Elementary Amiga BASIC

C. Regena

A clear and concise guide to the Commodore Amiga personal computer's BASIC. Includes dozens of program examples to type in and use. For beginning and intermediate programmers.

A **COMPUTE! Books** Publication $14.95


Elementary Amiga BASIC

C. Regena

COMPUTE! Publications, Inc.
Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies
Greensboro, North Carolina

Printed in the United States of America

10 9 8 7 6 5 4

ISBN 0-87455-041-6

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is part of ABC Consumer Magazines, Inc., one of the ABC Publishing Companies, and is not associated with any manufacturer of personal computers. Amiga is a trademark of Commodore-Amiga, Inc.

Contents


Foreword

Elementary Amiga BASIC is a complete primer in the fundamentals of BASIC programming. Whether you are just starting to program or have worked with BASIC on another computer, you'll find all the commands, statements, and techniques you will need for effective programming on the Amiga. Every aspect of Amiga BASIC is demonstrated, with plenty of practical example programs which you can type in and use or modify to meet your own needs.

You will learn to use the powerful features of Commodore's versatile 16-bit computer to create your own screens and windows, create and display menus, display text, and control the mouse in order to exploit the Amiga's sophisticated screen display capabilities.

You will learn to create colorful graphics in several modes, including sprites and animated objects. And controlling the Amiga's advanced sound chips from BASIC, you will be able to create sound effects, compose and play music, and program the computer to speak in different voices and accents—even to speak a foreign language.

Elementary Amiga BASIC also offers a solid grounding in educational programming, providing a wide variety of easy-to-understand, interactive programs which can be used in the classroom or at home by students of all ages.

> All the programs in this book are ready to type in and run. If you prefer not to type in the programs, however, you can order a disk which includes the programs in the book. Call toll-free 1-800-346-6767 (in New York, call 212-887-8525) or use the coupon found in the back of this book.


Preface

I appreciate the opportunity to try BASIC programming on another computer. It's fun to experiment with new commands and new features. The main purpose of this book is to help you enjoy your Amiga computer by making it do what you want it to do. I hope to give you explanations of how Amiga BASIC works and to get you started on your own programming. I have tried to include a variety of programs illustrating the versatility of the Amiga. Feel free to take these programming ideas and customize them for your own needs.

I offer a special thanks to Stephen Levy, Editor of COMPUTE! Books, who tried to keep me informed about the Amiga as the computer and the BASIC programming language were being updated and revised. Thanks also to members of the COMPUTE! staff who helped make this book possible.

This book is dedicated to my baby son, Brett Lynn Whitelaw, who made his arrival the same week as my Amiga came. He has had to share his new life with the computer as I have been writing this book.

C. Regena
January, 1986


Chapter 1

Getting Started

Chapter 1

Getting Started

In the few years since microcomputers were introduced, home computers have become more powerful and less expensive and much easier to use. It's possible to use a computer without knowing a thing about programming, because of the wide variety of ready-to-use application programs which only require you to select the program you want, load it, and run it.

However, to get the computer to do exactly what you want it to do, you may want to customize someone else's program or write your own. This book will teach you the fundamental skills necessary to program the Amiga in BASIC. I have included a variety of programs which you may type in and use as they are or modify to better fit your needs.

We will discuss the Amiga BASIC programming language that is packaged with your Amiga, so you won't need to buy anything extra to start programming. In addition, all of the instructions in this book will work without expanded memory or any extra peripherals.

If you already know how to program in BASIC for another computer, you will find that most Amiga BASIC commands are familiar, although the syntax of some may be slightly different and there are some new commands which give you an introduction to the Amiga's special graphics, sound, and speech features.

We can't possibly include everything here, of course, but COMPUTE! will be publishing other books and articles for more advanced programmers which will go into more detail for the Amiga's special features.

I will assume you have the Amiga BASIC disk and manual that came with your computer. The manual is a handy reference guide, containing descriptions and syntax of the available commands, and should be used along with this book.

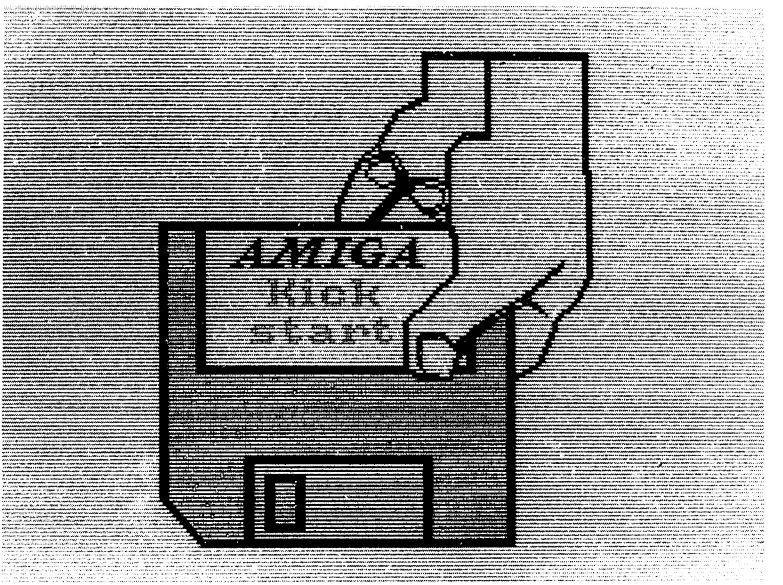
How to Load Amiga BASIC

To begin, you should sit at the Amiga, with the manual and this book both handy so that you can refer to them as you practice on the computer. Then follow these six steps to *boot up*, or start, your computer:

1. Turn on the monitor and Amiga.
2. At the Kickstart prompt, insert the Kickstart disk.
3. At the Workbench prompt, insert the Workbench disk.
4. When the disk drive light goes out and the screen shows the Workbench disk icon, insert the Amiga BASIC disk. *Warning: Never remove a disk while the disk drive light is on.*
5. Move the mouse arrow pointer onto the Amiga BASIC disk icon and press the left mouse button twice (called a *double-click*) to select and open this disk.
6. A window will appear. Move the pointer to the Amiga BASIC programming icon and select it by double-clicking the left mouse button.

   The arrow pointer will change to the busy symbol and the disk drive light will go on as Amiga BASIC is loaded. Next, a blue screen appears with two windows. The main window is the Output window, in which you will see a statement about Commodore Amiga BASIC. When you run a program, the results appear on this window. At the right of the screen is a List window. When you have a program in memory, the listing appears in this window. While the program is running, this window disappears.

Figure 1-1. The Amiga Startup Screen



You are now ready to start programming. A vertical orange line (the cursor) will appear in the List window. As you type, the letters are displayed and the cursor moves along to the next position on the screen line. You start off in the List window because that's where you will be entering your programs.

New Terms and Commands

Before we get too involved in programming, here are a few terms which you will be seeing a lot of. As you work your way through the book, you may want to flip back to this page to refresh your memory of what they mean.

REM is the keyword for REMark. This word allows you to put explanations or comments within a program. The computer will essentially ignore these statements. Examples:

```
REM TITLE
200 REM Draw wheels
CALC: REM Perform calculations
```

END is a command to stop the program. This command is optional in a program because the computer will automatically stop when it finishes going through the instructions. Some programmers like to use END before subroutines so the computer won't accidentally execute subroutines and cause errors. A STOP command will also stop execution. Some computers require the END statement to save the program properly. In this book END is used as the last statement in each program so that you can tell when you have entered a complete program.

LIST shows a listing of your program on the List screen and is used in command mode (in the Output window). If you are in the Output window after a program has run, and you want to see the List window again, you may type the command LIST and press RETURN.

RUN tells the computer to execute the program in memory. One of the functions of a computer is that you can run a program as many times as you want and the computer won't mind.

CLS is the command to CLear the Screen (Output window). I often use this command near the beginning of a program to make sure the screen will be clear. The Amiga automatically clears the screen when a program is run, but

you would want to use this command if the program is repeated or if you want a clear screen between sections of a program.

NEW is the command to *clear*, or discard, the old program and get ready for a new program. In the Output window, type NEW and press RETURN. If you have a program in memory, a system message will appear asking if you want to save the program. Use the mouse to select your answer. After a NEW command, both windows clear.

Statements and commands. A *statement* is an instruction in the program. A *command* is a particular kind of statement which tells the computer to do something.

Logical line. A *logical line* is an instruction in the program. It may actually consist of several statements separated by colons.

Line numbering. Although most computer versions require that each program line start with a line number, Amiga BASIC does not. You may use numbers or alphanumeric labels to reference a line, although this is not necessary. Line numbers are treated as labels by the computer.

Labels. Using descriptive labels can make a program easier to understand. The label is a word followed by a colon, such as

```
INIT: REM Initialize variables
```

You do not use the colon when you reference the line label in another part of the program, such as

```
GOSUB INIT
```

The familiar line numbers are used in this book to simplify the descriptions of what is happening in the programs. The numbers also make it a little easier to keep track of the lines as you enter the programs. Also, if you have the earlier version of ABasiC or are translating to a different version of BASIC, line numbers are required.

In computers that require line numbers, it doesn't matter what order you type the lines in; the computer will arrange the lines in numeric order. In Amiga BASIC, however, if you need to insert a line, you must move the cursor to the appropriate place in the listing and insert the line in its proper spot.

Constants and variables. Two more terms you'll hear a lot when you are programming are *constant* and *variable*. A constant never changes throughout the program. It may be a

numeric variable (such as 14, 8.5, 20, and so on) or a *string variable*, which may contain characters other than numbers (such as JOHN or 123-4567).

A variable is a name that may have its value changed as the program is running. For example, a numeric variable may be SCORE, which initially starts at zero then increments by one each time a hit is made in a game. A string variable may be NAME$, which starts with the value "CHERY", then is changed to "RICHARD", then later to "CINDY" as the program is run.

A string variable name ends with a dollar sign. A string may be the null string (it contains nothing and is represented as " "), or it may contain as many as 255 characters. All letters and numerals are valid in variable names, but the name must not be a *reserved word* (a command or keyword).

What's a Program?

There are two ways to make the Amiga *execute*, or perform, commands. The simplest is to type a command in the Output window and press RETURN. The computer will execute that command immediately. But a computer program is a whole *series* of instructions which tell the computer what to do. To write a set of instructions without having the computer execute them as soon as you type them in, you need to be in the List window. The computer keeps track of the things you have entered and won't execute the instructions until you run the program.

There are two ways to run a program. One way is to move from the List window to the Output window by moving the mouse arrow to the Output window and pressing the left mouse button once. Type RUN, then press the RETURN key.

Another way is to use the menu options which are at the top of the screen. To make the menu selections appear, press the *right* mouse button, and you will see the main menu headings—Project, Edit, Run, and Windows. Holding the right button down, move the pointer to the Run menu. A secondary menu appears. Point to Start and release the button.

If you started with the cursor in the List window, it will return after the program ends. If the Output window was active (if the cursor was in that window), the List window will not reappear when the program is finished. To get the List window back, type LIST and press RETURN.

Initializing a Disk

If your program is more than a few lines long, you won't want to type it in each time you use it. Instead, after it is typed in once, you can save it on a disk, then at any later time you can load it back in to run it.

To save a program, you first need an initialized disk. It's a good idea to have several initialized disks on hand while you're programming. If you prefer, you may save a program on the same disk that Amiga BASIC is on.

To initialize a disk:

1. Turn on the monitor and computer.
2. At the Kickstart prompt, insert the Kickstart disk.
3. At the Workbench prompt, insert the Workbench disk.
4. Select the Workbench disk icon by moving the mouse pointer to the disk icon and double-clicking the left mouse button. The Workbench window will appear.
5. Remove the Workbench disk and put in a new disk. An icon of a disk will appear with the label DF0:BAD.
6. Select the new disk by moving the arrow onto the disk icon and double-clicking the left mouse button. The white disk icon will change to black.
7. Press the right mouse button and move to the menu bar. The three menus are Workbench, Disk, and Special. Under Disk, select Initialize by releasing the mouse button when Initialize is highlighted.
8. You will see a message: *Please replace volume Workbench in any drive.* Take out the new disk and put in the Workbench disk.
9. This message then appears: *Please insert disk to be initialized in drive 0.* Take out the Workbench disk and put in the new disk. Move the mouse so the arrow points to Continue and press the left mouse button.
10. When you see *OK to initialize disk in drive 0 (all data will be erased)?*, move the arrow to Continue and press the left mouse button. The drive light will go on and the disk label on the disk icon will change to DF0:BUSY. When the light goes off, the label changes to EMPTY and you are finished.

    To give your new disk a meaningful name, move the arrow to the Empty disk icon and press the left mouse button. The disk turns black. Holding the right mouse button

down, move to the menu bar. Under the Workbench menu choose Rename by releasing the right mouse button when Rename is highlighted. A line will appear with the title Empty in it. Move the arrow to the box and press the left mouse button. Use the arrow keys and BACK SPACE key to erase Empty, then type the name of your new disk. Press RETURN when you are finished typing, and the disk is renamed.

Saving Programs

To save a program on the same disk that you used to load Amiga BASIC, move the cursor to the Output window by moving the arrow to the left screen and pressing the left mouse button. Choose a title for your program and then type the command

SAVE "TITLE"

and press RETURN. Notice that the title of the window changes from *BASIC* to the title you have chosen. Once you have a title on the window and want to save another copy, you just need the command SAVE.

To save the program on a different initialized disk, use the command

SAVE "DF0:TITLE"

DF0: tells the computer to save the program onto whatever disk is in the disk drive. Without DF0: the computer will ask you for the disk that you used to load Amiga BASIC.

Another way to save the program is to use the mouse to move to the menu bar. Under the Project menu, select Save. A message box appears and asks you to type in a title.

Loading a Program

To use a program that has previously been saved, you need to load the program. Put the cursor in the Output window. Using the name of the particular program you want, enter the command

LOAD "NAME"

If the program is on a different disk from the one you used to load Amiga BASIC, use the command

LOAD "DF0:NAME"

If the program you want is filed in a drawer, first specify the name of the drawer. For example, to load one of the demonstration programs in the BasicDemos drawer when you are in Amiga BASIC, use the command

LOAD "BasicDemos/music"

or, if it is on a different disk,

LOAD "DF0:BasicDemos/music"

When the program is successfully loaded, the listing will appear in the List window.

It is also possible to load and automatically run a program from a disk by using the command

RUN "NAME"

or

RUN "DF0:NAME"

However, the List window does not clear if this method is used.

Typing In Programs

Computers are rather particular in understanding what you type, so the programs in this book must be entered exactly as listed in order to work properly.

To be safe, you should save your work at last every half-hour or so. A power failure or a *brownout* (a dip in voltage) can be frustrating if it causes you to lose many hours of work. It's also a good idea to use two separate disks as you save your programs. It is possible for a power failure to occur (or the machine to lock up) right as you are saving a new version over another version, and both versions will be lost. If that happens, you will be very glad to have a backup copy on another disk.

The programs in this book have short lines, even though a program line may contain as many as 255 characters and may have several statements separated by colons. Many program lines could be combined, but shorter lines are easier to read. If you decide to use longer lines, you should know that as you are typing a longer line, the List screen will scroll to the right. You can use the arrow keys to scroll the display left and right in order to see a different section of the long line.

Be especially careful in typing DATA statements. Read the numbers carefully and be sure to copy the commas exactly as they are shown.

Editing Features

To make changes in your programs, you activate the List window by moving the mouse pointer to the window and clicking the left button. Then you use the arrow keys or the mouse to position the cursor in the line you want to change. To use the mouse, simply move the arrow to where you want the cursor to be, then press the left button.

To insert characters, place the cursor in the desired position and start typing. For example, enter this program segment into the List window:

```
10 REM EDIT PRACTICE
20 PRINT "HI"
30 A=3:B=4
40 C=A+B:PRINT C
50 END
```

Let's say you want to add D=5 to line 30. Move the cursor to the end of line 30, right after the 4. Now type

```
:D=5
```

You do not need to press RETURN (doing so will insert a blank line). If you want to add another line, you may press RETURN to get the blank line and then start typing. For examaple, you may add a line:

```
35 PRINT D
```

by pressing RETURN after the 5 on line 30, then entering line 35. You don't have to create the blank line first, but you may find it less confusing.

Now let's add something in the middle of a line. At line 20, add a name after HI inside the quotation marks. Place the cursor after the *I* and before the closing quotation mark. Now type a space, then a name, such as LEWIS. Notice how the

rest of the line moves over as you type. Do not press RETURN. Line 20 should now look like this:

```
20 PRINT "HI LEWIS"
```

To delete a character, place the cursor to the right of the unwanted character, then press the BACK SPACE key. To illustrate, change HI in line 20 to HELLO. First, place the cursor right after the *I* in HI. Press the BACK SPACE key to erase the *I*. Now type ELLO, and line 20 should look like this:

```
20 PRINT "HELLO LEWIS"
```

You also can edit a line by using the highlighting feature. Use the mouse to position the cursor at the point where you want to start editing. Keep holding the left mouse button down as you move it, and you'll notice that the background of the characters changes to yellow. Any characters in the yellow band will be changed. Release the left mouse button when you have highlighted all the characters to be changed. Next, type the correct characters. All the yellow characters will disappear.

If you make a mistake and highlight more characters that you mean to, you can cancel the highlighting by releasing the button and moving the pointer to another line. Click the left mouse button to shift the cursor out of the highlighted area, then start over by repositioning the cursor at the start of the characters you want to highlight.

Let's try an example. Change the name LEWIS in line 20 to DEAN. Use the mouse to move the arrow tip to the *L* in LEWIS. Press the left mouse button to start the cursor at the *L*. Keep holding the left mouse button down and move the arrow across the name to the *S*—the name LEWIS will be in black letters with a yellow background. Now release the mouse button. Type a different name, such as DEAN. As soon as you press D, all of the yellow letters disappear, and you can finish typing EAN.

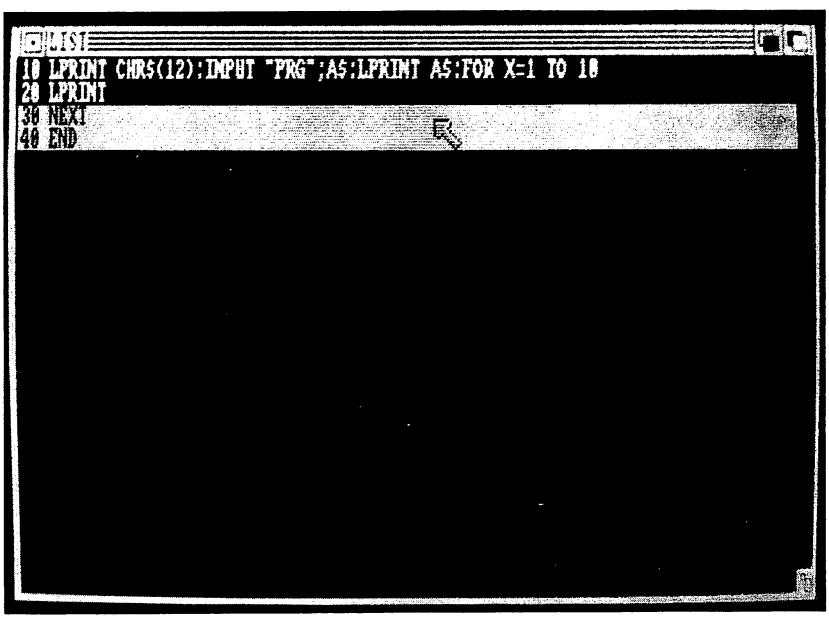
You may use this same method to change just one letter. For example, suppose you need to change DEAN to JEAN. Use the mouse to move the cursor to the *D* in DEAN. Press the left mouse button and move it just slightly so only the letter *D* is highlighted. Now release the mouse button and type the letter *J*.

You may use the mouse and the yellow highlighted areas to delete more than one character at a time as you do with the BACK SPACE key. Move the mouse to highlight all the characters to be deleted, then press the BACK SPACE key. For example, delete the word EDIT in line 10. Use the mouse to place the arrow at the *E* in EDIT. Hold the left mouse button down while you move to the space after EDIT. Now release the mouse button and press the BACK SPACE key once. The line should now look like this:

```
10 REM PRACTICE
```

You may use this method to delete several lines. Let's say you want to delete lines 35 and 40. Move the mouse anywhere on line 35 and press the left mouse button. Holding the button down, move downward to line 40. Notice that the whole two lines will turn yellow. Now let go of the mouse button. You may press either the RETURN key or the BACK SPACE key to get rid of both lines. If you use the RETURN key, you will get one blank line before line 50.

Figure 1-2. Using Highlighting to Edit Lines



Another way to delete several lines is to activate the Output window (move the arrow to the Output window and click the left mouse button). Now type the DELETE command with the specified line numbers or labels you wish to delete, such as DELETE 300–500, and then move back to the List window. This method is quick if you have to delete a range of lines which are not all showing on the screen at once.

Chapter 2

PRINT Statements


Chapter 2

PRINT Statements

To get started programming, let's try a few PRINT statements. PRINT is a command to display something on the screen. If you want to print a message, simply put it in quotation marks:

```
PRINT "HELLO"

20 PRINT "I like to program."

PRINT "Bye for now."
```

When you run the program, the messages inside the quotation marks will be printed on the screen.

You can print actual messages enclosed in quotation marks, numbers, or variables (either string or numeric). Here are some example PRINT statements:

```
PRINT NAME$

PRINT 3

PRINT 5+8

PRINT x
```

Try writing a program to print a message. Or you can try drawing pictures using symbols. Program 2-1 is an example.

