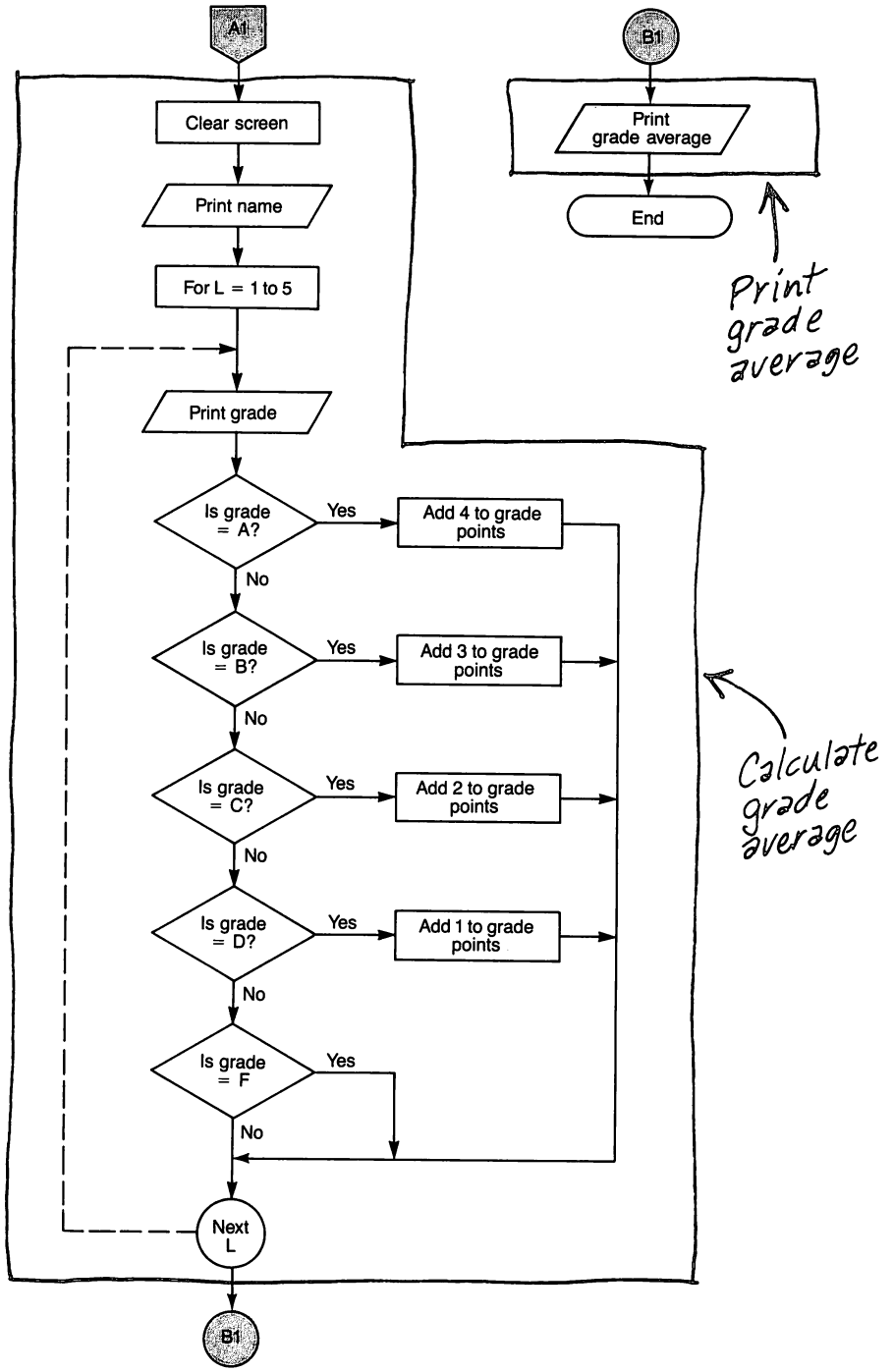
Outputs
NAME$
GRADE$(L)
T/5 = Average grade



Initialize variables

Get name and grades

Flowchart continued on next page


A1
Clear screen
Print name
For L = 1 to 5
Print grade
Is grade = A?
Yes
Add 4 to grade points
No
Is grade = B?
Yes
Add 3 to grade points
No
Is grade = C?
Yes
Add 2 to grade points
No
Is grade = D?
Yes
Add 1 to grade points
No
Is grade = F
Yes
No
Next L
B1
B1
Print grade average
End
Print grade average
Calculate grade average

Step 4
Write the code

Instruction	Explanation
10 REM GRADE BOOK PROGRAM	
20 DIM GRADE$(5)	Dimension 5 element array for alphabetic grades
30 CALL CLEAR	Clears the screen
40 INPUT "NAME ":NAME$	Print header for NAME and input student's name for variable NAME$
50 INPUT "IS NAME CORRECT? " :A$	Print question and enter Y or N
60 IF A$ < "Y" THEN 30	If name is not correct, go to line 30 and re-enter
70 CALL CLEAR	Clears the screen
80 FOR L = 1 TO 5	Begin grade input loop
90 PRINT: "GRADE";L;	Print GRADE header and number of grade being entered (from 1 to 5)
100 INPUT GRADE$(L)	Enter letter grade
110 IF GRADE$(L) = "A" THEN 170	Check for legal grade; if legal, go to line 170
120 IF GRADE$(L) = "B" THEN 170 130 IF GRADE$(L) = "C" THEN 170 140 IF GRADE$(L) = "D" THEN 170 150 IF GRADE$(L) = "F" THEN 170 160 GOTO 90	For illegal grade, go to line 90 and re-enter
170 NEXT L	End grade input loop
180 CALL CLEAR	Clears the screen
190 PRINT NAME$	Print student's name
200 FOR L = 1 TO 5	
210 PRINT:"GRADE";L;GRADE$ (L)	Print letter grade of each element
220 IF GRADE$(L) = "A" THEN 320	If grade is A go to line 320
230 IF GRADE$(L) = "B" THEN 340	If grade is B go to line 340
240 IF GRADE$(L) = "C" THEN 360	If grade is C go to line 360
250 IF GRADE$(L) = "D" THEN 380	If grade is D go to line 380
260 IF GRADE$(L) = "F" THEN 270	If grade is an F go to line 270 (no points added)
270 NEXT L	End grade print loop
280 PRINT ::"GRADE AVERAGE = " ;T/5	Print the grade average in grade points on a 4.0 scale
290 PRINT: "PRESS ENTER TO STOP";	Print ending message
300 INPUT A$	Press ENTER key to terminate program
310 END	
320 T = T + 4	Add 4 points for an A
330 GOTO 270	Go to end of loop
340 T = T + 3	Add 3 points for a B
350 GOTO 270	Go to end of loop
360 T = T + 2	Add 2 points for a C
370 GOTO 270	Go to end of loop
380 T = T + 1	Add 1 point for a D
390 GOTO 270	Go to end of loop

The input section, the calculation section, and report section could be done with separate subroutines. This would allow the input subroutine to be written so that errors in data entry could be corrected without affecting scores.

Grades could be entered one at a time by assignment for different students rather than several grades for one student.

Because each letter grade has a different point value, execution must go to the appropriate line to add the assigned number of points to the grade point accumulator.

Line 260 can be deleted without consequence; however, it was left in for clarity.

The input data and the output results could be stored in a data base. The report routine then could retrieve the data upon request to print the data.

Step 5
Run the program and debug it

```
NAME JOHN DOE
IS NAME CORRECT? Y

GRADE 1 ? A
GRADE 2 ? B
GRADE 3 ? B
GRADE 4 ? C
GRADE 5 ? C

JOHN DOE
GRADE 1 A
GRADE 2 B
GRADE 3 B
GRADE 4 C
GRADE 5 C

GRADE AVERAGE = 2.8
PRESS ENTER TO STOP?
** DONE **
```

ABS function

The format for this function is ABS(numeric expression). The ABSolute value function gives the absolute value of the value obtained when the numeric expression is evaluated. The effect of the ABS function is to remove the sign of a number; thus, ABS(+3)=3 and ABS(−3)=3.

Example six:
Solving an equation

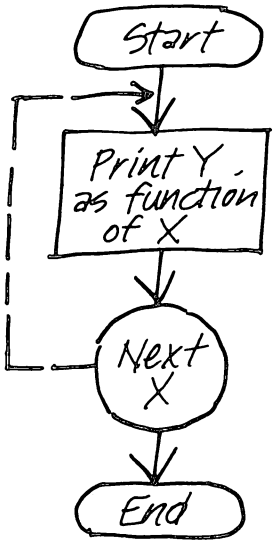
Step 1
Identify the problem
Write a program to print solutions to the equation Y = ABS(X) for different values of X.

Step 2
Outline the solution

Inputs
None

Process
Compute ABS(X) for different values of X and store absolute value in Y.

Outputs
Print X and Y.



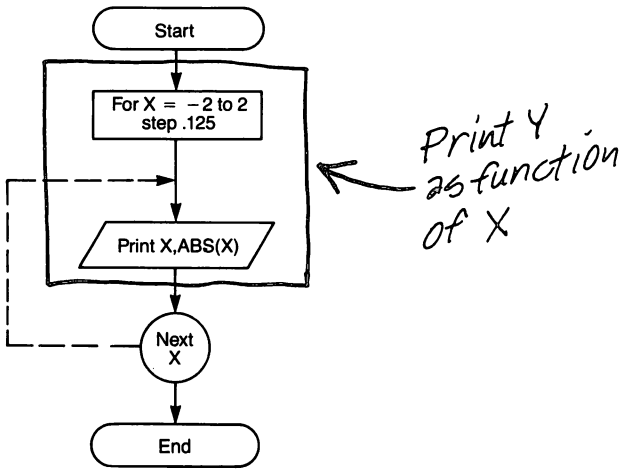
Step 3
Prepare a flowchart

Assign variable names

Inputs
None

Process
Y = ABS(X)

Outputs
X
ABS(X)



Step 4
Write the code

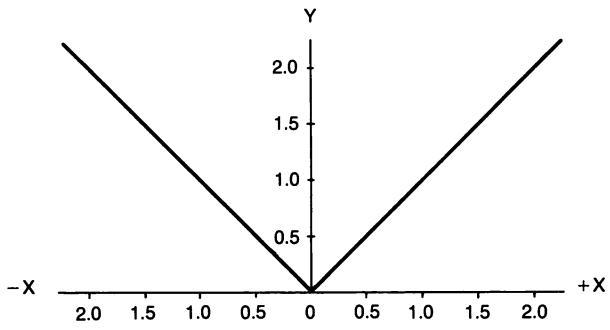
Instruction	Explanation
10 REM Y IS THE FUNCTION OF X	
20 PRINT " X", " Y"::	Prints column headers and skips a line
30 FOR X = −2 TO 2 STEP .125	Begin FOR-NEXT loop from −2 to 2 with a step increment of 0.125
40 PRINT X, ABS (X)	Print the value of X and Y (the absolute of X)
50 NEXT X	Loop to line 30 until value exceeds 2, then go to line 60
60 END	

Step 5
Run the program and debug it

X	Y
−2	2
−1.875	1.875
−1.75	1.75
−1.625	1.625
−1.5	1.5
−1.375	1.375
−1.25	1.25
−1.125	1.125
−1	1
−.875	.875
−.75	.75
−.625	.625
−.5	.5
−.375	.375
−.25	.25
−.125	.125
0	0
.125	.125
.25	.25
.375	.375
.5	.5
.625	.625
.75	.75
.875	.875
1	1
1.125	1.125
1.25	1.25
1.375	1.375
1.5	1.5
1.625	1.625
1.75	1.75
1.875	1.875
2	2

** DONE **

The graph was prepared by hand, but the computer can be programmed to present the graph on the screen, or output the values to a printer/plotter which will plot the graph on paper.



Introduction

Computer assisted instruction

Drill programs are an effective use of the computer. The computer is patient and consistent.

- Drill
- Concepts
- Eye-hand coordination
- Simulation

Computer games can improve eye-hand coordination and self-confidence.

Computer literacy

Children are learning about computers and programming today so they can function in the computerized world of tomorrow.

Administrative applications

The computer can help the busy classroom teacher and administrative staff with records and reports.

- Grade records
- Special education applications
- Word processing
- Attendance records

Educational program examples

Math drill programs can help a student master the basics.

Example one:	Math drill of addition facts.

RES command

Shows how to renumber a program with the RES command.

Example one:	Renumbered
Example two:	Math drill of addition, subtraction and multiplication facts
Example three:	Multiple choice tests
Example four:	Compute the chronological age of a person
Example five:	A gradebook program

Give the computer the current date and a birthdate and it will print the person's age in years and months.

ABS function

The ABS function is demonstrated in an equation.

Example six:	Solving an equation

Summary map

2 Applications for information and resource management

Checkbook records

Time management

Calendar records

Personal record keeping

Equation computation

Financial Planning

Health record keeping

Example two

Example three

4 Application programs

Example one:	Calculate calories
Example two:	Checkbook accounting
Example three:	Compute amount of paint to cover walls of a room
Example four:	Version two of paint coverage program
Example five:	Recipe program

Example five

Creating Information Management Programs

Applications for information and resource management

Computers are useful to help manage family information and resources to save time and/or money. Some management applications in the home for the home computer are:

1. Keeping checkbook records
2. Managing time
3. Remembering important dates
4. Keeping personal records
5. Using formulas
6. Planning finances
7. Keeping health care records

Checkbook records

The computer can keep your checkbook balanced.

Most households use one or more checkbooks. Each month the bank's computers generate a report of the account and send it to the account holder. It is the responsibility of the account holder to be sure all figures are correct. This task is dreaded by many people, and is even frightening to some. Having a personal checkbook computer program can ease the pain of dealing with this important function in household management.

Time management

Good management of time involves scheduling and setting priorities. The computer can be a valuable asset for these purposes.

Scheduling events such as Little League games, Boy Scouts, Girl Scouts, music lessons, swimming lessons, birthdays, outings, religious activities, etc. can be tedious. In addition, appointments for hair styling, car repairs, doctors, and dentists and schedules of meetings, social engagements, and charitable activities require keeping a good calendar of events. Even simple daily chores require time management. Some individuals need reminders to take books back to the library, meet a friend for a luncheon appointment, or simply to put the trash out for collection.

Most households maintain busy schedules and a time management computer program may help avoid some of the problems in scheduling. Fewer scheduling problems will result if all of the people involved have access to the information. That way it should be easier to make plans that won't conflict with one another. For example, someone can check to see if a round of golf on Saturday will conflict with a planned family event.

Another important function of time management is assigning an order of priority to each function that needs to be accomplished. This is important in daily life as well as the business world for getting the maximum benefit from your time. By reviewing the time management schedule periodically, the family can manage their time more effectively.

Calendar records

Don't forget that special person's birthday!

Remembering important dates is another application for a home computer. Some common dates are birthdates, anniversaries, Mother's day, and Father's day. Other important dates to include are graduations, weddings, etc. Your computer calendar program can remind you to get a gift or card for these events. The program can be run daily or weekly so that the important dates for the selected time period will be shown. Past events can be deleted and new ones added.

Personal record keeping

The computer is good at sorting records.

Record keeping is an important information management function for all of us. Records can be kept on everything from taxes to Christmas card mailing lists. Sometimes people have so much information in their collection and it is so disorganized that they don't know all that is in the collection. Examples of personal records include:

1. Expenditures for items that are tax deductible
2. Phonograph record and/or audio tape collection
3. Books in personal library
4. Recipes
5. Automobile expenses
6. Utility consumption
7. Stamp collection or other hobby inventory
8. Photo album data
9. List of household appliances with serial numbers
10. List of jewelry and other valuable items
11. Insurance policies

The list is limited only by your imagination and your particular needs. Even though it takes time to initially enter these records into a computer storage file such as a cassette or diskette, the time will be more than recovered in the long run. You may even save some money or be able to collect some money that you couldn't have otherwise.

Equation computation

Number crunching is the computer's specialty.

There are times when certain tasks require the use of mathematical equations. Since these equations are infrequently used or complicated, some of us may have to spend a lot of time finding the right equation or working out the solution. For example, you may wish to calculate how many yards of material are needed to make drapes. This, in turn, may require the calculation of the number of pleats that will be in the drapes. For another example, you may want to build a stairway and need to calculate the dimensions of the riser and tread. If you have a computer program that contains the necessary equations, all you have to do is enter the data into the computer as requested and the computer will make the calculations and provide the answers in seconds. Minutes or hours of drudgery can be avoided and accurate answers can be obtained quickly and easily.

Another example of using equations is in the conversion of numbers from the English system of measurement to the metric system or vice versa. These conversions may be required to select a tool for a job or to use cookbook recipes. Most Americans have not become comfortable with doing these conversions and a program to do the conversions can make life easier.

Using equations can help save money by allowing you to purchase only the material necessary to do the job. An example is to determine how much paint is required to paint a room. A program to do this will be presented later in this chapter.

Financial planning

Loan calculations and budgets are easier with the computer.

Another important area of family information and resource management is financial planning. This is especially needed with the high prices and inflation of today. The average household needs to know how much money it has to spend and how to spend it wisely. A good way to accomplish this is to set up a budget.

A budget can involve long range objectives as well as short range objectives. Long range objectives could include a college education for a family member, a new home, or retirement income. Short range objectives might include purchase of appliances or furniture, a vacation, or a night out on the town. A budget is a good application for a home computer. A good budget program can be a definite asset to your financial health.

A loan interest and payment calculation program can also be of value in financial planning. Many of us purchase items on credit and a loan program can quickly calculate interest cost, total cost, and payments to allow us to shop for the best financing.

Health record keeping

You can keep your own medical records up-to-date.

Accurate health records are not only important for the doctor, but also should be a part of your personal records. Each person should have a complete health history. Items to include would be illnesses, allergies, checkup dates, etc. Children's records also could include height, weight, and immunization records. It is easy to forget about some event that later may help a doctor diagnose a current illness.

Application programs

Example one:
Calculate calories

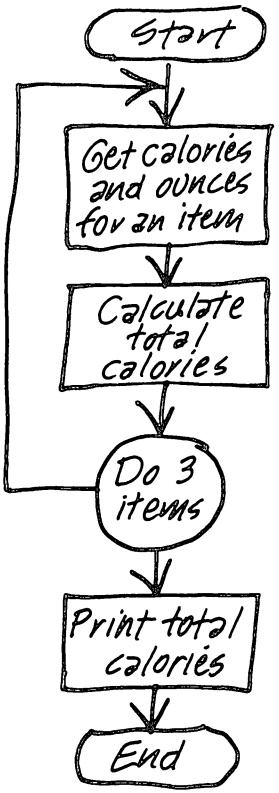
Step 1
Identify the problem
Calculate and print the total number of calories for three items.

Step 2
Outline the solution

Inputs
1. Get calories per ounce.
2. Get number of ounces.

Process
1. Calculate calories for each of three items.
2. Calculate total calories for all items.

Outputs
1. Print total calories for all items.



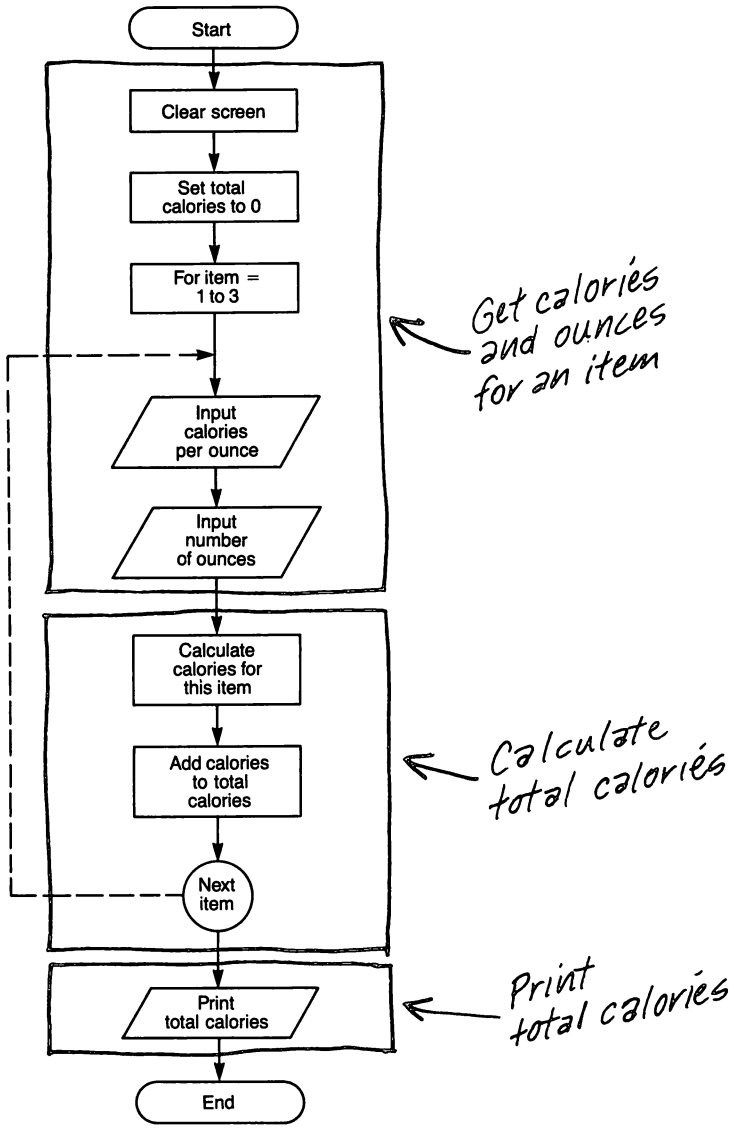
Step 3
Prepare a flowchart

Assign variable names

Inputs
CALORIES = No. of calories per ounce
OUNCES = No. of ounces of an item

Process
TCAL = No. of calories in an item
TAMT = Total calories for all items
ITEM = Loop counter for three items

Outputs
TAMT = Total calories for all items



Step 4
Write the code

Instruction	Explanation
100 REM CALORIE COUNTER	
110 CALL CLEAR	
120 TAMT = 0	Set initial total to zero
130 FOR ITEM = 1 TO 3	Begin loop to calculate total calories
140 INPUT "CALORIES PER OUNCE? ":CALORIES	Get number of calories for this item
150 INPUT "NO. OUNCES? ": OUNCES	Get number of ounces of this item
160 TCAL = CALORIES*OUNCES	Calculate total calories for this item
170 TAMT = TAMT + TCAL	Add this item's calories to total
180 NEXT ITEM	Loop until 3 items have been entered
190 PRINT :"TOTAL CALORIES = " ;TAMT	Print total calories
200 END	

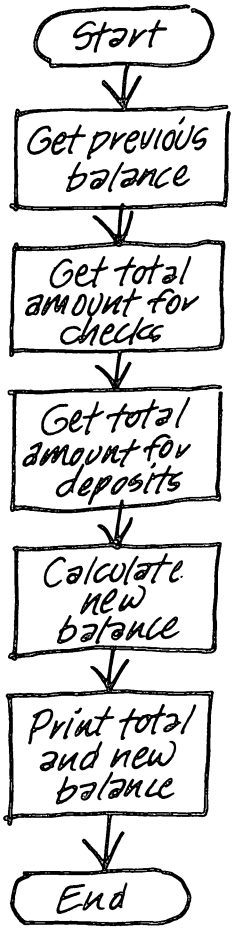
The number of items to be allowed could be made variable and dependent upon an INPUT statement.

Step 5
Run the program and debug it

```
CALORIES PER OUNCE? 100
NO. OUNCES? 3
CALORIES PER OUNCE? 200
NO. OUNCES? 2
CALORIES PER OUNCE? 100
NO. OUNCES? 4

TOTAL CALORIES = 1100

** DONE **
```

Example two:
Checkbook accounting

This program is a simple version of a checkbook balancing program which can be used to check the bank statement against one's personal checking record. Only the essentials are included in the sample program.



Step 1
Identify the problem
Write a computer program that will accept inputs of beginning balance, checks and deposits, then calculate the ending balance. Print out totals and ending balance.

Step 2
Outline the solution

Inputs
1. Get previous balance.
2. Get amount for checks.
   (A zero entry indicates end of entries.)
3. Get amount for deposits.
   (A zero entry indicates end of entries.)
4. Get responses to questions.

Process
1. Count number of checks.
2. Calculate total amount of checks.
3. Count number of deposits.
4. Calculate total amount of deposits.
5. Calculate new balance.

Outputs
1. Print previous balance, checks, deposits and new balance.