Program 2-1. Face

```
REM FACE
CLS
PRINT
PRINT "@@@@@@@"
PRINT "/     \"
PRINT "| ^ ^ |"
PRINT "| o o |"
PRINT "|  .  |"
PRINT "|  o  |"
PRINT "\_____/"
PRINT
END
```

Unless you specify otherwise, each PRINT statement starts printing on a new line. In the examples above, each PRINT statement displayed one item per line. To print several items with one PRINT statement, the items must be separated by *delimiters* (semicolons or commas). Program 2-2 illustrates various forms of printing.

Figure 2-1. Making a Face with PRINT Statements

Program 2-2. Printing

```
10 REM PRINTING
20 CLS
30 PRINT "HELLO"
40 PRINT
50 PRINT "FIRST";"SECOND"
60 PRINT "THIRD ";"FOURTH"
70 PRINT "FIVE","SIX","SEVEN"
80 PRINT 7-5
90 END
```

Line 10 is a REMark stating the title of the program. Line 20 clears the screen, then line 30 prints one word. Line 40 prints a blank line. Line 50 illustrates that the semicolon prints the second item right after the first. If you want a space between two items, you need to put the space within the quotation marks, as in line 60. Line 70 shows what happens when you separate items with commas—the next item starts in the next print region. Line 80 prints a numeric calculation.

Formatting the Display

To get spaces in your printing or to line up the words and numbers into columns, you can use spaces within quotation marks, or you can use the TAB and SPC functions. TAB is like

a typewriter tabulator that indents to a certain column. TAB*(n)* will start your printing in column *n*. SPC*(n)* will print *n* number of spaces before your next item.

A PRINT statement can contain several TAB and SPC functions. Program 2-3 is a short program that numbers the columns and illustrates the use of TAB and SPC.

Program 2-3. TAB and SPC

```
10 REM TAB AND SPC
20 CLS
30 PRINT "123456789012345678901234567 8"
40 PRINT TAB(5);"START"
50 PRINT TAB(8);"A";SPC(6);"B"
60 PRINT
70 END
```

Another useful formatting function is STRING$, which is used to print several repeats of the same character along a line. The syntax is STRING$*(n,c)*. The first value inside the parentheses is the number of characters you need, and *c* is the ASCII value of the character (or the actual character within quotation marks). For example, STRING$(10,65) tells the computer to print a string of ten letter *A*'s—the ASCII code of *A* is 65. Another way to write this is STRING$(10,"A").

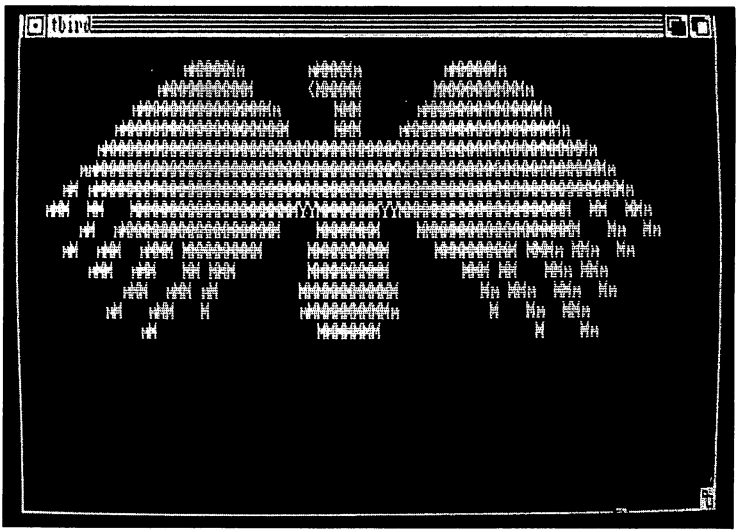
"Tbird" is longer program that illustrates how to get a kind of low-resolution drawing by printing symbols. It illustrates delimiters between printed items, the TAB function to start printing in a certain column, the SPC function to print a number of spaces between items, and the STRING$ function to print strings of characters.

Program 2-4. Tbird

```
10 REM TBIRD
20 CLS:PRINT
30 PRINT TAB(20);"mMMMMMm";SPC(7);"mMMMMm";SPC(9);"mMMMMMm"
40 PRINT TAB(17);"mMMMMMMMMM";SPC(6);"<MMMM";SPC(8);"MMMMMM
MMMMm"
50 PRINT TAB(14);"mMMMMMMMMMMMMMMMm";SPC(6);"MMM";SPC(6);"mMM
MMMMMMMMMMm"
60 PRINT TAB(12);"mMMMMMMMMMMMMMMMMMM";SPC(5);"MMM";SPC(4);"
mMMMMMMMMMMMMMMMMm"
70 PRINT TAB(10);"m";STRING$(54,"M");"m"
80 PRINT TAB(8);"mm";STRING$(57,"M");"m"
90 PRINT TAB(6);"mM m";STRING$(59,"M");"m"
100 PRINT TAB(4);"mMM  MM   ";STRING$(19,"M");"YYMMMMMMMYY";S
TRING$(19,"M");"  MM  MMm"
110 PRINT TAB(8);"mM  m";STRING$(18,"M");SPC(4);"MMMMMMM";SPC
(4);STRING$(18,"M");"   Mm  Mm"
```

```
120 PRINT TAB(6);"mM  mMM  mMMM MMMMMMMMM      MMMMMMMMM      M
MMMMMMMM MMMm MMm  Mm"
130 PRINT TAB(9);"mMM  mMM   MM MMM";SPC(8);"MMMMMMMMM";SPC(8
);"MMM MM   MMm MMm"
140 PRINT TAB(13);"mMM  mMM mM";SPC(9);STRING$(11,"M");SPC(9)
;"Mm MMm  MMm  Mm"
150 PRINT TAB(11);"mM   mMM   M";SPC(10);"mMMMMMMMMMm";SPC(10
);"M   Mm  MMm"
160 PRINT TAB(15);"mM";SPC(18);"MMMMMMM";SPC(17);"M    Mm"
170 PRINT
190 END
```

Figure 2-2. The Thunderbird Drawn by Tbird Program



The LOCATE command provides a more efficient way to print at a certain place on the screen or to relocate the cursor (rather than printing blank lines and TABulating to a column). This command is of the form LOCATE *r,c* where *r* is the row number and *c* is the column number of the starting position on the screen.

```
LOCATE 5,10:PRINT "MESSAGE"
```

starts printing MESSAGE in column 10 of the fifth row from the top.

You may wish to highlight your printing by using different colors. COLOR *f,b* requires a foreground color *f* and a

background color *b*, where *f* and *b* are numbers from 0 through 3. The default print color is COLOR 1,0, or white on blue. The blue screen color is color 0, white is 1, black is 2, and orange is 3 (without defining new screens and palettes). Try this sequence:

```
COLOR 2,1

PRINT "TRY THIS"

COLOR 3,2:PRINT "AND THIS"

COLOR Ø,1:PRINT "Inverse"

COLOR 1,Ø:PRINT "BACK TO NORMAL"
```

The PRINT USING Statement

PRINT USING is a handy statement to use when you want to format your printing. The Amiga BASIC manual lists all the different forms and options for printing numbers and strings. Program 2-5 illustrates different forms. In printing numbers, the number sign (#) indicates the placement of a numeral.

Program 2-5. Using

```
1Ø REM PRINT USING NUMBERS
2Ø CLS
3Ø A=123.456
4Ø B=75
5Ø C=.2
6Ø D=-1.35Ø67
7Ø PRINT USING "###";A
8Ø PRINT USING "###";B
9Ø PRINT USING "###";C
1ØØ PRINT USING "###";D
11Ø PRINT
12Ø PRINT USING "$$###.##";A,B,C,D
13Ø PRINT USING "$$###.##-";A,B,C,D
14Ø PRINT USING "+###.#";A,B,C,D
15Ø PRINT USING "###-";A,B,C,D
16Ø PRINT USING "########,.##";1234567.856#
17Ø PRINT USING "###.# A";A,B,C,D
18Ø PRINT
19Ø PRINT USING "**###.##";A
2ØØ PRINT USING "**###.##";B
21Ø PRINT USING "**###.##";C
22Ø PRINT USING "**###.##";D
23Ø PRINT
24Ø END
```

Lines 30–60 assign numbers to the variable names A, B, C, and D. Lines 70–100 print the numbers using the format "###", indicating a whole number of three digits. Notice that the numbers are rounded and right-justified.

Lines 120–130 illustrate how money amounts are printed. Two dollar signs precede the # signs. Lines 140–150 print signed numbers. In line 160, a comma before the decimal indicates that a comma is to be placed every three digits to the left of the decimal place. In line 170, another character, A, is printed after the number. It is included in the format within the quotation marks. Lines 190–220 have two asterisks before the number, which tells the computer to print leading asterisks in the field.

Program 2-6 illustrates some of the formatting options for strings.

Program 2-6. Using Strings

```
10 REM PRINT USING STRINGS
20 CLS
30 A$="RICHARD"
40 B$="BOB"
50 C$="RANDY"
60 PRINT USING "!";A$,B$,C$
70 PRINT USING "! ";A$,B$,C$
80 PRINT
90 PRINT USING "\\";A$,B$,C$
100 PRINT
110 PRINT USING "\  \";A$
120 PRINT
130 PRINT USING "\        \";A$
140 PRINT USING "\        \";B$
150 PRINT USING "\        \";C$
160 PRINT
170 PRINT USING "&";A$,B$,C$
180 PRINT USING "& ";A$,B$,C$
190 PRINT
200 PRINT USING "HIS INITIAL IS !.";A$
210 END
```

Lines 30–50 define string variables A$, B$, and C$. The exclamation point in the PRINT USING format indicates that the first character only of a string is to be printed. Line 60 illustrates this. Line 70 also uses ! to print the first character, but places a space after the character.

Two back slash marks indicate that the first two characters of a string are to be printed, as in line 90. To print more than the first two characters, insert spaces between the back slashes. The total number of characters to be printed will be

the number of spaces between the back slashes plus two for the back slashes. Line 110 indicates that four characters are to be printed. Lines 130–150 print ten characters each. Note that the strings are printed left-justified with spaces in the rest of the field.

The ampersand (&) indicates printing the whole string no matter what length. Line 170 uses & to print the three strings. If you need a space, it can be included. Line 180 uses &, then a space to separate the names.

With PRINT USING, other characters included within the quotation marks will be printed as is, so you can combine a title with a format. Line 200 includes a message, then ! to indicate printing the first character of the string A$.

Setting Line Width

WIDTH is used to limit the number of characters that can be printed per line. If you do not specify a width and print a long sentence, you can see only part of it on the screen. The other part is printed, but you must scroll the display to see it. If you have a specified width within the 80 columns of the screen, the long sentence will be split and limited to the visible screen. The size of the characters is not affected. Try this example.

```
WIDTH 15

PRINT "THIS IS A SENTENCE OF MORE THAN 15 CHARACTERS."

WIDTH 40

PRINT "NOW TRY THIS SENTENCE TO SEE HOW IT IS PRINTED."
```

Remember that you can use extra spaces in the printed messages so that words are not split. For example, in the last printed message above, insert an extra space before the word PRINTED.


Chapter 3

Random Numbers


Chapter 3

Random Numbers

One of the functions of a computer is to make random selections. Games may start with screens of random obstacles, quizzes may print questions in a random order, and school exercises may present random activities.

RND is the function to obtain random numers. RND returns a random decimal fraction between 0 and 1. To see an example, type PRINT RND in the Output window and press RETURN.

Usually, you will prefer to work with whole numbers. The INT function yields the INTeger, or whole number, portion of a number. For example, INT(3.21239) is the whole number 3. Since RND gives a decimal fraction, multiply it by a whole number, then take the INTeger portion. For example,

```
PRINT INT(10*RND)
```

will give a random number from 0 through 9 because RND is a fraction and INT takes the lower integer (it does not round the number). Now, if you really want random numbers from 1 through 10 instead of 0 through 9, use INT(10*RND)+1.

In throwing a die, you get numbers from 1 through 6, so the random number would be INT(6*RND)+1. If you have two dice, the possibilities are 1 through 12, or INT(12*RND)+1.

Try this program:

Program 3-1. Random

```
10 FOR C=1 TO 5
20 PRINT INT(10*RND)+1
30 NEXT C
40 END
```

Run it several times. Notice that you always get the same sequence of random numbers. This can be helpful when you are testing a program, but most times you will want different numbers each time. The RANDOMIZE command is used to mix up the numbers (technically, this is called using a different *seed*). Add this line to the program and try running it again:

```
5 RANDOMIZE
```

The RANDOMIZE command should come before the RND function in the program.

This time when you run the program, the computer asks you to enter a number. If you enter a different number each time, you will get a different sequence of random numbers.

Most of the time you will want your programs to generate random numbers without having the user enter numbers. The Amiga has a built-in TIMER which changes automatically. We can use TIMER as a way to get a different number each time the program is run. The randomization command becomes

```
5 RANDOMIZE TIMER
```

Program 3-2, "Simple Drill," illustrates a way to write a simple study drill. As you learn to program, you can improve this program by adding sound and graphics, or you can use the general idea to develop a drill about something else.

First, a random number from 0 through 9 is chosen and called A (line 130). Line 140 chooses another random number, B, from 0 through 9. Line 150 prints the problem of A+B, and the next line asks the student for the answer. Line 170 compares the student's answer with the correct answer; then the computer either prints the correct answer or a message that the student was correct.

A scoring feature is added by using a counter SCORE which starts at zero and is incremented by one each time the answer is correct. A FOR-NEXT loop presents ten problems for the quiz. After the quiz, the student's score is printed. The student then has the option to try again.

Program 3-2. Simple Drill

```
10 REM SIMPLE DRILL
20 CLS
30 PRINT "ADDITION DRILL"
40 PRINT:PRINT "You will see a problem."
50 PRINT:PRINT "Type the answer and press <RETURN>."
60 PRINT:PRINT
70 PRINT "Press any key to start."
80 K$=INKEY$:IF K$="" THEN 80
90 SCORE=0
100 FOR P=1 TO 10
110 CLS
120 RANDOMIZE TIMER
130 A=INT(10*RND)
140 B=INT(10*RND)
150 PRINT A;"+";B;
160 INPUT "= ",T
```

```
170 IF T=A+B THEN 200
180 PRINT:PRINT "NO, THE TOTAL IS";A+B
190 GOTO 220
200 PRINT:PRINT "CORRECT!"
210 SCORE=SCORE+1
220 PRINT:PRINT "PRESS <RETURN>"
230 K$=INKEY$:IF K$="" THEN 230
240 IF ASC(K$)<>13 THEN 230
250 NEXT P
260 CLS
270 PRINT "YOUR SCORE WAS"
280 PRINT SCORE;"CORRECT"
290 PRINT "OUT OF 10 PROBLEMS."
300 PRINT:PRINT "TRY AGAIN? (Y/N)"
310 K$=INKEY$:IF K$="Y" THEN 90
320 IF K$<>"N" THEN 310
330 PRINT:PRINT "PROGRAM ENDED"
340 END
```

Outline of Simple Drill Program

Lines	Explanation
10	REMark—title of program.
20	Clears screen.
30–70	Print title and instructions.
80	Waits for student to press any key.
90	Initializes SCORE at zero.
100	Performs quiz of ten problems.
110	Clears screen.
120–140	Randomly choose two numbers A and B.
150	Prints problem.
160	Receives student's answer.
170	Compares student's answer with correct sum.
180–190	If answer is incorrect, print correct answer; branch.
200–210	If answer is correct, print message; increment score.
220–240	Wait for student to press RETURN key.
250	Goes to next problem.
260–290	Clear screen; print score.
300–320	Print option to try again; branch appropriately.
330–340	End program.

Chapter 4

Interactive Programming

Chapter 4

Interactive Programming

You can print all kinds of messages on the screen now, but eventually you will want the user to type something and the program to react to it. This is called *interactive programming.* One way we get a messsage into the computer within a program is by using INPUT. For example,

```
PRINT "Type a number"

INPUT N
```

The computer will print the message and then wait for the user to type a number and then press RETURN. The computer will assign that number to the variable called N. The variable with the name N is a numeric variable, so a number must be entered—no symbols or letters.

To accept a string, a string variable name (a variable name with a dollar sign at the end) must be used with INPUT, such as INPUT N$. With a string variable, any kind of characters may be entered, and the variable N$ will equal whatever is entered.

You can use INPUT in a variety of ways in your programming—entering names and addresses for a file, entering numbers to figure a mortgage payment, entering answers to a math quiz, entering numbers to be played as a song, typing answers to a grammar quiz, and so forth.

In the previous example we used a PRINT statement to tell the user what to enter. The next statement was an INPUT to receive the answer. If you prefer, you can combine these statements using an *input prompt.* After the INPUT command, put your prompting message in quotation marks. Follow the last quotation mark with a semicolon, then the variable name:

```
INPUT "What is the answer";A
```

This method keeps the input cursor on the same line as the prompt message. The first method put the input on the next line. You could also print a message and put a semicolon

after the printed message, then use an INPUT command. Notice that a question mark is automatically printed.

```
PRINT "Enter a word.";

INPUT W$
```

If you do not want INPUT to print a question mark, you can use a comma instead of the semicolon after the input prompt:

```
INPUT "The answer is ",A
```

Experiment to get used to the spacing involved.

Illustrations of INPUT

Program 4-1 is a short interactive program that illustrates different ways to use INPUT. Line 30 asks a question, then line 40 receives a string variable. Line 70 uses an input prompt with a semicolon, then a numeric variable. Lines 110 and 120 use the input prompt with a comma.

Program 4-1. Input

```
10 REM INPUT
20 CLS
30 PRINT "WHAT IS YOUR NAME?"
40 INPUT N$
50 PRINT "HELLO, ";N$
60 PRINT
70 INPUT "HOW OLD ARE YOU";A
80 PRINT A;"IS A GOOD AGE."
90 PRINT
100 PRINT "NOW ADD TWO NUMBERS."
110 INPUT "FIRST NUMBER IS  ",B
120 INPUT "SECOND NUMBER IS ",C
130 PRINT
140 PRINT "THE SUM IS";B+C
150 END
```

Better Input Control

INPUT receives whatever the user types in before pressing the RETURN key—whether it is one character, several lines of characters, or nothing. With INPUT, it is easy for the user to cause errors by entering something the program is not expecting. INKEY$ is a method of receiving input that is more controllable. INKEY$ checks the keyboard to see if a key is

pressed. The character pressed is not printed on the screen unless the program specifies so with a PRINT command. Program 4-2 is an example of how INKEY$ is used.

Program 4-2. INKEY$ Example

```
10 PRINT "Press a key."
20 K$=INKEY$:IF K$="" THEN 20
30 PRINT K$
40 END
```

Line 10 prints the message to press a key. Line 20 checks to see if a key K$ is pressed. If no key is pressed, then K$ will be the null string " " and the computer will branch back to the same line. Only if a key is pressed will the program continue. Line 30 prints the character generated by the keypress.

You may wait until the user presses a certain key before continuing:

Program 4-3. Wait for a Certain Key

```
10 PRINT "Press the space bar."
20 K$=INKEY$:IF K$<> " " THEN 20
30 PRINT "Press the return key."
40 K$=INKEY$:IF K$<>CHR$(13) THEN 40
```

I often use INKEY$ to get a response and ignore all invalid responses. For example, this routine, "Get a Number," will ignore all keys that are not numbers:

Program 4-4. Get a Number

```
10 PRINT "Press a number."
20 N$=INKEY$
30 IF N$<"0" OR N$>"9" THEN 20
40 PRINT N$
```

Program 4-5 is a quiz about converting roman numerals and shows several types of interactive programming commands. The user may first choose converting from a roman

numeral to an arabic number, converting from an arabic number to a roman numeral, or ending the program. INKEY$ is used to receive the choice, which must be a number from 1 through 3.

Line 370 receives a numeric answer using INPUT. Line 480 receives an INPUT string variable answer for the roman numeral. If an answer is incorrect, the correct answer is printed, then the user must press RETURN to continue the program. Lines 400–420 use INKEY$ to wait for RETURN to be pressed; then another problem of the same type is printed.

Outline of Roman Numerals Program

Lines	Explanation
20	Clears screen.
30–60	Print title.
70–130	READ from DATA the roman numeral equivalents for hundreds, tens, and ones.
140–160	Print menu screen.
170–200	Receive choice and branch appropriately.
210–310	Calculate roman numeral equivalent of random number N.
320	Branches for second choice.
330–360	Print problem.
370	Receives answer.
380–390	If answer is incorrect, print correct answer.
400–420	Wait for user to press RETURN.
430	Prints message for correct answer.
440–450	Present option for another problem; branch to menu.
460–470	Print problem.
480	Receives roman numeral.
490	If answer is correct, branches.
500–510	Print correct answer.
520	Branches to line 400.
530–540	Clear screen and end.

Program 4-5. Roman Numerals

```
10 REM ROMAN NUMERALS
20 CLS
30 PRINT TAB(31);"********************"
40 PRINT TAB(31);"* ROMAN NUMERALS *"
50 PRINT TAB(31);"********************"
60 PRINT:PRINT
70 FOR C=1 TO 9
80 READ H$(C),T$(C),S$(C)
```

```
90 NEXT C
100 DATA C,X,I,CC,XX,II,CCC,XXX,III
110 DATA CD,XL,IV,D,L,V,DC,LX,VI
120 DATA DCC,LXX,VII,DCCC,LXXX,VIII
130 DATA CM,XC,IX
140 PRINT TAB(24);"CHOOSE:  1 ROMAN TO ARABIC"
150 PRINT TAB(33);"2 ARABIC TO ROMAN"
160 PRINT TAB(33);"3 END PROGRAM"
170 A$=INKEY$
180 IF A$<"1" OR A$>"3" THEN 170
190 RANDOMIZE TIMER
200 ON VAL(A$) GOTO 210,210,530
210 CLS:R$=""
220 N=INT(1999*RND)+1:NN=N
230 IF N<1000 THEN 250
240 R$="M":N=N-1000
250 IF N<100 THEN 280
260 NR=INT(N/100)
270 R$=R$+H$(NR):N=N-NR*100
280 IF N<10 THEN 310
290 NR=INT(N/10)
300 R$=R$+T$(NR):N=N-NR*10
310 IF N>0 THEN R$=R$+S$(N)
320 IF A$="2" THEN 460
330 PRINT "GIVEN THE ROMAN NUMERAL"
340 PRINT:PRINT:PRINT R$
350 PRINT:PRINT:PRINT
360 PRINT "WHAT IS THE CORRESPONDING NUMBER?"
370 INPUT A
380 IF A=NN THEN 430
390 PRINT:PRINT "THE NUMBER IS ";NN
400 PRINT:PRINT "PRESS <RETURN>"
410 E$=INKEY$
420 IF E$=CHR$(13) THEN 210 ELSE 410
430 PRINT:PRINT "CORRECT!":PRINT
440 PRINT:PRINT "ANOTHER PROBLEM?":PRINT
450 GOTO 140
460 PRINT "GIVEN THE NUMBER";NN
470 PRINT:PRINT "TYPE THE CORRESPONDING ROMAN NUMERAL."
480 INPUT E$:E$=UCASE$(E$)
490 IF E$=R$ THEN 430
500 PRINT:PRINT "THE CORRECT NUMBER IS"
510 PRINT R$
520 GOTO 400
530 CLS
540 END
```

Chapter 5

Program Transfer

Chapter 5

Program Transfer

The Amiga will execute the lines in a program in listed order unless the program tells the computer otherwise. In this chapter, we will look at several ways you can transfer control to a different place in your programs and discuss the advantages for doing so.

The GOTO Statement

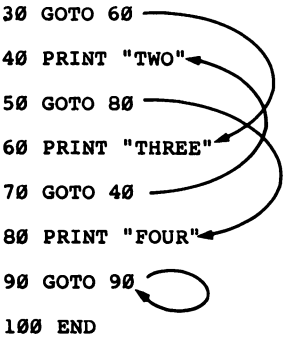
GOTO is a statement that tells the computer to GO TO a different line (as in GO DIRECTLY TO JAIL, DO NOT PASS GO). In Amiga BASIC, you can go to a line by specifying a line number or a line label. If you use an alphanumeric label, the label in the line must have a colon after it, but the GOTO statement does not use the colon after the label:

```
10 GOTO SAMPLE
SAMPLE: GOTO 10
```

You can transfer to a previous line, a later line, or even put the computer in a loop going to the same line.

Program 5-1. GOTO

```
10 REM GOTO
20 PRINT "ONE"
30 GOTO 60
40 PRINT "TWO"
50 GOTO 80
60 PRINT "THREE"
70 GOTO 40
80 PRINT "FOUR"
90 GOTO 90
100 END
```



Program 5-1 does not print the words in the order shown in the program because the GOTO commands transfer to different lines. The arrows show how the program is executed.

This program does not end because the computer keeps going to line 90. Press CTRL-C (Break) to stop execution.

Using GOTO to create loops can create lots of printing effects with short programs:

Program 5-2. Loop

```
10 REM LOOP
20 CLS
30 PRINT "HELLO"
40 GOTO 30
50 END
```

Keep in mind that GOTO commands can make a program hard to follow and less efficient. If you have to debug a program with many GOTO commands, you pretty much have to play computer to follow the logic, tracing each GOTO to the indicated line.

FOR-NEXT Loops

The FOR-NEXT statement creates a loop that is executed a certain number of times, then the program continues.

Program 5-3. FOR1

```
10 REM FOR1
20 FOR T=1 TO 5
30 PRINT "HI"
40 NEXT T
50 END
```

The variable T (use any name you wish) is a *counter*. Line 20 says to start T at the value 1, then go until T is the limit of 5. Line 30 prints HI. Line 40 says NEXT T, which increments T by one. The computer checks to see whether T has reached the limit of 5. If not, the program transfers to the statement directly following the FOR statement. This process continues until the limit is exceeded, then the program continues with the line after NEXT. In this example, T will be 1, 2, 3, 4, and 5

when HI is printed. When T is 6, the loop finishes and the program ends.

Program 5-4 is another example of a FOR-NEXT loop.

Program 5-4. FOR2

```
10 REM FOR2
20 FOR C=0 TO 9
30 PRINT C,C*C
40 NEXT C
50 END
```

The variable C starts at 0 and goes to 9. In this example, the counter is actually used within the loop.

The counter does not have to be incremented by one. You can specify a STEP size. Suppose you want to count by twos:

Program 5-5. FOR3

```
10 REM FOR3
20 FOR N=0 TO 10 STEP 2
30 PRINT N
40 NEXT N
50 END
```

The STEP size can be a fraction:

Program 5-6. FOR4

```
10 REM FOR4
20 FOR X=1 TO 3 STEP .5
30 PRINT X
40 NEXT X
50 END
```

The STEP size can be negative, which would be decreasing the index:

Program 5-7. FOR5

```
10 REM FOR5
20 FOR B=10 TO 0 STEP -1
30 PRINT B
40 NEXT B
50 END
```

Any of the numbers in the FOR statement can be variables. An example is

```
FOR X=A TO B STEP S
```

There is no limit to how long your FOR-NEXT loop is, but you do need to make sure there is a NEXT statement to correspond with each FOR statement. You can also have nested loops—again, make sure the FORs and NEXTs are matched.

Program 5-8. FOR6

```
10 REM FOR6
20 FOR A=1 TO 3
30 FOR B=1 TO 5
40 PRINT A;"*";B;"=";A*B
50 NEXT B
60 PRINT
70 NEXT A
80 END
```

Subroutines

If you want the program to perform the same series of steps in different places, there's no need to enter identical lines of code several times in the program. You can put the process in a subroutine and then use GOSUB to perform the routine each time you want it.

GOSUB is similar to GOTO except that GOSUB goes out to a subroutine, then returns. GOSUB is followed by a line number or line label, and when the program comes to the GOSUB statement, it will branch to the specified line, just as it

does with GOTO. However, with GOSUB the computer will remember where it branched from. When it gets to the command RETURN in the subroutine, it will branch back to the first line after the GOSUB statement.

Be sure that every GOSUB is matched with a RETURN. Otherwise, your program will stop running and an error message will be displayed. You can have GOSUBs within other GOSUBs—each RETURN branches back to the GOSUB it most recently executed.

Program 5-9 illustrates the use of subroutines by drawing dice. Five random numbers are chosen for dice, and the five dice are drawn. Subroutines are used to draw the dots. Lines 170–180 draw one dot. Lines 190–210 draw two dots. Lines 220–230 are the subroutine to draw three dots by first using the one-dot subroutine, then the two-dot subroutine.

Lines 240–270 draw four dots by first using the two-dot subroutine, then drawing the other two dots. Lines 280–290 draw five dots by using the four-dot subroutine (which in turn uses the two-dot subroutine) and the one-dot subroutine. Lines 300–330 draw six dots by using the four-dot subroutine and then drawing two more dots.

Program 5-9. Dice

```
10 REM DICE
20 CLS
30 X=30:Y=30:R=6
40 RANDOMIZE TIMER
50 FOR I=1 TO 5
60 D=INT(6*RND)+1
70 LINE (X,Y)-(X+80,Y+40),1,BF
80 ON D GOSUB 150,170,200,220,260,280
90 X=X+120
100 NEXT I
110 LOCATE 15,20:PRINT "PRESS 1 TO TOSS"
120 LOCATE 16,20:PRINT "PRESS 2 TO END"
130 E$=INKEY$:IF E$="1" THEN 20
140 IF E$="2" THEN 320 ELSE 130
150 CIRCLE (X+40,Y+20),R,2
160 RETURN
170 CIRCLE (X+20,Y+10),R,2
180 CIRCLE (X+60,Y+30),R,2
190 RETURN
200 GOSUB 150:GOSUB 170
210 RETURN
220 GOSUB 170
230 CIRCLE (X+20,Y+30),R,2
240 CIRCLE (X+60,Y+10),R,2
250 RETURN
260 GOSUB 220:GOSUB 150
270 RETURN
```

```
280 GOSUB 220
290 CIRCLE (X+20,Y+20),R,2
300 CIRCLE (X+60,Y+20),R,2
310 RETURN
320 CLS
330 END
```

Conditional Branching

Conditional branching is what makes a computer seem intelligent. Actually, of course, the programmer has to tell the computer what to do. IF-THEN statements direct the computer to branch a certain way if a certain condition is true. Amiga BASIC also allows ELSE, which directs a branch if the condition is not true.

The basic form is IF *expression* THEN *clause* ELSE *clause*, where *expression* is a numeric expression or a condition to be tested and *clause* can be either another BASIC command or a line number or label.

IF SCORE=10 THEN 370

tells the computer to check whether the variable SCORE is equal to 10. If so, the program branches to line 370. If not, the program ignores everything after THEN and simply goes to the next line.

IF SCORE=10 THEN 370 ELSE 180

will branch to line 370 if the expression SCORE=10 is true or it will branch to line 180 if the expression is false.

Commands also are allowed after THEN and ELSE, and there can be several commands separated by colons:

IF SCORE=10 THEN PRINT "YOU WIN":G=G+1:GOTO 470 ELSE GOTO Play

A numeric expression containing any of the arithmetic operators can be used, or string expressions can be compared. If a condition is false, the value of the expression is 0; if it is true, the value is −1. These are acceptable expressions:

```
200 IF A THEN 250

IF N$<>"ZZZ" THEN GOTO 500

IF LEN(P$)>3 THEN PRINT P$

500 IF B/D<C*A THEN A=A+1

600 IF X-Y THEN Z=10 ELSE Z=5
```

Logical operators also can be used. The words accepted are NOT, AND, OR, XOR, EQV, and IMP. (See your manual for a detailed discussion of these operators.) For example,

```
700 IF A$<"1" OR A$>"4" THEN 650

IF A=10 AND N$="M" THEN PRINT N$
```

ON-GOTO and ON-GOSUB

ON-GOTO and ON-GOSUB are conditional transfer statements. They can take the place of several IF-THEN statements. Let's say the user has a choice, and then that choice is tested:

```
IF CH=1 THEN GOTO 1000

IF CH=2 THEN GOTO 2000

IF CH=3 THEN GOTO 3000

IF CH=4 THEN GOTO 4000
```

Since the value of CH can be 1, 2, 3, or 4, an ON-GOTO statement can be used to write this sequence more simply:

```
ON CH GOTO 1000,2000,3000,4000
```

This statement says that, depending on the value of CH, the program is to branch to certain lines. If CH is 1, go to the first line number; if CH is 2, go to the second line number; and so forth.

ON-GOSUB is the same idea, but the program will go to a subroutine, then return to the next statement after the ON-GOSUB call.

```
ON X+1 GOSUB 250,250,300,400,650
```

This statement checks the value of X+1. If it is 1, the computer goes to subroutine 250; if it is 2, GOSUB 250; if it is 3, GOSUB 300; if it is 4, GOSUB 400; and if it is 5, GOSUB 650.

Applying What We've Learned

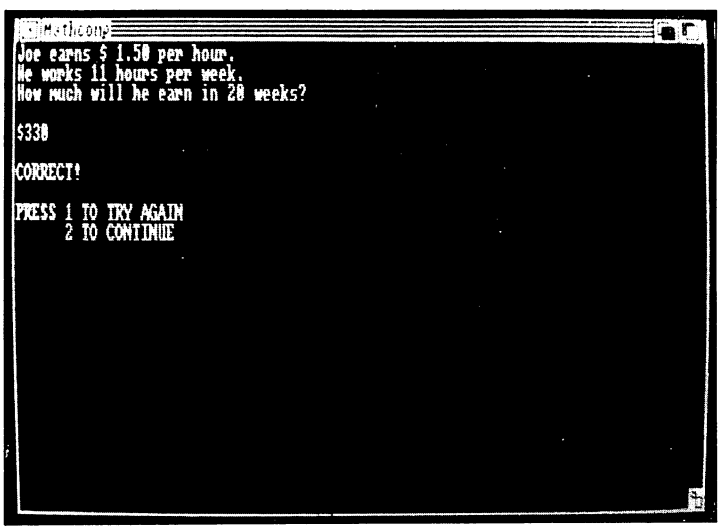
Program 5-10, "Math Competency," illustrates the use of program transfer in several forms. GOTO is used to branch past several lines in the program. FOR-NEXT loops are used to

read in data (more about DATA statements in the next chapter) and to print similar lines in some of the problems. GOSUB is used to call a subroutine that is used in several places. ON-GOTO is used to branch to seven different lines after a choice from the main menu is made. IF-THEN statements are used in a variety of ways to print the problems.

This program presents problems which are similar to problems on a mathematics competency examination such as standard achievement tests. Although these tests are designed for high-school students, the mathematics are at an elementary-school level. The problems involve addition, multiplication, and division.

Random names and numbers are used in the problems. The computer generates problems in word form, or *story problems*. If an answer is incorrect, the correct answer is given, usually with an explanation, and another problem is presented. If the answer is correct, the program continues or gives the student the option of having a similar problem.

Figure 5-1. An Interactive Program—Math Competency



There are six basic types of problems:

1. *Buying Items.* A list of items and their prices is printed. What would it cost to buy all the items on the list? If you had a certain amount of money, which two items could you buy?
2. *Sales Tax.* If you buy a list of items with a certain sales tax rate, what is the total cost?
3. *Earning Money.* A person earns a certain amount per hour and works a given number of hours per week. What are the total earnings per week or for a given number of weeks?
4. *Weekly Expenses.* A list of expenses for one week is given. What is the total expense for the week? What would be the total for several weeks?
5. *Saving Money.* An item costs a certain amount of money. If a person saves for a given number of weeks, how much per week must be saved?
6. *Averages.* Several numbers in a category are listed. What is their average?

Lines 80 to 480 use DATA statements and READ statements in FOR-NEXT loops to define arrays which hold the possible names, number limits, and phrases for the problems. (Arrays and related commands will be covered in the next chapter.) The variable names have numbers which relate to the section of the program in which they are used. Each array of names has both girls and boys, and the subscripts are used later to determine whether the feminine or masculine pronoun is needed.

To determine prices for items, the minimum values are read in from the data, and the computer adds a random number within limits to that base minimum number.

PRINT USING is a helpful command to print money or to line up columns and to make sure zero amounts are printed in the columns.

Random numbers are used to choose the wording of the problems. Either different phrases in arrays are printed, or branching with IF-THEN statements determines the printing.

Outline of Math Competency Program

Lines	Explanation
12–17	DIMension arrays for variables.
20	Branches past subroutine.
30–50	Subroutine to wait for student to press RETURN.
60–70	Clear screen; print title.
80–480	Define elements of arrays for possible names, numbers, and phrases used in writing problems.
480–610	Print main menu screen and branch.
620–840	First problem for Buying Items.
850–1160	Second problem for Buying Items.
1170–1400	Problem for Sales Tax.
1410–1610	First problem for Earning Money.
1620–1830	Second problem for Earning Money.
1840–2010	Third problem for Earning Money.
2020–2070	Print options; branch.
2080–2360	Problem for Weekly Expenses.
2370–2570	Problem for Saving Money.
2580–3200	Problem for Averages.
3210–3220	Clear screen; end.