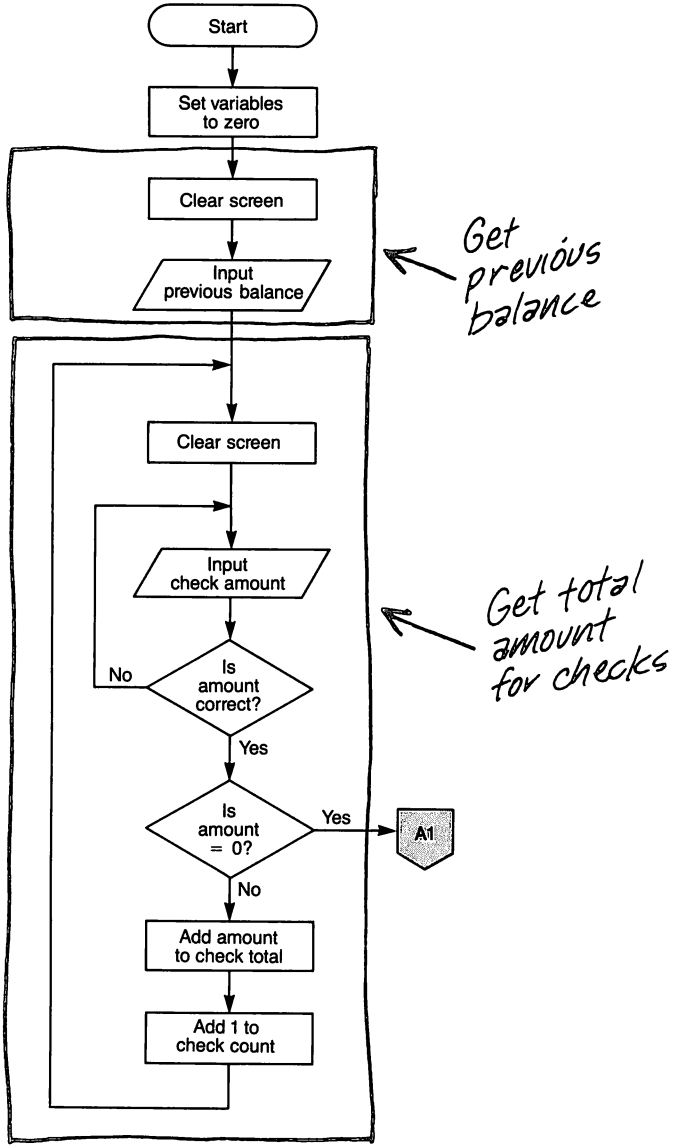
Step 3
Prepare a flowchart

Assign variable names

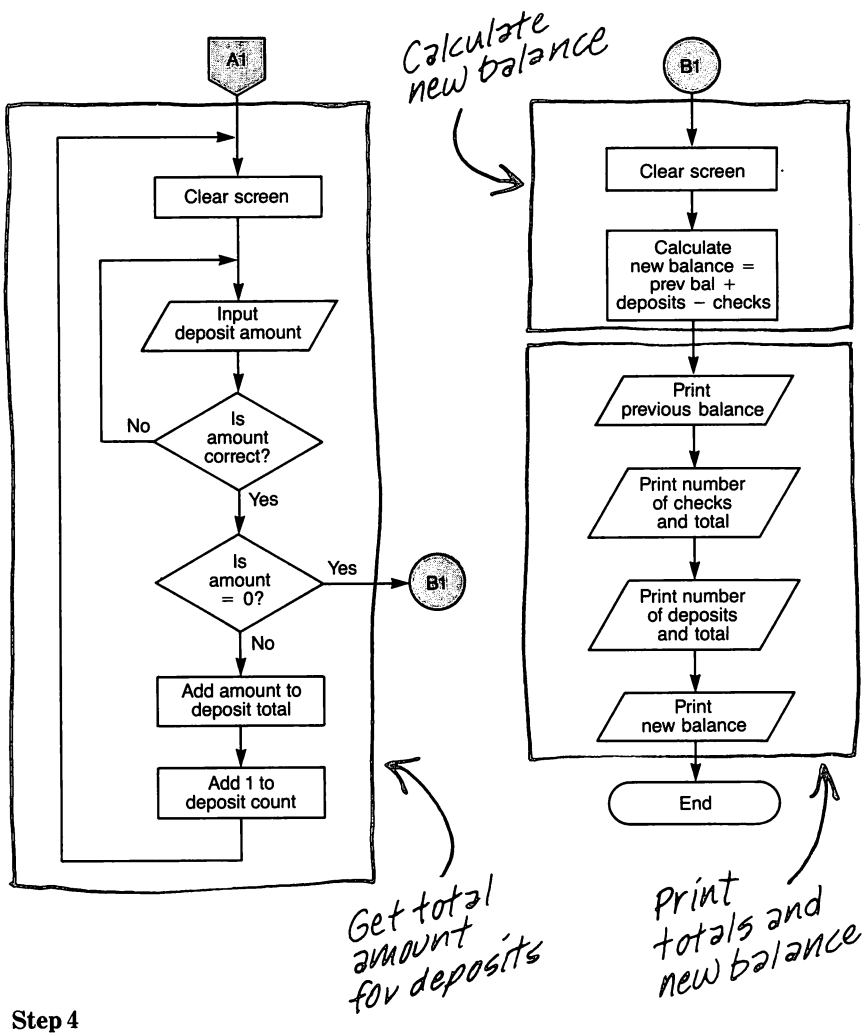
Inputs
PRVBAL = Previous balance
CAMT = Amount of a check
DAMT = Amount of a deposit
A$ = Response to questions

Process
TCHK = Total of all checks
CHCOUNT = Number of checks
TDEP = Total of all deposits
DCOUNT = Number of deposits
NEWBAL = New balance

Outputs
PRVBAL = Previous balance
CHCOUNT = Number of checks
TCHK = Total of all checks
DCOUNT = Number of deposits
TDEP = Total of all deposits
NEWBAL = New balance



Flowchart continued on next page



Step 4
Write the code

Instruction	Explanation
100 REM CHECKBOOK BALANCER	
110 REM ZERO THE VARIABLES	
120 TCHK = 0	Set variables to zero
130 TDEP = 0	
140 CHCOUNT = 0	
150 DCOUNT = 0	
160 REM CHECKS ROUTINE	
170 CALL CLEAR	
180 INPUT "PREVIOUS BALANCE " :PRVBAL	Print prompt and enter beginning balance
190 CALL CLEAR	

A dimensioned array could be set up so you could enter check number, date, payee, and amount for each check. These could be saved on cassette or diskette as a permanent record.

Instruction	Explanation
200 INPUT "CHECK AMOUNT ":CAMT	Print prompt and enter amount of check
210 INPUT "IS CHECK AMOUNT CORRECT? ":A$	Print prompt and enter answer, Y or N
220 IF A$ = "N" THEN 200	If amount is not correct, re-enter
230 IF CAMT = 0 THEN 300	If amount of check = 0, it indicates all checks have been entered so go to line 300
240 TCHK = TCHK + CAMT	Add amount of check to total amount of checks
250 CHCOUNT = CHCOUNT + 1	Add 1 to number of checks written
260 GOTO 190	Go to line 190 for next check
300 REM DEPOSITS ROUTINE	
310 CALL CLEAR	
320 INPUT "DEPOSIT AMOUNT " :DAMT	Print prompt and enter deposit amount
330 INPUT "IS DEPOSIT CORRECT? ":A$	Print prompt and answer Y or N
340 IF A$ = "N" THEN 320	If deposit is not correct, re-enter
350 IF DAMT = 0 THEN 400	If deposit = 0, it indicates all deposits have been entered so go to line 400
360 TDEP = TDEP + DAMT	Add deposit amount to total deposit amount
370 DCOUNT = DCOUNT + 1	Add 1 to number of deposits
380 GOTO 310	Go to line 310 for next deposit
400 REM TOTALS ROUTINE	
410 CALL CLEAR	
420 NEWBAL = PRVBAL + TDEP - TCHK	Compute total balance
430 PRINT "PREVIOUS BALANCE "; PRVBAL	Print beginning balance
440 PRINT:"NO. CHECKS", "TOTAL AMOUNT"	
450 PRINT CHCOUNT, TCHK	Print total number of checks and total amount of checks
460 PRINT:"NO. DEPOSITS", "TOTAL AMOUNT"	
470 PRINT DCOUNT, TDEP	Print number of deposits and total deposit amount
480 PRINT:"NEW BALANCE = " ;NEWBAL	Print new balance
490 PRINT	
500 INPUT"PRESS ENTER TO QUIT":A$	Wait for user to press ENTER key to end program
510 END	

Step 5
Run the program and debug it

```
PREVIOUS BALANCE 800
```

```
CHECK AMOUNT 100
IS CHECK AMOUNT CORRECT? Y
```

```
CHECK AMOUNT 50
IS CHECK AMOUNT CORRECT? N
CHECK AMOUNT 100
IS CHECK AMOUNT CORRECT? Y
```

```
CHECK AMOUNT 0
IS CHECK AMOUNT CORRECT? Y
```

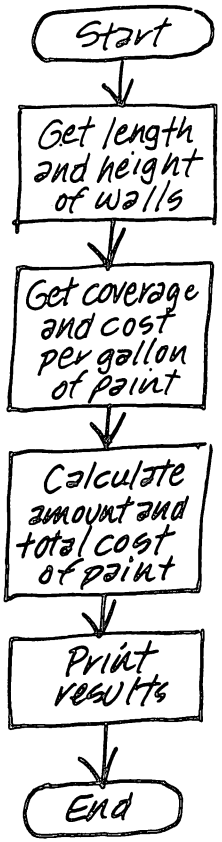
```
DEPOSIT AMOUNT 400
IS DEPOSIT CORRECT? Y
```

```
DEPOSIT AMOUNT 0
IS DEPOSIT CORRECT? Y
```

```
PREVIOUS BALANCE 800

NO. CHECKS      TOTAL AMOUNT
 2                200

NO. DEPOSITS    TOTAL AMOUNT
 1                400

NEW BALANCE = 1000

PRESS ENTER TO QUIT

** DONE **
```

Example three:
Compute amount of paint needed to cover walls of a room

The third program is an example of using an equation. In this case, a program is written to calculate how much paint is needed to cover the walls of a room.



Step 1
Identify the problem
Write a program that will calculate the amount and cost of paint required to cover the walls of a room. Print the total square feet to be painted, total gallons required, and total cost of paint.

Step 2
Outline the solution

Inputs
1. Get the length of each wall.
2. Get the height of all walls.
3. Get the number of square feet each gallon of paint will cover.
4. Get the cost per gallon of paint.
5. Get responses to questions.

Process
1. Calculate total square feet of all walls.
2. Calculate number of gallons required to paint all walls.
3. Calculate total cost of paint.

Outputs
1. Print total square footage of walls.
2. Print number of gallons required.
3. Print total cost of paint.

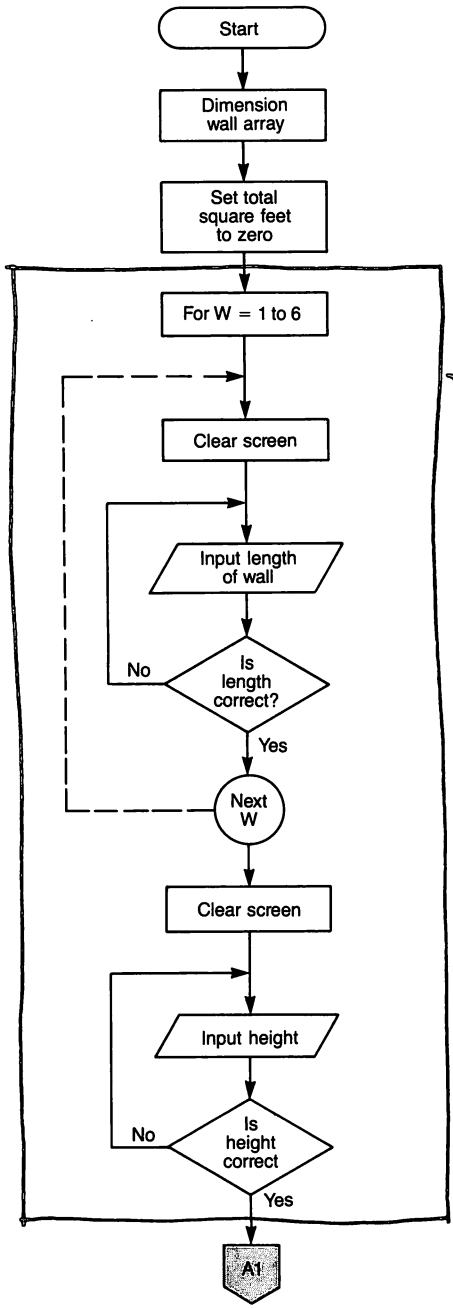
Step 3
Prepare a flowchart

Assign variable names

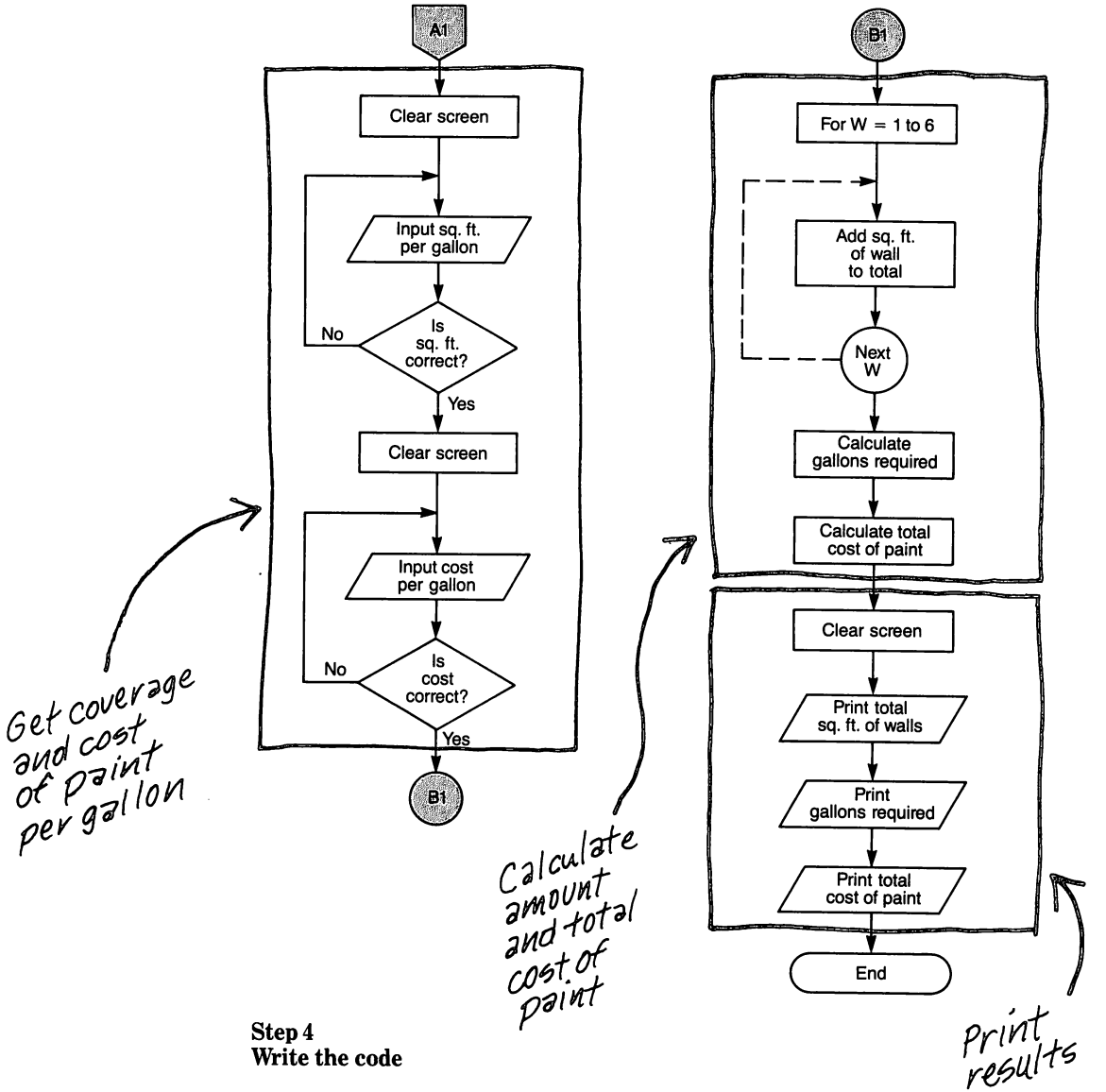
Inputs
WALL( )= Length of each wall to be painted
HEIGHT = Height of all walls
SQFT = Square feet of coverage for a gallon of paint
CST = Cost of a gallon of paint
A$ = Response to questions

Process
TOTSQFT = Total square feet of all walls
GALLONS = Number of gallons required
TCST = Total cost of paint
W = Loop counter for number of walls

Outputs
TOTSQFT = Total square feet of all walls
GALLONS = Number of gallons required
TCST = Total cost of paint



Flowchart continued on next page



Step 4
Write the code

The program could also be used to determine how much glue is needed to hang wallpaper.

Instruction	Explanation
100 REM PAINT PROGRAM (VERSION 1)	
110 REM CALCULATES AMOUNT OF PAINT	
120 REM REQUIRED FOR UP TO SIX WALLS	
130 REM AND THE APPROXIMATE COST	
140 REM FOR THE PAINT SELECTED	
150 REM DIM WALL & SET VARIABLES TO ZERO	

	Instruction	Explanation
The number of walls could be made a variable and input at the beginning.	160 DIM WALL(6)	Dimension array for 6 walls
	170 TOTSQFT = 0	
	180 REM INPUT WALL LENGTHS	
	190 FOR W = 1 TO 6	Begin FOR-NEXT loop to enter length for each wall
	200 CALL CLEAR	
	210 PRINT "LENGTH OF WALL";W;	Print prompt for each wall in turn
	220 INPUT WALL(W)	Enter length of wall for each wall in turn
	230 INPUT "IS LENGTH CORRECT ? ": A$	Verify accuracy of length
	240 IF A$ = "N" THEN 210	If not correct, re-enter length of wall
	250 NEXT W	Loop until all 6 lengths are entered
	260 REM INPUT HEIGHT OF WALLS	
	270 CALL CLEAR	
	280 PRINT "HEIGHT OF WALLS";	
The height of each wall could be entered individually for irregular ceiling heights.	290 INPUT HEIGHT	Enter height of wall
	300 INPUT "IS HEIGHT CORRECT? " :A$	Verify accuracy of height
	310 IF A$ = "N" THEN 280	If height not correct, re-enter
	320 REM INPUT PAINT COVERAGE AND COST PER GALLON	
	330 CALL CLEAR	
	340 PRINT "NO. SQ. FEET PER GALLON";	
	350 INPUT SQFT	Enter number of square feet per gallon that paint covers
	360 INPUT "IS SQ. FT. FIGURE CORRECT? ":A$	Verify accuracy of square feet
	370 IF A$ = "N" THEN 340	If square feet not correct, re-enter
	380 CALL CLEAR	
	390 INPUT "COST PER GALLON? " :CST	Enter cost per gallon
	400 INPUT "IS COST CORRECT? ":A$	Verify accuracy of cost
	410 IF A$ = "N" THEN 390	If cost not correct, re-enter
	420 REM CALCULATE AND PRINT VALUES	
	430 FOR W = 1 TO 6	Begin loop to calculate total square footage of all walls
	440 TOTSQFT = TOTSQFT + WALL(W) *HEIGHT	
	450 NEXT W	
	460 GALLONS = TOTSQFT/SQFT	Calculate the number of gallons required
	470 GALLONS = INT(GALLONS + .9)	Round up the gallons
	480 TCST = GALLONS*CST	Compute the total cost of the paint
	490 CALL CLEAR	
	500 PRINT "TOTAL SQUARE FOOTAGE" ;TOTSQFT	Print total square footage to be painted
	510 PRINT::"GALLONS", "COST"	
	520 PRINT GALLONS,TCST	Print amount of paint required and total cost of the paint
	530 PRINT::	
	540 INPUT "PRESS ENTER TO END":A$	Wait for user to press ENTER to quit
	550 END	

Step 5
Run the program and debug it

```
LENGTH OF WALL 1 ? 10
IS LENGTH CORRECT? Y

LENGTH OF WALL 2 ? 10
IS LENGTH CORRECT? Y

LENGTH OF WALL 3 ? 8
IS LENGTH CORRECT? Y

LENGTH OF WALL 4 ? 0
IS LENGTH CORRECT? Y

LENGTH OF WALL 5 ? 0
IS LENGTH CORRECT? Y

LENGTH OF WALL 6 ? 0
IS LENGTH CORRECT? Y

HEIGHT OF WALLS? 8
IS HEIGHT CORRECT? Y

NO. SQ. FEET PER GALLON? 100
IS SQ. FT. FIGURE CORRECT? Y

COST PER GALLON? 10.00
IS COST CORRECT? Y

TOTAL SQUARE FOOTAGE 224

GALLONS                 COST
  3                       30

PRESS ENTER TO END

** DONE **
```

Example four:
Version two of paint coverage program

The following code is the same program rewritten to permit entry of the height of each wall and to permit a variable number of walls.

Step 4
Write the code

	Instruction	Explanation
	100 REM PAINT PROGRAM (VERSION 2)	
	110 REM CALCULATES THE AMOUNT OF PAINT	
	120 REM REQUIRED FOR THE NUMBER OF	
	130 REM WALLS DESIRED AND THE APPROXIMATE	
	140 REM COST FOR THE PAINT SELECTED	
In this program, the number of walls is variable.	150 REM ENTER NUMBER OF WALLS TO BE PAINTED	
	160 INPUT "NO. OF WALLS (1-9) ? ": N	Enter number of walls
	170 IF N>9 THEN 160	Number of walls cannot exceed array dimensions
	180 REM INITIALIZE	
	190 DIM HEIGHT (9)	Dimension arrays for maximum anticipated number of walls
	200 DIM WALL (9)	
	210 TOTSQFT = 0	
	220 REM ENTER WALL HEIGHT AND LENGTH	
	230 REM AND CALCULATE TOTAL AREA	
	240 FOR W = 1 TO N	Begin loop; range determined by number of walls (N)
	250 CALL CLEAR	
	260 PRINT "LENGTH OF WALL";W;	
Length and height of each wall are entered.	270 INPUT WALL (W)	Enter length of wall number W
	280 PRINT "HEIGHT OF WALL";W;	
	290 INPUT HEIGHT(W)	Enter height of wall number W
	300 INPUT "ARE LENGTH AND HEIGHT CORRECT (Y/N)? ": A$	Check data entries Answer Y or N
	310 IF A$ = "N" THEN 260	If not correct, re-enter
	320 TOTSQFT = TOTSQFT + (HEIGHT (W) * WALL (W))	Calculate area of each wall and add to total accumulator
	330 NEXT W	
	340 REM ENTER PAINT COVERAGE AND COST PER GALLON	
	350 CALL CLEAR	
	360 INPUT "NO. SQ. FEET PER GALLON? ":SQFT	Enter paint coverage per gallon
	370 INPUT "COST PER GALLON? " :CST	Enter paint cost per gallon

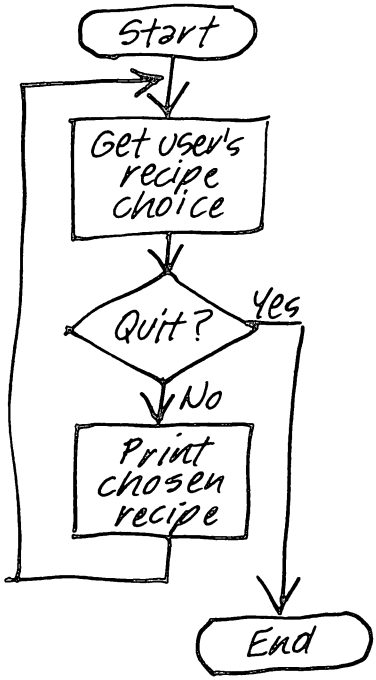
Instruction	Explanation
380 INPUT "ARE SQ. FT. AND COST CORRECT (Y/N)? ":A$	Check data entries; answer Y or N
390 IF A$ = "N" THEN 360	If not correct, re-enter
400 REM CALCULATE GALLONS NEEDED 410 REM AND TOTAL COST 420 GALLONS = INT (TOTSQFT/SQFT + .9)	Divide total wall area by paint coverage; add .9 to round up; use integer function to obtain whole gallons
430 TCST = GALLONS * CST	Total cost equals gallons needed times cost per gallon
440 REM PRINT RESULTS 450 CALL CLEAR 460 PRINT "TOTAL WALL AREA"; TOTSQFT;"SQ. FT."	Print total square footage
470 PRINT::"GALLONS","COST" 480 PRINT GALLONS, TCST	Print number of gallons needed and total cost of paint
490 PRINT:: 500 INPUT "PRESS ENTER TO END":A$	Wait for user to press ENTER to end program
510 END	

Step 5
Run the program and debug it

```
NO. OF WALLS (1-9)? 1

LENGTH OF WALL 1 ? 10
HEIGHT OF WALL 1 ? 8
ARE LENGTH AND HEIGHT
CORRECT (Y/N)? Y

NO. SQ. FEET PER
GALLON? 100
COST PER GALLON? 10.00
ARE SQ. FT. AND COST
CORRECT (Y/N)? Y

TOTAL WALL AREA 80 SQ. FT.

GALLONS          COST
  1               10

PRESS ENTER TO END

** DONE **
```

Example five:
Recipe program

The next program is a simple recipe program using print statements for the recipes.



Step 1
Identify the problem
Write a program that prints the choice of two recipes from a selection menu.

Step 2
Outline the solution

Inputs
1. Get recipe choice.
2. Get response to continue.

Process
1. Decide which recipe was chosen.

Outputs
1. Print recipe according to user's choice.

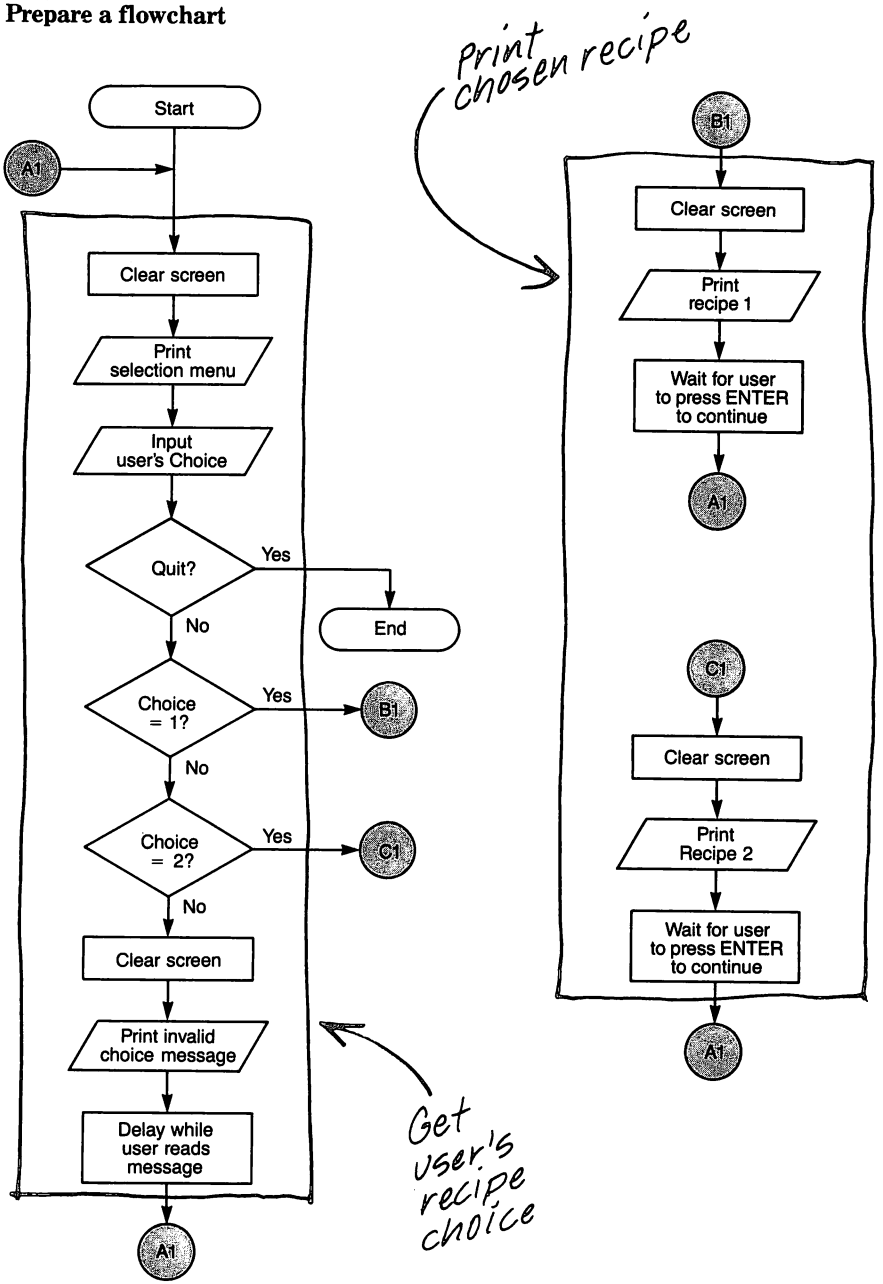
Step 3
Prepare a flowchart

Assign variable names

Inputs
CHOICE = User's recipe choice
A$ = User's response to continue

Process
CHOICE = User's recipe choice
DELAY = Loop counter for delay loop

Outputs
None



Step 4
Write the code

Instruction	Explanation
100 REM RECIPE PROGRAM	
110 REM MAIN DRIVER ROUTINE	
120 CALL CLEAR	
130 PRINT "BREAKFAST RECIPES"	Print selection menu
140 PRINT :"1 EGGS (REGULAR)"	
150 PRINT "2 SCRAMBLED EGGS"	
160 PRINT :"TYPE YOUR CHOICE OR"	
170 PRINT "99 TO QUIT"	
180 PRINT ::	
190 INPUT "YOUR CHOICE? ":CHOICE	Get user's choice
200 IF CHOICE = 99 THEN 999	If CHOICE = 99, then go to end
210 IF CHOICE = 1 THEN 300	Go to line 300 for recipe 1
220 IF CHOICE = 2 THEN 500	Go to line 500 for recipe 2
230 REM INVALID RECIPE CHOICE	
240 CALL CLEAR	
250 PRINT "MAKE ANOTHER CHOICE"	Inform user of invalid choice
260 FOR DELAY = 1 TO 1000	Delay while user reads message
270 NEXT DELAY	
280 GOTO 110	Go back to start of program
300 REM REGULAR EGGS ROUTINE	
310 CALL CLEAR	
320 PRINT "EGGS AS YOU LIKE THEM"	Print recipe 1
330 PRINT :"CRACK EGG ON EDGE OF PAN"	
340 PRINT "OR DISH."	
350 PRINT "PLACE CONTENTS OF EGG"	
360 PRINT "IN HOT GREASED PAN."	
370 PRINT "COOK EGG UNTIL WHITE IS"	
380 PRINT "FIRM BUT NOT BURNING."	
390 PRINT "REMOVE EGG FROM PAN WITH"	
400 PRINT "SPATULA AND PUT ON PLATE."	
410 PRINT :"FOR SUNNY SIDE DOWN EGGS"	
420 PRINT "FLIP EGG OVER WITH"	
430 PRINT "SPATULA"	
440 PRINT "AND COOK FOR ABOUT 30"	
450 PRINT "SECONDS."	
460 PRINT ::	
470 INPUT "PRESS ENTER TO CONTINUE":A$	Wait for user to press ENTER to continue
480 GOTO 110	Go back to start of program

It would be better to have the actual recipes stored as data in an external file rather than contained within the program. Recipes could be added to the file for updating and they could be accessed at any time quickly and easily. Programming schemes could be used to classify the recipes by type such as meats, salads, etc.