Program 5-10. Math Competency

```
10 REM MATH COMPETENCY
12 DIM J1$(3,5),J1(3,5,2),N1$(6)
13 DIM T2$(4),B2$(4,4),B2(4,4)
14 DIM N3$(5),J3$(5),T3$(5)
15 DIM N4$(6),B4$(3),A4$(3,5)
16 DIM N5$(6),A5(3),B5(3),M5(3),F5(3)
17 DIM N6$(8)
20 GOTO 60
30 PRINT:PRINT "PRESS <RETURN>"
40 E$=INKEY$:IF E$<>CHR$(13) THEN 40
50 RETURN
60 CLS
70 PRINT TAB(7);"** MATH COMPETENCY **"
80 FOR A=1 TO 3:FOR C=1 TO 5
90 READ J1$(A,C),J1(A,C,1),J1(A,C,2)
100 NEXT C,A
110 DATA PENCIL,8,15,ERASER,2,10,NOTEBOOK,35,99,RULER,29,49
120 DATA PAPER,59,90,DOLL,249,599,BALL,49,89,TRUCK,100,150,GA
ME,270,500
130 DATA MODEL,300,700,CANDY,20,50,MEAT,123,425,FRUIT,24,50
140 DATA CHIPS,100,257,BREAD,100,179
150 FOR A=1 TO 6:READ N1$(A):NEXT A
160 DATA Laura,Cindy,Chery,David,Randy,Brett
170 H$(1)="PENCIL AND ERASER"
180 H$(2)="BALL AND TRUCK"
190 H$(3)="CANDY AND FRUIT"
200 REM
210 FOR C=1 TO 4:READ T2$(C)
```

```
220 FOR A=1 TO 4:READ B2$(C,A),B2(C,A):NEXT A,C
230 DATA HARDWARE,HAMMER,15,PLIERS,3,SAW,6,NAILS,1
240 DATA CLOTHES,BELT,4,TIE,5,SHIRT,6,PANTS,20
250 DATA TOYS,BALL,1,CAR,2,GAME,5,DOLL,6
260 DATA SUPPLIES,PAPER,4,CLIPS,1,PENCILS,1,ENVELOPES,2
270 REM
280 FOR A=0 TO 5:READ N3$(A),J3$(A),T3$(A):NEXT A
290 DATA Sam,doing odd jobs.,Paul,Joe,mowing lawns.,Jack,Bob,
tending children.
300 DATA Mark,Ann,running errands.,Jane,Sue,doing housework.
310 DATA Judy,Kim,delivering ads.,Dawn
320 REM
330 FOR A=1 TO 3:READ N4$(A),N4$(A+3),B4$(A)
340 FOR C=1 TO 5:READ A4$(A,C):NEXT C
350 NEXT A
360 DATA Lena,Andy,is going to,Camp Beaver,Camp fee,Horse rid
ing
370 DATA Tennis lessons,Craft supplies,Laura,Bill,will atten
d,Sports Clinic
380 DATA Tuition,Uniform fee,Equipment fee,Special events
390 DATA Jodi,John,will stay at,Logan Canyon,Camp fee
400 DATA T-shirts,Activity fee,Supplies
410 REM
420 FOR C=1 TO 6:READ N5$(C) :NEXT C
430 DATA Jenny,Angie,Chris,Brent,Grant,Chuck
440 FOR C=1 TO 3:READ A5$(C),B5(C),M5(C),F5(C):NEXT C
450 DATA bike,80,5,7,stereo,90,5,14,computer,100,10,10
460 REM
470 FOR C=1 TO 8:READ N6$(C):NEXT C
480 DATA Sue,Pat,Rita,June,Bob,Ron,Kent,Mike
490 REM
500 PRINT:PRINT:PRINT "CHOOSE:"
510 PRINT:PRINT TAB(8);"1 BUYING ITEMS"
520 PRINT:PRINT TAB(8);"2 SALES TAX"
530 PRINT:PRINT TAB(8);"3 EARNING MONEY"
540 PRINT:PRINT TAB(8);"4 WEEKLY EXPENSES"
550 PRINT:PRINT TAB(8);"5 SAVING MONEY"
560 PRINT:PRINT TAB(8);"6 AVERAGES"
570 PRINT:PRINT TAB(8);"7 END PROGRAM"
580 RANDOMIZE TIMER
590 E$=INKEY$
600 IF E$<"1" OR E$>"7" THEN 590
610 ON VAL(E$) GOTO 630,1180,1420,2090,2380,2590,3210
620 REM BUYING ITEMS
630 CLS:PRINT "Given this price list:":PRINT
640 A=INT(3*RND+1):TP=0
650 FOR C=1 TO 5:D=J1(A,C,2)-J1(A,C,1)
660 P=J1(A,C,1)+INT(D*RND+1):TP=TP+P:XX(C)=P
670 PRINT TAB(4);J1$(A,C);TAB(20);
680 PRINT USING "##.##";P/100:NEXT C
690 F=INT(2*RND+1)
700 IF F=2 THEN 740
710 PRINT:PRINT "How much will it cost to buy all"
720 PRINT "the items on the list?"
730 GOTO 770
740 N=INT(6*RND+1)
750 PRINT:PRINT N1$(N);" wants to buy everything on the"
760 PRINT "list.  What would the total cost be?"
770 INPUT "$",X
```

```
780 IF ABS(X-TP/100)<.001 THEN 830
790 PRINT:PRINT "ADD ALL FIVE NUMBERS."
800 PRINT "THE TOTAL IS ";
810 PRINT USING "$##.##";TP/100
820 GOSUB 30:GOTO 630
830 PRINT:PRINT "CORRECT!"
840 GOSUB 30
850 CLS:PRINT:PRINT
860 FOR C=1 TO 5:PRINT TAB(4);J1$(A,C);TAB(20);
870 PRINT USING "##.##";XX(C)/100:NEXT C
880 PRINT
890 IF F=1 THEN PRINT "If you could only ";:GOTO 910
900 PRINT "If ";N1$(N);" could only ";
910 IF A=1 THEN M=INT(5*RND+25):GOTO 940
920 IF A=2 THEN M=INT(36*RND)+239:GOTO 940
930 M=INT(18*RND+100)
940 PRINT USING "spend $##.##";M/100
950 PRINT "which of these pairs of items on the"
960 IF F=1 THEN PRINT "list could you buy?":GOTO 990
970 IF N<4 THEN PRINT "list could she buy?":GOTO 990
980 PRINT "list could he buy?"
990 R=INT(4*RND+1):PRINT:PRINT
1000 FOR V=1 TO 4:IF V=R THEN S$(V)=H$(A):GOTO 1060
1010 X=INT(2*RND+4):S$(V)=J1$(A,X):X=INT(3*RND+1)
1020 S$(V)=S$(V)+" AND "+J1$(A,X)
1030 IF V=1 THEN 1060
1040 FOR V1=1 TO V-1:IF S$(V1)=S$(V) THEN 1010
1050 NEXT V1
1060 PRINT CHR$(64+V);" "+S$(V):NEXT V
1070 E$=INKEY$:IF E$="" THEN 1070
1080 PRINT E$:IF ASC(E$)<>64+R THEN 1140
1090 PRINT:PRINT "CORRECT!"
1100 PRINT:PRINT "TRY AGAIN? (Y/N)"
1110 E$=INKEY$:IF E$="Y" OR E$="y" THEN 630
1120 IF E$<>"N" AND E$<>"n" THEN 1110
1130 CLS:GOTO 500
1140 PRINT:PRINT "THE TOTAL OF THE TWO ITEMS MUST BE"
1150 PRINT USING "LESS THAN $##.##";M/100
1160 GOTO 1100
1170 REM SALES TAX
1180 CLS
1190 A=0:T=INT(4*RND+2)
1200 PRINT "Sales tax on the following items"
1210 PRINT "is";T;"per cent, or $.0";RIGHT$(STR$(T),1)
1220 PRINT "for each dollar spent."
1230 PRINT "What is the total cost?":PRINT
1240 I=INT(4*RND+1):PRINT TAB(5);T2$(I):PRINT
1250 FOR J=1 TO 4:P=B2(I,J)+.25*(INT(4*RND))
1260 PRINT B2$(I,J),:PRINT USING "$###.##";P:A=A+P:NEXT J
1270 PRINT "                         ":PRINT
1280 INPUT " TOTAL COST = $",B
1290 PRINT:TX=1+T/100:TA=A*TX+.005
1300 IF ABS(B-TA)<.01 THEN 1360
1310 PRINT:PRINT "ADD COSTS FOR TOTAL."
1320 PRINT "PRICE OF ITEMS = ";:PRINT USING "$###.##";A
1330 PRINT "MULTIPLY BY ";T/100;" FOR TAX, THEN ADD.":PRINT
1340 PRINT USING "TOTAL COST = $###.##";TA
1350 GOSUB 30:GOTO 1180
1360 PRINT:PRINT "CORRECT!"
```

```
1370 PRINT:PRINT "TRY AGAIN? (Y/N)"
1380 E$=INKEY$:IF E$="Y" OR E$="y" THEN 1180
1390 IF E$<>"N" AND E$<>"n" THEN 1380
1400 CLS:GOTO 500
1410 REM EARNING MONEY
1420 CLS
1430 N=INT(6*RND):H=8+INT(11*RND)
1440 P=1+.25*INT(10*RND)
1450 PRINT N3$(N);" works";H;"hours per week."
1460 IF N<3 THEN PRINT "He earns ";:GOTO 1480
1470 PRINT "She earns ";
1480 PRINT USING "$##.## per hour.";P
1490 IF N<3 THEN PRINT "How much does he earn";:GOTO 1510
1500 PRINT "How much does she earn";
1510 PRINT " in a week?":PRINT
1520 INPUT "$",D:D1=P*H
1530 IF ABS(D-D1)<.001 THEN 1570
1540 PRINT:PRINT "MULTIPLY";H;"HOURS BY";:PRINT USING "$##.##
 PER HOUR.";P
1550 PRINT:PRINT USING "THE ANSWER IS $###.##";D1
1560 GOSUB 30:GOTO 1420
1570 PRINT:PRINT "CORRECT!"
1580 PRINT:PRINT "PRESS 1 TO TRY AGAIN"
1590 PRINT TAB(7);"2 TO CONTINUE"
1600 E$=INKEY$:IF E$="1" THEN 1420
1610 IF E$<>"2" THEN 1600
1620 CLS
1630 N=INT(5*RND):H=INT(11*RND+8)
1640 P=1+.25*INT(10*RND)
1650 PRINT N3$(N);" earns ";:PRINT USING "$##.## per hour.";P
1660 IF N<3 THEN PRINT "He works";:GOTO 1680
1670 PRINT "She works";
1680 PRINT H;"hours per week."
1690 IF N<3 THEN PRINT "How much will he earn in";:GOTO 1710
1700 PRINT "How much will she earn in";
1710 W=INT(19*RND+2):PRINT W;"weeks?":PRINT
1720 INPUT "$",D
1730 D1=P*H*W:IF ABS(D-D1)<.001 THEN 1790
1740 PRINT:PRINT "MULTIPLY";H;"HOURS BY";
1750 PRINT USING "$##.## PER HOUR.";P
1760 PRINT "THEN MULTIPLY BY";W;"WEEKS."
1770 PRINT USING "THE ANSWER IS $###.##";D1
1780 GOSUB 30:GOTO 1620
1790 PRINT:PRINT "CORRECT!"
1800 PRINT:PRINT "PRESS 1 TO TRY AGAIN"
1810 PRINT TAB(7);"2 TO CONTINUE"
1820 E$=INKEY$:IF E$="1" THEN 1620
1830 IF E$<>"2" THEN 1820
1840 CLS
1850 J=INT(5*RND):T=INT(5*RND)
1860 P=1+.25*INT(10*RND)
1870 W=INT(8*RND)+2
1880 PRINT T3$(T);" earned ";
1890 PRINT USING "$##.## last week";P:PRINT J3$(J):PRINT
1900 IF T<3 THEN PRINT "If he";:GOTO 1920
1910 PRINT "If she";
1920 PRINT " earned this amount every week,"
1930 PRINT "what would the total income be"
1940 PRINT "for";W;"weeks?":PRINT
```

```
1950 INPUT "$",D
1960 D1=P*W:IF ABS(D-D1)<.001 THEN 2010
1970 PRINT:PRINT USING "MULTIPLY $##.## PER WEEK";P
1980 PRINT "BY";W;"WEEKS."
1990 PRINT:PRINT USING "THE ANSWER IS $###.##";D1
2000 GOSUB 30:GOTO 1840
2010 PRINT:PRINT "CORRECT!"
2020 PRINT:PRINT "PRESS 1 TO TRY AGAIN"
2030 PRINT TAB(7);"2 START 'EARNING MONEY'' OVER"
2040 PRINT TAB(7);"3 RETURN TO MAIN MENU SCREEN"
2050 E$=INKEY$:IF E$="1" THEN 1840
2060 IF E$="2" THEN 1420
2070 IF E$<>"3" THEN 2050 ELSE CLS:GOTO 500
2080 REM WEEKLY EXPENSES
2090 CLS
2100 I=INT(3*RND+1)
2110 PRINT "Here are the expenses for one week at":PRINT A4$(
I,1);".":PRINT
2120 P=10*INT(5*RND+1)+40
2130 PRINT A4$(I,2);TAB(20);:PRINT USING "$##.##";P:T=P
2140 P=.25*INT(12*RND+1)+2.75
2150 PRINT A4$(I,3);TAB(20);:PRINT USING "$##.##";P:T=T+P
2160 P=.5*INT(8*RND+1)+1.5
2170 PRINT A4$(I,4);TAB(20);:PRINT USING "$##.##";P:T=T+P
2180 P=.5*INT(5*RND+1)+.5
2190 PRINT A4$(I,5);TAB(20);:PRINT USING "$##.##";P:T=T+P
2200 PRINT:INPUT "Total expenses for one week = $",D
2210 IF ABS(D-T)<.001 THEN 2230
2220 PRINT:PRINT USING "ADD THE NUMBERS TO GET TOTAL $###.##"
;T:PRINT
2230 W=INT(7*RND+2)
2240 PRINT:PRINT N4$(INT(6*RND+1));" ";B4$(INT(3*RND+1));" ";
A4$(I,1)
2250 PRINT "for";W;"weeks.  What will it cost?"
2260 INPUT "$",D
2270 IF ABS(D-W*T)<.001 THEN 2320
2280 PRINT:PRINT "MULTIPLY TOTAL EXPENSE PER WEEK"
2290 PRINT "TIMES";W;"WEEKS."
2300 PRINT USING "$###.## * # = $###.##";T,W,T*W
2310 GOSUB 30:GOTO 2090
2320 PRINT:PRINT "CORRECT!"
2330 PRINT:PRINT "TRY AGAIN? (Y/N)"
2340 E$=INKEY$:IF E$="Y" OR E$="y" THEN 2090
2350 IF E$<>"N" AND E$<>"n" THEN 2340
2360 CLS:GOTO 500
2370 REM SAVING MONEY
2380 CLS
2390 R6=INT(6*RND+1)
2400 PRINT N5$(R6);" wants to buy a ";
2410 R3=INT(3*RND+1):PRINT A5$(R3);"."
2420 P=B5(R3)+M5(R3)*INT(F5(R3)*RND+1)
2430 PRINT USING "It will cost $###.##.";P
2440 IF R6>=4 THEN E$="he" ELSE E$="she"
2450 W=10*INT(4*RND+1)
2460 PRINT "If ";E$;" saves for";W;"weeks,"
2470 PRINT "how much will ";N5$(R6)" need to save"
2480 PRINT "each week?":PRINT
2490 INPUT "$",D:IF ABS(D-P/W)<.01 THEN 2530
2500 PRINT:PRINT USING "TOTAL COST $###.## DIVIDED BY ##";P,W
```

```
2510 PRINT USING "WEEKS = $###.##";P/W
2520 GOSUB 30:GOTO 2380
2530 PRINT:PRINT "CORRECT!"
2540 PRINT:PRINT "ANOTHER PROBLEM? (Y/N)"
2550 E$=INKEY$:IF E$="Y" OR E$="y" THEN 2380
2560 IF E$<>"N" AND E$<>"n" THEN 2550
2570 CLS:GOTO 500
2580 REM AVERAGES
2590 CLS
2600 Z=INT(3*RND+1):T=0
2610 ON Z GOTO 2620,2710,2810
2620 PRINT "A bowling team had the following scores for one g
ame.":PRINT
2630 X=INT(2*RND)
2640 FOR I=1 TO 4
2650 S=115+INT(40*RND):T=T+S
2660 PRINT N6$(I+X*R);TAB(8);S
2670 NEXT I
2680 PRINT:PRINT "What was the team's average score"
2690 PRINT "for the game?"
2700 N=4:F=10:GOTO 2910
2710 PRINT "A basketball team won the following"
2720 PRINT "number of games.":PRINT
2730 N=4+INT(3*RND+1):Y=1983-N
2740 FOR I=1 TO N
2750 S=50+INT(20*RND):T=T+S:Y=Y+1
2760 PRINT Y;TAB(9);S
2770 NEXT I
2780 PRINT:PRINT "What was the average number of games"
2790 PRINT "per year the team won during these   years?"
2800 F=6:GOTO 2910
2810 PRINT "A fullback gained the following"
2820 PRINT "number of yards in several football  games."
2830 N=4+INT(3*RND+1)
2840 FOR I=1 TO N
2850 S=60+INT(30*RND):T=T+S
2860 PRINT TAB(5);S
2870 NEXT I
2880 PRINT:PRINT "What was the fullback's average"
2890 PRINT "number of yards gained per game?"
2900 F=10
2910 A=INT(T/N+.5):PRINT
2920 C=INT(4*RND+1):ON C GOTO 2930,2960,3010,3050
2930 PRINT "A ";A
2940 FOR I=1 TO 3:A=A+INT(F*RND+1):PRINT CHR$(65+I)+" ";A:NEX
T I
2950 GOTO 3080
2960 PRINT "A ";A-INT(F*RND+1)
2970 PRINT "B ";A
2980 A=A+INT(F*RND+1):PRINT "C ";A
2990 A=A+INT(F*RND+1):PRINT "D ";A
3000 GOTO 3080
3010 I=A-INT(F*RND+1):J=I-INT(F*RND+1)
3020 PRINT "A ";J:PRINT "B ";I:PRINT "C ";A
3030 PRINT "D ";A+INT(F*RND+1)
3040 GOTO 3080
3050 I=A-INT(F*RND+1):J=I-INT(F*RND+1):K=J-INT(F*RND+1)
3060 PRINT "A ";K:PRINT "B ";J
3070 PRINT "C ";I:PRINT "D ";A
```

```
3080 E$=INKEY$
3090 IF E$>="a" AND E$<="d" THEN CE=ASC(E$)-96:GOTO 3120
3100 IF E$<"A" OR E$>"D" THEN 3080
3110 CE=ASC(E$)-64
3120 IF CE=C THEN 3160
3130 PRINT:PRINT "NO, THE ANSWER IS ";CHR$(64+C)
3140 PRINT "DIVIDE TOTAL BY NUMBER OF ITEMS."
3150 GOSUB 30:GOTO 2590
3160 PRINT:PRINT "CORRECT!"
3170 PRINT:PRINT "ANOTHER PROBLEM? (Y/N)"
3180 E$=INKEY$:IF E$="Y" OR E$="y" THEN 2590
3190 IF E$<>"N" AND E$<>"n" THEN 3180
3200 CLS:GOTO 500
3210 CLS
3220 END
```

Chapter 6

Arrays and DATA Statements

Chapter 6

Arrays and DATA Statements

Memory locations, or addresses, are like a wallful of post office boxes, each with its own name or label. Each location contains a value. For example, suppose we have these values assigned to these locations at the beginning of a program:

```
A=3
B=4
X=10
```

The boxes would look like this:

A	B	X
3	4	10

Later in the program you may change the values:

```
A=7
B=A+2
X=A+B
```

The values in the boxes change; they become

A	B	X
7	9	16

Each of these boxes has a name, and each name represents only one box.

Now, just as in the post office, some boxes can be bigger than others:

A	B	X
C		

The C box can be divided into smaller parts, but they are still parts of C. In this case, the C box holds an *array*, and different values can go into each section of C. We specify each

part of C with a subscript, a number in parentheses. The names of the elements of the array C are C(1), C(2), and C(3).

A	B	X
C(1)	C(2)	C(3)

Boxes can be even larger—representing one, two, or even more dimensions. Here is a chart of D, which has two dimensions. The first subscript may be 1 or 2, and the second subscript may be 1, 2, 3, or 4.

A	B	X	D(1,1)	D(1,2)	D(1,3)	D(1,4)
C(1)	C(2)	C(3)	D(2,1)	D(2,2)	D(2,3)	D(2,4)

Efficient Programming

Arrays can make a repetitive computer program more efficient. Suppose you are describing three children whose names are Richard, Robert, and Randy. We can say:

```
NAME$(1)="Richard"
NAME$(2)="Robert"
NAME$(3)="Randy"
```

Now we wish to list some things about these children:

```
AGE(1)=14
AGE(2)=9
AGE(3)=5

COLOR$(1)="Black"
COLOR$(2)="Red"
COLOR$(3)="Blue"
```

```
SPORT$(1)="Baseball"
SPORT$(2)="Football"
SPORT$(3)="Basketball"
```

We now have our information about the children in four arrays. You can print a list of the children by using a single loop and a variable subscript:

```
200 FOR J=1 TO 3
210 PRINT NAME$(J);AGE(J),SPORT$(J)
220 NEXT J
```

If you wish to know about a particular child, print only his or her information by searching the arrays for a particular subscript.

```
N=2:PRINT NAME$(N),COLOR$(N)
```

If you have a longer list, you could sort. For example, to find all the nine-year-olds, for any given total number (T) of children:

```
400 FOR J=1 TO T
410 IF AGE(J)<>9 THEN 430
420 PRINT NAME$(J)
430 NEXT J
```

The computer will execute only line 420 and print a name when the value of AGE(J) is 9.

Two-dimensional arrays. This information about the children could be in a two-dimensional array rather than in the four one-dimensional arrays above. Call the main array PERSON$. The data may be arranged like this:

```
PERSON$(1,1)="Richard"
PERSON$(1,2)="14"
PERSON$(1,3)="Black"
PERSON$(1,4)="Baseball"
```

```
PERSON$(2,1)="Robert"
PERSON$(2,2)="9"
PERSON$(2,3)="Red"
PERSON$(2,4)="Football"

PERSON$(3,1)="Randy"
PERSON$(3,2)="5"
PERSON$(3,3)="Blue"
PERSON$(3,4)="Basketball"
```

The first subscript tells us which child's data is held in that variable, and the second subscript identifies the category of information, name, age, color, and sport. The word or number in quotation marks is the string placed in each address of our post office boxes.

Arrays can contain both numeric variables and string variables.

Creating Dimensions

If you use a variable name with a subscript without first DIMensioning that variable, the computer automatically reserves 11 elements for the array (subscripts 0 through 10). If you need more than 11, use a DIM statement to clear enough space:

```
DIM D(3Ø)
```

If you want to conserve memory and you do not need all 11 elements, you can save memory by DIMensioning the array for fewer elements:

```
1Ø DIM A(6)
```

The DIMension statement must appear before any reference to the array; it is wise to put all DIMension statements near the beginning of the program.

The computer automatically starts numbering all subscripts with zero. In other words, there can be elements such as D(0) and E(1,0). Since the zero variable counts as one element, a statement like DIM A(10) reserves 11 subscripted vari-

ables, A(0) through A(10). If you prefer to use only elements numbered one and above, you may use the OPTION BASE statement:

```
OPTION BASE 1

DIM A(5),B(12)
```

DATA, READ, and RESTORE Statements

A DATA statement is always associated with a READ statement, and together they essentially perform LET, or assignment, processes, assigning values to variables. (By the way, the command LET is optional in Amiga BASIC; LET A=4 usually is written A=4.)

Suppose we want to initialize several variables, then print some combinations of the numbers. The program segment would be like this:

```
10 A=4

20 B=7

30 C=3

40 D=5

50 E=12

60 F=2

70 PRINT A+B,C*D,E/F
```

Using DATA and READ, lines 10–60 may be combined like this:

```
10 READ A,B,C,D,E,F

20 DATA 4,7,3,5,12,2

70 PRINT A+B,C*D,E/F
```

Both of these sequences do the same thing. Defining the variables without using DATA statements can sometimes make a program easier to understand, with less chance for error. However, using DATA statements can combine many repetitious lines and thus save memory and make the program more efficient.

When the computer comes to a READ statement, it looks for the first DATA statement. The first variable in the READ statement will correspond with the first number in the DATA statement. In this case, the computer will read 4 for the value of A. The computer will then READ B and will go to the very next DATA item to assign 7 to B, and so on.

The DATA statement may be placed anywhere in the program. For example, you may change line 20 and put the DATA statement at line 5 and place it before the READ statement. Or you may change it to line 80 after the PRINT statement, and the program will work exactly the same. The computer ignores DATA statements until a READ statement is encountered; then the computer will look at the DATA statements in order.

You won't actually see anything while the computer is reading data, but your variables will be assigned the values and you can use them in calculations. To see the results, you will need to PRINT.

Your data items may also be strings. Quite often, you will see a READ statement in a loop to perform repeated operations, perhaps using subscripted variables or variables in an array:

```
10 REM DATA1

20 FOR C=1 TO 10

30 READ N$(C):PRINT N$(C)

40 NEXT C

50 DATA CHERY,RICHARD,CINDY,BOB,RANDY

60 DATA BRETT,ED,BILL,JOHN,JIM
```

In this program segment, N$(1) will be CHERY, N$(2) will be RICHARD, and so on. Since long lines (up to 255 characters) are accepted, all of the data may be typed on one line. Or you may use several lines for the DATA statements. The computer keeps track of a data pointer to know how much data is used and which is the next item to be used. If all the data is finished in one line, the computer goes to the next DATA statement if it needs more. If you do not have enough data items, however, the computer will display an error message.

Your job as a programmer is to make sure that the data matches the READ statements and that items are read in the

right order. You may combine numbers and strings in the same statements as long as you make sure the numbers go to numeric variable names and the strings go to string variable names. If you have extra data items, the computer simply ignores them.

Reusing data. The RESTORE statement lets you reuse the data items or makes sure the computer starts with the very first DATA statement in the program. RESTORE moves the data pointer from whatever data items have already been used back to the first item. Program 6-1 is an example.

Program 6-1. DATA2

```
10 REM DATA2
20 FOR C=1 TO 3
30 READ A,B
40 PRINT A;"+";B;"=";A+B
50 NEXT C
60 PRINT
70 RESTORE
80 FOR C=1 TO 2
90 READ X,Y
100 PRINT X;"*";Y;"=";X*Y
110 NEXT C
120 DATA 2,4,8,5,7,3
130 END
```

When you run this program, the values of A and B will first be 2 and 4, then 8 and 5, then 7 and 3. Line 70 RESTOREs the data. X and Y will then be read as 2 and 4 the first time through the loop, and 8 and 5 the second time through the loop. We didn't need to use all the data. If there were another READ statement later in the program, the data value would be 7.

In the program above, it wouldn't matter if we interchanged lines 60 and 70; the result would be the same. The RESTORE and READ statements may be separated by other statements. Notice, however, that the program would be different if you put the RESTORE statement between lines 80 and 90.

If you spend a little time experimenting with DATA, READ, and RESTORE statements, you'll soon understand how they work. Try putting your statements in different places in the program. Try using different numbers of items in the DATA and READ statements.

A specific starting point. Another useful feature of the RESTORE statement is that you can specify a data line num-

ber. RESTORE alone will start the data over with the first DATA line in the program. RESTORE *n*, where *n* is a line number, will start the data over with the data in line *n*. For example, RESTORE 800 in a program means that the next READ statement will start with the data in line 800. This command can really help you keep track of DATA statements. If you have a long program with a lot of data, you can arrange your data properly, then use RESTORE *n* before each READ statement so that you know exactly which data goes with which segment of the program.

Following is an example of using RESTORE in a music program. The data items are frequencies for the SOUND command. Some of the musical phrases will be used more than once. Rather than making you type more DATA statements (which would be repetitious), RESTORE allows the same data to be used over again.

In this example, GOSUB 120 will perform the subroutine in lines 120–160. This subroutine READs a frequency F, then plays the note, and repeats this process for seven notes. To make it easier to understand, the data items are arranged seven to a line. Line 20 will read the data in line 40. Line 30 will read the data in line 50. Line 60 RESTOREs the data in line 40 so that line 70 will use the data in line 40 again. Since line 40 is also the first DATA statement, I could have used either RESTORE or RESTORE 40. Line 80 says to RESTORE 100, so line 90 will use the data of line 100.

Program 6-2. DATA3

```
10 REM DATA3
20 GOSUB 120
30 GOSUB 120
40 DATA 262,330,294,262,330,392,349
50 DATA 440,392,494,523,587,494,523
60 RESTORE 40
70 GOSUB 120
80 RESTORE 100
90 GOSUB 120
100 DATA 262,330,294,262,392,330,262
110 GOTO 170
120 FOR C=1 TO 7
130 READ F
140 SOUND F,10
150 NEXT C
160 RETURN
170 END
```

If you have problems with this sample program, the most likely place to look for typing errors is in the DATA state-

ments. Remember that those numbers are frequencies, so they will all be three-digit numbers. Make sure you have the commas placed correctly, and make sure you don't have a comma at the end of a line.

One thing to keep in mind with programming is that there are several ways to write a program to accomplish the same thing. The DATA/READ process is just one method, and even then there are different ways of arranging your DATA and READ statements to make the computer do what you want it to do.

The "Math Competency" program in the previous chapter illustrates the use of arrays for the variables.

The "Roman Numerals" program is another program that uses arrays and DATA statements. Each element of the array contains the roman numeral equivalent.

Next, let's look at a program which creates the Braille alphabet—you could use the same idea for Morse code or for lowercase letters corresponding to capital letters or your own symbol code.

This program illustrates using arrays for the variables. Instead of using A=100000, B=101000, C=110000, and so forth, the variables are B$(1), B$(2),..., through B$(26) for the 26 letters of the alphabet. Each B$ holds a code for the Braille dots. There are six positions for dots in Braille. The code for each letter consists of ones and zeros; a one indicates a dot and a zero indicates a blank space.

Line 70 is a FOR-NEXT loop that reads in the 26 values. The data is contained in lines 80–120. To help you in typing the lines, these DATA statements contain five numbers each, except the last statement, which has six.

Lines 210–280 contain a subroutine that converts the ones and zeros to the graphic representation of the Braille symbol. One by one, each number of the six-digit code is examined. If the number is one, a lowercase *o* is printed representing a dot. If the number is zero, a space is printed.

The program consists of three parts. The first part prints the Braille alphabet in order. (If you want to quit and return to the main menu, press RETURN.) The second part allows you to press any letter key to see the equivalent Braille symbol. Part 3 is a quiz. A random letter is shown. The user presses the letter key for the Braille pattern. In any section you can get back to the main menu screen by pressing RETURN.

Outline of Braille Program

Lines	Explanation
20–50	Clear screen and print title.
60	Dimensions arrays B$(26) for the Braille equivalents of the 26 letters and N(26) for use in the quiz.
70	READs in the values for B$.
80–120	DATA containing the codes for the Braille letters.
130–200	Print main menu screen and branch.
210–280	Subroutine to print the Braille symbol in three rows of two positions each. COLOR 2,1 changes to black printing on a white background. MID$ looks at the code one digit at a time.
290–300	Subroutine to delay.
330–350	Print introduction for printing Braille alphabet.
360–400	For the 26 letters, print the Braille equivalent. INKEY$ checks to see if the RETURN key is pressed.
410–430	Wait for user to press RETURN.
440–480	Print instructions for second part of program.
490–540	Detect key pressed by user, which must be RETURN or one of the alphabet letters.
550	Prints the letter pressed and its Braille equivalent.
560	Branches back to line 490.
570–630	Print instructions for quiz.
640	Initializes N array to keep track of which letters have been used; initializes G for number of guesses.
650	Performs quiz for 26 letters.
660	Chooses a random letter that has not been chosen before.
670	Initializes flag F for use in keeping track of missed letters.
680	Increments number of guesses, G.
690–700	Print Braille letter.
710–760	Receive answer.
770	If answer is correct, prints message.
780–820	If answer is incorrect, return for another guess. If answer is incorrect twice, show correct letter. Wait for user to press RETURN.
830	If answer is correct, sets N(T) to zero so the letter will not be chosen again.
840	Goes to next letter.
850–860	Print score.
870	Transfers to line 410.
880	Ends.

Program 6-3. Braille

```
10 REM BRAILLE
20 CLS
30 PRINT TAB(20);STRING$(20,"*")
40 PRINT TAB(20);"* BRAILLE ALPHABET *"
50 PRINT TAB(20);STRING$(20,"*")
60 DIM B$(26),N(26)
70 FOR T=1 TO 26:READ B$(T):NEXT T
80 DATA 100000,101000,110000,110100,100100
90 DATA 111000,111100,101100,011000,011100
100 DATA 100010,101010,110010,110110,100110
110 DATA 111010,111110,101110,011010,011110
120 DATA 100011,101011,011101,110011,110111,100111
130 PRINT:PRINT:PRINT TAB(5);"CHOOSE:"
140 PRINT TAB(15);"1  SEE COMPLETE ALPHABET"
150 PRINT TAB(15);"2  CHOOSE LETTERS"
160 PRINT TAB(15);"3  QUIZ"
170 PRINT TAB(15);"4  END PROGRAM"
180 E$=INKEY$:IF E$<"1" OR E$>"4" THEN 180
190 CLS
200 ON VAL(E$) GOTO 330,440,570,880
210 FOR J=1 TO 3:COLOR 2,1
220 A$=MID$(B$(T),J*2-1,1)
230 C$=MID$(B$(T),J*2,1)
240 IF A$="1" THEN PRINT " o "; ELSE PRINT "   ";
250 IF C$="1" THEN PRINT "o "; ELSE PRINT "  ";
260 COLOR 1,0:PRINT:PRINT TAB(16);
270 NEXT J:PRINT:PRINT
280 RETURN
290 FOR DELAY=1 TO 2000:NEXT DELAY
300 RETURN
330 PRINT TAB(20);"ALPHABET"
340 PRINT:PRINT "PRESS <RETURN> TO RETURN TO MAIN MENU."
350 PRINT:PRINT
360 FOR T=1 TO 26
370 PRINT TAB(8);CHR$(64+T);TAB(16);:GOSUB 210
380 E$=INKEY$:IF E$=CHR$(13) THEN CLS:GOTO 120
390 GOSUB 290
400 NEXT T
410 PRINT:PRINT "PRESS <RETURN>."
420 E$=INKEY$:IF E$<>CHR$(13) THEN 420
430 CLS:GOTO 120
440 PRINT "Press a letter.  The Braille equivalent will be s
hown."
450 PRINT
460 PRINT "Press <RETURN> to get back to the main menu scree
n."
470 PRINT
480 PRINT "Start by pressing any letter.":PRINT:PRINT
490 E$=INKEY$
500 IF E$="" THEN 490
510 IF E$=CHR$(13) THEN CLS:GOTO 120
520 IF E$<"A" OR E$>"z" THEN 490
530 IF E$>"Z" AND E$<"a" THEN 490
540 IF E$<"a" THEN T=ASC(E$)-64 ELSE T=ASC(E$)-96
550 PRINT TAB(8);CHR$(T+64);TAB(16);:GOSUB 210
560 GOTO 490
570 PRINT TAB(20);"BRAILLE QUIZ"
580 PRINT:PRINT "You will see a Braille representation of on
e"
```

```
590 PRINT:PRINT "of the letters of the alphabet."
600 PRINT:PRINT "Type the letter."
610 PRINT:PRINT "The quiz consists of 26 letters."
620 PRINT:PRINT "Press <RETURN> if you prefer to stop the qu
iz."
630 PRINT:PRINT:PRINT:PRINT
640 FOR T=1 TO 26:N(T)=1:NEXT T:G=0
650 FOR Z=1 TO 26
660 T=INT(26*RND)+1:IF N(T)=0 THEN 660
670 F=0
680 G=G+1
690 PRINT TAB(16);
700 GOSUB 210
710 E$=INKEY$
720 IF E$="" THEN 710
730 IF E$=CHR$(13) THEN CLS:GOTO 120
740 IF E$>="A" AND E$<="Z" THEN E=ASC(E$)-64:GOTO 760
750 IF E$>="a" AND E$<="z" THEN E=ASC(E$)-96 ELSE 710
760 PRINT TAB(16);E$:PRINT
770 IF E=T THEN PRINT "CORRECT":PRINT:GOTO 830
780 F=F+1:IF F<2 THEN PRINT "SORRY, TRY AGAIN.":PRINT:PRINT:
GOTO 680
790 PRINT "THE CORRECT ANSWER IS ";CHR$(T+64)
800 PRINT:PRINT "PRESS <SPACE BAR>":PRINT:PRINT
810 E$=INKEY$:IF E$<>" " THEN 810
820 GOTO 660
830 N(T)=0
840 NEXT Z
850 PRINT:PRINT "OUT OF 26 LETTERS,"
860 PRINT "YOU REQUIRED";G;"GUESSES."
870 GOTO 410
880 END
```

The "States and Capitals" program in the sample program section (Chapter 14) illustrates the use of READ, DATA, and RESTORE in drawing the states. Line labels are used to help keep track of which data goes with which state.

Arrays in Recipes

Here is another program, "Cookie File," (Program 6-4), that illustrates the use of READ and DATA statements. This program uses line numbers instead of labels. Arrays are used to keep track of an ingredient list and an inventory of ingredients. DATA statements are used to hold recipes for cookies. RESTORE is used to get to a certain recipe.

The first section lists the ingredients. You may specify Y for yes or N for no, indicating whether you have the ingredient or not. After you have gone through the list of ingredients, the computer will let you know which cookies can be made with those ingredients.

The second section of the program prints the recipe for

the cookie you choose. Fifteen recipes are included. After the recipe is printed, you may convert the recipe if you wish—multiply the recipe by two or three or one half, or even a number with a fraction such as 7.5.

Lines 1310–1350 list the ingredients with their measurements. This information is READ in in lines 30–60. The ingredients are saved in the INV$ array, and the ingredients with the measurements are in the ING$ array. INV$(*n*,0) stores the names of the ingredients for the *n* ingredients, and INV$(*n*,1) stores a Y or N for the inventory list.

The cookie recipes are in the DATA statements in lines 1360–1500. Following the name of the cookie are the amounts of the ingredients in the same order as they are in the ingredient list. Only those ingredients used will have numbers. In typing these statements, you need to be very careful to get the commas in the right places or you might get data errors or wrong recipes. The last number in the line is the oven temperature. Lines 810–960 RESTORE the proper data for each type of cookie.

When the computer is checking to see which cookies can be made, the data for the cookies is restored. If a there is a number in the recipe indicating an ingredient, the corresponding ingredient in the inventory list is checked and has to be a Y. If it is an N, indicating you don't have the ingredient, that cookie name will not be printed.

Outline of Cookie File Program

Lines	Explanation
20	DIMensions arrays for 20 ingredients and inventory items.
30–60	READ in from DATA the measurements and names of the ingredients.
70–150	Clear screen, print main menu screen, then branch.
160–200	Print instructions for inventory list.
210	Initializes number of Y (Yes) ingredients.
220–320	For each ingredient print the name and record response of Y or N.
330	Initializes C as the number of cookies that can be made.
340–380	If the user does not have flour or sugar or if the number of available ingredients is four or fewer, print message for no cookies.

Lines	Explanation
390–410	Wait for user to press RETURN; branch to main menu.
420	RESTOREs data for cookie recipes.
430	READs name of cookie.
440–480	Compare ingredients in recipe with inventory list.
490–500	Print name of cookie that can be made; increment counter.
510–550	Read next data item, checking for next cookie.
560	If no cookies can be made, branches back to message.
570–580	Print message and branch to wait procedure.
590–760	Print list of available cookie recipes.
770–790	Receive choice, making sure key pressed is available letter for recipe.
800–810	Branch to desired cookie.
820–960	RESTORE proper DATA statement for recipe.
970	Prints name of cookie.
980–1060	READ data for ingredient. If a number is read, that is the measurement for the corresponding ingredient.
1070–1080	READ and print temperature.
1090	Prints instruction for certain recipes.
1100–1180	Ask if user wants to convert recipe, and if so, by what number or fraction.
1190–1240	Print converted recipe.
1250–1260	Ask if user wants to convert again.
1270–1290	Wait for user to press RETURN before going to main menu.
1300–1350	DATA for measures and ingredients. Quotation marks are used because measures have a trailing space.
1360–1510	DATA for cookie recipes. Be careful typing commas.
1520–1530	Clear screen and end.

Program 6-4. Cookie File

```
10 REM COOKIE FILE
20 DIM ING$(19),INV$(19,1)
30 FOR I=0 TO 19
40 READ A$,INV$(I,0)
50 ING$(I)=A$+INV$(I,0)
60 NEXT I
70 CLS
80 PRINT TAB(12);"COOKIE FILE"
90 PRINT:PRINT "CHOOSE:"
100 PRINT:PRINT "1  NEED TO KNOW WHAT CAN BE MADE"
```

```
110 PRINT:PRINT "2  WANT TO SEE A CERTAIN RECIPE"
120 PRINT:PRINT "3  END PROGRAM"
130 K$=INKEY$:IF K$<"1" OR K$>"3" THEN 130
140 ON VAL(K$) GOTO 160,600,1520
150 REM
160 CLS
170 PRINT "IN THE FOLLOWING LIST"
180 PRINT "PRESS 'Y' IF YOU HAVE THE INGREDIENT."
190 PRINT "PRESS 'N' IF YOU DO NOT."
200 PRINT "PRESS 'S' TO START OVER.":PRINT:PRINT
210 YS=0
220 FOR K=0 TO 19
230 PRINT INV$(K,0);
240 K$=INKEY$
250 IF K$="S" OR K$="s" THEN 160
260 IF K$="N" OR K$="n" THEN 290
270 IF K$<>"Y" AND K$<>"y" THEN 240
280 YS=YS+1
290 PRINT TAB(20);K$
300 IF K$="n" THEN K$="N"
310 INV$(K,1)=K$
320 NEXT K
330 C=0
340 PRINT:PRINT "YOU CAN MAKE:":PRINT
350 IF INV$(0,1)="N" THEN 380
360 IF INV$(7,1)="N" THEN 380
370 IF YS>4 THEN 420
380 PRINT "NOTHING TODAY.":PRINT "YOU NEED MORE SUPPLIES."
390 PRINT:PRINT "PRESS <RETURN> TO CONTINUE."
400 K$=INKEY$:IF K$<>CHR$(13) THEN 400
410 GOTO 70
420 RESTORE 1360
430 READ A$
440 FOR J=0 TO 19
450 READ B$
460 IF B$="" OR B$="0" THEN 480
470 IF INV$(J,1)="N" THEN 510
480 NEXT J
490 PRINT A$
500 C=C+1
510 READ D$
520 IF D$="ZZZ" THEN 570
530 IF LEN(D$)<6 THEN 510
540 A$=D$
550 GOTO 440
560 IF C=0 THEN 380
570 PRINT:PRINT "GO AHEAD AND BAKE!"
580 GOTO 390
590 REM
600 CLS
610 PRINT "CHOOSE:":PRINT
620 PRINT "A  ALMOND COOKIES"
630 PRINT "B  BALL COOKIES"
640 PRINT "C  BROWNIES"
650 PRINT "D  BUTTERSCOTCH BARS"
660 PRINT "E  CHOCOLATE CHIP BARS"
670 PRINT "F  CHOCOLATE CHIP COOKIES"
680 PRINT "G  CHOCOLATE DROP COOKIES"
690 PRINT "H  HONEY BALLS"
700 PRINT "I  HONEY SPICE COOKIES"
710 PRINT "J  MEXICAN WEDDING COOKIES"
```

```
720 PRINT "K  OATMEAL CHOCOLATE CHIPS"
730 PRINT "L  OATMEAL CRISPS"
740 PRINT "M  SNICKERDOODLES"
750 PRINT "N  SUGAR COOKIES"
760 PRINT "O  TOFFEE BARS"
770 C$=INKEY$
780 IF C$>="a" AND C$<="o" THEN C=ASC(C$)-96:GOTO 800
790 IF C$<"A" OR C$>"Z" THEN 770 ELSE C=ASC(C$)-64
800 CLS
810 ON C GOTO 820,830,840,850,860,870,880,890,900,910,920,93
0,940,950,960
820 RESTORE 1360:GOTO 970
830 RESTORE 1370:GOTO 970
840 RESTORE 1380:GOTO 970
850 RESTORE 1390:GOTO 970
860 RESTORE 1400:GOTO 970
870 RESTORE 1410:GOTO 970
880 RESTORE 1420:GOTO 970
890 RESTORE 1430:GOTO 970
900 RESTORE 1440:GOTO 970
910 RESTORE 1450:GOTO 970
920 RESTORE 1460:GOTO 970
930 RESTORE 1470:GOTO 970
940 RESTORE 1480:GOTO 970
950 RESTORE 1490:GOTO 970
960 RESTORE 1500:GOTO 970
970 READ A$:PRINT A$:PRINT
980 I=0
990 FOR J=0 TO 19
1000 READ B$
1010 IF B$="" OR B$="0" THEN 1060
1020 AMT(I)=VAL(B$)
1030 INGR$(I)=ING$(J)
1040 PRINT AMT(I);INGR$(I)
1050 I=I+1
1060 NEXT J
1070 READ T
1080 PRINT:PRINT "Bake at";T;"degrees."
1090 IF C=8 OR C=10 THEN PRINT "Roll in powdered sugar."
1100 PRINT:PRINT "WANT TO CONVERT RECIPE? (Y/N)"
1110 K$=INKEY$
1120 IF K$="N" OR K$="n" THEN 1270
1130 IF K$<>"Y" AND K$<>"y" THEN 1110
1140 PRINT:PRINT "MULTIPLY BY WHAT NUMBER"
1150 INPUT "OR DECIMAL FRACTION";F
1160 IF F>0 THEN 1190
1170 PRINT:PRINT "SORRY, F>0"
1180 GOTO 1140
1190 CLS
1200 PRINT F;"TIMES ORIGINAL RECIPE":PRINT:PRINT
1210 PRINT A$:PRINT
1220 FOR K=0 TO I-1
1230 PRINT F*AMT(K);INGR$(K)
1240 NEXT K
1250 PRINT:PRINT "CONVERT AGAIN? (Y/N)"
1260 GOTO 1110
1270 PRINT:PRINT "PRESS <RETURN> TO CONTINUE."
1280 K$=INKEY$:IF K$<>CHR$(13) THEN 1280
1290 GOTO 70
1300 REM
```

```
1310 DATA "c. ",shortening,"c. ",sugar,"c. ",brown sugar
1315 DATA "c. ",powdered sugar,"tbsp. ",honey,"",eggs
1320 DATA "tsp. ",vanilla,"c. ",flour,"tsp. ",baking powder
1325 DATA "tsp. ",baking soda,"tsp. ",salt
1330 DATA "tsp. ",cinnamon,"tbsp. ",cocoa,"tsp. ",almond ext
ract
1335 DATA "c. ",milk,"c. ",oatmeal
1340 DATA "oz. ",chocolate chips,"doz. ",almonds
1350 DATA "tsp. ",cake decors,"c. ",cinnamon & sugar
1360 DATA ALMOND COOKIES,2,2,,,,2,,3,2,,,,,2,,,,4,,,375
1370 DATA BALL COOKIES,.5,.33,,,,1,.5,.75,,,,,,,,,,2,,375
1380 DATA BROWNIES,.5,1,,,,2,1,.75,.5,,.5,,6,,,,,,,,350
1390 DATA BUTTERSCOTCH BARS,.5,,2,,,2,1,1.75,2,,.25,,,,,,,,
,375
1400 DATA CHOCOLATE CHIP BARS,.5,,1,,,1,1,1.75,,.5,.5,,,,.5,
,12,,,,350
1410 DATA CHOCOLATE CHIP COOKIES,.5,.25,.5,,,1,.5,1,,.5,.5,,
,,,,6,,,,375
1420 DATA CHOCOLATE DROP COOKIES,.5,,1,,,1,1,1.67,,.5,.5,,6,
,.5,,,,,,350
1430 DATA HONEY BALLS,.5,,,,2,,1,1,,,.25,,,,,,,,,300
1440 DATA HONEY SPICE COOKIES,.5,.75,,,4,,.5,1,,,,.5,,,,,,,,
,375
1450 DATA MEXICAN WEDDING COOKIES,.75,,,.67,,,1,1.5,,,.25,1,
,,,.75,,,,,325
1460 DATA OATMEAL CHOCOLATE CHIPS,1,1,.5,,,2,1,2,,1,1,,,,,2,
6,,,,350
1470 DATA OATMEAL CRISPS,1,1,1,,,2,1,1.5,,1,1,,,,,3,,,,,350
1480 DATA SNICKERDOODLES,1,1.5,,,,2,,2.75,3,,.5,,,,,,,,,.5,4
00
1490 DATA SUGAR COOKIES,.67,.75,,,,1,.5,2,1.5,,.25,,,,,.25,,,
,,,375
1500 DATA TOFFEE BARS,1,,1,,,,1,2,,,,,,,,,6,,,,350
1510 DATA ZZZ
1520 CLS
1530 END
```

Chapter 7

Menus, Windows, and the Mouse

Chapter 7

Menus, Windows, and the Mouse

You can look at how menus work on the Amiga when you first turn on the computer. After inserting the Workbench disk, push the right mouse button and move toward the top of the screen. The top highlighted line will change and menu titles will appear. As you touch a title, a menu will drop down with several options. As you move the mouse downward, the subtitles will be highlighted. To make a selection, you place the pointer over the item you want and then release the right mouse button.