Instruction	Explanation
500 REM SCRAMBLED EGGS ROUTINE	
510 CALL CLEAR	
520 PRINT "SCRAMBLED EGGS"	Print recipe 2
530 PRINT :"HEAT 1 TABLESPOON OIL IN"	
540 PRINT "PAN."	
550 PRINT "BREAK EGGS INTO DISH."	
560 PRINT "ADD 1 TABLESPOON MILK TO EGGS."	
570 PRINT "MIX MILK AND EGGS."	
580 PRINT "POUR MIXTURE INTO PAN."	
590 PRINT :"STIR MIXTURE TOGETHER"	
600 PRINT "UNTIL EGGS BECOME FIRM."	
610 PRINT ::	
620 INPUT "PRESS ENTER TO CONTINUE":A$	Wait for user to press ENTER to continue
630 GOTO 110	Go back to start of program
999 END	

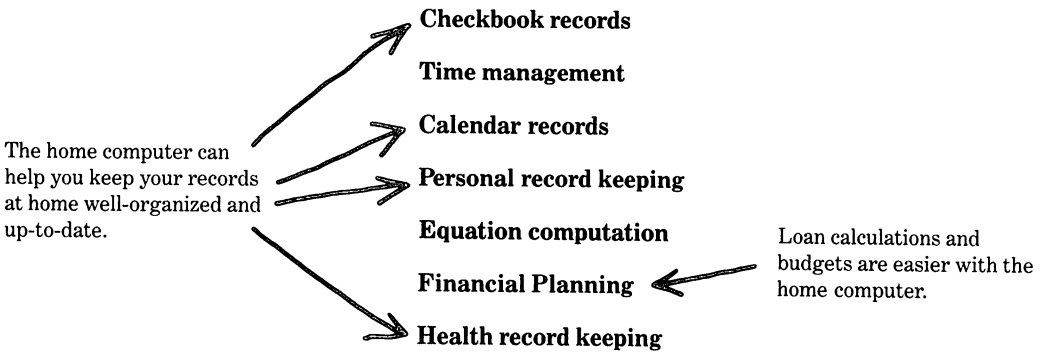
Step 5
Run the program and debug it

```
BREAKFAST RECIPES

1 EGGS (REGULAR)
2 SCRAMBLED EGGS

TYPE YOUR CHOICE OR
99 TO QUIT


YOUR CHOICE? 2
```

```
SCRAMBLED EGGS

HEAT 1 TABLESPOON OIL IN
PAN.
BREAK EGGS INTO DISH.
ADD 1 TABLESPOON MILK TO EGGS.
MIX MILK AND EGGS.
POUR MIXTURE INTO PAN.

STIR MIXTURE TOGETHER
UNTIL EGGS BECOME FIRM.


PRESS ENTER TO CONTINUE
```

```
BREAKFAST RECIPES

1 EGGS (REGULAR)
2 SCRAMBLED EGGS

TYPE YOUR CHOICE OR
99 TO QUIT


YOUR CHOICE? 99

** DONE **
```

Applications for information and resource management



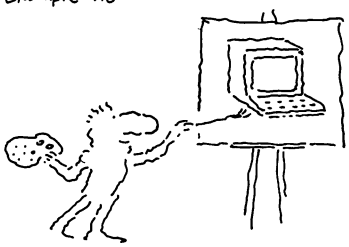
Application programs

The computer can be programmed to help with the common ordinary tasks around the house.

Example one:	Calculate calories
Example two:	Checkbook accounting
Example three:	Compute amount of paint to cover walls of a room
Example four:	Version two of paint coverage program
Example five:	Recipe program

2 **Value of entertainment programs**

Example one



Logical thinking

Spatial awareness

Quick decisions

Simulated conditions

Educational games

3 **Entertainment programs**

Example one: Making a picture

Example two

7 **SIN function**

Example two: Print a word using the sine function

Example three: Simulation of rolling a pair of dice

Example four: Card game

Example five: Quiz program

Example five

30 **Summary map**

Creating Entertainment Programs

Value of entertainment programs

A common conclusion is that game programs are fine for fun and to pass the time, but have no other value. As pointed out in Chapter 1, games can provide significant benefits while being entertaining. Some of these benefits are discussed next.

Logical thinking

Logical thinking and planning skills are gained from playing games

Games such as checkers or chess not only require that rules be followed, but also require strategy because one must be able to plan for several moves ahead. What makes it even more interesting is that the strategy might have to change after each move. The abilities to do logical thinking and planning are significant advantages that can be applied to various occupations and real life situations. Many adults sometimes forget that play is the child's learning ground for the working world of tomorrow. Adults can also benefit when playing such strategy games because the games keep the mind active. For all ages, games can be an escape from the problems and pressures of the real world.

Spatial awareness

Games help children relate to their surroundings.

Eye-hand coordination was mentioned in Chapter 1, but coupled to this is the recognition of different shapes at different angles (spatial awareness). Such activity can help children relate to their surroundings, give them three-dimensional recognition, and help them identify shapes and directions.

Quick decisions

Games improve a person's ability to make quick decisions.

Many games require that visual information be processed quickly and a decision made to make some small movement quickly and accurately. The ability to assess a situation and make quick decisions will aid children as they enter into competitive sports or problem situations in everyday life.

Simulated conditions

Games provide opportunities to explore activities that may not be experienced in real life.

There are many cases in life where some previous experience helps us cope with a new situation. Some computer games allow the player to gain experience in unusual situations by using simulation. With the techniques of simulation, computers create conditions on the screen that appear as real life.

For example, how many people in real life would ever pilot a Boeing 747? Very few. But a computer can be made to simulate the pilot conditions for flying the plane, so that the player, by manipulating computer controls such as a joystick or paddle, actually feels as if he or she is flying the plane. Even kids can pilot the plane. If the plane crashes, no one is hurt and no property is damaged. With the push of a button, the player can try again.

Another example is a trip to the moon. Lunar lander games feature another type of vehicle that even fewer of us will be able to experience in real life. Still other examples place the player in the driver's seat of an Indianapolis racer, or behind a laser gunsight, or inside a tank.

Other simulation games are used to play "what if" type games. Some are just for fun, others present serious type situations. These games are based upon the occurrence of certain conditions. For instance, one game simulates nuclear conditions and what would happen if a nuclear explosion occurred. The player tries different options as solutions in order to find out the consequences of making particular decisions. Other simulation games place the player(s) in adventure trip settings, space voyages, on a pirate ship, trapped by monsters inside the earth, or hunting treasure in the pyramids of the East.

All of these have the advantage of being fun, but they require decisions to be made. They allow one to investigate critical decisions before they happen in real life. The computer age might be accused of causing the kids to grow up too fast, and maybe rightly so. However, if that is the way the world will be, perhaps it is best that the children are prepared for it.

Educational games

In educational games, various math, reading, spelling and language skills are being learned or reinforced as the game is played.

Educational games have been developed so that understanding and learning specific educational concepts are the main objectives of a game. As the game is played, skills in math, reading, spelling and language are learned or reinforced. When learning activities are written in game format, the kids (and adults too) think they're fun while the teacher and parents can enjoy the fact that the child is learning new skills and reinforcing skills already learned. In fact, some children may learn more through the educational games than through traditional teaching methods.

Educational games make learning fun.

Many of these programs interweave a game or gamelike motivational activity within a computer assisted instruction program. For example, animals are fed or space ships are shot down as questions are answered correctly. In more recent games, speech is included so the player is verbally encouraged, reminded, or given the correct answer after several incorrect tries.

Educational games are especially good for children that have experienced failure.

The educational games are especially good for children that have experienced failure in the usual learning environments because the computer game can give them an added incentive to learn. The players compete with themselves and don't have an adult passing judgement as the learning process continues. Errors are easily erased and no one remembers; although scores are maintained and performance against time is an ingredient in many games.

Educational games also are for adults.

Not all of the learning games are for children. Typing, foreign language, computer programming, chess, high-school science, and word processing skills all can be learned or reinforced by adult oriented games.

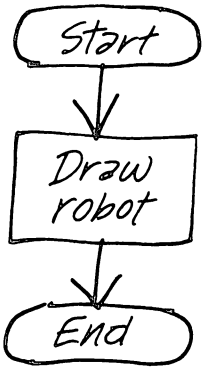
Thus, we see that computer games are valuable not only for fun, but also have many other benefits that may improve educational skills and help prepare people to face a competitive fast-moving real world. Computer games can provide an opportunity for people to explore activities by simulation that they may never experience in real life. Computer games help to motivate, remind and reinforce as new skills are being learned without being domineering and judgmental.

Entertainment programs

Let's develop some entertainment programs in the BASIC language. The first one is quite easy to understand and will demonstrate a simple form of computer art without using graphics. As in previous chapters, we'll show how to develop the program from beginning to end.

Example one:
Making a picture

Step 1
Identify the problem
Write a program to draw a robot figure on the screen using only an alphabetic character. The picture is to remain on the screen until cleared by the user.

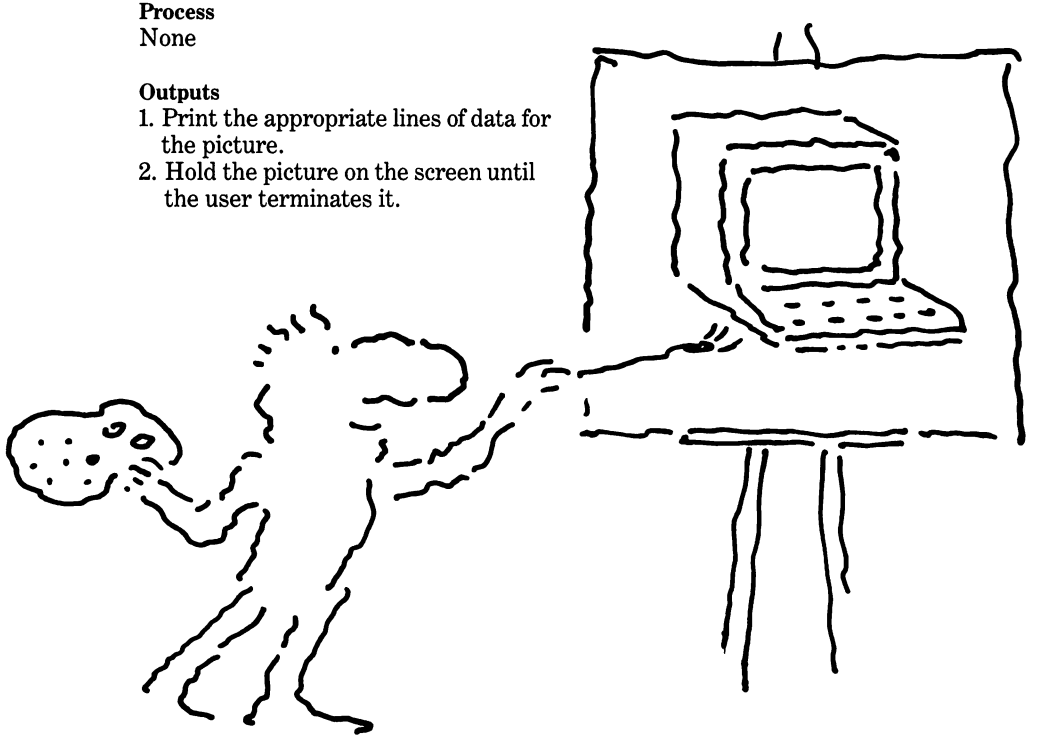


Step 2
Outline the solution
As shown on page 8-5, fill in the squares on a piece of linear graph paper with an alphabetic character to form the figure as you want it to appear on the screen. Each filled-in square will be represented by an alphabetic character on the screen. The maximum size is 28 columns by 24 rows (lines). There are no variables for the program. The program merely prints the chosen alphabetic character to fill in the positions selected in each line. Once the figure is mapped on the graph paper, it will be easy to write the program from it.

Inputs
None

Process
None

Outputs
1. Print the appropriate lines of data for the picture.
2. Hold the picture on the screen until the user terminates it.



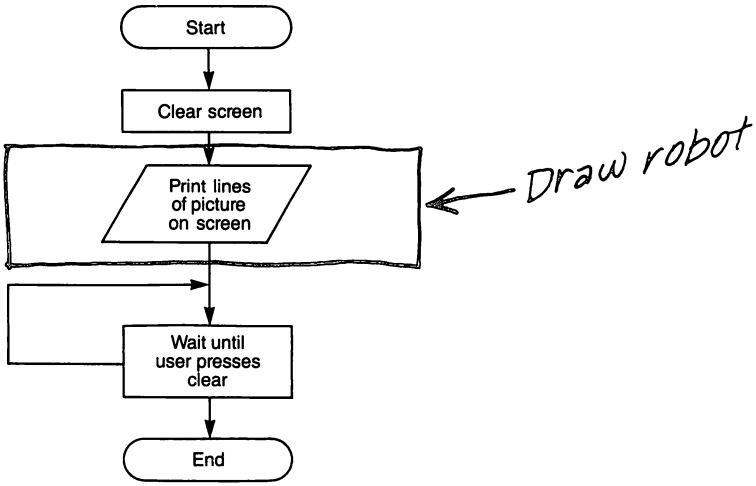
Step 3
Prepare a flowchart

Assign variable names

Inputs
None

Processing
None

Outputs
None



COLUMN

ROW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1				NOTE																								
				9						X	X	X	X	X	X	X	X											
				9						X	X	X	X	X	X	X	X											
				9						X	X		X	X		X	X											
5				8					X	X	X		X	X		X	X	X										
				8					X	X	X	X	X	X	X	X	X	X										
				8					X	X	X	X			X	X	X	X										
				9						X	X	X	X	X	X	X	X											
				9						X	X					X	X											
10				9						X	X	X	X	X	X	X	X											
				11								X	X	X	X													
				12									X	X														
				6			X	X	X	X	X	X	X	X	X	X	X	X	X	X								
				6			X			X	X	X	X	X	X	X	X			X								
15				6			X			X	X	X	X	X	X	X	X			X								
				6			X			X	X	X	X	X	X	X	X			X								
				5		X	X			X	X	X	X	X	X	X	X			X	X							
				5		X	X			X	X	X	X	X	X	X	X			X	X							
				9						X	X	X	X	X	X	X	X											
20				9						X	X	X	X	X	X	X	X											
				9						X	X					X	X											
				9						X	X					X	X											
				9						X	X					X	X											
24				8					X	X	X	X			X	X	X	X										

NOTE: Number of leading blanks (spaces) are listed in column 4 for convenience.

Step 4
Write the code

Use the information from the graph paper sketch that was made in outlining the solution as a guide to write the PRINT statements. The character is positioned in each line by writing the PRINT statement to contain the required number of blank spaces between the opening quotation marks and the first X of each line.

Instruction	Explanation
100 REM ROBOT PICTURE	
110 CALL CLEAR	Clears screen
120 PRINT " XXXXXXXX"	Lines 120-340 are print statements for the picture of the robot
130 PRINT " XXXXXXXX"	
140 PRINT " XX XX XX"	
150 PRINT " XXX XX XXX"	
160 PRINT " XXXXXXXXXX"	
170 PRINT " XXXX XXXX"	
180 PRINT " XXXXXXXX"	
190 PRINT " XX XX"	
200 PRINT " XXXXXXXX"	
210 PRINT " XXXX"	
220 PRINT " XX"	
230 PRINT " XXXXXXXXXXXXX"	
240 PRINT " X XXXXXXXX X"	
250 PRINT " X XXXXXXXX X"	
260 PRINT " X XXXXXXXX X"	
270 PRINT " XX XXXXXXXX XX"	
280 PRINT " XX XXXXXXXX XX"	
290 PRINT " XXXXXXXX"	
300 PRINT " XXXXXXXX"	
310 PRINT " XX XX"	
320 PRINT " XX XX"	
330 PRINT " XX XX"	
340 PRINT " XXXX XXXX"	
350 GOTO 350	This instruction simply keeps going to itself to keep the program running

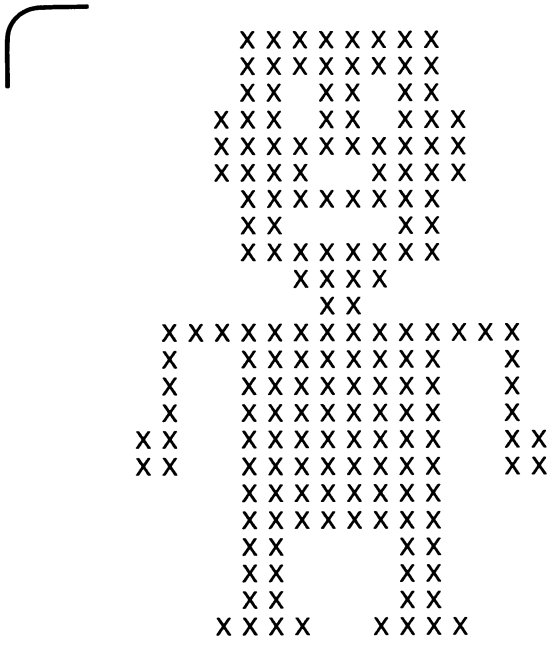
Note
Errors will occur if you leave out quotation marks.

You may want to try characters other than X for different parts of the figure.

You may wish to change the program to move the figure to a different position or change the shape.

Note
You can stop the program by holding down the function key and pressing the 4 key.

Step 5
Run the program and debug it



SIN Function

Most home computers can be used to calculate the solutions to equations in mathematics, physics, chemistry and other sciences. Three important trigonometric functions; the sine, cosine and tangent functions, are built into the computer. These functions are used in calculations involving angles. Most of us are familiar with angles and know that they are often measured in degrees. However, angles also can be measured in another unit called radians. The home computer's built-in functions require that radians be used; therefore, if an angle is given in degrees, it must be converted to radians. This conversion is easy to do and can be done on the home computer. The conversion factor is obtained by dividing π by 180. Since π is approximately equal to 3.1416, the factor is 3.1416/180 which is 0.0174533. To convert degrees to radians, simply multiply degrees by this conversion factor. Of course, this multiplication can be combined with the SIN function. For example, to find the sine of an angle of 40 degrees:

Type SIN(40* .0174533)
Press **ENTER**

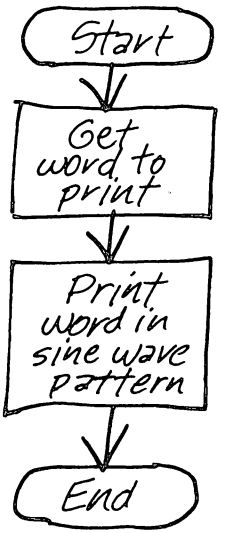
Of course, if radians are already known, no conversion is required. For example, to find the sine of an angle of 0.8 radians:

Type SIN (.8)
Press **ENTER**

The next example program uses the SIN function of the computer.

Example two:
Print a word using the sine function

This program uses the mathematical sine function to determine the print position of a word. Some versions of the game have a predetermined word or words to be printed; however, this program has an INPUT statement to allow any selected word of seven characters or less to be printed.



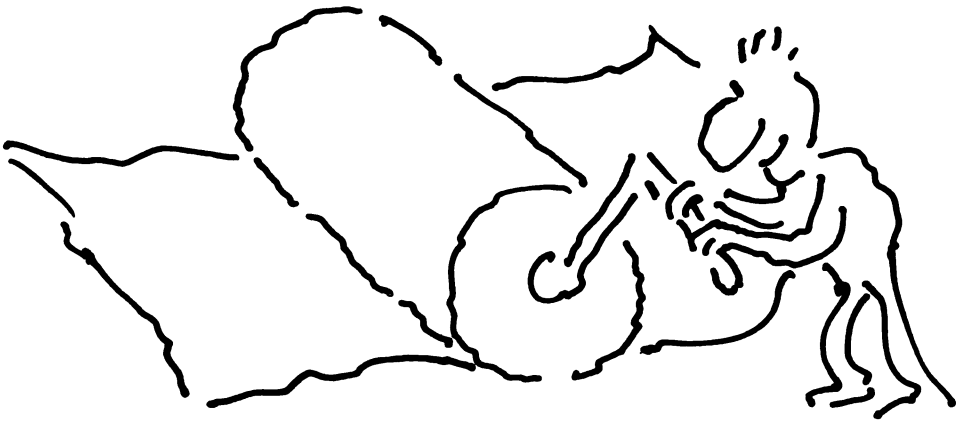
Step 1
Identify the problem
Write a program that will print a word on the screen. The starting position of the word on the screen is determined by the mathematical sine function.

Step 2
Outline the solution
The logic is quite simple. The starting position of a word printed on the screen is to vary so that the pattern of printed words is in the shape of a sine wave. The computer uses the SIN function of the BASIC language which automatically calculates the sine of a number.

Inputs
1. Get the word to print on the screen.

Process
1. Calculate the starting position for the word.
2. Repeat the cycle the specified number of times.

Outputs
1. Print the word at the calculated position.



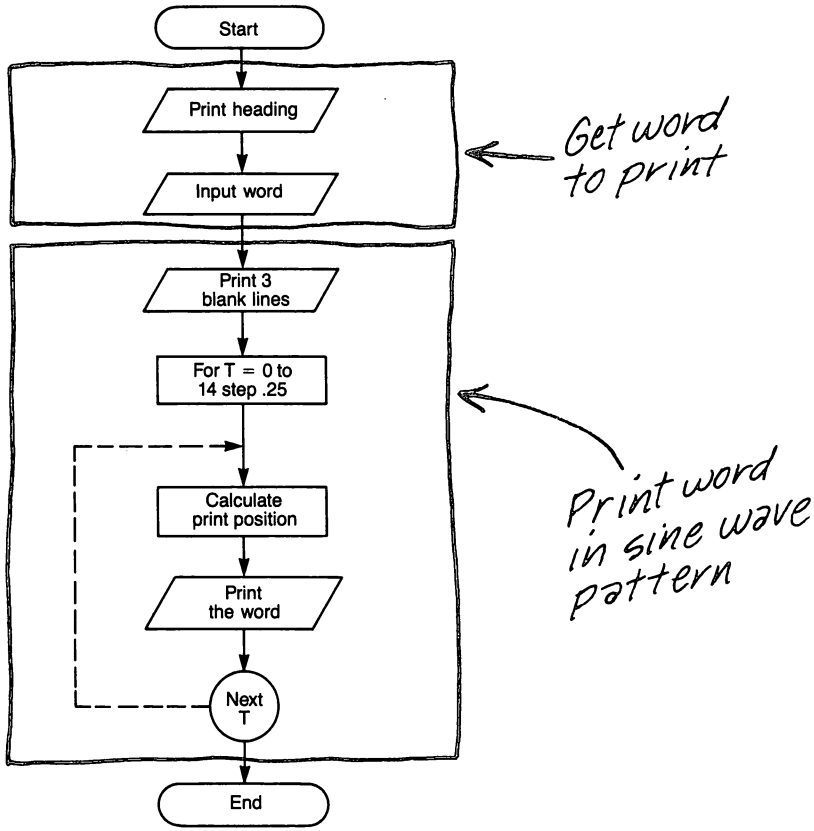
Step 3
Prepare a flowchart

Assign variable names

Inputs
WORD$ = Word to be printed by computer

Process
A = Calculated position to begin printing
T = Control variable for the number of times loop will be executed

Outputs
WORD$ = Word to be printed on the screen



Step 4
Write the code
As simple as the program is, it is powerful in that the same word can be reproduced any number of specified times in a specific pattern on the screen.

Instruction	Explanation
100 REM SINE WAVE PROGRAM	
110 PRINT "SINE WAVE GAME"	Prints title
120 INPUT "WORD (7 OR LESS CHARACTERS) ":WORD$	Prints prompt to request word
130 PRINT :::	Prints 3 blank lines
140 REM START LOOP	
150 FOR T=0 TO 14 STEP .25	Begin FOR-NEXT loop. Each time the loop is executed, .25 is added to the count so the loop is executed 57 times. The loop starts at zero and counts to 14 (i.e., 0, .25, .50, .75, 1.00, 1.25,....)
160 A=INT(12+11*SIN(T))	Uses the sine function to calculate where the beginning of the word will occur; if the end of the word would go beyond column 28, the whole word will be printed on the next line, so don't use more than 7 characters in your word
170 PRINT TAB(A);	Skip to the location calculated in line 160. The semicolon suppresses the line feed so that the word will print at the location specified
180 PRINT WORD$	
190 NEXT T	Loop to line 150 until the count exceeds 14.
200 END	Terminates program

This program is a favorite among beginning computer students. Other versions can be written using different values depending on the computer system.

Try changing the STEP size of .25 in line 150.

Now try changing the values in line 160; first the 11; then the 12.

Computers are very good at calculating mathematical functions.

This program illustrates how repetitious mathematical calculations are a useful application of the computer.

Step 5
Run the program and debug it

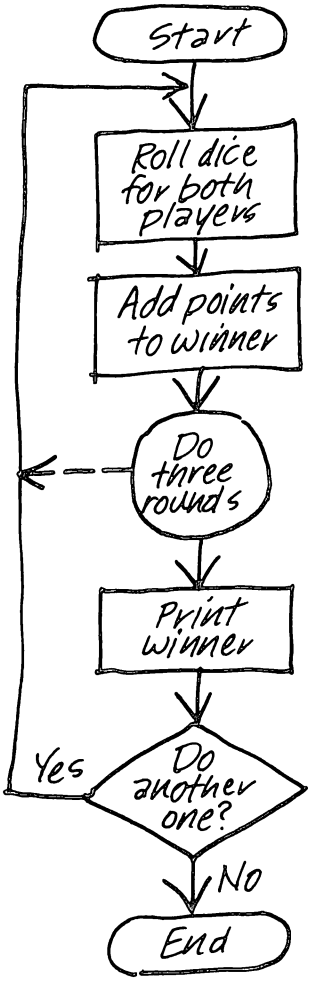
```
SINE WAVE GAME
WORD (7 OR LESS CHARACTERS)
COMPUTE

                COMPUTE
                 COMPUTE
                  COMPUTE
                    COMPUTE
                      COMPUTE
                      COMPUTE
                      COMPUTE
                      COMPUTE
                     COMPUTE
                    COMPUTE
                   COMPUTE
                  COMPUTE
                 COMPUTE
                COMPUTE
               COMPUTE
              COMPUTE
             COMPUTE
             COMPUTE
             COMPUTE
             COMPUTE
              COMPUTE
              COMPUTE
                 COMPUTE
                   COMPUTE
                     COMPUTE
                      COMPUTE
                        COMPUTE
                          COMPUTE
                            COMPUTE
                            COMPUTE
                            COMPUTE
                            COMPUTE
                           COMPUTE
                          COMPUTE
                         COMPUTE
                        COMPUTE
                       COMPUTE
                      COMPUTE
                     etc.
** DONE **
```

Note
The words will scroll off the top of the screen and disappear because the screen can display only 24 lines at one time.

The S shape is called a sine wave.

Example three:
Simulation of rolling a pair of dice

Many games require the use of a die or dice. This program uses the RANDOMIZE command and RND function (discussed in Chapter 4) to simulate rolling a pair of dice.



Step 1
Identify the problem
Write a game program that uses a pair of dice to determine the winner (computer or player) of two out of three rolls.

Step 2
Outline the solution
There are two players - the computer and a person named "player". The game winner is the one who wins two out of three rolls of the dice. The winner of each roll is the one with the largest sum of the dice. The entire game is to be controlled by the computer. Starting is initiated by the player. Play will continue until the player stops it. Development of the program requires one to consider that each die has six sides with values from 1 to 6 and that any side may turn up in a random manner when the die is rolled.

Inputs
1. User's response to play again question.

Process
1. Generate value for dice.
2. Decide who wins each round.
3. Add points to winner's score.
4. Decide whose score is larger.

Outputs
1. Print headings.
2. Print dice values for each round.
3. Print score for each player.
4. Print who won the game.
5. Print play again question.

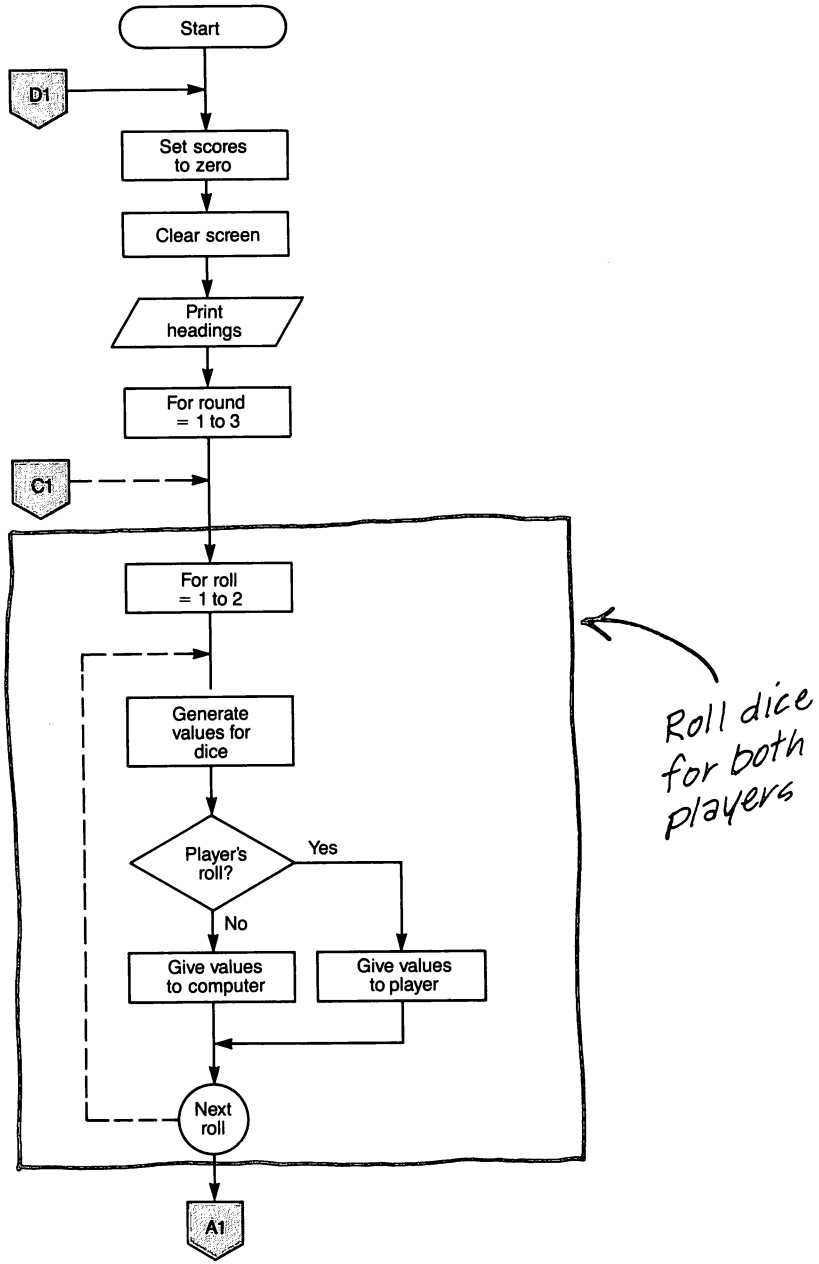
Step 3
Prepare a flowchart

Assign variable names

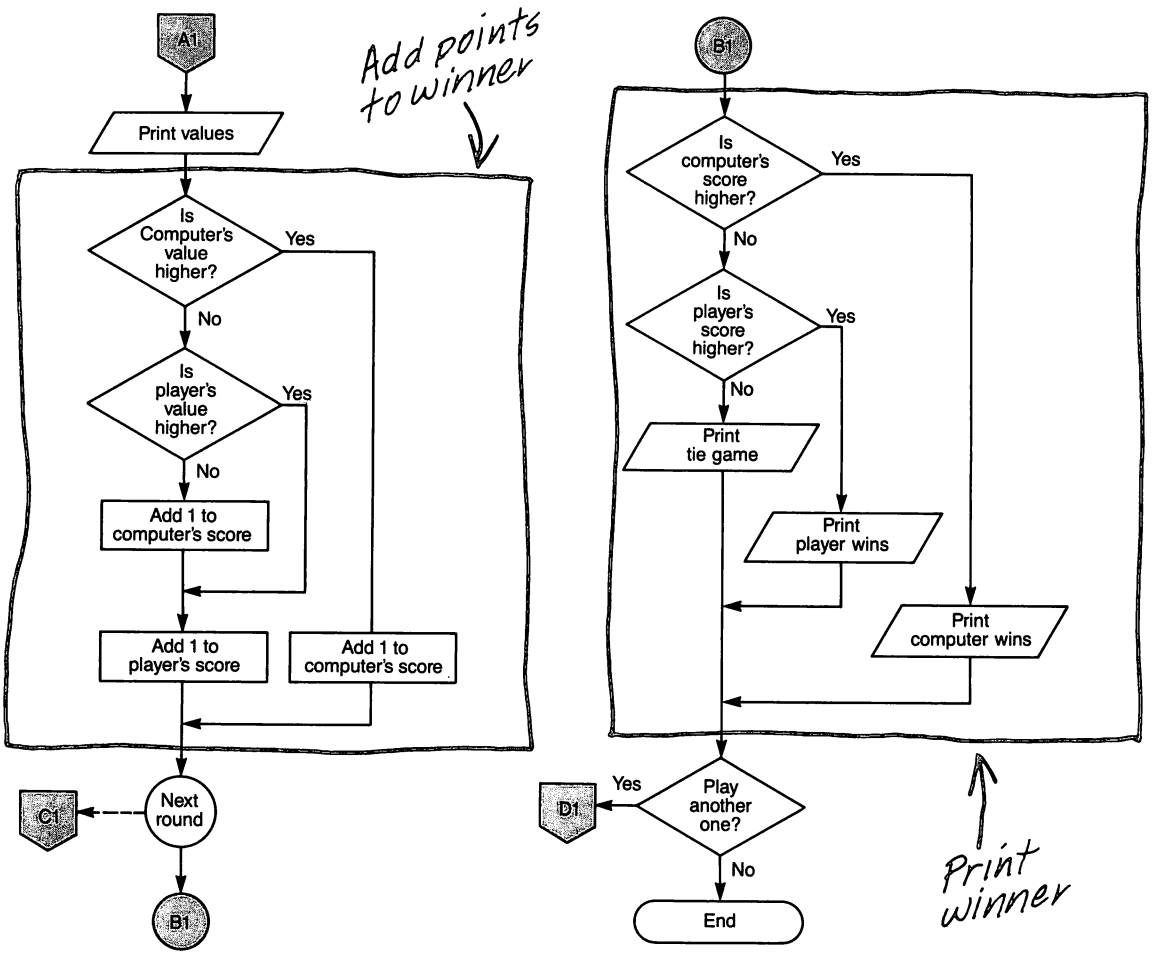
Inputs
A$ = User response

Process
COMPUTER = Computer's total score
PLAYER = Player's total score
ROUND = Counter for 3 rounds
ROLL = Counter for 2 rolls
DIE1 = Value for Die 1
DIE2 = Value for Die 2
SCORE1 = Computer's score for the round
SCORE2 = Player's score for the round

Outputs
SCORE1
SCORE2



Flow chart continued on next page


A1
Print values
Add points to winner
Is Computer's value higher?
Yes
No
Is player's value higher?
Yes
No
Add 1 to computer's score
Add 1 to player's score
Add 1 to computer's score
C1
Next round
B1
B1
Is computer's score higher?
Yes
No
Is player's score higher?
Yes
No
Print tie game
Print player wins
Print computer wins
D1
Yes
Play another one?
No
End
Print winner

Step 4
Write the code

Instruction	Explanation
100 REM DICE GAME	
110 REM START UP RANDOM NUMBER GENERATOR	
120 RANDOMIZE	Reseed the random number generator
130 REM START SCORES AT ZERO	
140 COMPUTER = 0	Start computer's total score at zero
150 PLAYER = 0	Start player's total score at zero
160 REM PRINT HEADINGS	
170 CALL CLEAR	Clear the screen
180 PRINT "TWO OUT OF THREE ROLLS WINS"	
190 PRINT :"COMPUTER", "PLAYER"	Print the heading lines
200 REM MAIN PROGRAM	
210 FOR ROUND = 1 TO 3	Begin loop for 3 rounds
220 FOR ROLL = 1 TO 2	Begin loop for 2 rolls
230 DIE1 = INT(RND*6) + 1	Generate a value for Die 1
240 DIE2 = INT(RND*6) + 1	Generate a value for Die 2
250 IF ROLL = 2 THEN 280	If player's turn, go to line 280
260 SCORE1 = DIE1 + DIE2	Add value of dice for computer's score
270 GOTO 290	Skip to end of roll loop
280 SCORE2 = DIE1 + DIE2	Add value of dice for player's score
290 NEXT ROLL	End of roll loop; terminates after both computer and player have rolled the dice
300 REM PRINT SCORES	
310 PRINT SCORE1, SCORE2	Print the scores for this round
320 REM ADD POINTS TO WINNER	
330 IF SCORE1>SCORE2 THEN 360	If computer's score is higher, go to line 360
340 IF SCORE2>SCORE1 THEN 380	If player's score is higher, go to line 380
350 PLAYER = PLAYER + 1	If neither is higher, add a point to both scores
360 COMPUTER = COMPUTER + 1	Add one to computer's score
370 GOTO 390	Skip to end of round loop
380 PLAYER = PLAYER + 1	Add one to player's score
390 NEXT ROUND	End of round loop; terminates after 3 rounds
400 REM PRINT WINNER	
410 IF COMPUTER>PLAYER THEN 450	If computer's total is higher, go to line 450
420 IF PLAYER>COMPUTER THEN 470	If player's total is higher, go to line 470
430 PRINT "TIE GAME"	If neither is higher, print tie game message

Note
An important feature is the nested FOR-NEXT loops. The outer loop controls the three rounds. The inner loop allows each player (the human or the computer) to roll the dice once in each round.

You might decide to use three dice instead of two.

You could change the program to have one person play against a second, or increase the number of players to more than two, or you could have the number of players as a variable to be chosen for each game.

The game possibilities are numerous. Each program can become your own creation limited only by your imagination.

Instruction	Explanation
440 GOTO 480	
450 PRINT "THE COMPUTER WINS!!"	Print computer wins message
460 GOTO 480	
470 PRINT "YOU BEAT THE COMPUTER!!"	Print player wins message
480 PRINT :"ANOTHER GAME";	Ask if player wants to play again
490 INPUT A$	Get player's response
500 IF A$>"NO" THEN 130	If player wants to play again, go back to line 130 and start the game over
510 PRINT :"OK, GOODBYE."	If not, say goodbye
520 END	

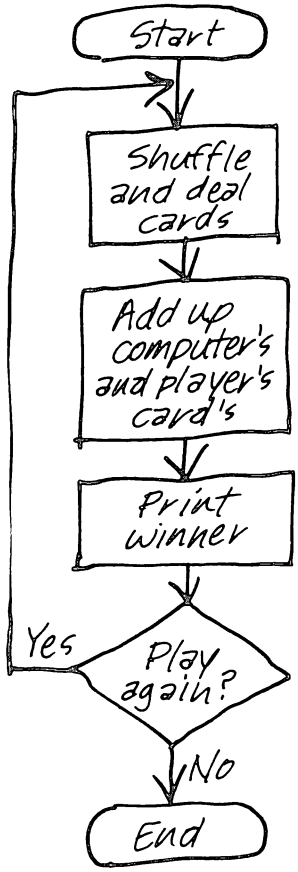
Note
When running larger programs or testing programs, it is not uncommon to encounter problems. Don't be discouraged, even the experienced programmer makes errors. You can learn a lot from your mistakes.

Step 5
Run the program and debug it

```
TWO OUT OF THREE ROLLS WINS

COMPUTER        PLAYER
  4               2
  7               9
  6               5
THE COMPUTER WINS!!

ANOTHER GAME? NO

OK, GOODBYE.

** DONE **
```

Example four:
Card game

This version of a card game is very limited in scope and is only the rudiments of Black Jack or 21. Space limitations prevent writing an entire program, but it would be good experience for you to expand it into a complete game of 21.



Step 1
Identify the problem
Write a program for the card game of 21 so that the computer is one player and you are the other. The computer also shuffles and deals the cards. The computer and player each draw two cards. The cards are evaluated and the score calculated. The highest score without going over 21 is the winner.

Step 2
Outline the solution
Several things must be kept in mind. The Ace is given a value of 11 unless that would put the score over 21; in that case, it is given a value of 1. Kings, Queens, and Jacks are given 10 points each. The deck of cards is a standard deck without the Jokers.

Inputs
1. User's response to play again question.

Process
1. Shuffle cards.
2. Deal cards.
3. Determine values for each player's hand.
4. Decide whose hand is larger.

Outputs
1. Print headings.
2. Print card values.
3. Print value for each player's hand.
4. Print who won the game.
5. Print play again question.

Step 3
Prepare a flowchart
The flowchart for this program is more complicated than the previous ones. There are three major FOR-NEXT loops. One builds the deck of 52 cards comprised of 13 cards in 4 suits. The other two loops are alike except that one is for the computer and one for the player. These loops evaluate the cards that have been drawn and compute the score.

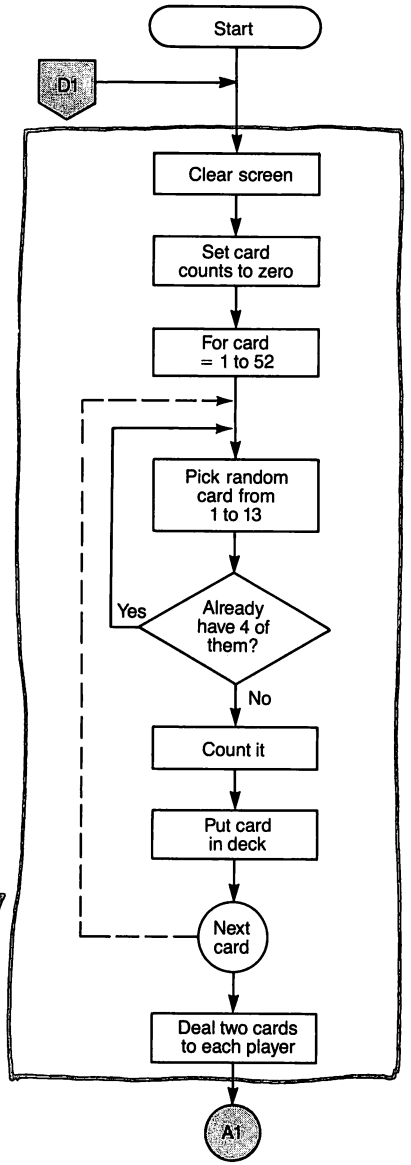
There is also a subroutine which is used to add up the cards in each hand. Since the logic to do this is the same for both the computer and the player, it is more convenient to use a subroutine to perform this function and call it for both the computer's and the player's cards, rather than have a separate subroutine for each.

Assign variable names

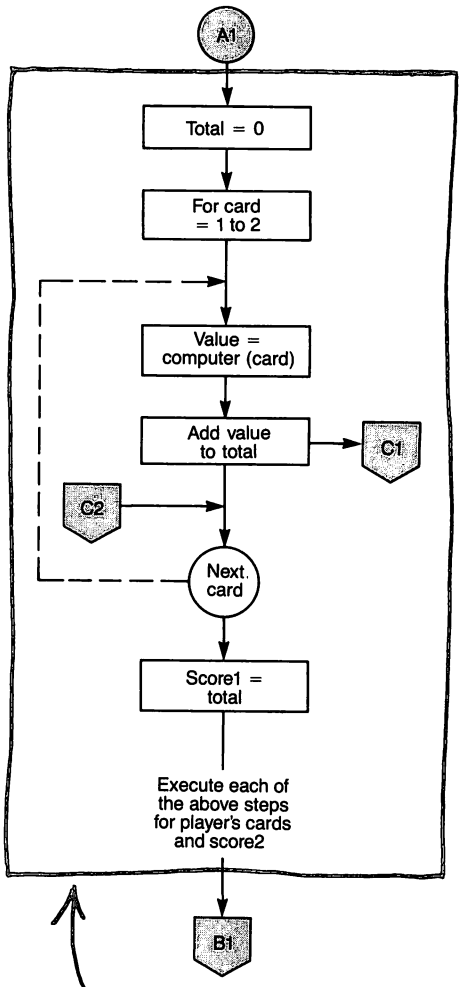
Inputs
A$ = User's response

Process
DECK = Array, deck of 52 cards
COUNT = Array, count of cards in deck
COMPUTER = Array, value for each of computer's cards
PLAYER = Array, value for each of player's cards
SCORE1 = Total value for computer's cards
SCORE2 = Total value for player's cards
CARD = Loop counter for cards
I = Loop counter for initialization
R1 = Temporary value for cards
TOTAL = Temporary total value of cards
VALUE = Temporary value of a card

Outputs
COMPUTER
PLAYER
SCORE1
SCORE2

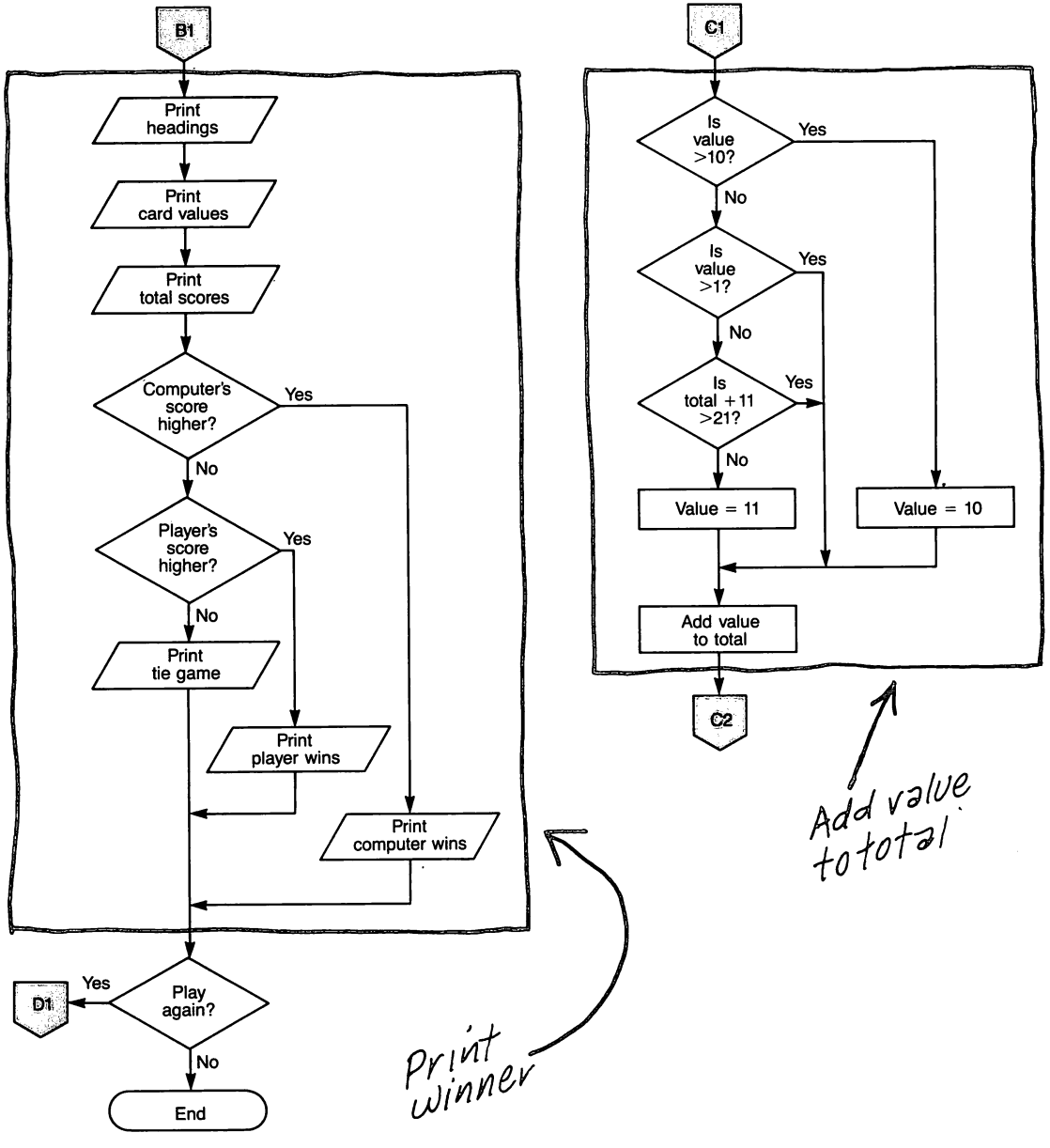


Shuffle and deal cards



Add up computer's and player's cards

Flowchart continued on next page


B1
Print headings
Print card values
Print total scores
Computer's score higher?
Yes
No
Player's score higher?
Yes
No
Print tie game
Print player wins
Print computer wins
D1
Yes
Play again?
No
End
C1
Is value >10?
Yes
No
Is value >1?
Yes
No
Is total +11 >21?
Yes
No
Value = 11
Value = 10
Add value to total
C2
Add value to total
Print winner

Step 4
Write the code

Instruction	Explanation
100 REM CARD GAME	
110 DIM DECK(52)	Dimension the array for the deck of cards
120 DIM COUNT(13)	Dimension the array for the card counter
130 DIM COMPUTER(2)	Dimension the array for the computer's cards
140 DIM PLAYER(2)	Dimension the array for the player's cards
150 REM START UP THE RANDOM NUMBER GENERATOR	
160 RANDOMIZE	Reseed the random number generator
170 REM MAIN PROGRAM	
180 CALL CLEAR	Clear the screen
190 PRINT "HIGHEST SCORE WINS"::: 200 REM SET ALL COUNTS TO ZERO	
210 FOR I = 1 TO 13	Loop to set counters for cards to zero
220 COUNT(I) = 0	
230 NEXT I	
240 REM SHUFFLE CARDS	
250 PRINT "SHUFFLING..."	
260 FOR CARD = 1 TO 52	Loop to shuffle cards
270 R1 = INT(RND*13) + 1	Generate a random value between 1 and 13
280 IF COUNT(R1) = 4 THEN 270	If there are already 4 cards of that value in the deck, get another value
290 COUNT(R1) = COUNT(R1) + 1	Otherwise, count the card
300 DECK(CARD) = R1	And add it to the deck
310 NEXT CARD	End of shuffle loop; terminates when all 52 cards have been chosen
320 REM DEAL 2 CARDS EACH FOR COMPUTER AND PLAYER	
330 PRINT ::"DEALING..."	
340 COMPUTER(1) = DECK(1)	Give first card to computer
350 PLAYER(1) = DECK(2)	Second card to player
360 COMPUTER(2) = DECK(3)	Third card to computer
370 PLAYER(2) = DECK(4)	Fourth card to player
380 REM ADD UP COMPUTER'S CARDS	
390 TOTAL = 0	Start total of hand at zero
400 FOR CARD = 1 TO 2	Loop to determine total value of computer's hand
410 VALUE = COMPUTER(CARD)	Assign computer's card value to temporary variable to be used by the subroutine
420 GOSUB 800	Add the card's value to the hand total
430 NEXT CARD	End of value loop; terminates when both cards have been added to total
440 SCORE1 = TOTAL	Assign total hand value to computer
450 REM ADD UP PLAYER'S CARDS	
460 TOTAL = 0	Start total of hand at zero
470 FOR CARD = 1 TO 2	Loop to determine total value of player's hand

The program can be structured differently to make it more efficient. It was presented in this manner to make it easier for you to follow the logic.

Enhancing the program to a full game of 21 will require a lot of thought to cover all possibilities.