You can use this same menu structure in your BASIC programs. The form for creating a menu is

MENU ***menu-id,item-id,state [,title]***

The *menu-id* is the number assigned to the menu bar selection and can be a number from 1 through 10. If you select 1, for example, your menu will be in the leftmost menu position.

The *item-id* is the number assigned to the menu item under the menu bar and can be a value from 0 through 20. Item-id 0 refers to the entire menu. The other numbers are for the choices under the main topic.

The *state* argument is 0 to disable, 1 to enable, or 2 to enable and place a checkmark.

The *title* is a string containing the title of the item chosen.

MENU Functions

MENU ON enables event trapping or use of the ON MENU GOSUB statement.

There are two functions involved with MENU. MENU(0) is similar to INKEY$ and is reset to zero every time it executes, but returns a number which corresponds to the number of the last menu bar selection made—the main menu chosen.

MENU(1) returns a number which corresponds to the number of the last menu item chosen or the subtopic chosen.

MENU RESET restores the original Amiga BASIC default menu bar.

Program 7-1 illustrates the use of the MENU statements.

This sample only prints a statement when an item is chosen, but this method could be used to choose actual menu items in a program.

Line 20 resets the second position menu to FRACTIONS. Under this title there will be five subtopics, which are defined in lines 30–70.

Lines 100–110 check to see whether the second menu has been selected. For our example, the program just stays at these lines until FRACTIONS is chosen.

Under FRACTIONS are the five choices. Line 120 uses MENU(1) to see which item has been chosen and calls it CHOICE. Line 130 branches depending on the item chosen. If you choose to end the program, line 610 resets the menu bar to the original second menu in Amiga BASIC.

Program 7-1. Menus

```
10 REM MENUS
20 MENU 2,0,1,"FRACTIONS"
30 MENU 2,1,1,"Simplify"
40 MENU 2,2,1,"Add"
50 MENU 2,3,1,"Multiply"
60 MENU 2,4,1,"Divide"
70 MENU 2,5,1,"End Program"
80 MENU ON
90 CLS
100 M=MENU(0)
110 IF M<>2 THEN 100
120 CHOICE=MENU(1)
130 ON CHOICE GOSUB 200,300,400,500,600
140 GOTO 100
200 PRINT "Simplify Fractions"
210 RETURN
300 PRINT "Add Fractions"
310 RETURN
400 PRINT "Multiply Fractions"
410 RETURN
500 PRINT "Divide Fractions"
510 RETURN
600 PRINT "End Program"
610 MENU RESET
620 END
```

Creating Windows

You can make your own windows in BASIC with the versatile WINDOW command. The basic command is

WINDOW *id*

where *id* is a number which identifies the window. For example, The Output window that appears while you are in

BASIC is window number 1, so for your own windows you should specify a number greater than 1.

You can add more information to a WINDOW statement to further define your program's window. In order, you can give the window a title, specify the size similar to the LINE or GET command, specify a type which sets up how much the user can do with the window, and choose a screen ID which can be a value from 1 through 4. When you use the WINDOW statement, a new Output window is created and displayed and brought to the front of the screen.

The title is a string expression that will show up in the title bar at the top of the window.

The type is a number from 0 through 31:

1	Window size can be changed and a sizing gadget appears in the lower right side of the window.
2	Window can be moved about using the title bar.
4	Window can be moved from front to back of other windows, and that gadget appears in the upper right corner.
8	Window can be closed using a close gadget.
16	Contents of window reappear after the window has temporarily been covered by another window.

To specify a type, add together two or more of these values.

To define the size of the window, use the rectangular coordinates of the form (x1,y1)–(x2,y2), where (x1,y1) are the coordinates of the upper left corner and (x2,y2) are the coordinates of the lower right corner on the full screen. If you don't specify coordinates for the size, the window appears at the current default for that window—whatever was previously set in the program. The initial defaults are for the full screen.

WINDOW CLOSE *id* is the command to make the named window invisible.

WINDOW OUTPUT *id* names the window for current output without moving the window to the front—direct output can go to a window that is behind another.

A Window Illustration

The next program illustrates some of the options of the WINDOW statement. It sets up three windows and puts something in each.

Line 30 defines window 2, entitled *Printing*. It will be at the rectangle from (10,10) to (250,50), which is the upper left section of the screen. Its type number is 14, which says the window can be moved about using the title bar; it has the front and back option; and it can be closed by the upper left corner gadget (2 + 4 + 8 = 14).

Line 40 sets up window 3 with the title *Lines*. It will be at the right side of the screen and has a type of 7. It can be moved about using the title bar; it can be moved from front to back; and the window size can be changed.

Line 50 defines window 4 with the title *Circles*. It will appear in the lower left section of the screen and has type 6. This window can be moved about and has the front and back gadget.

When you run the program and the windows appear, you can see the different gadgets available in the corners of each window. You can experiment with the mouse to see which options are available.

Lines 60–80 print a message in window 2. Notice that the COLOR command only applies to that window.

Lines 90–130 draw lines in window 3. Keep in mind that the coordinates specified in graphics commands are relative to that window wherever it is—not the whole output screen.

Lines 140–190 draw circles in window 4. Again, notice the coordinates are for that window, not the big screen.

Program 7-2. Windows

```
10 REM WINDOWS
20 CLS
30 WINDOW 2,"Printing",(10,10)-(250,50),14
40 WINDOW 3,"Lines",(265,15)-(500,65),7
50 WINDOW 4,"Circles",(15,65)-(300,180),6
60 WINDOW 2
70 COLOR 3,2:LOCATE 3,5
80 PRINT "This is Window 2"
90 WINDOW 3
100 X2=10
110 LINE(0,0)-(X2,100)
120 X2=X2+10
130 IF X2<400 THEN 110
140 WINDOW 4
150 X=15:Y=10
160 FOR I=1 TO 9
170 CIRCLE (X,Y),20
180 X=X+20:Y=Y+10
190 NEXT I
200 END
```

Controlling the Mouse

You can control cursor movements and make menu selections with either the keyboard or the mouse, but using the mouse gives both the programmer and the program user more flexibility. There are several commands and functions in Amiga BASIC that relate to the mouse and the position of the pointer arrow. You should refer to your manual for detailed discussion of these commands. Here we will summarize the commands and give some sample programs of how the mouse commands work.

The **MOUSE ON** statement enables event trapping based on the user's pressing the left mouse button.

Related to MOUSE ON is **ON MOUSE-GOSUB**, which directs the program for events.

MOUSE OFF disables the ON MOUSE event trapping.

MOUSE STOP suspends mouse event trapping—event trapping continues but the ON MOUSE–GOSUB statement is not executed until a subsequent MOUSE ON statement is executed.

The **MOUSE*(n)*** functions are listed in your Amiga BASIC manual in detail. This function returns values that indicate whether the left mouse button was pressed and give information about the position of the arrow. The function parameter *n* can be a number from 0 through 6.

This first short program (Program 7-3) checks the position of the mouse when the button is pressed. Line 30 uses MOUSE(0) to check whether the button is pressed. The program stays at this line until the value of the function is not zero. When you press the left button, the value is no longer zero and the program continues. Line 40 uses MOUSE(5) to determine the ending x coordinate and MOUSE(6) for the ending y coordinate. These values are printed on the screen.

Line 50 returns to line 30. As you run this program, move the mouse to various places to see the coordinates returned when you press the left mouse button. To stop the program, press CTRL-C (Break).

Program 7-3. Mouse

```
10 REM MOUSE
20 CLS
30 IF MOUSE(0)=0 THEN 30
40 PRINT MOUSE(5);",";MOUSE(6)
50 GOTO 30
60 END
```

Now, what do you do with those coordinates? Here's a short routine (Program 7-4) that shows how you can draw by pressing the left button and moving the mouse around.

Line 20 defines x and y to be integers using DEFINT. Line 30 clears the screen. Line 40 waits until the left mouse button is pressed. When the button is pressed, line 50 checks the current x coordinate with MOUSE(1) and the current y coordinate with MOUSE(2) and returns the values x and y. These coordinates are used in the PSET command in line 60 to turn on a point—place a white dot on the blue screen. Line 70 branches back to line 40 to keep checking the mouse button.

Program 7-4. Drawing with the Mouse

```
10 REM DRAWING WITH MOUSE
20 DEFINT X,Y
30 CLS
40 IF MOUSE(0)=0 THEN 40
50 X=MOUSE(1):Y=MOUSE(2)
60 PSET (X,Y)
70 GOTO 40
80 END
```

To move an object, use the GET and PUT commands. GET gets a rectangle of information or a picture from a specified area, then PUT places that rectangle back on the screen in a different place.

Use a DIMension statement to reserve an array large enough to keep track of the information in the rectangle you will be moving. GET is of the form

GET (x1,y1)-(x2,y2),A

where (x1,y1) are the coordinates of the upper left corner of the desired rectangle and (x2,y2) are the coordinates of the lower right corner. *A* is the array name given to this rectangle.

PUT specifies the coordinates of the upper left corner where you want to put the array A. The form is

PUT (x,y),A

Program 7-5 is an illustration of the use of GET and PUT. Line 20 dimensions the array A. Line 40 draws a box, and line 50 draws a circle in the box. Line 60 uses the coordinates of the box's outside dimensions and calls this rectangle the A array. Line 70 redraws the picture with PUT, starting at the upper left coordinates (100,100).

Program 7-5. GET and PUT

```
10 REM GETPUT
20 DIM A(100)
30 CLS
40 LINE (10,10)-(30,30),,BF
50 CIRCLE (20,20),8,0
60 GET (10,10)-(30,30),A
70 PUT (100,100),A
80 END
```

To move an object on the screen (such as an icon in your own program), we can use a combination of the MOUSE functions and the GET and PUT commands. Here's one way to do this (Program 7-6) . Lines 50–70 draw a simple picture on the screen in the upper left corner. Line 80 GETs the information and stores it in array A (dimensioned in line 30).

Line 100 checks to see whether the left mouse button has been pressed. If not, the program stays at line 100. When the button is pressed, the current x and y position is checked with MOUSE(1) and MOUSE(2). If it is different from the previous position, lines 130–150 redraw the picture with PUT, and x and y are reinitialized.

Program 7-6. Moving

```
10 REM MOVING
20 DEFINT A,X,Y
30 DIM A(1000)
40 CLS
50 LINE (0,0)-(50,50),,BF
60 CIRCLE (25,15),10,0
70 CIRCLE (25,25),15,0
80 GET (0,0)-(50,50),A
90 REM CHECK MOUSE
100 IF MOUSE(0)=0 THEN 100
110 IF ABS(X-MOUSE(1))>2 THEN 130
120 IF ABS(Y-MOUSE(2))<3 THEN 100
130 PUT(X,Y),A
140 X=MOUSE(1):Y=MOUSE(2)
150 PUT(X,Y),A
160 GOTO 100
170 END
```

If you want to move and not erase as you go, you can create some interesting graphics. To illustrate, we will draw a circle, then let you draw with that circle by moving the mouse and pressing the left button.

Program 7-7. Drawing2

```
10 REM DRAWING2
20 DEFINT A,X,Y:DIM A(300)
30 CLS
40 CIRCLE (8,8),8
50 GET (0,0)-(16,16),A
60 IF MOUSE(0)=0 THEN 60
70 X=MOUSE(1):Y=MOUSE(2)
80 PUT (X,Y),A
90 GOTO 60
100 END
```

Chapter 8

Graphics

Chapter 8

Graphics

Graphics on the Amiga can be a lot of fun. Many of the basic programs in this book can be enhanced by adding graphics. This book, however, is designed for beginning programmers and thus the graphics programming will be limited. COMPUTE! is publishing articles and other books which will go into more detail about the many graphics capabilities of the Amiga.

The standard output screen is 640 *pixels* (dots) wide and 200 pixels high, which gives you a total of 128,000 individual dots which can be used in your pictures.

Printing in Color

The simplest form of graphics involves printing text in different colors. This device can be used in regular, nongraphic programs to highlight error messages or other important prompts to the program user or to separate the program's messages from the user's input. To print in different colors on the standard screen, use the command

COLOR *f,b*

where *f* is the value of the foreground color and *b* is the background color value. The colors are numbered from 0 through 3. For example, this will print black letters on an orange background rather than the standard white on blue.

```
COLOR 2,3
PRINT "HELLO."
```

You can add interest and draw attention to certain lines by varying the COLOR command before your PRINT statements.

Drawing Lines

Many of the drawing commands use coordinates in parentheses, listed as (x,y) where *x* is the distance from the left of the screen going toward the right. The upper left corner of the screen is (0,0). The *y* coordinate is the distance down the screen. For example, (10,20) would be 10 pixels across and 20 pixels down.

The LINE command is the basic drawing command to go from one point to another. Program 8-1 illustrates several forms of the LINE command.

Line 20 is the basic LINE command to draw from the first point (10,10) to the second point (50,40) using coordinates (x,y) from the upper left corner of the screen. Line 30 uses the same type of command. Line 40 illustrates a LINE command that starts at the last point drawn and goes to the specified point. These two lines will create a drawing that will go from (20,50) to (60,75) and then to (70,60).

You'll notice that the previous lines are drawn in white, the default color. You may specify a color number right after the second coordinate if you prefer a different color. Line 50 draws a black line.

The LINE command has some added options. Line 60 illustrates the Box option, indicated by a *B* after the color number. This command will draw a box with the upper left corner at the first coordinate set (10,100) and the lower right corner at the second coordinates (30,120). It will be outlined in color number 3, which is orange.

In line 70, BF is the Box Filled option, or a box that is colored in. If you want just to use the default color, you don't need to specify the color number, but you do need to use the right number of commas. Line 80 draws a Box Filled with the default color white.

Program 8-1. Line

```
10 REM LINE
20 LINE (10,10)-(50,40)
30 LINE (20,50)-(60,75)
40 LINE -(70,60)
50 LINE (80,80)-(130,70),2
60 LINE (10,100)-(30,120),3,B
70 LINE (40,110)-(65,140),2,BF
80 LINE (80,105)-(100,130),,BF
90 END
```

With just the LINE graphics command you can create beautiful designs. Draw the lines in certain patterns or in a certain sequence. Program 8-2 draws lines using three nested FOR-NEXT loops.

Program 8-2. Lines

```
10 REM LINES
20 CLS
30 X1=320:Y1=0:X2=320:Y2=199
40 M=X1:N=X2
45 FOR J=1 TO 5
50 FOR C=0 TO 3
60 FOR I=1 TO 8
70 LINE (X1,Y1)-(X2,Y2),C
80 LINE (M,Y1)-(N,Y2),C
90 X1=X1-5:X2=X2+5
100 M=M+5:N=N-5
110 NEXT I
120 NEXT C
130 NEXT J
140 END
```

Next, in Program 8-3, we'll use the LINE command with the BF option to draw boxes. Two loops are used to draw the pattern. This time, IF-THEN statements are used instead of FOR-NEXT loops.

Program 8-3. Boxes

```
10 REM BOXES
20 C=1:X=0:Y=0:CLS
30 LINE(X,Y)-(X+50,Y+50),C,BF
40 C=C+1:IF C=4 THEN C=1
50 X=X+20:Y=Y+10
60 IF Y<=130 THEN 30
70 LINE(X,Y)-(X+50,Y+50),C,BF
80 C=C+1:IF C=4 THEN C=1
90 X=X+20:Y=Y-10
100 IF Y>0 THEN 70
110 END
```

A lot of interesting graphics effects are done by using the CIRCLE command. The basic form of the CIRCLE command is to specify a pair of coordinates which is the center of the circle, then a radius, then optionally the color of the circle.

Program 8-4 contains the CIRCLE command in lines 30 and 60. The center point is always the same—(320,100)—or about the middle of the screen. Concentric circles are drawn in a loop with the radius, R, increasing each time. Then the circles are drawn with the background color 0, which in effect erases the present circles. These circles are drawn with a decreasing radius. Line 90 changes the color of the drawing. Line 100 creates an endless loop, so to stop the program press CTRL-C to break or use the mouse to select Stop under the Run menu.

Program 8-4. Circles

```
10 REM CIRCLES
20 CLS:C=1
30 CIRCLE (320,100),R,C
40 R=R+4
50 IF R<100 THEN 30
60 CIRCLE (320,100),R,0
70 R=R-4
80 IF R>0 THEN 60
90 C=C+1:IF C=4 THEN C=1
100 GOTO 30
110 END
```

Just as there are several options in the LINE command, there are several in the CIRCLE command. The number right after the center coordinates is the radius. After the radius number is the color number. The next two numbers are a starting point and an ending point, so you may draw an arc, or part of a circle. The last number is the aspect, or the height/width ratio. The numbers after the radius are optional.

Here's a program (Program 8-5) which illustrates several kinds of CIRCLE commands. Line 20 draws a circle using color 2, black.

Line 30 draws a circle with color 3. This command specifies a starting point of 0 and an ending point of 3.14159 (which is approximately pi). Keep in mind that these numbers are expressed in radians. The starting point may be left out and would be assumed to be 0, which is at the three o'clock position on a round clock face. The direction is counterclockwise.

You do not have to start at 0, of course, and line 40 draws a circle starting at pi radians (at the left side of the circle) and going to 6.

The aspect number gives the height/width ratio to make the circle into an ellipse. If this number is not specified, the default value is 1, a circle. Line 50 draws an ellipse that has a ratio of 2. Note that if some of the optional numbers are left out, the commas are still necessary. Line 60 draws an ellipse with a fractional ratio, .33. Again, the starting and ending point are not specified, so the ellipse is complete.

Line 70 draws an arc of an ellipse by specifying the starting point of 1, the ending point of 3, and a height/width ratio of 1.5.

Program 8-5. Circles2

```
10 REM CIRCLES2
20 CIRCLE (30,50),20,2
30 CIRCLE (100,50),30,3,0,3.14159
40 CIRCLE (130,50),40,1,3.14159,6
50 CIRCLE (190,50),25,,,,2
60 CIRCLE (100,90),25,3,,,.33
70 CIRCLE (100,170),40,2,1,3,1.5
80 END
```

Painting on the Amiga

The PAINT command is used to fill in screen areas with color. (It must be used in a window that has been defined with a type of 16 through 31.) The command specifies the coordinates of a point where painting is to start, then a hue number. The next number, the border color at which to stop, is optional.

The following short program (Program 8-6) illustrates the PAINT command. Line 20 defines a WINDOW with the type of 24. Line 30 draws a circle; then the next line draws a line that goes through the circle. Line 40 paints starting at point (40,50), which is the lower part of the circle, and uses color number 3. The next line paints the upper part of the circle because it starts at the point (60,50). It paints with the color 2, black, and goes to the border of color 3, the orange circle outline.

Program 8-6. Paint

```
10 REM PAINT
20 WINDOW 2,"PAINTING",(100,10)-(200,100),24
30 CIRCLE (50,50),30,3
   LINE (20,20)-(100,100),3
40 PAINT (40,50),3
   PAINT (60,50),2,3
50 END
```

Graphic Patterns

When you draw a line or fill in an area with BF or AREAFILL, the default values are a solid line and a solid fill pattern. However, you can change both these patterns with the PATTERN statement. You can use graph paper to draw out a pattern of filled-in squares, then convert each row to its hexadecimal equivalent. The pattern numbers start with &H.

Here's a sample pattern using two defined pattern lines (Program 8-7). First, line 20 draws a box so that you will be able to see the pattern. Without a pattern specification it will be the solid color. Line 30 DIMensions the integer variable PAT% for two elements. Line 40 and the next line define the pattern for each of the PAT% elements. These will repeat in a filled area. Line 50 defines the pattern. The first number, &HFF, defines the pattern for a line. &HFFFF is a solid line, so this is the same as &HFF00 and will yield a dotted line. The second specification is the PAT% array for the filled areas.

Line 60 draws a line using the new pattern. Line 70 draws a box filled with the new pattern.

Program 8-7. Pattern

```
10 REM PATTERN
20 LINE (0,0)-(80,80),,BF
30 DIM PAT%(1)
40 PAT%(0)=&HAAAA
   PAT%(1)=&HFFFF
50 PATTERN &HFF,PAT%
60 LINE (0,90)-(90,90)
70 LINE (10,100)-(80,150),,BF
80 END
```

Try two different numbers in line 40, and a different pattern in line 50. You can see how using different patterns can make some beautiful effects. By the way, changing the line pattern by using PATTERN can affect the cursor when it is in the Output window. For example, the above pattern will make the cursor disappear because of the trailing zeros.

Your pattern in a filled area doesn't have to be just two alternating lines. This next program (Program 8-8) illustrates a pattern array P% that contains four elements to define even more intricate fill patterns.

The PATTERN command in line 40 first defines the line, then the fill pattern.

Program 8-8. Pattern2

```
10 REM PATTERN2
20 DIM P%(3)
30 P%(0)=&HFFFF
   P%(1)=&HAAAA
   P%(2)=&H5555
   P%(3)=&H3333
40 PATTERN &H3333,P%
50 LINE (10,10)-(80,80),,BF
60 LINE (0,90)-(90,90)
70 END
```

Now try changing the hexadecimal numbers in line 30 for the P% elements. For example,

```
&HFAFA
&HAFAF
&H5353
&H3535
```

Or another pattern would be

```
&H6666
&H1212
&H4444
&H7777
```

Area Fills

Because it takes up a lot of memory, the PAINT command is very limited with the 256K Amiga, but AREA and AREAFILL can be versatile enough to do the same thing. AREA commands specify points to be joined in a polygon, then AREAFILL joins those points and fills in the polygon with the default solid color or a specified pattern. You don't actually see anything on the screen after AREA statements until you use the AREAFILL command.

AREA commands can use actual numbers (or variables) specifying coordinates of points, or they can use the STEP option which gives relative distances.

Program 8-9 illustrates how AREA, AREAFILL, and PATTERN can work.

The three AREA statements in line 20 define points for a triangle, and line 30 fills in the triangle. Line 40 uses four AREA statements to define a quadrilateral. Line 50 uses AREAFILL 1 which will use the reverse. You can specify either 1 or 0, and the default is 0.

Line 60 DIMensions an array PAT% with four elements, then the four elements are defined with hexadecimal patterns. Line 70 redefines the fill patterns.

Line 80 starts with a specific AREA point; then STEP indicates go a relative distance from the last point. Line 90 fills in that polygon.

Line 100 illustrates how the PATTERN command changed the line, and line 110 illustrates the pattern in a filled box. Line 120 defines another triangle, and line 130 fills with the reverse.

Program 8-9. Areas

```
10 REM AREAS
20 AREA (25,10)
   AREA (50,20)
   AREA (0,20)
30 AREAFILL
40 AREA (50,25)
   AREA STEP (20,15)
   AREA STEP (-10,15)
   AREA STEP (-10,-10)
50 AREAFILL 1
60 DIM PAT%(3)
   PAT%(0)=&H505
   PAT%(1)=&HA0A
   PAT%(2)=&H505
   PAT%(3)=&HA0A
70 PATTERN &HFFF,PAT%
80 AREA (100,50)
   AREA STEP (50,20)
   AREA STEP (-50,20)
   AREA STEP (-50,-20)
   AREA STEP (50,-20)
90 AREAFILL
100 LINE (0,120)-(200,120)
110 LINE (10,130)-(150,170),3,BF
120 AREA (180,130)
    AREA STEP (30,20)
    AREA STEP (-50,20)
130 AREAFILL 1
140 END
```

Adding Colors to Your Palette

What about all the colors the Amiga is supposed to have? So far I have simplified the programs by just using the four default colors for the standard screen: 0, the background blue; 1, white; 2, black; and 3, orange. These colors can be changed by using Preferences on the Workbench, or you can use the PALETTE command.

Think of the PALETTE command as an artist's palette on which you mix colors. For each of the four possible colors, or paint buckets, you can mix a combination of red, green, and blue. The first number in the PALETTE command is one of the paint buckets. The next three numbers are the mixtures in order of red, green, and blue. The numbers can be thought of as

fractions from 0 through 1. Black is 0,0,0, or no colors, and white is 1,1,1, or a mixture of all colors.

Program 8-10 illustrates the PALETTE command by drawing four boxes of the four colors. PALETTE 0, or the background color, is a mixture of 1,1,1 which is white. PALETTE 1, the default drawing and printing color, is a mixture of 0,0,1. PALETTE 2 is 0,1,0. PALETTE 3 is 1,0,0.

Program 8-10. Palettes

```
10 REM PALETTES
20 PALETTE 0,1,1,1
   PALETTE 1,0,0,1
   PALETTE 2,0,1,0
   PALETTE 3,1,0,0
40 FOR C=0 TO 3
     LINE (C*20,80)-(C*20+20,120),C,BF
   NEXT C
50 END
```

Now try some fractional mixtures in the above program:

```
PALETTE 0,0,0,0

        1,.2,.4,.6

        2,.1,.5,.4

        3,.2,.2,.2
```

or

```
PALETTE 0,.1,.8,.8

        1,.3,.2,.4

        2,.4,.1,.6

        3,.8,.5,0
```

You can see that you could spend all day experimenting with colors.

The next program (Program 8-11) can help you experiment with colors. A box of colors appears at the left of the screen. There are three "tubes" for the three colors at the right. A circle appears above one of the paints. Use the arrow keys to move the level of color up or down. Press RETURN to move to the next tube of color. As you adjust the levels with the arrow keys, the numbers above the columns are the numbers to be used in the PALETTE statement to produce the color in the square. The colors start at 0,0,0.

Line 20 is a FOR-NEXT loop that draws the three boxes for the level indicators. Line 30 draws the large box of color. Line 40 defines the y coordinate for drawing the circle indicating which tube you can change the level on. Line 50 is the number of the tube T.

Line 60 defines XX and LL for the x position and level for the particular tube. The circle is drawn above the tube that can be adjusted.

Line 70 detects which key on the keyboard is pressed—the RETURN key, the up arrow key, or the down arrow key. All other keys are ignored. DL is the change in level, which can be −1 or +1. If the down arrow key is pressed, the line is erased, but if you are moving up, the lines stay drawn.

Line 80 defines LL for the changed level and checks the top and bottom positions. PALETTE changes the color in the box. The new values for the mixtures are printed.

Line 90 is the procedure when the RETURN key is pressed. The circle is erased, the tube number is incremented, and the next tube is available for input.

To stop this program you will need to press CTRL-C or Stop from the menu bar.

Program 8-11. Palettes2

```
10 REM PALETTES2
20 FOR C=1 TO 3
   CC=C*110+130
   LINE (CC,79)-(CC+10,181),,B
   LINE (CC,180)-(CC+10,180)
   X(C)=CC+5:L(C)=180
   NEXT C
30 LINE (40,10)-(120,60),2,BF
40 Y=70
50 T=1
60 XX=X(T):LL=L(T)
   CIRCLE (XX,Y),5
70 A$=INKEY$:IF A$="" THEN 70
   IF A$=CHR$(13) THEN 90
   IF A$=CHR$(28) THEN DL=-1:GOTO 80
   IF A$<>CHR$(29) THEN 70
   DL=1
   LINE (XX-4,LL)-(XX+4,LL),0
80 LL=LL+DL
   IF LL<80 THEN LL=80
   IF LL>180 THEN LL=180
   LINE (XX-4,LL)-(XX+4,LL),1
   P(T)=(180-LL)/100:L(T)=LL
   PALETTE 2,P(1),P(2),P(3)
   LOCATE 5,1:PRINT TAB(18+T*14);"   "
   LOCATE 5,30:PRINT P(1),P(2),P(3)
   GOTO 70
```

```
90 CIRCLE (XX,Y),5,0
   T=T+1:IF T>3 THEN 50 ELSE 60
100 END
```

Sprites and Bobs

A whole book can be written about programming moving objects—sprites and blitter objects (bobs). I'll just get you started here. Most of the commands start with OBJECT, and you can just sit at the computer and start experimenting.

To define a shape, use the Object Editor program that comes with the demonstration programs on the same disk as Amiga BASIC. If you are in BASIC, you can load this program with the command

LOAD "BasicDemos/ObjEdit"

For an example, press 1 to design a sprite. Now use the right mouse button to see the menus on the menu bar. Use the left mouse button to change the size of the object or to choose a color, then to draw with the pen (or use the different shapes). When you have designed an object, press the right mouse button to go to the Project menu and select Save. You will be asked for a title for your work. Remember, if you are saving on a different disk, to use "DF0:". I designed a snake and called it "DF0:SNAKE".

When you are finished designing objects, you can use NEW to get rid of the ObjEdit program and start your own program. The example below (Program 8-12) illustrates how to set up one sprite and shows how some of the commands are used.

```
OPEN "DF0:SNAKE" FOR INPUT AS 1
```

is used to OPEN the file containing the information about the object designed using the Object Editor program. The next line reads the information as a string with INPUT$(LOF(1),1). OBJECT.SHAPE 1 says to define shape number 1 with that previously saved string.

CLOSE 1 closes the file that we will no longer need.

OBJECT.X and OBJECT.Y define where the object will start on the screen. The object number 1 is specified, along with the x coordinate and y coordinate. I defined SX and SY to be the speed in the x direction and the speed in the y direction. OBJECT.VX and OBJECT.VY specify those speeds (velocities) for object number 1.

OBJECT ON makes our sprite visible. Without specifying a number, all objects would become visible. OBJECT.START starts the object in motion. Again, a number can be specified, and no number means all objects.

Line 40 tests to see whether the sprite collided with the border line. If K is 0, there is no collision and the sprite can keep moving. If K is −1 or −3, then the top or bottom border was hit and the y velocity needs to be changed; otherwise, the side borders were hit and the x velocity needs to be reversed. GOTO 30 continues the program until you choose Stop on the menu bar or press CTRL-C.

Program 8-12. Sprite

```
10 REM SPRITE
   OPEN "DF0:SNAKE" FOR INPUT AS 1
   OBJECT.SHAPE 1,INPUT$(LOF(1),1)
   CLOSE 1
   OBJECT.X 1,20
   OBJECT.Y 1,50
   SX=60:SY=50
   OBJECT.VX 1,SX
   OBJECT.VY 1,SY
   OBJECT.ON
30 OBJECT.START

40 K=COLLISION(1)
   IF K=0 THEN 40

   IF K=-1 OR K=-3 THEN SY=OBJECT.VY(1):OBJECT.VY 1,-SY:GOTO
 30
   SX=OBJECT.VX(1)
   OBJECT.VX 1,-SX
   GOTO 30
END
```

Use of the object commands is an advanced topic and beyond the scope of this book. See *COMPUTE!'s Advanced Amiga BASIC, Inside Amiga Graphics,* or *COMPUTE!'s Amiga Programmer's Guide* for more detailed information. Briefly, here are some of the other commands you might encounter:

OBJECT.AX and **OBJECT.AY** are accelerations of the object in the x and y directions.

OBJECT.CLIP (x1,y1)-(x2,y2) defines a rectangle, and objects cannot be drawn outside the area. The default value is the border of the current Output window.

OBJECT.CLOSE *id* is like closing a file—you use this command when you no longer need an object.

OBJECT.OFF makes an object invisible and stops an object if it was started with OBJECT.START. The object is still available if you use .OFF.

OBJECT.STOP freezes the motion of an object (it will still be visible).

OBJECT.HIT determines collision objects.

OBJECT.PLANES is used with blitter objects to set the bob's planePICK and place-on-off masks.

OBJECT.PRIORITY is used to set priority of bobs, which determines when an object is drawn in relation to other objects—or whether objects are in front of or behind other objects.

Chapter 9

Music, Sounds, and Speech

Chapter 9

Music, Sounds, and Speech

The basic command to produce a musical tone on the Amiga is

SOUND *f,d,v,c*

where *f* is a frequency (pitch), *d* is duration, *v* is volume, and *c* is the audio channel from 0 through 3.

The frequency is a number for the standard cycles per second for a tone, such as 440 for an A note. The Amiga BASIC manual has a chart, or you may want to make your own using musical staff paper.

The duration is a number for the length of time you want a tone to play.

The volume may be a number from 0 through 255, where 255 is the loudest. If you leave the volume parameter out of the statement, the default value is 127.

When you're using SOUND statements to program the computer to play a tune, it's a good idea to use a variable for the duration. For example, let T represent the length of a quarter note. T/2 would be an eighth note; T*2, a half note; and T*4, a whole note. Try this short tune:

Program 9-1. Music1

```
10 REM MUSIC1
20 T=10
30 SOUND 330,T
40 SOUND 294,T/2
50 SOUND 262,T/2
60 SOUND 294,T
70 SOUND 330,T*1.5
80 SOUND 349,T/2
90 SOUND 392,4*T
100 END
```

Now suppose you want to play the tune twice as fast. With a variable duration you need to change only line 20, not all of the SOUND statements. Try T=5.

To make the tune slower, try T=20. The notes stay in the right proportion. Line 30 is a quarter note, for example. Lines 40, 50, and 80 are eighth notes. Line 70 is a dotted quarter note. Line 90 represents a whole note.

The frequencies can also be variables specified at the beginning of the program. The note names can be used as the variables to make your program conform somewhat to regular musical notation. Program 9-1 can be written like this:

Program 9-2. Music2

```
10 REM MUSIC2
20 T=10:C=262:D=294
25 E=330:F=349:G=392
30 SOUND E,T
40 SOUND D,T/2
50 SOUND C,T/2
60 SOUND D,T
70 SOUND E,T*1.5
80 SOUND F,T/2
90 SOUND G,4*T
100 END
```

DATA statements can shorten the program if you are using lots of SOUND statements. Here's the same tune using DATA:

Program 9-3. Music3

```
10 REM MUSIC3
20 T=1
30 FOR N=1 TO 7
40  READ F,D
50 SOUND F,D*T
60 NEXT N
```

```
70 DATA 330,1,294,.5,262,.5,294,1,330,1.5,349,.5,392,4
80 END
```

If you read music, you can use this method to translate sheet music to the computer.

Switching Channels

If you want more than one tone at a time, as a chord, you can use the four channels of sound.

Here's a short tune (Program 9-4) that uses the four channels. The music information is contained in DATA statements, in the order duration factor D and then four frequency numbers to go with the four channels.

Line 20 sets a time of 4. When a duration factor D is read in, it is multiplied by T for the total duration. Line 30 reads the value for D. If D is 0, it indicates the end of the data and the program branches to the end.

Lines 40–70 are a FOR-NEXT loop that reads the four frequencies. For each frequency, the SOUND statement starts the music. F is the frequency, and the counter S is also used in the SOUND statement. S is first used to set the volume—the upper notes are played louder than lower notes. S is also used for the channel number.

Line 80 transfers control back to the READ statement for the next set of numbers. This music is from the Rondo section of Beethoven's Fifth Concerto.

Program 9-4. Rondo

```
10 REM RONDO
20 T=4
30 READ D:IF D=0 THEN 1000
40 FOR S=0 TO 3
50 READ F
60 SOUND F,D*T,150-S*20,S
70 NEXT S
80 GOTO 20
90 DATA 2,466,392,311,156
100 DATA 2,622,392,311,233
110 DATA 2,622,466,392,196
120 DATA 2,784,466,392,196
130 DATA 2,20,20,20,233
140 DATA 1,784,622,466,196
150 DATA 1,932,20,20,20
160 DATA 1,932,784,622,156
170 DATA 1,1244,20,20,20
180 DATA 2,1244,932,784,233
190 DATA 2,1244,932,784,196
```

```
200 DATA 2,1244,932,784,156
210 DATA 2,20,20,20,233
220 DATA 1,1244,932,784,156
230 DATA 1,1568,932,784,156
240 DATA 2,1396,932,698,349
250 DATA 2,20,20,20,233
260 DATA 1,1108,932,698,208
270 DATA 1,1396,932,698,208
280 DATA 2,1244,932,784,156
290 DATA 2,1244,932,784,233
300 DATA 1,784,622,196,156
310 DATA 1,932,622,196,156
320 DATA 2,932,698,587,117
330 DATA 2,932,698,587,233
340 DATA 1,932,698,587,175
350 DATA 1,880,698,587,175
360 DATA 4,932,698,587,117
370 DATA 0
1000 END
```

The Amiga's musical abilities can be used in a wide variety of ways—for fun, for learning basic skills, and for sharpening your musicianship. You can put music into the computer and then sing with it. Or you can play a solo instrument with the computer playing the accompaniment. The Amiga is especially good for learning music, because you can immediately hear any changes you want to implement as you are composing.

Or, if you are trying to learn a piece that has a difficult rhythm, program it on your Amiga. Play it first at a slow tempo, then gradually increase the tempo as you practice along with the computer.

In addition, musical tones work well in interactive programs. You can use a sound for a prompt or a happy musical interlude for correct responses.

With some experimentation you can make many different sounds with the computer. The best way to learn to program the music is to sit at the computer and experiment.

"Notes" is an educational program that illustrates the use of the SOUND command. This program is designed for the beginning music student. The first option, Keyboard, shows the letter names of the keys on a piano or organ keyboard and then presents a drill of ten keys chosen at random. A question mark appears under a key, and the student must press the correct letter name. When the correct letter is pressed, the name of the key appears and that tone is played.

The second and third options are Treble Clef and Bass Clef. These two sections display the appropriate staff and clef, and present words and phrases to help the student remember

the letter names of the notes. A drill of ten notes is then presented.

An array S$ is used to keep track of the letter names of the notes, and S is the array to save the frequencies of the corresponding notes. In the Keyboard section, notes from 0 through 18 are used. In the Treble Clef and Bass Clef sections, notes from 1 through 9 are used.

Program 9-5. Notes

```
10 REM NOTES
20 DIM S$(18),S(18)
50 CLS
60 LOCATE 5,10:PRINT "LET'S LEARN NOTES"
70 PRINT:PRINT "CHOOSE:":PRINT
80 PRINT "1  KEYBOARD"
90 PRINT "2  TREBLE CLEF"
100 PRINT "3  BASS CLEF"
110 PRINT "4  END PROGRAM"
120 E$=INKEY$:IF E$="" THEN 120
130 IF E$<"1" OR E$>"4" THEN 120
140 CLS:RANDOMIZE TIMER
145 ON VAL(E$) GOTO 750,1280,1880,2120
150 REM KEYBOARD
160 CLS:LINE(0,0)-(640,70),1,BF
170 FOR I=24 TO 640 STEP 32
180 LINE(I,0)-(I,70),2
190 NEXT I
200 RESTORE 210
210 DATA 48,80,144,176,208,272,304
215 DATA 368,400,432,496,528,596
220 FOR I=1 TO 13
230 READ A
240 LINE(A,0)-(A+16,48),2,BF
250 NEXT I
260 RETURN
270 LOCATE 23,1:PRINT "PRESS <RETURN> TO CONTINUE.";
290 E$=INKEY$:IF E$="" THEN 290
300 IF ASC(E$)<>13 THEN 290
310 CLS:RETURN
320 REM CDE
330 LINE(192,36)-(336,116),1,BF
340 LINE(240,96)-(240,116),2
350 LINE(288,96)-(288,116),2
360 LINE(228,36)-(252,96),2,BF
370 LINE(276,36)-(300,96),2,BF
380 LOCATE 16,28:PRINT "C"
390 RETURN
400 REM STAFF
410 FOR I=43 TO 107 STEP 16
420 LINE (0,I)-(640,I),1
430 NEXT I
440 RETURN
450 REM TREBLE CLEF
460 LOCATE 2,18:PRINT "TREBLE CLEF NOTES"
470 LINE (76,118)-(84,122)
```

```
480 LINE -(86,116):LINE -(70,36)
490 LINE -(76,28):LINE -(84,26)
500 LINE -(90,30):LINE -(88,40)
510 LINE -(72,54):LINE -(40,75)
520 LINE -(34,84):LINE -(38,95)
530 LINE -(48,102):LINE -(64,106)
540 LINE -(100,106):LINE -(116,100)
550 LINE -(124,94):LINE -(122,86)
560 LINE -(116,80):LINE -(104,76)
570 LINE -(80,76):LINE -(64,82)
580 LINE -(60,88):LINE -(64,98)
590 RETURN
600 REM BASS CLEF
610 LOCATE 2,20:PRINT "BASS CLEF NOTES"
620 LINE (50,56)-(62,62),,BF
630 PRESET (64,62):PRESET (62,56):PRESET (50,62)
640 LINE (52,53)-(58,55),,BF
650 LINE (54,52)-(68,47)
660 LINE (58,52)-(68,47)
670 LINE -(88,44):LINE -(104,44)
680 LINE -(126,50):LINE -(136,59)
690 LINE -(136,72):LINE -(126,82)
700 LINE -(102,95):LINE -(82,101)
710 LINE (160,52)-(164,55),,BF
720 LINE (160,64)-(164,67),,BF
730 RETURN
740 REM KEYBOARD
750 REM
760 GOSUB 160
770 LOCATE 10,2:PRINT "B   C   D   E   F   G   A   B   C   D
   E   F   G   A   B   C   D   E   F   G"
780 PRINT:PRINT "  The musical keyboard has groups of 3 blac
k keys alternating"
790 PRINT "  with groups of 2 black keys."
800 PRINT:PRINT "  Each white key has a letter name."
810 PRINT "  We use the letters  A  B  C  D  E  F  G."
820 RESTORE 840
830 FOR I=0 TO 18:READ S$(I),S(I):NEXT I
840 DATA B,247,C,262,D,293,E,330,F,349,G,393,A,440,B,494
850 DATA C,523,D,587,E,659,F,698,G,783,A,880,B,988,C,1047
860 DATA D,1175,E,1319,F,1397
870 GOSUB 270
880 PRINT "One of the easiest keys to find is 'C'."
890 GOSUB 330
900 GOSUB 270
910 GOSUB 160
920 LOCATE 10,6:PRINT "C";TAB(34);"C";TAB(62);"C"
930 GOSUB 270
940 PRINT "'D' is between the two black keys and 'E' is on t
he right."
960 GOSUB 330
970 LOCATE 16,34:PRINT "D    E"
980 GOSUB 270
990 GOSUB 160
1000 LOCATE 10,2:PRINT "B   C   D   E   F   G   A   B   C
D   E   F   G   A   B   C   D   E   F   G"
1010 PRINT:PRINT "  Notice the letter names repeat."
1020 PRINT:PRINT:PRINT "     NOW FOR A QUIZ ..."
1030 GOSUB 270
1040 GOSUB 160
```

```
105Ø LOCATE 15,1Ø:PRINT "NAME THE NOTE"
1Ø6Ø PK=-1
1Ø7Ø FOR T=1 TO 1Ø
1Ø8Ø N=INT(19*RND):IF N=PK THEN 1Ø8Ø
1Ø9Ø PK=N:C=4*N+1
111Ø LOCATE 1Ø,C:PRINT "?"
112Ø LOCATE 1Ø,C:PRINT " "
113Ø A$=INKEY$:IF A$="" THEN 111Ø
114Ø IF A$=S$(N) THEN 117Ø
115Ø SOUND 25Ø,4:SOUND 2ØØ,4
116Ø GOTO 111Ø
117Ø SOUND S(N),4Ø
118Ø LOCATE 1Ø,C:PRINT S$(N)
119Ø FOR DELAY=1 TO 5ØØØ:NEXT DELAY
12ØØ LOCATE 1Ø,C:PRINT " "
121Ø NEXT T
122Ø LOCATE 15,1Ø:PRINT "G O O D   W O R K !"
123Ø FOR I=1 TO 2Ø
124Ø SOUND 1ØØØ*RND+523,2
125Ø NEXT I
126Ø GOTO 5Ø
127Ø REM TREBLE CLEF
128Ø REM
129Ø GOSUB 41Ø:GOSUB 46Ø
13ØØ RESTORE 131Ø
131Ø DATA F,698,E,659,D,587,C,523,B,494,A,44Ø,G,392,F,349,E,
33Ø
132Ø FOR I=1 TO 9:READ S$(I),S(I):NEXT I
133Ø LOCATE 18,2:PRINT "The names of the notes on the spaces
 spell the word FACE."
135Ø SOUND 349,6:LOCATE 13,28:PRINT "F"
136Ø SOUND 44Ø,6:LOCATE 11,32:PRINT "A"
137Ø SOUND 523,6:LOCATE 9,36:PRINT "C"
138Ø SOUND 659,6:LOCATE 7,4Ø:PRINT "E"
139Ø GOSUB 27Ø
14ØØ GOSUB 41Ø:GOSUB 46Ø
141Ø LOCATE 17,2:PRINT "Learn this phrase to help you rememb
er the notes on lines."
142Ø PRINT:PRINT " The first letter of each word is the lett
er name of the note."
143Ø PRINT:PRINT TAB(24);"EVERY GOOD BOY DOES FINE."
144Ø SOUND 33Ø,6:LOCATE 14,24:PRINT "Every"
145Ø SOUND 392,6:LOCATE 12,36:PRINT "Good"
146Ø SOUND 494,6:LOCATE 1Ø,46:PRINT "Boy"
147Ø SOUND 587,6:LOCATE 8,54:PRINT "Does"
148Ø SOUND 698,6:LOCATE 6,64:PRINT "Fine"
149Ø GOSUB 27Ø
15ØØ GOSUB 41Ø:GOSUB 46Ø
151Ø LOCATE 18,2Ø:PRINT "NAME THE NOTE"
152Ø PN=Ø
153Ø FOR T=1 TO 1Ø
154Ø N=INT(9*RND)+1:IF N=PN THEN 154Ø
155Ø PN=N:C=5+N:D=C*8-11
156Ø I=26Ø:J=28Ø
157Ø FOR R=D TO D+4
158Ø LINE (I,R)-(J,R),3
159Ø I=I-2:J=J+2:NEXT R
16ØØ LINE (I,D+5)-(J,D+8),3,BF
161Ø FOR R=D+9 TO D+13
162Ø I=I+2:J=J-2
163Ø LINE (I,R)-(J,R),3
```

```
1640 NEXT R
1660 A$=INKEY$:IF A$="" THEN 1660
1670 IF A$=S$(N) THEN 1700
1680 SOUND 250,5:SOUND 200,5
1690 GOTO 1660
1700 SOUND S(N),40,255
1710 LOCATE C,44:PRINT S$(N)
1720 FOR DELAY=1 TO 5000:NEXT DELAY
1730 LINE (250,D)-(360,D+13),0,BF
1740 FOR I=43 TO 107 STEP 16
1750 LINE (250,I)-(360,I)
1760 NEXT I
1770 NEXT T
1780 LOCATE 18,20:PRINT "G R E A T  ! !"
1790 FOR T=1 TO 20
1800 SOUND 1000*RND+523,2
1810 NEXT T
1820 PRINT:PRINT "TRY AGAIN? (Y/N)"
1830 E$=INKEY$:IF E$="N" THEN 50
1840 IF E$<>"Y" THEN 1830
1850 LOCATE 20,1:PRINT "                    "
1860 GOTO 1510
1870 REM BASS CLEF
1880 REM
1890 GOSUB 410:GOSUB 610
1900 LOCATE 16,2:PRINT "Phrases to learn the bass clef notes
 refer to animals."
1910 PRINT " For the notes on spaces, remember:"
1920 PRINT:PRINT TAB(28);"ALL COWS EAT GRASS."
1930 SOUND 110,6,255:LOCATE 13,28:PRINT "All"
1940 SOUND 131,6,255:LOCATE 11,38:PRINT "Cows"
1950 SOUND 165,6,255:LOCATE 9,50:PRINT "Eat"
1960 SOUND 196,6,255:LOCATE 7,60:PRINT "Grass"
1970 GOSUB 270
1980 GOSUB 410:GOSUB 610
1990 LOCATE 17,1:PRINT "For the line notes use this phrase:"
2000 PRINT:PRINT TAB(24);"GREAT BIG DOGS FIGHT ANIMALS."
2010 SOUND 98,6,255:LOCATE 14,24:PRINT "Great"
2020 SOUND 123,6,255:LOCATE 12,36:PRINT "Big"
2030 SOUND 147,6,255:LOCATE 10,44:PRINT "Dogs"
2040 SOUND 175,6,255:LOCATE 8,54:PRINT "Fight"
2050 SOUND 220,6,255:LOCATE 6,66:PRINT "Animals"
2060 RESTORE 2070
2070 DATA A,220,G,196,F,175,E,165,D,147,C,131,B,123,A,110,G,
98
2080 FOR I=1 TO 9:READ S$(I),S(I):NEXT I
2090 GOSUB 270
2100 GOSUB 410:GOSUB 610
2110 GOTO 1510
2120 END
```