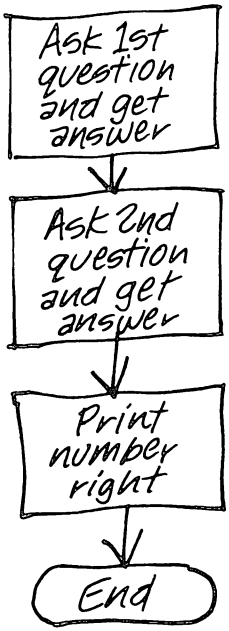
Instruction	Explanation
480 VALUE = PLAYER(CARD)	Assign player's card value to temporary variable
490 GOSUB 800	Add the card's value to the hand total
500 NEXT CARD	End of value loop; terminates when both cards have been added to total
510 SCORE2 = TOTAL	Assign total hand value to player
520 REM PRINT WINNING RESULTS	
530 CALL CLEAR	Clear the screen
540 PRINT TAB(10);"COMPUTER";	Print headings
550 PRINT TAB(20);"PLAYER"::	
560 FOR CARD = 1 TO 2	Loop to print values of each card
570 PRINT "CARD";CARD;	Print card number
580 PRINT TAB(12); COMPUTER(CARD);	Print computer's card
590 PRINT TAB(21);PLAYER(CARD)	Print player's card
600 NEXT CARD	End of print loop; terminates when all cards have been printed
610 PRINT :"SCORE";	
620 PRINT TAB(12);SCORE1;	Print total of computer's hand
630 PRINT TAB(21);SCORE2	Print total of player's hand
640 IF SCORE1>SCORE2 THEN 680	If computer's score is higher, go to line 680
650 IF SCORE2>SCORE1 THEN 700	If player's score is higher, go to line 700
660 PRINT ::"TIE GAME."	If neither score is higher, print tie game message
670 GOTO 710	
680 PRINT ::"THE COMPUTER WINS!!"	Print computer wins message
690 GOTO 710	
700 PRINT ::"YOU WIN!!"	Print player wins message
710 PRINT ::"DO YOU WANT TO PLAY AGAIN";	Ask if player wants to play again
720 INPUT A$	Get player's response
730 IF A > "NO" THEN 170	If player wants to play again, go to line 170
740 PRINT ::"OK, GOODBYE!!"	Print goodbye
750 END	Stop program
800 REM SUBROUTINE TO ADD UP CARD VALUES	
810 IF VALUE>10 THEN 870	If card is a face card (11,12,13), go to line 870
820 IF VALUE>1 THEN 880	If card is a numeric card (2-10), go to line 880
830 REM DETERMINE VALUE FOR ACE	Otherwise, the card is an ace (1)
840 IF TOTAL + 11>21 THEN 880	If 11 would put the total over 21, leave the value for the ace at 1, go to line 880
850 VALUE = 11	Otherwise, give the ace a value of 11
860 GOTO 880	
870 VALUE = 10	Give face cards a value of 10
880 TOTAL = TOTAL + VALUE	Add the value of the card to the total
890 RETURN	Go back to the main program

Step 5
Run the program and debug it

```
HIGHEST SCORE WINS

SHUFFLING...

DEALING...

                COMPUTER        PLAYER

CARD 1          7               4
CARD 2          12              1

SCORE           17              15

THE COMPUTER WINS!!

DO YOU WANT TO PLAY AGAIN?
NO

OK, GOODBYE!!

** DONE **
```

Example five:
Quiz program

This example is a quiz with a little humor injected in the responses from the computer. Quizzes of this type offer a fun way to learn.



Step 1
Identify the problem
Ask two questions, one at a time. Compare student's answer with correct answer. If correct, add 1 point to score. If not correct, repeat the question for a second try. If correct on second try, do not add a point. If not correct, give the correct answer. Use humorous response messages for correct and incorrect answers.

Step 2
Outline the solution

Inputs
1. Answers to questions.

Process
1. Determine if answer is correct.
2. Determine how many answers the student got right.
3. Delay while student reads messages.
4. Add points to student's score for correct answers.

Outputs
1. Print questions.
2. Print right/wrong messages.
3. Print student's score.
4. Print evalution messages.

Step 3
Prepare a flowchart
The flowchart quickly shows the many decision points and branching required in the code.

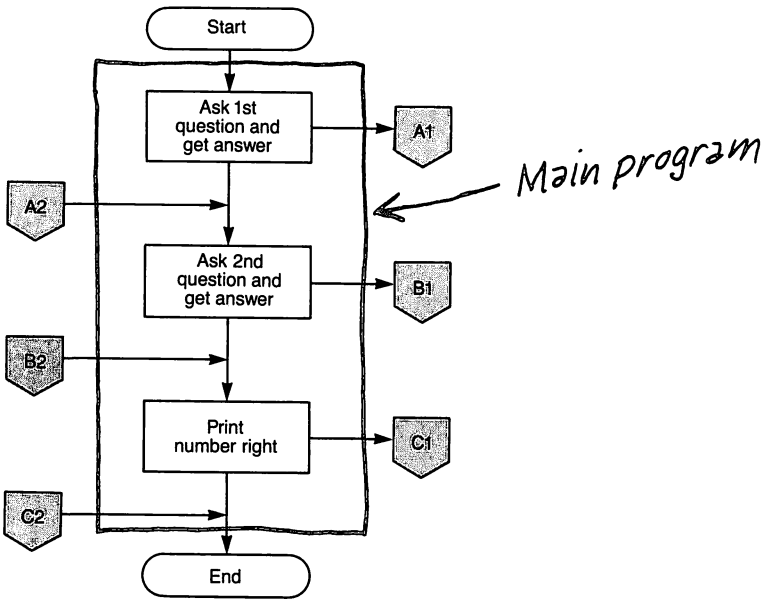
You can see that this program uses a separate subroutine for each question. Notice how similar the two subroutines are in their structure. The only differences are the words within the quotes. Perhaps you could rewrite this as one subroutine with the text messages and answers stored in variables. This would allow you to add more questions without duplicating the whole subroutine for each one.

Assign variable names

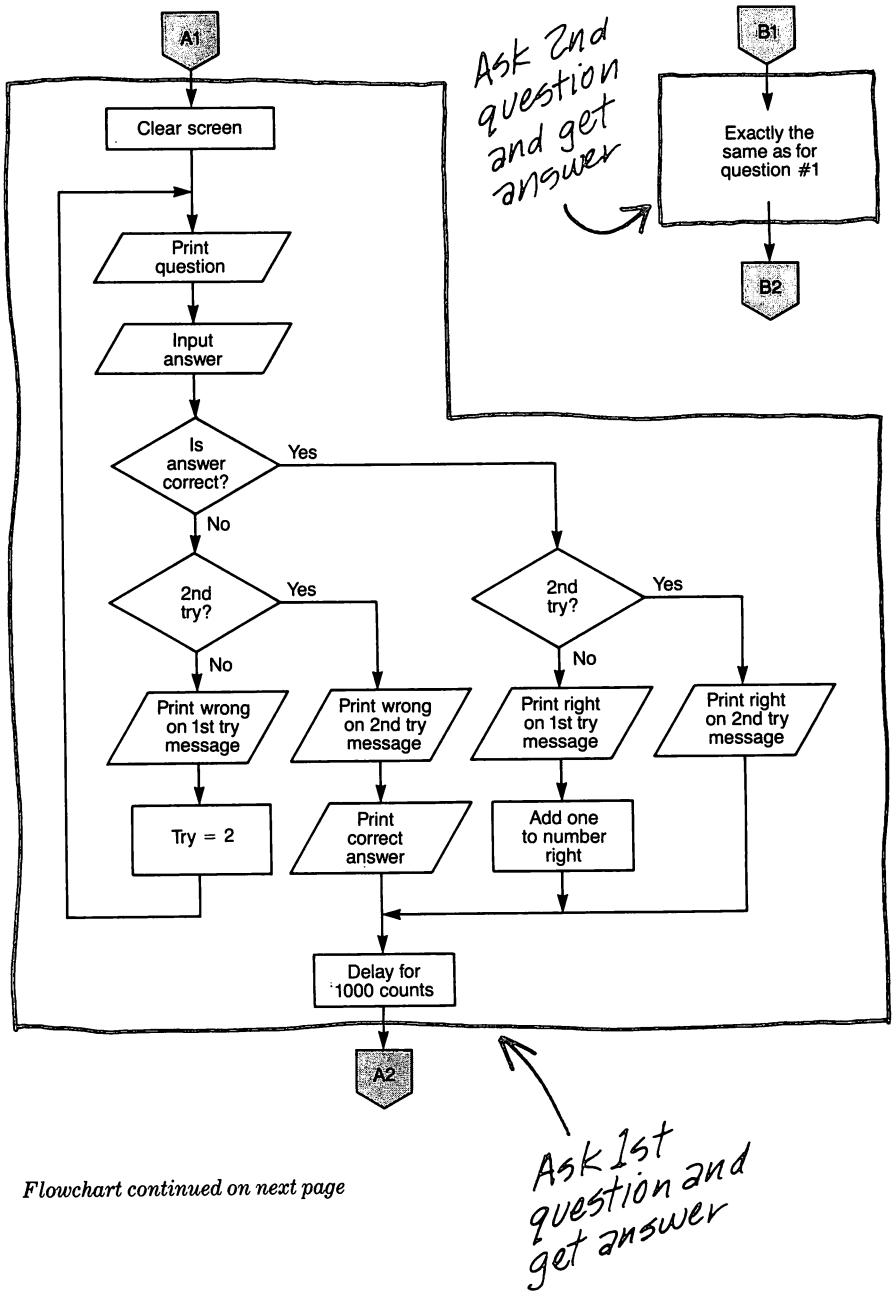
Inputs
A$ = Student's answer

Process
RIGHT = Number of questions answered correctly on first try
TRY = Number of tries for correct answer
DELAY = Loop counter for delay

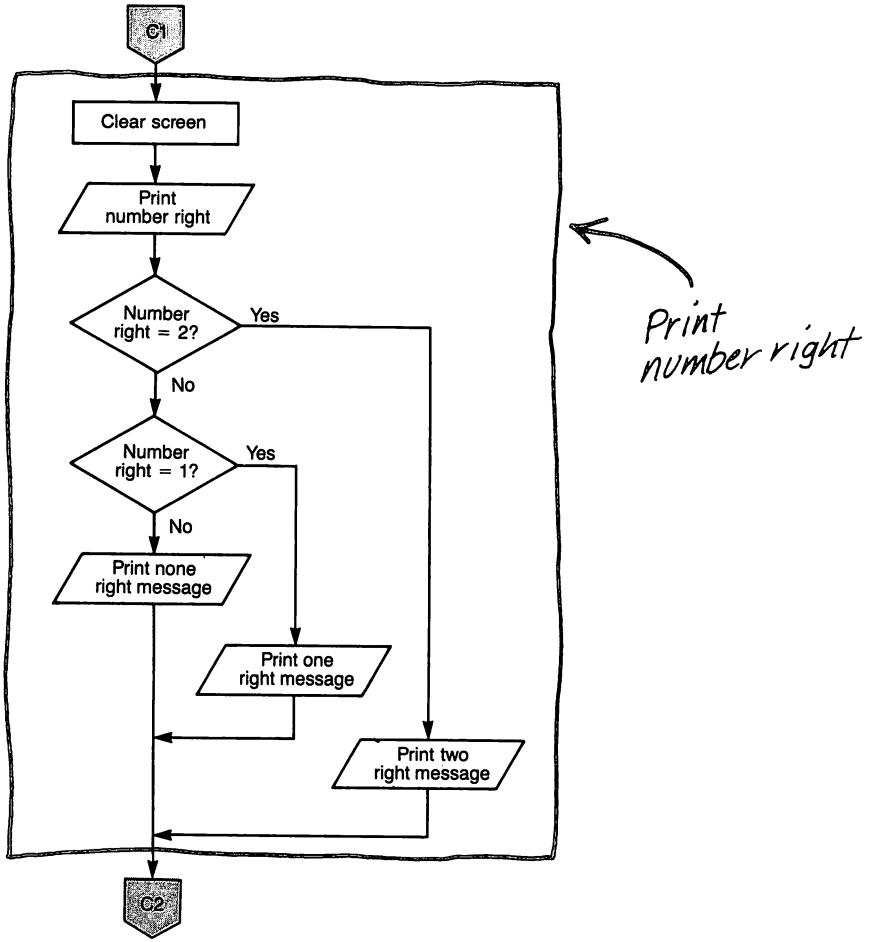
Outputs
RIGHT = Number of questions answered correctly on first try



Flowchart continued on next page



Flowchart continued on next page



Step 4
Write the code
The coding is rather long, primarily due to several PRINT statements for the responses. A critical variable in this program is TRY which counts the number of attempts at the answers. The value of TRY determines the branching to the response messages. TRY must be reset to one for each new question.

Instruction	Explanation
100 REM COMIC QUIZ	
110 RIGHT=0	Start right answer counter at zero
120 REM MAIN ROUTINE	
130 REM FIRST QUESTION	
140 GOSUB 300	Perform 1st question subroutine; will return to line 150
150 REM SECOND QUESTION	
160 GOSUB 700	Perform 2nd question subroutine; will return to line 170

Note
If you want to add questions to this program, it will require additional GOSUBs to the appropriate subroutines (if the same program logic is used).

	Instruction	Explanation
	170 REM PRINT CLOSING REMARKS	
	180 GOSUB 1100	Perform closing remarks subroutine; will return to line 190
	190 PRINT ::	
	200 INPUT "PRESS ENTER TO END":A$	Wait until student presses ENTER key
	210 END	
External storage devices could be used to store the questions and answers.	300 REM FIRST QUESTION SUBROUTINE	
	310 TRY = 1	Indicate student's first try at this question
	320 CALL CLEAR	Clear the screen
For variety in responses, several responses could be available and chosen at random.	330 PRINT "IN WHAT STATE IS ST. LOUIS?"	Ask the first question
	340 INPUT A$	Get student's answer
	350 IF A$ = "MISSOURI" THEN 490	If student's answer is correct, go to line 490
	360 REM WRONG ANSWER	
	370 IF TRY = 2 THEN 440	If this is student's second try, go to line 440
	380 REM WRONG ON FIRST TRY	
	390 PRINT :"GOSH, YOUR BEARINGS ARE OFF!"	
	400 PRINT "TRY IT AGAIN."	
	410 PRINT ::	
	420 TRY = TRY + 1	Give student another try; indicate second try
	430 GOTO 330	Go back and ask the question again
	440 REM WRONG ON SECOND TRY	
	450 PRINT :"YOU SURE AREN'T FROM THE"	Print wrong on second try message
	460 PRINT "SHOW ME STATE!"	
	470 PRINT :"IT'S GOOD OLD MISSOURI."	
	480 GOTO 580	Skip ahead to the delay loop
	490 REM RIGHT ANSWER	
	500 IF TRY = 2 THEN 550	If this is student's second try, go to line 550
	510 REM RIGHT ON FIRST TRY	
	520 PRINT :"RIGHT ON!!"	
The scoring could be changed to allow 2 points for a correct answer on the first try and one point for a correct answer on the second try with an appropriate message.	530 RIGHT = RIGHT + 1	Student answered right on first try; add a point to student's score
	540 GOTO 580	Skip ahead to the delay loop
	550 REM RIGHT ON SECOND TRY	
	560 PRINT :"NOW YOU'RE RIGHT!"	Print right on second try message
	570 PRINT "I GUESS YOU'VE BEEN THERE."	
	580 FOR DELAY = 1 TO 1000	Loop to delay so student can read the messages
	590 NEXT DELAY	
	600 RETURN	Go back to the main program
	700 REM SECOND QUESTION SUBROUTINE	
	710 TRY = 1	Indicate student's first try at this question
	720 CALL CLEAR	Clear the screen

Instruction	Explanation
730 PRINT "WHAT IS THE AREA OF"	Ask the second question
740 PRINT "10 FT. BY 10 FT?"	
750 INPUT A$	Get student's answer
760 IF A$ = "100" THEN 910	If student's answer is correct, go to line 910
770 REM WRONG ANSWER	
780 IF TRY = 2 THEN 860	If this is student's second try, go to line 860
790 REM WRONG ON FIRST TRY	
800 PRINT :"I SAID WHAT'S THE AREA,"	
810 PRINT "NOT WHAT'S THE TEMPERATURE!"	
820 PRINT "TRY IT AGAIN."	
830 PRINT ::	
840 TRY = TRY + 1	Give student another try; indicate second try
850 GOTO 730	Go back and ask the question again
860 REM WRONG ON SECOND TRY	
870 PRINT :"I GUESS YOU'VE NEVER SEEN"	Print wrong on second try message
880 PRINT "A TEN FOOT SQUARE!"	
890 PRINT :"THE ANSWER IS 100."	
900 GOTO 990	Skip ahead to the delay loop
910 REM RIGHT ANSWER	
920 IF TRY = 2 THEN 970	If this is student's second try, go to line 970
930 REM RIGHT ON FIRST TRY	
940 PRINT :"JUST WHAT THE DOCTOR SAID!!"	
950 RIGHT = RIGHT + 1	Student answered right on first try; add a point to student's score
960 GOTO 990	Skip ahead to the delay loop
970 REM RIGHT ON SECOND TRY	
980 PRINT :"BY GEORGE, YOU GOT IT!"	Print right on second try message
990 FOR DELAY = 1 TO 1000	Loop to delay so student can read the messages
1000 NEXT DELAY	
1010 RETURN	Go back to the main program
1100 REM PRINT CLOSING REMARKS SUBROUTINE	
1110 CALL CLEAR	Clear the screen
1120 PRINT "YOU GOT";RIGHT; "RIGHT"	Print number right on first try
1130 PRINT "ON THE FIRST TRY."	
1140 IF RIGHT = 2 THEN 1240	If student got 2 right, go to line 1240
1150 IF RIGHT = 1 THEN 1200	If student got 1 right, go to line 1200
1160 REM NONE RIGHT	
1170 PRINT :"AW, SHUCKS!"	Print none right message
1180 PRINT "BETTER LUCK NEXT TIME."	

Use your imagination and change the program; you'll learn a lot by experimenting.

Instruction	Explanation
1190 GOTO 1270	Skip ahead to the end of the subroutine
1200 REM ONE RIGHT	
1210 PRINT:"I GUESS YOU KNOW"	Print one right message
1220 PRINT "A FEW THINGS."	
1230 GOTO 1270	Skip ahead to the end of the subroutine
1240 REM TWO RIGHT	
1250 PRINT :"YOU MUST THINK YOU'RE SMART."	Print two right message
1260 PRINT "WELL, YOU'RE RIGHT!!"	
1270 RETURN	Go back to the main program

Because this program is longer and more complex, you are more likely to have errors.

Step 5
Run the program and debug it

```
IN WHAT STATE IS ST. LOUIS?
?MONTANA

GOSH YOUR BEARINGS ARE OFF!
TRY IT AGAIN.


IN WHAT STATE IS ST. LOUIS?
?MISSISSIPPI

YOU SURE AREN'T FROM THE
SHOW ME STATE!

IT'S GOOD OLD MISSOURI.


WHAT IS THE AREA OF
10 FT. BY 10 FT.?
?100

JUST WHAT THE DOCTOR SAID!!


YOU GOT 1 RIGHT
ON THE FIRST TRY.

I GUESS YOU KNOW
A FEW THINGS.


PRESS ENTER TO END

** DONE **
```

Value of entertainment programs

Is developed and improved by recognition of different shapes at different angles.

Logical thinking ← Strategy games develop this in order to plan ahead.

Spatial awareness

Many games have fast action that require quick decisions to win.

Quick decisions

Simulated conditions ← Provides realistic experiences that a person otherwise would miss.

Educational games ← Make learning fun and are especially good for slow learners.

Entertainment programs

Example one: Making a picture

SIN function

Example two: Print a word using the sine function

Example three: Simulation of rolling a pair of dice

Example four: Card game

Try your hand at making up your own game.

Example five: Quiz program

An interesting S pattern.

Summary map

3 **Introduction**

3 **Speech**

Punctuation

Combining words and phrases

Homographs

Speech in the program mode

Example one

5 **The CALL SAY instruction**

Example one: A program with speech using string expressions

Example two: Modifying example one to use string variables

11 **The CALL SPGET instruction**

11 **Uses of the speech synthesizer**

12 **The internal sound system**

12 **The CALL SOUND instruction**

Example three: Playing the C major chord

Example four: Playing three notes separately and as a chord

17 **Experimenting with time and volume**

Example five: Playing a melody

20 **Uses of computer sound**

Alerting tones

Music

Noise

21 **Graphics**

22 **The CALL VCHAR instruction**

23 **The CALL HCHAR instruction**

24 **The CALL SCREEN instruction**

24 **The CALL COLOR instruction**

Example six:	Display all the screen colors
Example seven:	Experimenting with color
Example eight:	Print hello in color
Example nine:	Generating both graphics and sound

36 **Developing your own character sets**

36 **Uses of graphics**

37 **Summary map**

Example six

Example seven

Example nine

Sound, Graphics and More

Introduction

The sound capability of the TI Home Computer includes musical tones and speech with the optional Solid State Speech™ Synthesizer. The graphics capability provides 16 colors that can be used for background, foreground, and character color in any combination. In addition to the standard set of characters, special characters also can be generated for use in graphics.

Speech

With speech capability, the computer can give audible prompts instead of, or in addition to, visual prompts. Verbal reinforcement can be especially important to younger children and non-readers.

Writing programs which make the computer talk is similar to writing any other program. Both the optional speech synthesizer and Speech Editor cartridge must be used with the computer in all speech programs. The speech vocabulary is about 250 words and phrases.

To use the speech capability, insert the Speech Editor cartridge into the console slot and select the desired option. For use in the immediate mode, select option 2 and the Speech Editor message will appear at the top of the screen.

Type HELLO
Press **ENTER**

Note
Be sure the volume control on the monitor or TV set is turned up enough to hear the sound.

The computer says "HELLO."

If you type words that are not recognized by the computer, it will halt and indicate the word that is not acceptable.

Punctuation

Pauses in computer speech are produced by punctuation symbols.

When writing sentences on paper, it is customary to use punctuation to assist the reader. Punctuation is also used in speech commands for the TI Home Computer. This punctuation includes commas, semicolons, colons, periods, spaces between words, and the + and − symbols. Each of these will cause a pause in the speech. The table below shows the actual pause time values for each of the acceptable punctuation symbols.

Speech separator characters and pause time intervals

Symbol	Pause (seconds)
+	0
SPACE	.1
−	.2
,	.3
;	.5
:	.8
.	1.0

Experimenting with these in either the immediate execution mode or in the program mode will allow you to hear the effects of the punctuation. Try typing the following in the immediate mode.

I THINK. I KNOW, I... THINK-. I KNOW.

The time values of punctuation used in combination are additive.

You can imagine how this will sound just by looking at the line and the table. Note that when the punctuation symbols are used in combination, the pause times are added together to produce longer pause intervals. This is true in both the immediate and program modes.

Combining words and phrases

Words in the vocabulary can be combined to create other words.

Another important topic is that of combining words and phrases. Although, the vocabulary of the Speech Editor is small, it can be expanded by combining words which are in the vocabulary to make other words which are not in the vocabulary. The spelling may not be correct, but the spoken word will sound like the desired word.

Combining words in this manner requires use of the + sign between words. For example, the word "therefore" does not exist in the computer's vocabulary, but the words "there" and "four" do exist. To make the computer say "therefore", type in THERE + FOUR. Another example is seaside. To make this word, type SEE + SIDE; overdone is OVER + DONE, etc.

There are also some programmed phrases in the speech vocabulary. Examples are:

1. HANDHELD UNIT
2. TEXAS INSTRUMENTS
3. GOOD WORK
4. I WIN
5. SUPPOSED TO

You must begin and end these phrases with the number symbol. Thus, the phrase TEXAS INSTRUMENTS is written, #TEXAS INSTRUMENTS#, and in a sentence, it is written as:

I AM A #TEXAS INSTRUMENTS# COMPUTER.

Homographs

Homographs are words that are spelled the same but pronounced differently. Each pronunciation also has its own meaning. For example, the word "read" is pronounced "reed" and "red". To resolve this situation with computerized speech, two forms of each homograph are defined for use in programs.

Here are some examples:

READ (reed)	READ1 (red)
THE (thē)	THE1 (thuh)
A (ā)	A1 (uh)

Speech in the program mode

In order to write programs with speech, you must select option 1 on the master selection screen. Remember that the optional speech synthesizer and the Speech Editor cartridge must be operational and installed.

The CALL SAY instruction

Programs that use speech require the CALL SAY instruction. The format is:

CALL SAY ("string expression") or CALL SAY (string variable).

Example one:
A program with speech using string expressions

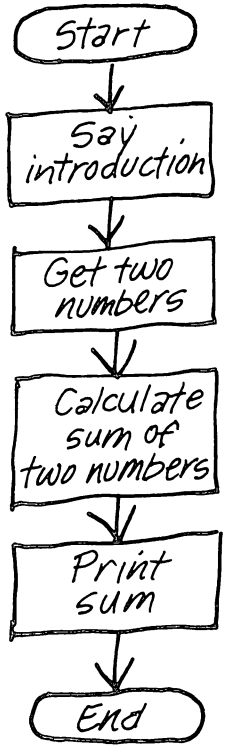
Step 1
Identify the problem
Write a program to visually and audibly request two numbers and calculate the sum. The computer gives an audible message for the answer and prints the answer on the screen.

Step 2
Outline the solution

Inputs
1. Get two numbers from user.

Process
1. Calculate the sum of the two numbers.

Outputs
1. Print and say prompts for numbers.
2. Print answer and say prompt



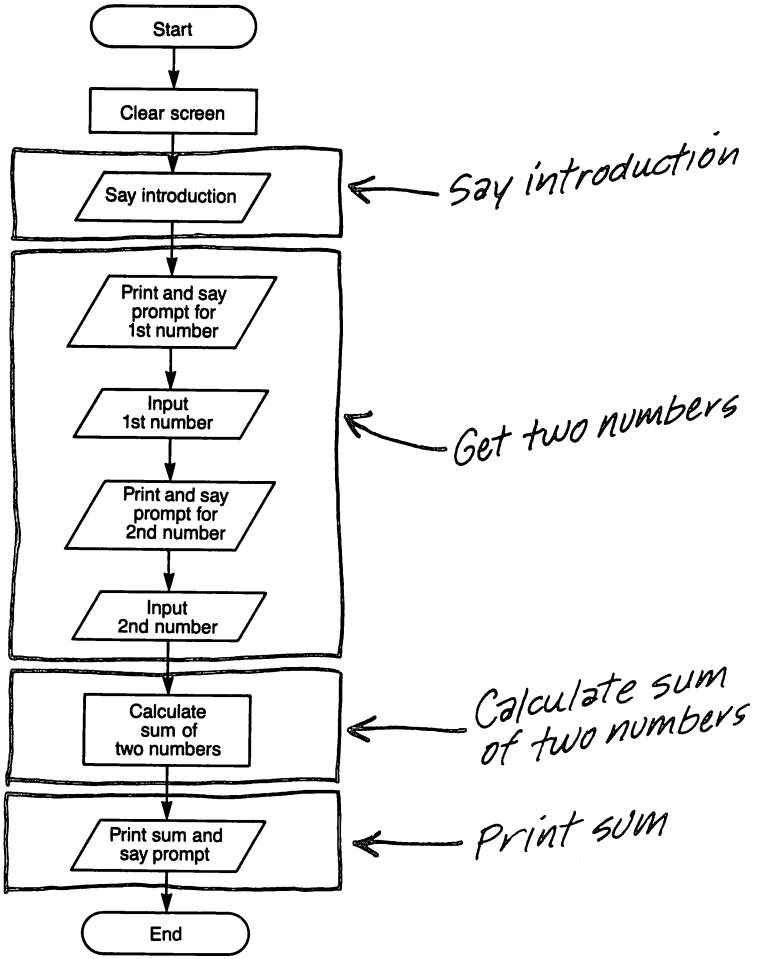
Step 3
Prepare a flowchart

Assign variable names

Inputs
N1 = First number entered
N2 = Second number entered

Process
ANSWER = Sum of the two numbers

Outputs
ANSWER = Sum of the two numbers



Step 4
Write the code

The program can be written another way by using string variables instead of having the words to be spoken in quotes.

A larger and more complicated program could give the answer verbally, too. Then the program could be extended to a flash card addition drill which provides both visual and audible cues.

Instruction	Explanation
10 CALL CLEAR	
20 CALL SAY ("HELLO.")	Computer says "Hello"
30 CALL SAY ("I AM A #TEXAS INSTRUMENTS# COMPUTER.")	Computer speaks "I am a Texas Instruments Computer"
40 PRINT "TYPE ANY NUMBER"	Computer prints screen prompt
50 CALL SAY ("TYPE ANY NUMBER.")	The computer says "Type any number"
60 INPUT N1	The user types in a number
70 PRINT "TYPE A SECOND NUMBER"	The computer prints screen prompt for second number
80 CALL SAY ("TYPE A1 SECOND NUMBER.")	The computer says "Type a second number"
90 INPUT N2	User types in a second number
100 ANSWER = N1 + N2	The computer computes the sum
110 CALL SAY ("THE ANSWER IS.")	The computer says "The answer is"
120 PRINT: "THE ANSWER IS"; ANSWER	
130 END	

Step 5
Run the program and debug it

Note
Be sure the volume control on the monitor or TV set is turned up enough to hear the sound.

Visual	Audible
	Hello. I am a Texas Instruments Computer.
TYPE ANY NUMBER	Type any number.
? **2**	
TYPE A SECOND NUMBER	Type a second number.
? **3**	
THE ANSWER IS 5	The answer is
** DONE **	

Example two:
Modifying example one to use string variables

Step 1
Identify the problem

Write a program to visually and audibly request two numbers and find the sum. The computer gives an audible message for the answer and prints the answer on the screen.

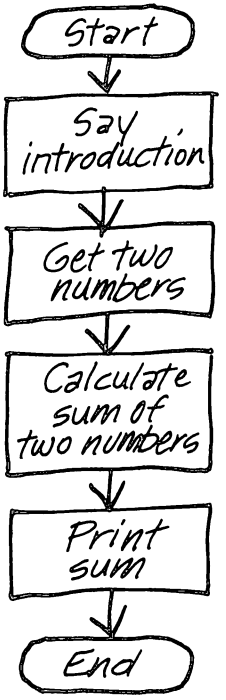
Step 2
Outline the solution

The change from example one is that the strings are assigned to variable names. Although the string expressions are different in this example, programs often use the same strings again and again. In such programs, computer memory is conserved and coding is easier when the strings are assigned to variable names.

Inputs
1. Get two numbers from user.

Process
1. Calculate the sum of the two numbers.

Outputs
1. Print and say prompts for numbers.
2. Print answer and say prompt



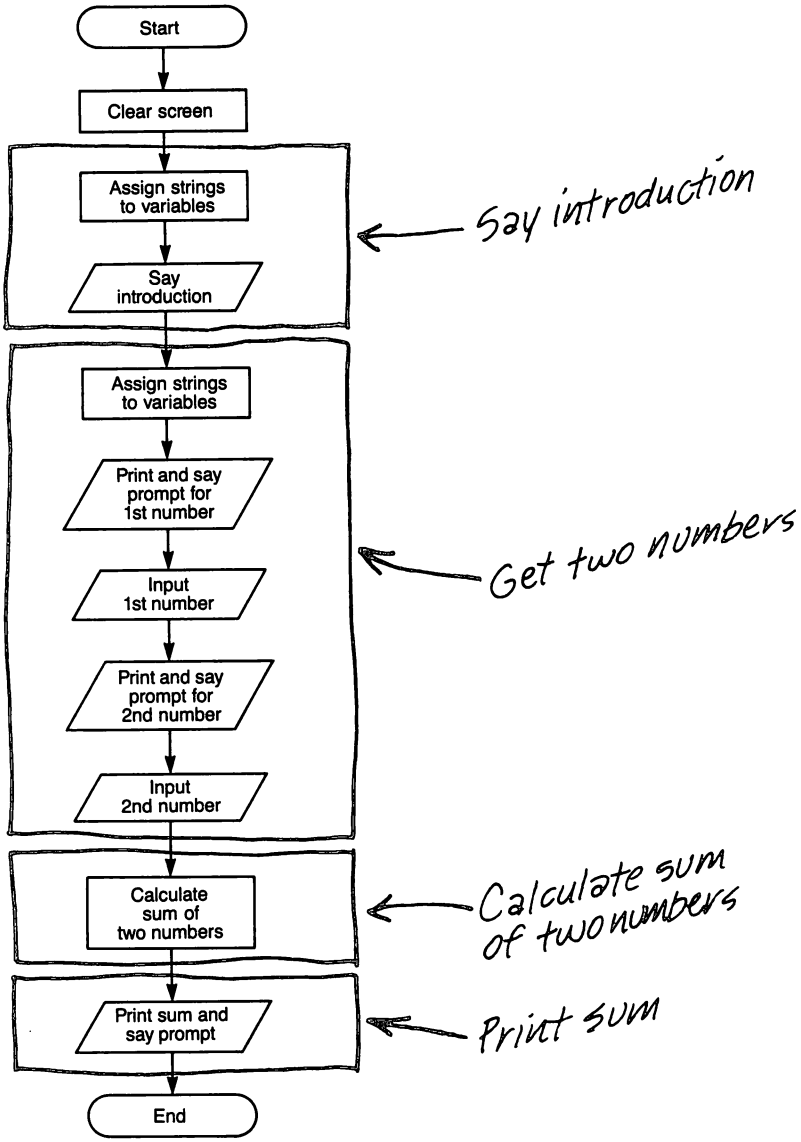
Step 3
Prepare a flowchart

Assign variable names

Inputs
N1 = First number entered
N2 = Second number entered

Process
ANSWER = Sum of the two numbers
A$ = "Hello"
B$ = "I am a"
C$ = "Texas Instruments"
D$ = "Computer"
E$ = "Type any number"
F$ = "Type a second number"
G$ = "The answer is"

Outputs
ANSWER = Sum of the two numbers



Step 4
Write the code

Instruction	Explanation
10 CALL CLEAR	
20 A$ = "HELLO."	Assign words to string variables
30 B$ = "I AM A"	
40 C$ = "#TEXAS INSTRUMENTS#"	
50 D$ = "COMPUTER."	
60 CALL SAY (A$,"",B$,"", C$,"",D$)	Computer says , "Hello. I am a Texas Instruments computer."
70 PRINT "TYPE ANY NUMBER"	
80 E$ = "TYPE ANY NUMBER."	Assign words to string variables
90 F$ = "TYPE A1 SECOND NUMBER."	
100 G$ = "THE ANSWER IS"	
110 CALL SAY (E$)	Computer says "Type any number."
120 INPUT N1	User types in first number
130 PRINT "TYPE A SECOND NUMBER"	
140 CALL SAY (F$)	Computer says "Type a second number."
150 INPUT N2	User enters second number
160 ANSWER = N1 + N2	Computes the sum
170 CALL SAY (G$)	Computer says "The answer is"
180 PRINT: "THE ANSWER IS"; ANSWER	Prints the answer
190 END	

Note
The quotation marks are not required in the CALL SAY program line since the variable names already contain the quoted words. The "" (open and close quotation marks with no space between) used in line 60 to connect the string variables together is called a null string. The null string must be used to connect string variables in CALL SAY statements. The & symbol cannot be used as it is in PRINT statements.

Step 5
Run the program and debug it

Visual	Audible
	Hello. I am a Texas Instruments Computer.
TYPE ANY NUMBER	Type any number.
?**2**	
TYPE A SECOND NUMBER	Type a second number.
?**3**	
THE ANSWER IS 5	The answer is
** DONE **	

The CALL SPGET instruction

Up to this point, we have been using words or phrases called word strings. The CALL SPGET instruction uses a type of string variable called a direct string. The direct string is stored directly as a speech pattern and has the advantage of providing better overall quality of the speech synthesizer sound.

If you use both word strings and direct strings with CALL SAY, then you must alternate between the types. The first word or phrase must be a word string. The general format is:

CALL SAY (word string, direct string, word string)

Here is an example program to say "I am a computer."

Note
The word to be assigned to the variable is enclosed by the quotation marks and is separated from the variable by a comma. The entire expression is enclosed in parentheses.

Instruction	Explanation
10 CALL SPGET("AM" ,B$)	Assigns AM to B$
20 CALL SPGET("COMPUTER", C$)	Assigns COMPUTER to C$
30 CALL SAY("I" ,B$,"A" ,C$)	Word strings and direct strings are alternated
40 END	

You can use SPGET to make a master vocabulary for a program that uses a lot of speech, then use CALL SAY with the appropriate variable names to build phrases and sentences as required.

You may use any of the methods discussed to program speech into your program using CALL SAY. However, more natural sounding speech is produced by first using CALL SPGET to assign the speech code for each word to a direct string variable, and then using the direct string variable with CALL SAY. The next example program illustrates this technique using SPGET for all words rather than only some of the words.

Note
Notice that a null string is used first in the expression in line 60.

Compare the output of this version using SPGET with one using only word strings and you should be able to notice the difference in speech quality.

Instruction	Explanation
10 CALL SPGET("HELLO",A$)	Assign word between quotes to the variable using the SPGET command
20 CALL SPGET("I",B$)	
30 CALL SPGET("AM",C$)	
40 CALL SPGET("A",D$)	
50 CALL SPGET("COMPUTER", E$)	
60 CALL SAY("",A$,"",B$,"", C$,"",D$,"",E$)	The null string is alternated with the variables as required
70 END	

Uses of the speech synthesizer

Computer speech can be used in business, education and entertainment.

In the business world, the speech synthesizer can alert the computer operator by audible commands if problems or errors are encountered. Also prompts for input information can be requested by speech as well as by screen. However, care must be taken so that the speech does not slow the operator.

The most useful application for computer speech could be in the educational area. It is especially appropriate for teaching reading and spelling.

In entertainment applications, the speech synthesizer can liven up a game with spoken responses. Speech also can be used to teach a new game by giving instructions and directions audibly.

Computer speech is still in the infant stage and progress is being made toward better sounding speech and unlimited vocabulary. Who knows what the future may bring. Speech synthesizers will read books to the blind, have robots speak to us, and have machines give us directions for their operation. Some of these are already available and other innovations are on the horizon. However, the TI Home Computer provides speech capability right now for your use and experimentation.

The internal sound system

Single tones, multiple tones and noise can be produced internally under control of your program.

The internal sound system of the TI Home Computer has a range of 5 octaves and any three tones can be played simultaneously. In addition to the musical tones, a noise generator can be used to produce sound effects. A common use for the noise generator is to add realism to the games played on the computer.

Before we write any programs using the TI sound capabilities, a discussion of the sound generator and the program instruction used to make sound is needed. The frequency range of the computer's tone generator is from 110 to more than 44,000 hertz. (One hertz (abbreviated Hz) is equal to one cycle per second.) This allows you to produce tones with the computer that vary from 110 Hz (the A below low C on a piano) to over 44,000 Hz (well above the range of human hearing).

The duration and volume of the sound can also be controlled by the program. The duration (how long the sound is "on") ranges from 1 to 4,275 milliseconds. One thousand (1,000) milliseconds is equal to one second; thus, the duration range is from 0.001 to 4.275 seconds.

The volume is chosen from a range of 0 to 30. Zero and one produce the same volume level which is the loudest. Thirty (30) is the quietest sound available with the TI Home Computer. Of course, the volume control on the monitor or TV set must be turned up enough to hear, and its setting also affects the loudness of the tones.

The CALL SOUND instruction

The instruction for sound is CALL SOUND. The format is:

CALL SOUND (duration in milliseconds, frequency in Hz, loudness)

Here is an example to produce a single tone:

CALL SOUND(1000,440,2)

Analyzing this instruction, we see that the sound lasts for 1,000 milliseconds which is one second. The 440 Hz tone produced is the A above middle C on the piano. (Frequencies for some musical notes are given in the User's Reference Guide for the TI Home Computer.) The loudness is relatively loud since it is set at 2.

The next example plays two tones simultaneously:

CALL SOUND(1000,440,2,659,2)

Only one duration may be specified in a CALL SOUND statement; therefore, all tones in a single statement are always of the same duration. In this example, the duration is 1,000 milliseconds. The first note (440) again is the A above middle C with a loudness setting of 2. The second note (659) is the E above high C. In this example, E is also the same loudness as the first note. To make the second note quieter, change the statement as follows:

```
CALL SOUND(1000,440,2,659,20)
```

Now the E (659) is played considerably quieter (20) than in the previous example.

Continuing on with the progression, add a third note of 880 with a loudness of 2 and change the loudness of the 659 note back to 2:

```
CALL SOUND(1000,440,2,659,2,880,2)
```

Again the first two notes are the same as in the previous examples (440 and 659). The third note at 880 Hz is high A. The volume of all three notes is the same at 2.

The internal sound system includes a noise generator. Noise is hard to define since noise to one person may be music to another. To put noise into a program, the frequency value is replaced with a negative integer from −1 to −8. For example:

```
CALL SOUND(1000, -4,2)
```

produces a very loud noise for one second.

You may have up to three tones and one noise generated in a statement. For example:

```
CALL SOUND(1000,440,2,659,2,880,2, -4,2)
```

The addition of the noise (-4,2) changes the effect of the sound.

Variables can be used in a statement rather than fixed values for the duration, tone or noise and volume. Of course, the variables must be assigned values elsewhere in the program. Sometimes it is easier to program using variables rather than the actual values. This can also save computer memory so the program is more efficient. Any valid variable name can be used, but the following variable names can be used to make them easy to remember:

T = time (duration)
V = volume (loudness)

Designate tones by the name of the note; for example, A1, A2, C, E, G, etc.

Example three:
Playing the C major chord

Step 1
Identify the problem
Write a program to play the C major chord for one second.

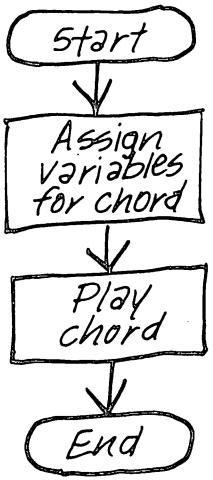
Step 2
Outline the solution

Inputs
1. None

Process
1. Assign appropriate values to variables.

Outputs
1. Play a C major chord.

Step 3
Prepare a flowchart

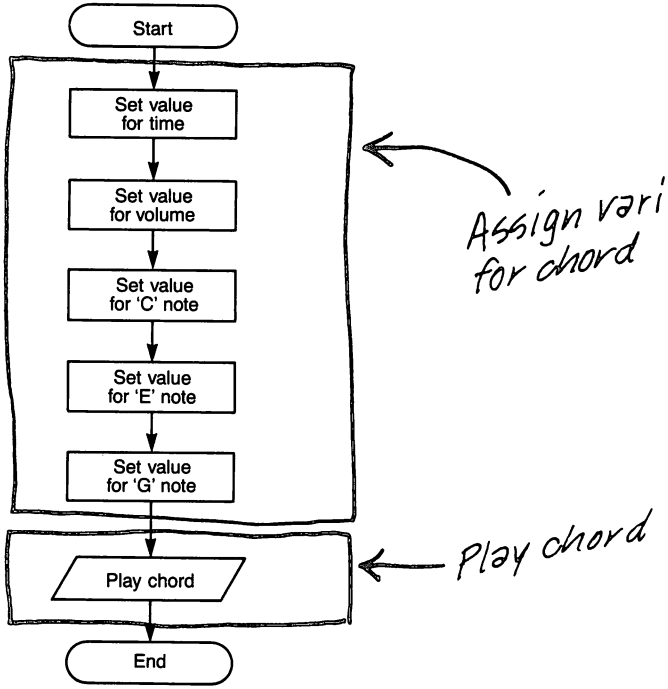


Assign variable names

Inputs
None

Process
V = Volume (loudness)
T = Time (length of play)
C = Middle C
E = E above middle C
G = G above middle C

Outputs
V = Volume (loudness)
T = Time (length of play)
C = Middle C
E = E above middle C
G = G above middle C



Step 4
Write the code

Instruction	Explanation
10 T = 1000	Time is set for one second
20 V = 1	Volume is set to very loud
30 C = 262	Middle C is set for one note
40 E = 330	E above middle C is set for second note
50 G = 392	G above middle C is set for third note
60 CALL SOUND(T,C,V,E,V,G,V)	The C major chord is played for 1 second
70 END	

Step 5
Run the program and debug it
In this program, the computer plays the C major chord for a period of one second.

Example four:
Playing three notes separately and as a chord

This is a good exercise for the beginning pianist or those learning music theory.

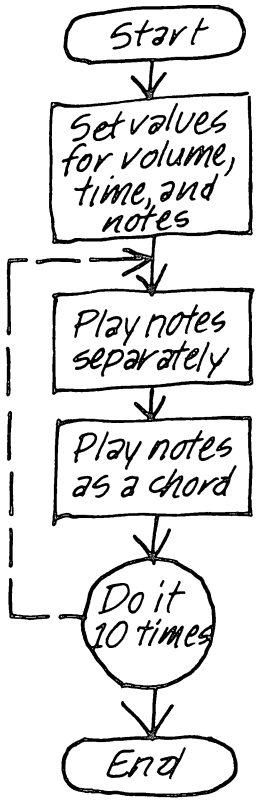
Step 1
Identify the problem
Write a program to play the three separate notes of the C major chord, then play the same notes together as a chord for a longer period of time. Repeat the process 10 times.

Step 2
Outline the solution

Inputs
1. None

Process
1. Assign appropriate values to variables.
2. Play sequence 10 times.

Outputs
1. Play three separate notes.
2. Play the notes as a chord.



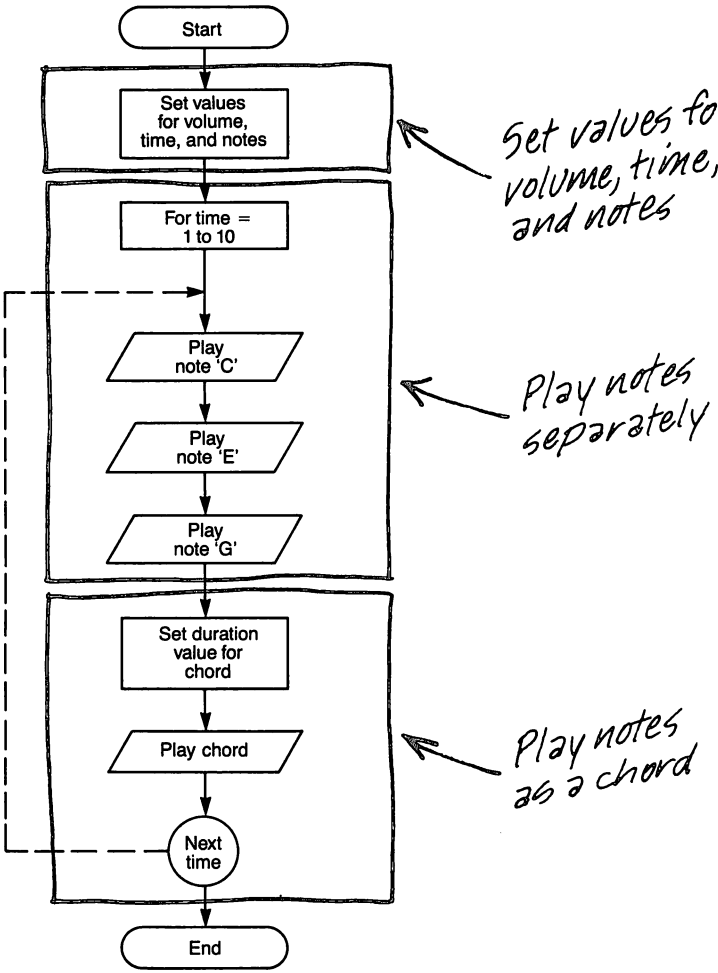
Step 3
Prepare a flowchart

Assign variable names

Inputs
None

Process
T = Length of time tone or chord is played
V = Loudness of the tone or chord
C = Middle C
E = E above middle C
G = G above middle C
TIME = Counter that keeps track of how many times the notes and chords have been played

Outputs
T = Length of time tone or chord is played
V = Loudness of the tone or chord
C = Middle C
E = E above middle C
G = G above middle C
TIME = Counter that keeps track of how many times the notes and chords have been played



Step 4
Write the code

This program illustrates that time variations can be programmed to simulate the different time values of notes and chords required to play a song.

Instruction	Explanation
10 V=10	Set the tone to loud
20 C=262	Set middle C
30 E=330	Set E above middle C
40 G=392	Set G above middle C
50 FOR TIME=1 TO 10	Loop to play sequence 10 times
60 T=1000	Set the time for one second
70 CALL SOUND(T,C,V)	Play C note for one second
80 CALL SOUND(T,E,V)	Play E note for one second
90 CALL SOUND(T,G,V)	Play G note for one second
100 T=3000	Set time for 3 seconds
110 CALL SOUND(T,C,V,E,V,G,V)	Play the C major chord for 3 seconds
120 NEXT TIME	Do it 10 times
130 END	

Step 5
Run the program and debug it
The individual notes are played separately and then simultaneously to make the chord. The chord is played longer than any of the individual notes. The sequence is repeated 10 times.

Experimenting with time and volume

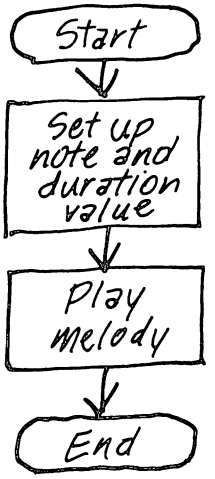
In order to program music, the programmer must determine how long and how loud each note or group of notes needs to be played. When first experimenting, try dividing a four-beat measure into 4 equal parts of 1,000 milliseconds, then vary the time duration to obtain the desired effect.

As stated earlier, zero and one produce the loudest volume level and 30 produces the quietest. Experimenting is the best way to find the volume level that you like. Remember, it is possible to play up to three notes simultaneously with each one at a different volume level. This adds variety to musical programs which use the sound generator. Experimenting can be done in the immediate mode if desired; however, it may be easier to set up a program with INPUT statements to change the desired variables.

Now that we have a repeating chord and know something about the different note values, let's try writing a program to produce a melody.

Example five:
Playing a melody

Step 1
Identify the problem
Create a program to play a simple melody using single notes and notes played simultaneously. Vary the time of the notes.



Step 2
Outline the solution

Inputs
1. None

Process
1. Assign note values to variables.
2. Play beginning of melody twice.
3. Play end of melody once.

Outputs
1. Play the notes and chords of the melody.

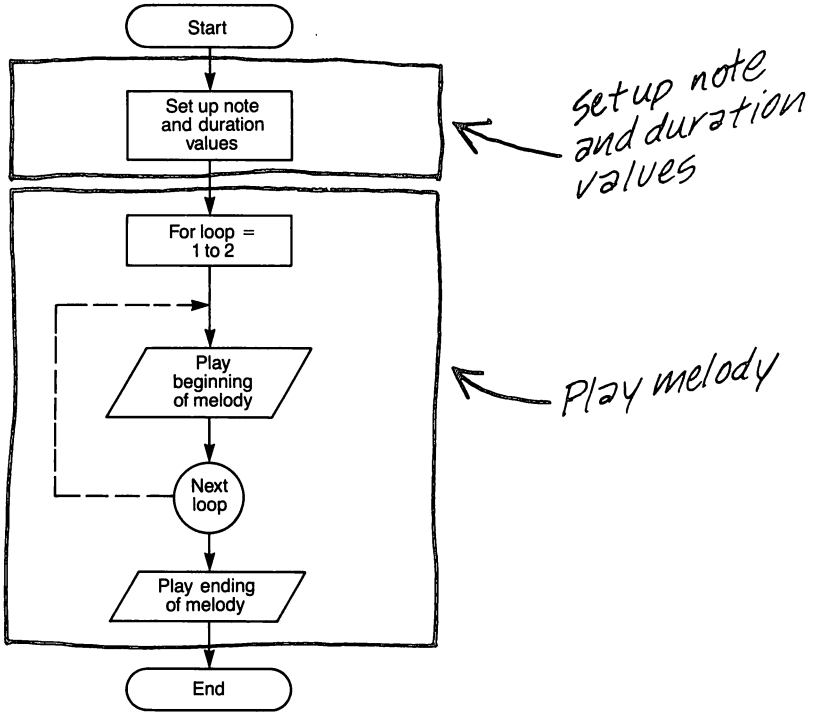
Step 3
Prepare a flowchart

Assign variable names

Inputs
None

Process
T = Length of time to play note(s)
V = Loudness of note(s)
LOOP = Controls how many times loop repeats
C = Middle C
D = D above middle C
E = E above middle C
F = F above middle C
G = G above middle C

Outputs
T = Length of time to play note(s)
V = Loudness of note(s)
LOOP = Controls how many times loop repeats
C = Middle C
D = D above middle C
E = E above middle C
F = F above middle C
G = G above middle C



Step 4
Write the code

Instruction	Explanation
10 REM A VERY SHORT TUNE	
20 T = 1000	Set time to 1 second
30 V = 10	Set volume to moderately loud
40 C = 262	Set C to middle C
50 D = 294	Set D above middle C
60 E = 330	Set E above middle C
70 F = 349	Set F above middle C
80 G = 392	Set G above middle C
90 FOR LOOP = 1 TO 2	Begin FOR-NEXT loop to play beginning of melody twice
100 CALL SOUND(T,E,V)	
110 T = 500	Set time to 1/2 second
120 CALL SOUND(T,E,V)	Play note E for 1/2 second
130 CALL SOUND(T,E,V)	Play note E again for 1/2 second
140 T = 1000	Set time to 1 second
150 CALL SOUND(T,E,V,G,V)	Play notes E and G simultaneously for 1 second
160 T = 500	Set time to 1/2 second
170 CALL SOUND(T,E,V,G,V)	Play E and G simultaneously for 1/2 second
180 CALL SOUND(T,E,V,G,V)	Play E and G simultaneously again for 1/2 second
190 T = 1000	Set time to 1 second
200 CALL SOUND(T,E,V)	Play notes E,F, C and D separately for 1 second each
210 CALL SOUND(T,F,V)	
220 CALL SOUND(T,C,V)	
230 CALL SOUND(T,D,V)	
240 NEXT LOOP	Repeat loop to play same sequence again, then go to next sequence
250 CALL SOUND(T,F,V)	Play notes F, E, D and C separately for 1 second each
260 CALL SOUND(T,E,V)	
270 CALL SOUND(T,D,V)	
280 CALL SOUND(T,C,V)	
290 T = 4000	Set time to 4 seconds
300 CALL SOUND(T,C,V)	Play note C for 4 seconds
310 END	

Note
Notice that some time durations are used more than once and a separate statement is used each time the duration is changed. A more efficient way to do this is to assign each of these durations to a variable; for example, T1 = 500 and T2 = 1000 at the beginning of the program. Then use the appropriate variable name in each statement; that is, use T2 in lines 100, 150, 200, 210, 220, 230, 250, 260, 270 and 280 instead of T, and use T1 in lines 120, 130, 170 and 180 instead of T. Then lines 20, 110, 140, 160 and 190 can be deleted.

Step 5
Run the program and debug it
This program shows how music can be created on the TI Home Computer.

Uses of computer sound

Some uses of sound from the TI Home Computer have been hinted at, but let's now take a closer look at some of them.

Alerting tones

Single tones of short duration are effective warning signals.

Sound can be used to alert a computer operator to an entry error. For example, some entries must be numeric only and entry of an alphabetic character will cause an error. Sometimes only numeric values within a certain range are allowed in a data entry, such as 1 through 12 for months. Therefore, if 13 is entered, it is an error. With a sound warning in addition to a visual warning, the operator is alerted that the data is unacceptable and must be re-entered. Another possibility for using sound as an operator alert is when a delete function is requested by the operator. If the delete is carried through, data will be destroyed, so an audible tone can get the attention of the operator to be sure the delete is what is desired.

Music

Short musical melodies are effective in games for either winning or losing messages.

We have discussed music using the examples given above. Music adds to the excitement of a game when a brief melody is played for winning. It also sparks interests in educational programs by giving a musical reward for correct answers. You can even program "sad" music for a wrong response that will gently let the student know that he/she has made an error.

The TI Home Computer also can be used to teach music because the tones are produced according to the musical scale. Individual notes over a range of five octaves can be produced and any three can be combined and played together to produce chords. One can also teach the basics of music theory by demonstrating various concepts.

Noise

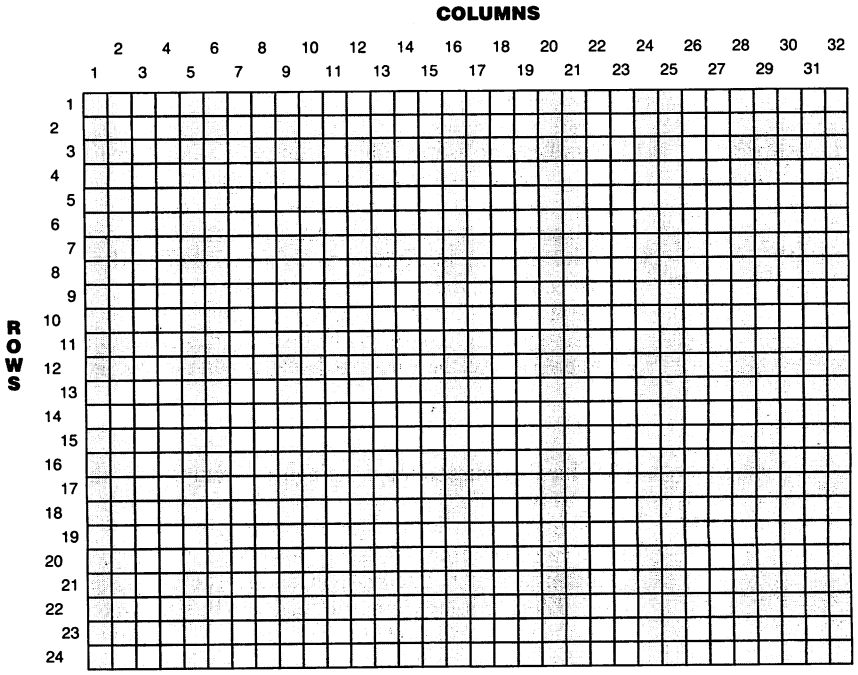
Sound effects make a game more interesting.

The noise generator is another important sound effect. It, along with the tones, can add to the feeling of realism in computer games. Sound effects add excitement to the high moments of a game. It can indicate such things as increasing speed of a moving object, the bounce of a ball against objects, the explosion of a bomb, and the ricochet of bullets.

Graphics

Graphics are plotted on a grid using two instructions.

Several fundamentals must be understood before trying to program graphics. The first is that the monitor or TV screen must be thought of as a grid which contains 24 rows and 32 columns as illustrated below.



Row and column layout of screen

On some monitor screens, columns 1, 2, 31 and 32 may be clipped off; thus, limiting you to 28 columns, 3 through 30. Experiment to determine how many are available on your screen. Each square on the grid is identified by a row number and a column number. These two numbers are called coordinates. For example, the coordinates 1, 1 mean the upper left-hand corner, the coordinates 8, 9 mean the 8th row and the 9th column, and the coordinates 14, 15 mean the 14th row and the 15th column.

The next step to understanding graphics is to learn the instructions to place information on the screen. There are two instructions:

CALL VCHAR for vertical characters
CALL HCHAR for horizontal characters

The CALL VCHAR instruction

The format for CALL VCHAR is illustrated in the following example:

CALL VCHAR(12,16,42)

This instruction places the character whose code number is 42 at a vertical and horizontal position defined by the first two numbers; that is, row 12 and column 16 (which is the center of the screen). The 42 in the third position is the assigned code number for the asterisk. All numbers, letters, symbols and the space are assigned code numbers from 32 to 127. These are defined in the Appendix. Here is another example:

CALL VCHAR(8,15,67)

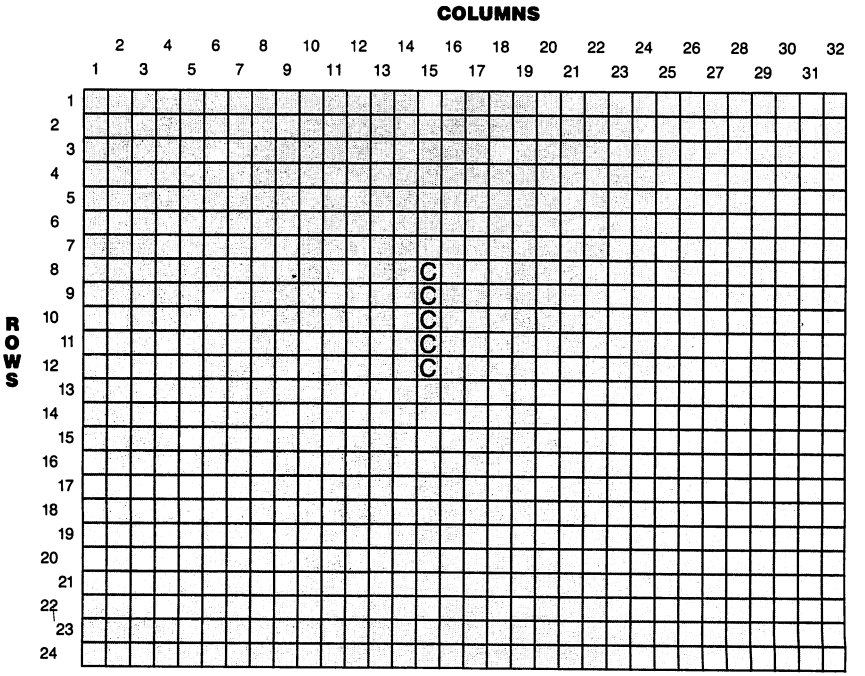
This statement tells the computer to place a C, defined by the number 67, in the eighth row and the fifteenth column.

This instruction can be expanded to have the specified character (including spaces) repeatedly printed in the vertical direction without having to write the instruction each time.

The following example illustrates this:

CALL VCHAR(8,15,67,5)

The statement is the same as in the above example except that a 5 has been added in the fourth position. The 5 specifies how many times to print the character. Therefore, this instruction tells the computer to print 5 C's vertically beginning at row 8, column 15, as shown below.



Output produced by CALL VCHAR (8, 15, 67, 5)

Printing continues at top of screen when position exceeds bottom of screen.

If the number of repetitions is greater than 24, printing continues at the top of the screen in the next column to the right after the bottom of the screen is reached in the specified column.

The CALL HCHAR instruction

The CALL HCHAR instruction is similar to the CALL VCHAR instruction. The format is:

CALL HCHAR(8,5,42)

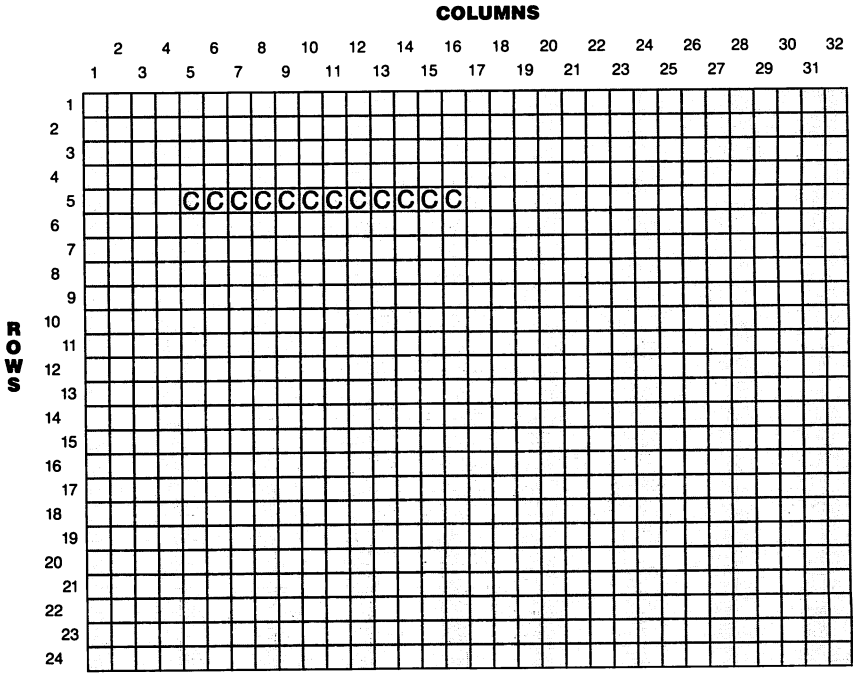
This statement will place an asterisk, (specified by the 42) at row eight and column 5. Another example is:

CALL HCHAR(5,5,67)

This statement prints a C (specified by the 67) at row 5, column 5. Repeated horizontal printing is available with CALL HCHAR. The number in the fourth position specifies how many times to print the character. For example:

CALL HCHAR(5,5,67,12)

This statement prints 12 C's horizontally across the screen beginning at row 5, column 5 as shown below.



Output produced by CALL HCHAR (5, 5, 67, 12)

Printing continues to left of screen when position exceeds right of screen.

If the number of repetitions is greater than 32, printing continues at the left edge of the screen in the next lower row after the right edge of the screen is reached in the specified row.

The CALL SCREEN instruction

Two more instructions allow graphics in 16 colors.

The CALL SCREEN instruction is needed to write a graphics program in color. The format is:

CALL SCREEN(n)

The parentheses enclose a number from 1 to 16 which represents one of the 16 available colors. The 16 colors and their codes are listed in the table below. If a CALL SCREEN statement is not used, the screen color is automatically set to light green (code 4) during a program run.

Color Codes

Color	Code
Transparent	1
Black	2
Med. green	3
Light green	4
Dark blue	5
Light blue	6
Dark red	7
Cyan	8
Med. red	9
Light red	10
Dark yellow	11
Light yellow	12
Dark green	13
Magenta	14
Gray	15
White	16

The CALL COLOR instruction

Another instruction needed for color is CALL COLOR. The format is:

CALL COLOR(2,5,15)

The characters are divided into twelve sets which include the numbers, letters, special symbols and the space. (The sets are defined in the Appendix.) Another four sets are reserved for you to create special characters. More on that later.

The first number within the parentheses in the instruction specifies the character set. In the example above, the 2 specifies character set 2 which includes character codes 40 through 47. Therefore, a number from 40 to 47 will need to be placed in a CALL VCHAR or CALL HCHAR statement in order for the character to be printed.

The second number within the parentheses specifies the foreground color. This is the color in which the characters are printed. The third number within the parentheses specifies the background color on which the character is printed. It is the part of the small rectangular space allowed for a character that is not filled up by the character. (It is *not* the screen color which is light green during a program run unless otherwise specified in a CALL SCREEN statement.) The same color codes shown in the table above are used for the foreground and background colors.

Code 1 is transparent; therefore, if the background color code is 1, the background color is the same color as the screen color. Thus, if a CALL SCREEN statement is not used, a transparent background has the light green color for the unfilled portion of the character printed on the screen. Any other screen color used with a transparent background color will produce similar results.

Example six:
Display all the screen colors

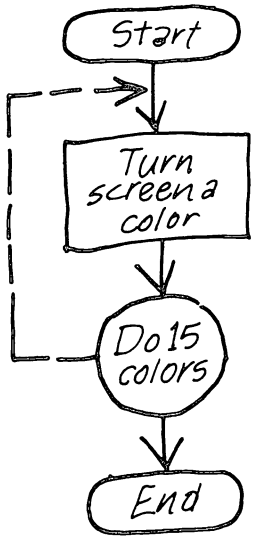
Step 1
Identify the problem
Show the 15 visible colors of the screen.
Omit color code 1.

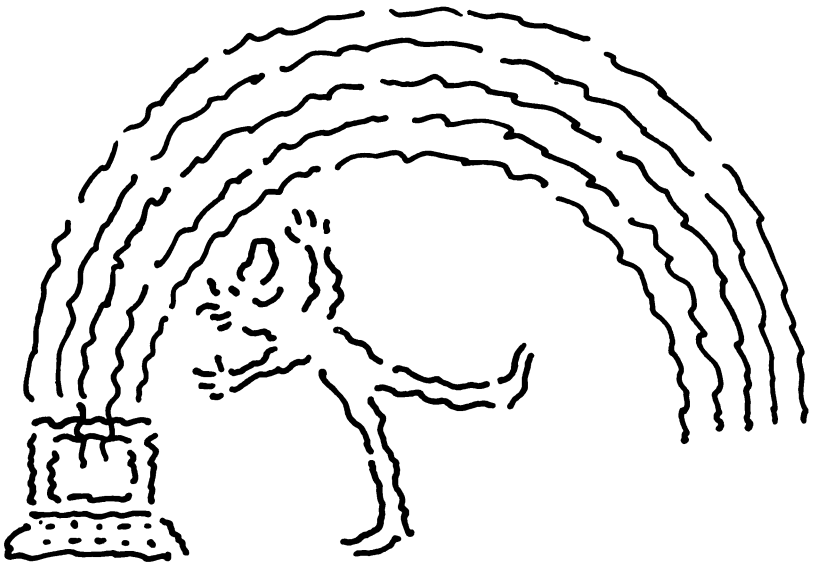
Step 2
Outline the solution

Inputs
1. None

Process
1. Do all screen colors except 1.
2. Hold each color for a few seconds.

Outputs
1. Have screen turn all colors except 1.





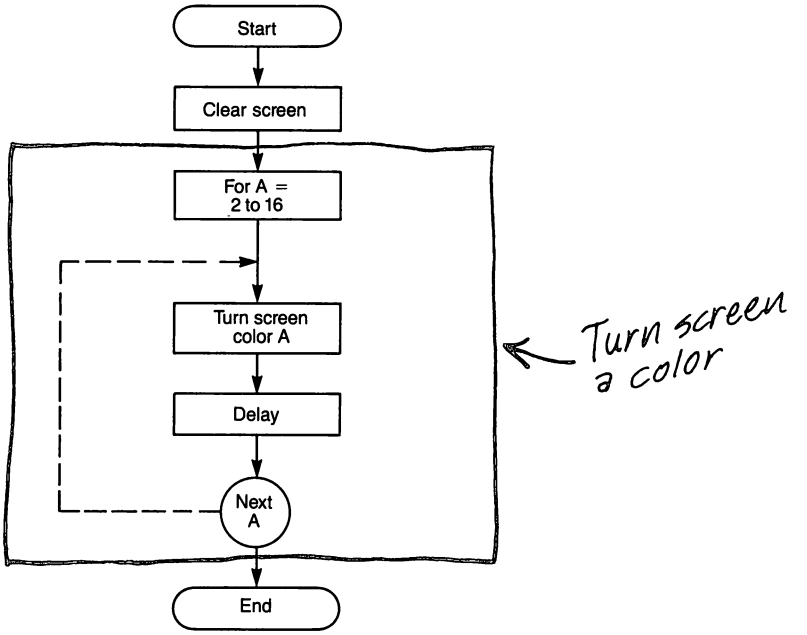
Step 3
Prepare a flowchart

Assign variable names

Input
None

Process
A = The value of the color to be displayed on the screen.
DELAY = Holds the color on the screen for a few seconds.

Outputs
None



Step 4
Write the code

Instruction	Explanation
10 REM SCREEN COLOR PROGRAM	
20 CALL CLEAR	
30 FOR A=2 TO 16	Begin loop to execute 15 times
40 CALL SCREEN(A)	Prints the screen color corresponding to the value in A
50 FOR DELAY=1 TO 200	Delay loop to keep screen color on screen
60 NEXT DELAY	
70 NEXT A	Repeat loop with new color
80 END	

Step 5
Run the program and debug it
Run the program to see the available screen colors.

Example seven:
Experimenting with color

The next program presents a different way of coloring the screen.

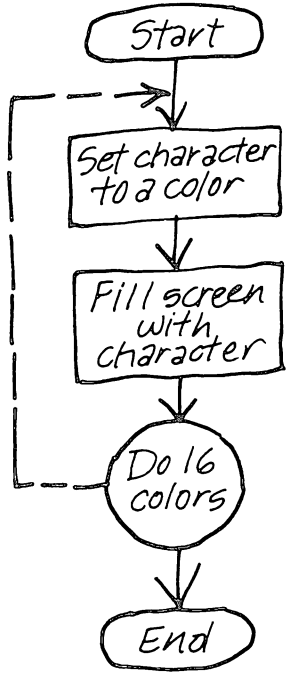
Step 1
Identify the problem
Fill the screen with the asterisk character using the same color for foreground and background.

Step 2
Outline the solution

Inputs
1. None

Process
1. Set foreground and background of character to the same color.
2. Do all colors.
3. Hold each color for a few seconds.

Outputs
1. Fill the screen with the character.



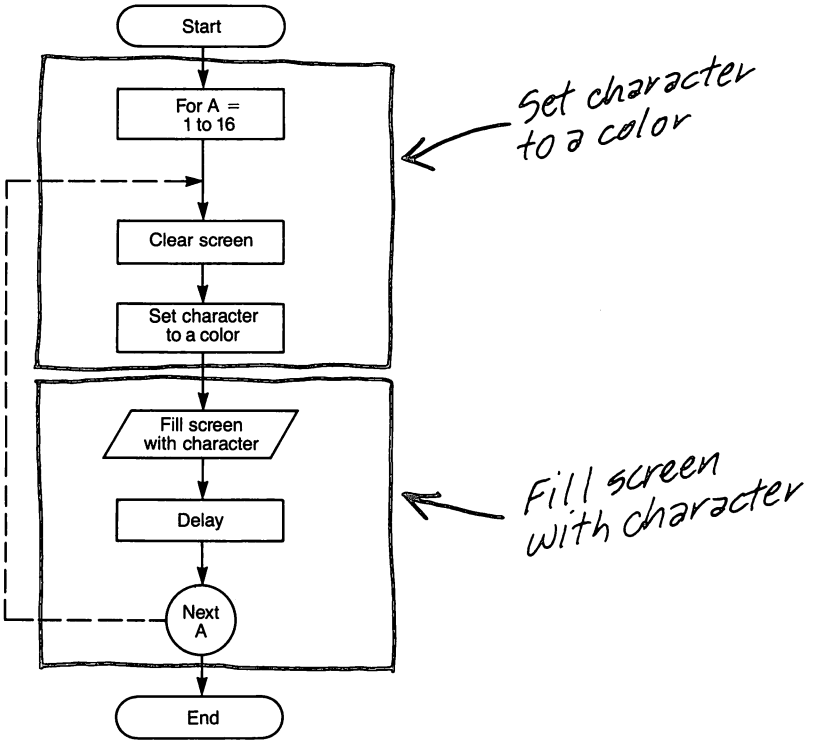
Step 3
Prepare a flowchart

Assign variable names

Inputs
None

Process
A = Loop variable which controls the colors
DELAY = Counter that holds the data on the screen

Outputs
None



Step 4
Write the code

Instruction	Explanation
10 REM COLOR PROGRAM	
20 FOR A = 1 TO 16	The loop repeats for 16 times to display all available colors
30 CALL CLEAR	
40 CALL COLOR(2,A,A)	The 2 specifies the second character set; both the foreground and background colors are set to value A which is the loop index
50 CALL HCHAR(1,1,42,768)	Starts printing asterisks at row 1, column 1; the asterisk is 42, and there are 768 repetitions, which means the screen is filled with asterisks printed row by row
60 FOR DELAY = 1 TO 200	Time delay loop
70 NEXT DELAY	
80 NEXT A	Loop for next foreground-background color
90 END	

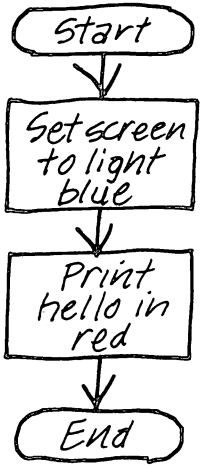
Step 5
Run the program and debug it
The program generates 16 different screens. The asterisk can't be seen because it is the same color as the background. Now add lines 25 and 75 and change line 40 as follows:

```
25 FOR B=1 TO 16
40 CALL COLOR(2,B,A)
75 NEXT B
```

and run the program again.

Example eight:
Print hello in color

Step 1
Identify the problem
Print the word HELLO on the screen using a light blue screen, transparent background and dark red letters.



Step 2
Outline the solution

Inputs
1. None

Process
1. Set characters to dark red color.

Outputs
1. Turn the screen to light blue color.
2. Place characters on the screen.

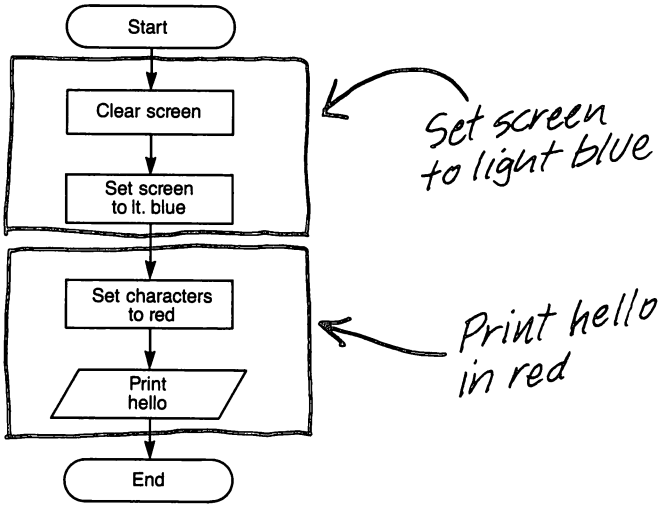
Step 3
Prepare a flowchart

Assign variable names

Inputs
None

Process
None

Outputs
None



Step 4
Write the code

Instruction	Explanation
10 REM PRINT HELLO ON SCREEN IN COLOR	
20 CALL CLEAR	
30 CALL SCREEN(6)	Sets screen color to light blue
40 CALL COLOR(6,7,1)	The H is in character set 6; 7 is dark red color of the character; 1 sets the character background to transparent
50 CALL COLOR(5,7,1)	Same as line 40 except E is in character set 5
60 CALL HCHAR(12,13,72)	72 is the code value for the letter H which is printed in row 12, column 13
70 CALL HCHAR(12,14,69)	69 is the code value of E which is printed in row 12, column 14
80 CALL HCHAR(12,15,76)	76 is the code value of L, which is printed in row 12, column 15
90 CALL HCHAR(12,16,76)	Same as line 80 except second L is printed in column 16
100 CALL HCHAR(12,17,79)	Letter O is also in character set 6; code value for O is 79; O is printed in row 12, column 17
110 GOTO 110	Loops on itself to hold display until the CLEAR function (FCTN 4) is used

Step 5
Run the program and debug it
The program prints HELLO in dark red on a transparent background. Since the background is transparent, the light blue screen color is also the background color.

Example nine:
Generating both graphics and sound

Step 1
Identify the problem

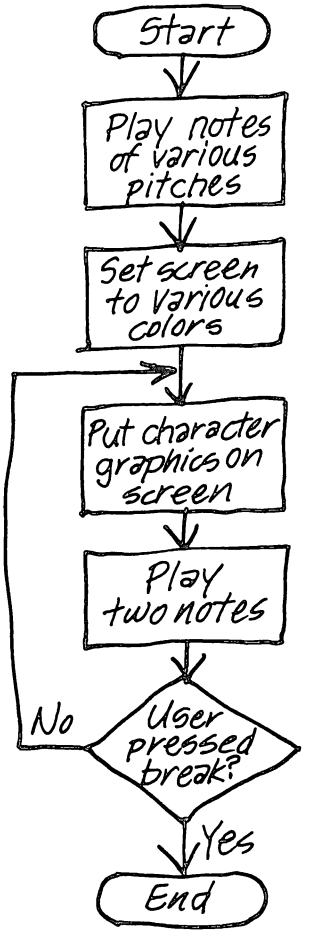
Write a program which prints a controlled sequence of characters in both the horizontal and vertical directions. In addition, generate sound effects during each sequence.

Step 2
Outline the solution

Inputs
1. None

Process
1. Calculate various pitches to play.
2. Generate sequences of characters to print on the screen.
3. Delay between colors and pitches.

Outputs
1. Play various notes.
2. Turn screen various colors.
3. Draw horizontal and vertical rows of characters on the screen.



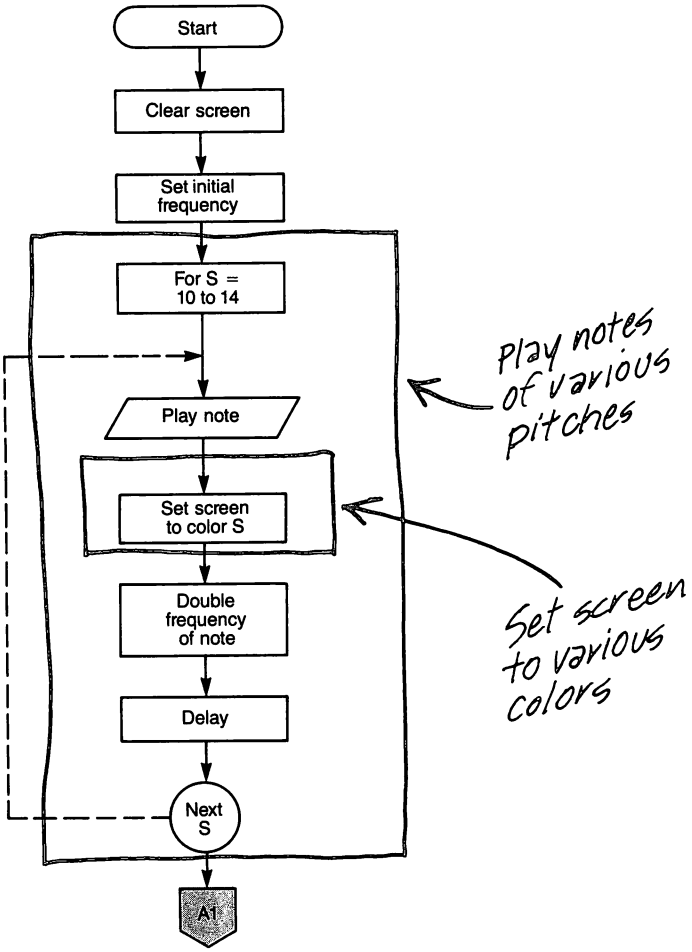
Step 3
Prepare a flowchart

Assign variable names

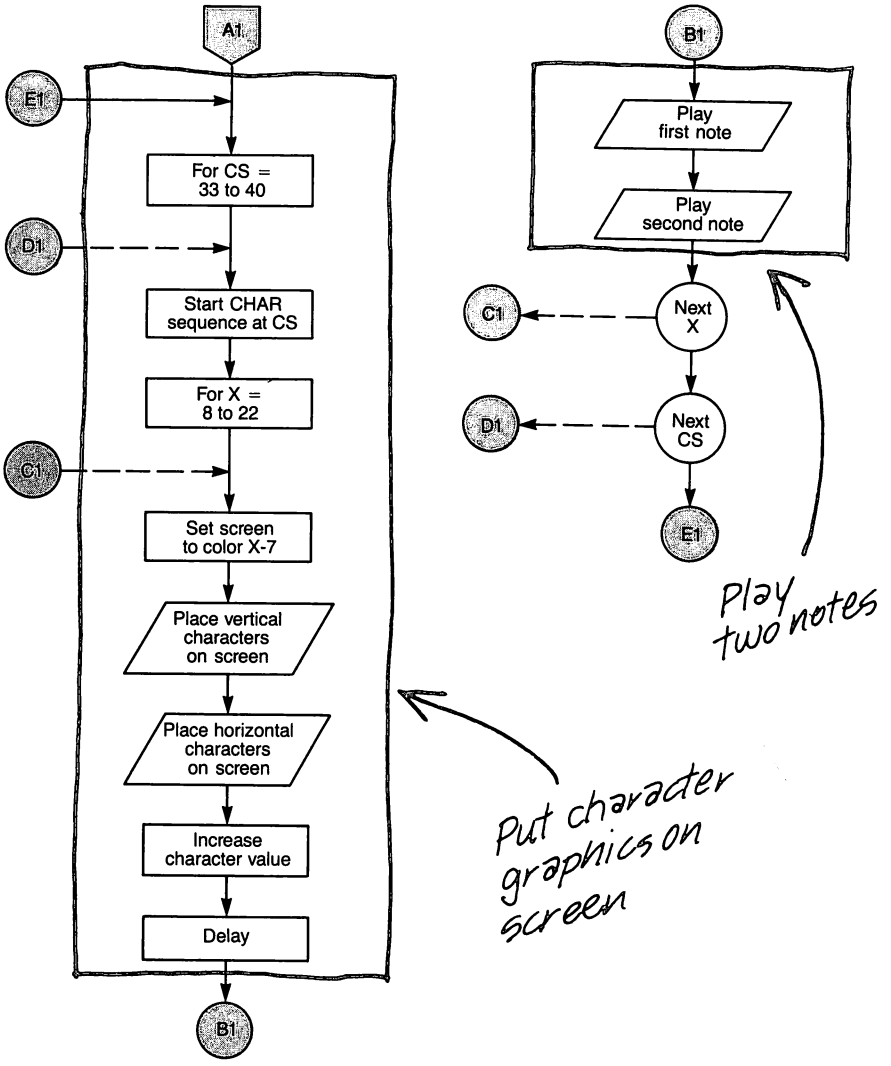
Inputs
None

Process
F = Frequency of notes
S = Screen color
DELAY = Loop counter for delay loop
CS = Starting value of character number
CHR = Character number for plotting
X = Loop counter for graphics loop

Outputs
None



Flowchart continued on next page


A1
E1
For CS = 33 to 40
D1
Start CHAR sequence at CS
For X = 8 to 22
C1
Set screen to color X-7
Place vertical characters on screen
Place horizontal characters on screen
Increase character value
Delay
B1
B1
Play first note
Play second note
C1
Next X
D1
Next CS
E1
Play two notes
Put character graphics on screen

Step 4
Write the code

Instruction	Explanation
100 CALL CLEAR	
110 F = 110	Set beginning frequency
120 FOR S = 10 TO 14	Begin loop to print screens in varying color
130 CALL SOUND(500,F,2)	Sound tones of increasing frequency for ½ second
140 CALL SCREEN(S)	Change screen color
150 F = F*2	Increase frequency of tone
160 FOR DELAY = 1 TO 200	Delay loop
170 NEXT DELAY	
180 NEXT S	
190 FOR CS = 33 TO 40	Loop to step through characters
200 CHR = CS	Set initial character value
210 FOR X=8 TO 22	Loop to control positioning of character and patterns
220 CALL SCREEN(X-7)	Change screen color
230 CALL VCHAR(4,X,CHR,15)	Position the character and repeat 15 times vertically
240 CALL HCHAR(X-4,8,CHR,15)	Position the character and repeat 15 times horizontally
250 CHR=CHR+8	Add 8 for a new character value
260 FOR DELAY = 1 TO 200	Delay loop
270 NEXT DELAY	
280 CALL SOUND(1000,330,2)	Sound 330 Hz tone for one second
290 CALL SOUND(456,456,2)	Sound 456 Hz tone for slightly less than ½ second
300 NEXT X	
310 NEXT CS	
320 GOTO 190	Start sequence all over again; continues until the CLEAR function (FCTN 4) is used

The first loop sequences the screen through five different colors accompanied by tones of increasing pitch.

The second loop prints various characters in vertical and horizontal patterns and sounds two tones.

Step 5
Run the program and debug it
The program demonstrates combined sound and graphics in a nonsense application.

Developing your own character sets

Earlier we mentioned that some character set numbers are reserved so programmers can define character sets of their own. This permits them to write special applications that otherwise might not be possible. Some examples of special character sets are the letters of a foreign language, common mathematical symbols and figures for games. Development of character sets takes thought and experience, but interesting results can be achieved.

Uses of graphics

Computer graphics can produce animation for games and simulation, and bar charts for business reports. The use of color adds interest and versatility.

Most of us have heard the expression "one picture is worth a thousand words." The same can be said of computer graphics. Computer graphics can show simulation of activities that otherwise would be impossible to produce because of safety or economic considerations. Graphics that utilize good format and choice of colors add interest and appeal to programs.

Animation is also possible with the TI Home Computer graphics. Animation is accomplished by showing a series of fixed pictures that rapidly follow one another in sequence so that they appear to be a moving picture. The technique is similar to that used to produce motion pictures on film. Animation takes time, patience and experimentation.

Graphics can be used in educational, entertainment and resource management programs. In education, they present the teaching material in a pleasing and motivating manner. In entertainment, color graphics add excitement and realism to games. In resource management, they make the screens more pleasant and efficient to use. For bar graphs, colored bars can be keyed to represent different things. Thus, well-planned and -developed graphics can greatly enhance any computer applications.

Introduction

Speech

The optional speech synthesizer and Speech Editor cartridge provide the capability for the computer to talk.

Words and phrases in the vocabulary can be combined to produce sentences that the computer can say. Pauses are controlled by punctuation symbols.

Punctuation

Combining words and phrases

Homographs

Speech in the program mode

The CALL SAY instruction

These instructions are used in programs to tell the computer to talk.

Example one:	A program with speech using string expressions
Example two:	Modifying example one to use string variables

The CALL SPGET instruction

Uses of the speech synthesizer

The internal sound system

The CALL SOUND instruction

Single tones, multiple tones, and noise can be produced internally under control of your program.

Example three:	Playing the C major chord
Example four:	Playing three notes separately and as a chord

Experimenting with time and volume

By varying time and volume, music can be produced.

Example five:	Playing a melody

Uses of computer sound

Short "beeps" are useful for alerting people.

Alerting tones

Music

Noise

Sound effects for games can be produced by the noise generator.

Graphics

The CALL VCHAR instruction

The CALL HCHAR instruction

These instructions control the print position of characters by specifying coordinates of screen locations.

The CALL SCREEN instruction

The CALL COLOR instruction

These instructions control the colors produced on the screen.

Example six:	Display all the screen colors
Example seven:	Experimenting with color
Example eight:	Print hello in color
Example nine:	Generating both graphics and sound

Developing your own character sets

Uses of graphics

Summary map

Well-planned and well-developed graphics, especially in color, can greatly enhance any application.

Glossary

Address: A group of bits, usually 8 or 16, that identify a unique location in memory, or a peripheral device such as a printer.

Alphanumeric: The letters, numbers or combination of letters and numbers which are used by the computer in a form called strings.

Application: A particular use for a computer program.

Arithmetic and Logic Unit (ALU): A portion of the Central Processor Unit which performs mathematical operations for calculations and logical operations to make decisions.

Arrays: A collection of information all pertaining to the same subject. Arrays can be one-dimensional (such as a grocery list) or two-dimensional (such as a weight-distance postage chart).

ASCII: American Standard Code for Information Interchange. This is the standard code used to define the letters, numbers and special functions of the computer keyboard in the language of the computer.

BASIC: Beginners' All-Purpose Symbolic Instruction Code. A user-friendly language used in most home and many personal and small business computers which allows the user to communicate with the computer.

Binary Numbers: Base 2 numbers have two values, a 1 and a 0. They are used to make machine language instructions.

Bit: An acronym meaning binary digit. A bit is a 1 or a 0 and is the smallest possible piece of computer information.

Branch: A place in the computer program where the computer quits following the normal step-by-step sequence and "branches" to a step directed by the program itself.

Bus: A path inside the computer which is common to many devices (such as the address or data bus). When an address is placed on the address bus, it goes to every device connected to the bus at the same time, but only the device which has that address will respond.

Byte: A group of eight bits which the computer treats like one unit.

Central Processor Unit (CPU): The brain of the computer. All information flows into and out of the CPU. It uses the information to make decisions, to perform mathematical operations, or to make another device operate (such as the printer).

Character: Any single symbol which can appear on the screen such as letters, numbers, punctuation marks, etc.

Code: A term often used to describe the user's computer language (such as BASIC). When you "write code", you are writing a computer program in a user language. Also, the combination of 1's and 0's that represent digital signals to convert keyboard inputs to machine language.

Command: An operation the computer performs in the immediate mode.

Computer: A device that uses a stored program for problem-solving, record keeping, entertainment, education, etc. (such as the TI-99/4A Home Computer).

Computer Aided Instruction (CAI): Term used for educational instruction by using a computer with special software.

Data: Another name for information that the computer uses.

Documentation: The printed materials such as instruction manuals that explain hardware and software.

Fetch: The act of the CPU when it obtains the next instruction from memory.

Firmware: A name for software stored in read-only memory.

Floppy Disk or Diskette: A circular, flat device used to store data. For home computers, the magnetic floppy disk or diskette that is 5¼ inches in diameter is often used.

Flowchart: A diagram showing the logical flow of a program.

Graphics: Refers to the ability of a computer to create and display non-alphanumeric characters such as bar graphs and pictures.

Hardware: Refers to all the external and internal physical parts of a computer system such as the keyboard, monitor, cassette recorder, cables, etc.

Home Computer: A desk top or smaller computer designed for education, entertainment and general household computing needs.

Input/Output (I/O): Term used to define the direction of external data flow to/from the CPU. Data going into the CPU is said to be input, data coming from the CPU is said to be output.

Instruction: Term used to define an operation or conditional operation in a computer.

Integrated Circuit: A circuit whose connections and components are fabricated into one integrated structure on a certain material such as silicon.

K: In computers, due to the use of the binary number system, K indicates a multiplier of 1,024. Thus, 1K = 1,024, 2K = 2,408, etc.

Load: Computer term used to describe the placement of data from some source into memory or internal CPU registers.

LOGO: A special computer language that is useful for learning to program.

Loop: Term used to describe an operation in a computer program which is performed many times in succession by executing a set of selected instructions over and over again.

Machine Language: The language that the computer understands. See binary numbers.

Memory: The part of a computer where programs and data are stored internally. Also, external media such as cassette tape and magnetic disk.

Menu: A display on the screen which lists the choices available to the user. (Also called a directory or catalog.)

Microcomputer: The name usually given to a small computer which uses a microprocessor for its CPU.

Microprocessor: A large-scale integrated circuit which contains all of the functions of a computer processor. These include CPU, ALU, timing and control circuits.

Modem: A device that converts computer signals to a type of signal that can be transmitted over telephone lines.

Operating System: A program contained inside the computer to manage how the units of a computer system operate with each other.

Output: See Input/Output.

Peripheral: A device or subsystem external to the computer such as a printer, monitor, cassette recorder, etc.

Personal Computer (also Professional Computer): A small computer that is generally used for home or business applications. It usually requires more knowledge to operate than a home computer.

Preprogrammed Software: Software that has been prepared and tested and is ready to load and use.

Program: Term used to describe the set of instructions the computer uses to perform a specific function.

Register: A small memory usually storing only 8 or 16 bits to hold instructions or data temporarily.

Software: Term used to describe the computer programs that make the hardware work.

Speech Synthesizer: A solid-state electronic device that artificially forms human speech from basic sounds or words stored in a memory.

String: A group of alphanumeric characters in a program which form words or number patterns.

Subroutine: A small portion of a computer program which performs a specific function. For example, a subroutine may perform all of the mathematical computations as a part of a program to determine the total cost of a loan.

User-Friendly: Term which describes a computer and software that is easy to use by a person who knows very little about computers.

Variable: Term used to describe a symbol which can be defined and changed in any way the user wants within a program. For example, A = 5, or A = 29.

Word: A computer word is composed of one or more bytes which is treated by the computer as a whole when performing its operations.

Word Processor Program: Software that allows a computer to be used to prepare and edit text material.

Appendix

Available Software for the TI-99/4A Home Computer

Information Management

Cartridges
Home Financial Decisions
Household Budget Management
Securities Analysis
Personal Record Keeping
Tax/Investment Record Keeping
Personal Real Estate
Personal Report Generator
TI Writer
Microsoft™[1] Multiplan™[1]
Terminal Emulator II

Diskettes and Cassettes
Mailing List
Personal Finance Aids
Checkbook Manager
Business Aids Library
Financial Management
Inventory Management
Invoice Management
Cash Management
Lease/Purchase Decisions
Personal Tax Plan[2]
TI-Mini-Writer

Education

Cartridges
Early Learning Fun
Beginning Grammar
Number Magic
Video Graphs
Early Reading[3]
Reading Fun[3]
Reading On[3]
Reading Roundup[3]
Reading Rally[3]
Reading Flight[3]
Addition/Subtraction 1 and 2[3]
Multiplication 1[3]
Division 1[3]
Numeration I and II[3]
*Computer Math Games I, II, III, IV, VI[4]
Alien Addition[5]
Minus Mission[5]
Alligator Mix[5]
Meteor Multiplication[5]
Demolition Division[5]
Dragon Mix[5]
Word Invasion[5]
Word Radar[5]
Milliken Math Series[6] (11 Cartridges)
Scholastic Spelling[7] (4 Cartridges)
TI LOGO II
Early LOGO Learning Fun
Key to Spanish[8]
Weight Control and Nutrition
Physical Fitness
Music Maker
Touch Typing Tutor
Video Chess

Diskettes and Cassettes
Music Skills Trainer
Computer Music Box
Market Simulation
Basketball Satistician
Bridge Bidding I, II and III
Speak & Spell[9] Program
Speak & Math[9] Program
Spell Writer
PLATO®[10]

Entertainment

Cartridges
Parsec
Tombstone City: 21st Century
TI Invaders
MunchMan
Munch Mobile™[11]
MoonMine
Sneggit
Car Wars
Alpiner
Othello[12]
Chisolm Trail
Football
Video Games I
Hunt the Wumpus
Indoor Soccer
Mind Challengers
A-MAZE-ING
The Attack[13]
Blasto[13]
Blackjack and Poker[13]
Hustle[13]
ZeroZap[13]
Hangman[13]
Connect Four[13]
Yahtzee
Terry Turtle's Adventure[13]
I'm Hiding[13]
Honey Hunt[13]
Sound Track Trolley[13]
Championship Baseball[13]
Space Bandit[13]
Big Foot[13]
Super Fly[13]
Sewermania[13]
Meteor Belt[13]
M*A*S*H[14]

Diskettes and Cassettes
Tunnels of Doom
Entrapment
Adventure[15]
Adventureland[15]
Secret Mission[15]
Voodoo Castle[15]
The Count[15]
Strange Odyssey[15]
Mystery Fun House[15]
Pyramid Of Doom[15]
Ghost Town[15]
Savage Island I & II[15]
Golden Voyage[15]
Mystery Melody
Oldies But Goodies I and II
Saturday Night Bingo
Draw Poker

Computer Programming

Cartridges
Speech Editor
Editor/Assembler
Mini-Memory
Extended BASIC

Diskettes and Cassettes
Text-to-Speech
Programming Aids, I, II and III
Pascal Development System
UCSD[16] Pascal Compiler
UCSD P-System[16] Assembler/Linker
UCSD P-System Editor/ Filer/Utilities
Beginner's BASIC Tutor
Teach Yourself BASIC[17]
Teach Yourself Extended BASIC
TI PILOT
Course Designer Authoring System

Math and Engineering

Cartridges
Statistics

Diskettes and Cassettes
Math Routines Library
Graphing Package
Electrical Engineering Library
AC Circuit Analysis Library
Structural Engineering Library

*Computer Math Games I will not be available until mid 1984.

[1]Microsoft and Multiplan are trademarks of Microsoft Corp.
[2]Developed by Aardvark Software, inc.
[3]Developed by Scott, Foresman and Co.
[4]Developed by Addison-Wesley Publishing Co.
[5]Developed by Developmental Learning Materials, Inc. (DLM)
[6]Developed by Milliken Publishing Co.
[7]Developed by Scholastic, Inc.
[8]Developed by Westinghouse Learning Corp.
[9]Trademark of Texas Instruments Incorporated
[10]PLATO is a trademark of Control Data Corp. U.S.A. PLATO courseware is manufactured under license by Texas Instruments.
[11]Manufactured under license from SNK Electronics Corp.
[12]Trademark of Gabriel Industries
[13]Milton Bradley Voice Command Video Game Series Manufactured by Texas Instruments under license from Milton Bradley Company Trademarks of Milton Bradley Company.
[14]Manufactured under license from Fox Video Games, Inc.
[15]Developed by Scott Adams
[16]UCSD, UCSD Pascal and UCSD P-System are trademarks of the Regents of the University of California.
[17]Developed in conjunction with Wolfdata.

ASCII character codes

The defined characters on the TI-99/4A Home Computer are the standard ASCII characters for codes 32 through 127. The following chart lists these characters and their codes.

ASCII Code	Character	ASCII Code	Character	ASCII Code	Character
32	(space)	65	A	97	a
33	! (exclamation point)	66	B	98	b
34	" (quote)	67	C	99	c
35	# (number or pound sign)	68	D	100	d
36	$ (dollar)	69	E	101	e
37	% (percent)	70	F	102	f
38	& (ampersand)	71	G	103	g
39	' (apostrophe)	72	H	104	h
40	((open parenthesis)	73	I	105	i
41	) (close parenthesis)	74	J	106	j
42	* (asterisk)	75	K	107	k
43	+ (plus)	76	L	108	l
44	, (comma)	77	M	109	m
45	- (minus)	78	N	110	n
46	. (period)	79	O	111	o
47	/ (slant)	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z
58	: (colon)	91	[(open bracket)	123	{ (left brace)
59	; (semicolon)	92	/ (reverse slant)	124	¦
60	< (less than)	93	] (close bracket)	125	} (right brace)
61	= (equals)	94	^ (exponentiation)	126	~ (tilde)
62	> (greater than)	95	_ (line)	127	DEL (appears on screen as a blank)
63	? (question mark)	96	` (grave)		
64	@ (at sign)				

Character code sets

These character codes are grouped into twelve *sets* for use in color graphics programs.

Set #	Character Codes	Set #	Character Codes
1	32-39	7	80-87
2	40-47	8	88-95
3	48-55	9	96-103
4	56-63	10	104-111
5	64-71	11	112-119
6	72-79	12	120-127

Index

Arrays: 5-19

BASIC: 3-4
BASIC Interpreter: 3-4; 4-5
Bit: 3-3
Block Diagram: 3-4
Branch:
  Conditional: 4-23
  Unconditional: 4-22
Byte: 3-3, 8

CALL CLEAR: 4-9
CALL COLOR: 4-31; 9-24
CALL HCHAR: 4-32; 9-23
CALL SAY: 9-5
CALL SCREEN: 4-31; 9-24
CALL SOUND: 9-12
  Duration: 9-13, 17
  Volume: 9-13, 17
  Alerting Tones: 9-20
  Music: 9-20
  Noise: 9-20
CALL SPGET: 9-11
CALL VCHAR: 4-32; 9-22
Cassette Tape: 3-9
Central Processor Unit: 1-14
Computer Aided Instruction (CAI): 2-6
Computer:
  Computations: 7-3
  Data Management: 7-2
  Education: 6-3, 4, 6
  Financial Planning: 7-4
  Functions: 1-14; 6-2
  Immediate Mode: 4-4, 5, 8
  Literacy: 6-4
  Operation: 4-7
  Program Mode: 4-4, 5, 10, 5-2
  Records: 7-2, 3, 4
  Word Processing: 6-6

DIM: 5-20
Documentation: 3-2

EDIT: 4-14
Entertainment: 1-5; 8-2, 3
Expansion System: 3-11
Firmware: 3-2
Floppy Disk: 3-10
Flowchart: 5-2, 4, 6, 16, 17
Formatting: 4-19
FOR-NEXT: 4-26

GOSUB: 4-30
GOTO: 4-22
Graphics: 4-31; 9-21, 36
  Color Graphics: 4-31
  Developing Character Sets: 9-36

Home Computer: 1-5

IF-THEN: 4-23
Initialization Program: 3-6
INPUT: 4-17, 18
Input/Output: 1-14
INT: 4-35
Integrated Circuit: 1-12

LET: 4-16
Line Numbers: 4-10
LIST: 4-13
LOGO: 2-10
Loops: 4-22, 27, 28

Machine Language: 3-3
Memory: 1-14; 3-7
Menu: 2-5
Modem: 3-13

NEW: 4-15

PRINT: 4-8
Printer: 3-13
Processing: 4-6; 5-4
Program: 4-10
  Adding Lines: 4-12
  Creating: 5-2
  Debugging: 5-5
  Editing: 4-14
  Stopping and Starting: 4-11
Punctuation:
  INPUT Statements: 4-19
  PRINT Statements: 4-20, 21
  Speech: 9-3

Random Access: 3-8
RANDOMIZE: 4-34
Register: 3-6
REM: 4-17
RETURN: 4-30
RES: 6-10
Ribbon Cable: 3-12
RND: 4-33
ROM: 1-15
RS-232: 3-13
RUN: 4-10

Sample Programs:
  Accumulating a Sum: 4-29
  Adding Program Line: 4-13
  Auto Cost 1: 5-11
  Auto Cost 2: 5-14
  Auto Cost 3: 5-18
  Calculate Age: 6-26
  Calorie counter: 7-6
  Card Game: 8-19
  Checkbook Balancer: 7-9
  Color/Characters: 9-35
  Comic Quiz: 8-25
  Dice Game: 8-14
  Fahrenheit/Celcius Conversion: 4-36
  Function: 6-34
  Geography Test: 6-19
  Grade Averaging: 5-24
  Grade Book: 6-31
  Making Tea: 5-7
  Math Drill: 6-9
  Math Drill 1: 6-10
  Math Exercise: 6-13
  Paint Program: 7-14
  Paint Program 2: 7-17
  Print HELLO: 4-11
  Print HELLO (color): 9-31
  Printing, Adding: 4-15
  Recipe: 7-21
  Robot: 8-6
  Sample Computation: 4-24
  Screen Color: 9-26
  Screen Color 2: 9-29
  Short Tune: 9-19
  Simple Adding: 4-12
  Simple Sounds: 9-17
  Sine Wave: 8-10
  Speech Program: 9-7
  Speech Program 2: 9-10
  Time Delay Loop: 4-28
  Using CALL COLOR: 4-33
  Using CALL HCHAR: 4-33
  Using CALL SCREEN: 4-33
  Using CALL SOUND: 9-12, 9-15
  Using FOR-NEXT: 4-26
  Using GOSUB and RETURN: 4-30
  Using GOTO: 4-22
  Using IF-THEN: 4-23
  Using INPUT: 4-17, 18
  Using INT: 4-35
  Using LET: 4-16, 17
  Using Nested Loops: 4-27
  Using PRINT TAB: 4-22
  Using Punctuation: 4-20, 21
  Using RANDOMIZE: 4-34
  Using RND: 4-34
  Using SPGET: 9-11
  Using String Variables: 4-25
  Using Variables: 5-9
SIN: 8-7
Software: 1-14
Software List: A-1
Speech: 9-3
  Creating Phrases: 9-4
  Homographs: 9-4
  Punctuation: 9-3
Speech Synthesizer: 2-4; 3-12; 9-11
Spreadsheet: 2-23
Structured Programming: 5-16, 17

TAB: 4-21
Track, Disk: 3-11

Variables: 4-15
  String Variables: 4-16, 25

Word Processor: 2-23; 6-6

How to Feel at Home with a Home Computer

If you're thinking about buying

What are home computers for?
This book answers that question for anyone thinking about buying a home computer.

Here, in one clearly-written, fully-illustrated, easy-to-understand volume, you'll learn how easy it is to use a home computer — and, how a home computer, thanks to preprogrammed software, can help you educate your children, manage your personal finances and entertain the whole family.

The section on programming will encourage you to discover and explore the intricacies of writing your own programs. You'll be surprised at how easy it really is.

If you already own one

This book is for you, too.
If you already own a home computer, this book will greatly enhance its value by showing you how to do more with it — how to use your home computer to its fullest capability.

You'll get an in-depth look at and come away with a good feel for programming — i.e., having your home computer do exactly what you want it to do. You'll see how the rich, versatile BASIC programming language makes writing and executing your own programs rewarding . . . and fun.

Here's what else you'll like about this book:

- Fully illustrated
- Easy-to-read format
- Step-by-step instructions
- Written in clear, concise, easy-to-understand language — no computerese
- Content highlights throughout the text
- Guide maps begin and end each chapter
- Actual programs that run and perform a task
- Program examples in Education, Entertainment and Information Management

About the authors

Gary G. Bitter, Ph.D., is a Professor of Computer Education at Arizona State University who has concentrated for the last 15 years on teaching teachers about computers and the use of computers in the education process.

Roger S. Walker, Ph.D., is a Professor of Computer Science and Engineering at the University of Texas at Arlington who has a strong interest in promoting computer literacy in the home and school.

About the examples

The TI-99/4A is used for all examples. The examples are a valuable guide whether you have a computer or not, and help you get the most out of any home computer you may have.

TEXAS INSTRUMENTS

LCB 8183
ISBN 0-89512-097-6

Printed in USA
264 Pages