Creating Different Sounds

By varying the WAVE value of a voice, you can produce many different sounds on the Amiga. The default WAVE is a sine wave, a pure tone. To make the tone sound different, you can change the WAVE pattern. For example, for a noise the pattern is erratic. Then, of course, you can change the WAVE for each

of the four voices and combine those for all sorts of variations in sound.

The WAVE command uses an integer array of 256 numbers (elements 0–255). You will need a DIMension statement near the beginning of the program to reserve space. For example, if we call our array W, we can use

```
20 DEFINT W
30 DIM W(255)
```

where DEFINT defines W as an integer.

Now you can put different numbers in the W array. For example,

```
FOR C=0 TO 255
W(C)=1
NEXT C
```

Or you can make a calculation, perhaps a trigonometric function of C, for each value of W(C). Or you can read the values from DATA:

```
FOR C=0 TO 255
READ W(C)
NEXT C
```

Now use the WAVE command to set the waveform for a particular channel:

```
WAVE 0,W
WAVE 1,W
```

You could set each channel to a different array.

To give you an idea of how different numbers in the array affect the sound, try the next program. I have used the tone of concert A (frequency 440). First the tone is sounded with the default waveform:

```
WAVE 0,SIN
SOUND 440,60
```

A graph of the sine wave is drawn on the screen. The numbers in the array can be from −127 to +127, so the borders of the screen will be the boundaries. Use the mouse to

move to a particular point and press the left mouse button to change that particular array element to a different number. You can hold the left mouse button down as you slowly move in the x direction if you want each element to change gradually. The new element number values are graphed, and 440 is played.

If you want to know all the values for the elements so that you can put them in DATA statements, get the tone you want and then press RETURN. Then click the left mouse button once. The values are printed on the screen.

In the program, line 20 defines all variables starting with W, T, J, and K to be integers. Line 30 DIMensions the array W for the 256 elements. Line 40 is the default WAVE, and line 50 sounds 440 so that you can hear the default sound on channel 0.

Lines 60–130 draw the graph of the sine wave. Line 140 changes the waveform to the new array W, which at first is a scaled sine wave where the numbers are whole numbers between −127 and +127. Line 150 sounds the frequency of 440. The next two lines check to see whether the RETURN key was pressed.

Line 160 checks to see whether the left mouse button is pressed. Line 170 ignores any mouse activity beyond the right side of the graph. Any up and down movement will affect the last point of the graph. Line 180 erases the original graph line, and lines 190–200 draw the new graph line where the mouse button is pressed. Line 210 changes the particular W element to the new value, then line 220 branches back to change the W array in the WAVE statement and play the tone with the new sound.

Lines 300–310 print the values of all the W elements for use in DATA statements.

Program 9-6. Sound Wave

```
10 REM SOUND WAVE
20 DEFINT W,T,J,K
30 DIM W(255)
40 WAVE 0,SIN
50 SOUND 440,60
60 LINE (0,100)-(620,100)
70 PI=3.14159
80 F=2*PI/510
90 FOR X=0 TO 510 STEP 2
100 Y=SIN(F*X)
110 YY=100-40*Y
    W(X/2)=127-YY*127/100
120 LINE (X,YY)-(X,100)
```

```
130 NEXT X
140 WAVE 0,W
150 SOUND 440,2
    R$=INKEY$
    IF R$=CHR$(13) THEN 300
160 IF MOUSE(0)=0 THEN 160
170 J=MOUSE(1):IF J>510 THEN J=510
180 LINE(J,0)-(J,200),0
190 K=MOUSE(2)
200 LINE (J,K)-(J,100)
210 W(J/2)=127-K*127/100
220 GOTO 140
300 WIDTH 77
    LOCATE 20,1
310 FOR T=0 TO 255
      PRINT W(T);
    NEXT T
END
```

Making the Amiga Speak

The Amiga has impressive speech capabilities built in. All you need is the Workbench disk.

Just as we set up a wave array for the waveform to hear a certain kind of music, we can set up an array for speech. There are default values which you can use until you're ready to define your own. The voice characteristics we can define in the array are pitch, inflection, rate, voice, tuning, volume, channel, mode, and control.

Pitch is expressed in hertz and is a number between 65 and 320. The default is 110, which is a male speaking voice.

The **inflection** can be 0 for using inflections and emphasizing syllables or 1 for a robot-like monotone.

The **rate** is a number betwen 40 and 400 words per minute; the default is 150.

The **voice** is 0 for male and 1 for female.

The **tuning** number is the sampling frequency in hertz (Hz). The number can range from a low of 5000 to a high, squeaky 28000. The default is 22200.

Volume is a number from 0 for no sound to 64 as the loudest.

The **channel** is the combination of channels and is a value from 0 through 11.

The **mode** can be 0 for synchronous speech output or 1 for asynchronous speech output.

Control is used when the mode is 1. It can be 0 for saying one statement and then the next, 1 for canceling the previous statement, or 2 for immediately interrupting the first statement and executing the second one.

The easiest way to define the array (which must be integer) is to use data:

```
FOR T=Ø TO 8

READ S%(T)

NEXT T

DATA 11Ø,Ø,25Ø,Ø,22200,64,1,Ø,Ø
```

Once the conditions of speech are set up, you type something for the computer to say. One speech command is SAY TRANSLATE$. You can use a string of regular English words and the computer will translate it to sounds and speak them. Common pronunciations are used, so you may need to change some spellings to fit the computer's pronunciation. Program 9-7 is a short example. (Another example came with the Basic-Demos on the Amiga BASIC disk).

In the SAY TRANSLATE$ command, you can use a string in quotation marks, as in line 20. You can also set up a string variable and use that variable name, as in line 30. S% is the array of the speech conditions.

Remember to insert the Workbench disk to run speech programs.

Program 9-7. Speech

```
REM--SPEECH

1Ø FOR T=Ø TO 5
    READ S%(T)
   NEXT T
2Ø SAY TRANSLATE$("HELLO"),S%
3Ø M$="THIS IS A TEST."
   SAY TRANSLATE$(M$),S%

  DATA 1Ø5,Ø,144,Ø,2Ø59Ø,63
```

Another way to get the computer to speak is to use the SAY command using *phonemes,* which are unique letter combinations representing specific sounds. A description of phonemes is given in the Amiga BASIC manual. You must use only the phonemes on the chart or you will get an error message.

Our next example, Program 9-8, illustrates how you can make the Amiga speak different languages by spelling phonetically with the phoneme method.

Line 10 sets up the speech conditions array S%. Line 20 defines the five different messages. Line 30 prints the menu screen, and line 40 lets the user choose a language or end the program.

Program 9-8. Language

```
REM--LANGUAGE
10 FOR T=0 TO 5
     READ S%(T)
   NEXT T
   DATA 105,0,144,0,20590,63
20 TEXT$(1)="AHN DUH TWAA KAETR SEYNK SIYS SEHT WIYT NAHF DI
YS"
   TEXT$(2)="UWNOH DOHS TREYS KWAATROH SIYN KOH SEYS SIY EH
TEH OHCHOH NUWEHVEH DLIYEHS"
   TEXT$(3)="AYNS TZWAY DRAY FIYR FUWNF SEHKS ZIYBAXN AAKT N
OYN TSEYN"
   TEXT$(4)="IYCHIY NIY SAAN SHIY GOH /HIYCHIY /HAACHIY /HRO
HKUW KUW JUW"
   TEXT$(5)="WAHN TUW THRIY FOHR FAYV SIHKS SEHVAXN EYT NAYN
 TEHN"

30 PRINT "LANGUAGE"
   PRINT:PRINT
   PRINT "CHOOSE:"
   PRINT "  1  FRENCH"
   PRINT "  2  SPANISH"
   PRINT "  3  GERMAN"
   PRINT "  4  JAPANESE"
   PRINT "  5  ENGLISH"
   PRINT "  6  END PROGRAM"

40 A$=INKEY$
   IF A$<"1" OR A$>"6" THEN 40
   IF A$="6" THEN 50
   SAY TEXT$(VAL(A$)),S%
   GOTO 40

50 CLS:END
```

Chapter 10

Built-In Functions

Chapter 10

Built-In Functions

Commands like RUN, GOSUB, and PRINT start a BASIC statement and control what happens in that line. Functions, on the other hand, are like small subroutines within a statement. They return a value or a string. A statement which uses a function must also contain a command that tells what to do with that function.

String Functions

Usually, the computer treats information as numeric. Certain information, however, is treated as strings, or groups of characters that can be letters, numbers, or symbols. A string is contained in double quotation marks, it can be up to 255 characters, and a string variable name must end with a dollar sign, such as A$.

Strings in Amiga BASIC are combined, or concatenated, with the plus sign:

```
PRINT A$+B$

NAME$=FIRST$+" SMITH"
```

You cannot combine string and numeric expressions. If you do have numbers and want to use them as strings, use the STR$ function to change the numeric value to a string:

```
N$=STR$(N)

PHONE$"586-"+STR$(NUMBER)
```

If you have a string that contains a number character, you can convert it to a numeric value (for example, for calculations) with VAL:

```
A=VAL(A$)

X=X-VAL(Z$)
```

All characters handled by a computer have a unique numeric value called the ASCII value (American Standard Code for Information Interchange). ASC(X$) is a function which returns the ASCII value of the first character in a string. If the

string expression to be converted is a constant, it must be contained in quotation marks:

```
PRINT ASC("*")

PRINT ASC(C$)

A=ASC(A$)
```

This program returns the ASCII value of any character you press on the keyboard:

```
10 REM ASCII CODES

20 E$=INKEY$

30 IF E$="" THEN 20

40 PRINT E$;ASC(E$)

50 GOTO 20

60 END
```

Press CTRL-C, Break, to end the program.

CHR$ can be considered the inverse of ASC. CHR$(*x*) returns the character represented by the ASCII value *x*. Try these commands:

```
PRINT CHR$(48)

PRINT CHR$(65)
```

This program illustrates the CHR$ function by printing the characters corresponding to the ASCII values 54–67.

```
10 REM CHR$

20 FOR C=54 TO 67

30 PRINT C;CHR$(C)

40 NEXT C

50 END
```

Here is a program that illustrates more string functions:

Program 10-1. Strings

```
10 REM STRINGS
   A$="CHERY"
   B$="RICHARD"
   C$="CINDY"
   D$="ROBERT"
```

```
   E$="RANDY"
   F$="BRETT LYNN"
20 PRINT A$;" has a length of";LEN(A$)
30 PRINT LEFT$(B$,4);" is a nickname."
40 PRINT MID$(C$,2,3)
50 PRINT RIGHT$(D$,4)
60 PRINT RIGHT$(D$,4)+" "+LEFT$(E$,3)
70 P=INSTR(F$," "):PRINT P
80 R=INSTR(2,D$,"R"):PRINT R
90 PRINT STRING$(32,"*")
100 PRINT STRING$(10,65)
110 S$="NAME"+SPACE$(5)+"PHONE"
120 PRINT S$
130 IF LEN(D$)>LEN(E$) THEN PRINT D$
140 END
```

Line 20 illustrates the use of the function LEN. LEN(A$) returns the length of the string A$, or the number of characters in the string. LEN("GREETINGS") would be 9 because there are nine letters in the word. In the program, the string A$ is CHERY, which has five letters.

LEFT$, MID$, and RIGHT$ get certain portions of a string. LEFT$(X$,*n*) returns *n* characters starting at the left of the string—or the first *n* characters of the string. RIGHT$(X$,*n*) returns *n* characters from the right end of the string, or the last *n* characters. MID$(X$,*s*,*n*) returns a string from the middle of the given string. X$ is the original string, *s* is the number of the starting character, and *n* is the number of characters in the string you want. MID$(C$,2,3) says to look at string C$ and start with the second character and use three characters. Since C$ is CINDY, this function starts with the second character, I, and takes three letters, so the result is IND.

INSTR is a function used to locate a certain letter or string within another string. In line 70, INSTR(F$," ") wants to find the space " " in the string F$ and will return the numbered position of that space. The string F$ is "BRETT LYNN". The string we want to find is the space. It is in position 6, so the value returned will be 6. You can specify either constants or variables in the INSTR function. An example of variables is

X=INSTR$(X$,A$)

X will be the position of the first character of A$ found in X$.

You may not want to start at the beginning of a string to find another string. You can specify a number (or numeric variable) as the first parameter in the INSTR function and the searching will start with that character instead of the first

character. Line 80 uses this function. R will be the position of "R" in the string D$ starting with the second character.

STRING$ is a handy function if you want to print a long string of one character. STRING$*(n,c)* returns a string of *n* number of the character with ASCII value *c* or a character specified in quotation marks. Line 90 will print 32 asterisks. Line 100 will print ten of the character corresponding to ASCII value 65, which is the letter *A*.

Lines 60 and 110 illustrate how you can combine strings with the plus sign. The functions can be used in combinations. Line 130 illustrates the LEN function to show that you can use the string functions in calculations or comparisons.

Numeric Functions

The mathematical functions are three-letter abbreviations with an argument or numeric expression in parentheses. The numeric expression can be either a constant (number) or a variable or expression.

ABS*(x)* returns the absolute value of a numeric expression *x*. The absolute value of a number is the number itself without a plus or minus sign. ABS(−4) is 4. ABS(4) is 4. ABS(0) is 0.

ATN*(x)* returns the arctangent of the expression *x*. The arctangent of *x* is the angle whose tangent is *x*. In BASIC, angles are expressed as radians. If you want the equivalent angle in degrees, you can convert by multiplying the radians by 180/pi, or 57.2957795.

COS*(x)* returns the cosine of the expression *x*. Remember that the angle needs to be expressed in radians.

SIN*(x)* returns the sine of the angle *x*.

TAN*(x)* returns the tangent of the angle *x*.

EXP*(x)* gives the exponential function, or the value of e to the *x* power.

LOG*(x)* gives the natural logarithm of *x*, or log of *x* with the base e. Remember that the argument or expression *x* must be greater than zero. The logarithm and exponential functions are inverses:

X=LOG(EXP(X)) and **X=EXP(LOG(X))**

INT*(x)* gives the integer function of a number *x*, which is the whole number part of the number *x* if *x* is positive and the next smaller whole number if the number *x* is negative. Another way to think of the INTeger function is that the result is

the closest integer or whole number to the left of the decimal point of a number.

SGN(*x*) returns the sign of a number *x*. If *x* is negative, SGN(*x*) is equal to −1. If *x* is positive, SGN(*x*) is +1 or 1. If *x* is 0, SGN(*x*) is 0.

SQR(*x*) returns the square root of *x*.

DEF FN. If you wish to use a function that is not listed here, a combination of these functions, or any sort of formula or equation, you can define your own functions with a DEF statement. You need to define the function before you use the function in a program. It is usually simplest to put DEF statements near the beginning of the program. Here are some examples:

```
DEF FNF(X)=2*X*X-5*X+SQR(X)

DEF FNR(N)=INT(N*RND+1)
```

Our next program, "Stepping," uses the function SGN to see if one note is higher, lower, or the same as another. This program can be used with beginning music students who are learning to read music. Two notes are shown on the staff. The student presses the appropriate arrow key to indicate whether the notes are stepping up, stepping down, or staying the same. CR is the correct response and is calculated with SGN(N2−N1), where N2 is the second note and N1 is the first note. If CR is 1, the answer is up, if CR is 0, the notes are the same, and if CR is −1, the answer is down.

Lines 20–90 clear the screen, and print the title and instructions. Lines 100–130 read from data the frequency F and the y coordinate Y for each of nine notes. Lines 140–150 wait for the student to press the space bar to start.

Line 160 performs the quiz for ten problems. Lines 180–200 draw the staff of five lines. Lines 210–280 draw the labels and arrows.

Line 290 randomly chooses the two notes N1 and N2. Line 300 calcultes CR, which is the difference between the notes. Lines 310–380 draw the two notes using Z as the relative y coordinate.

Line 390 makes a prompting sound, then lines 400–425 detect which arrow key is pressed. Line 430 checks the answer, and, if the answer is incorrect, line 440 plays an *uh-oh* sound and line 450 transfers back to line 390 to get another

answer. The answer must be correct to continue. Line 460 prints the message for a correct answer, then line 470 plays the two notes shown. Line 480 delays, then line 490 goes to the next problem.

At the end of the quiz, lines 500–520 play a tune of random notes. Lines 530–550 present an option to try again. Lines 560–570 clear the screen and end the program.

Program 10-2. Stepping

```
10 REM STEPPING UP OR DOWN
20 CLS
30 PRINT:PRINT TAB(30);"STEPPING UP OR DOWN"
40 PRINT:PRINT:PRINT
50 PRINT "You will see two notes."
60 PRINT:PRINT "From the first one, do you step up, step down,"
70 PRINT:PRINT "or stay the same to play the second note?"
80 PRINT:PRINT "Use the arrow keys."
90 PRINT:PRINT:PRINT
100 FOR C=1 TO 9:READ F(C),Y(C):NEXT C
110 DATA 330,110,349,100,392,90
120 DATA 440,80,494,70,523,60
130 DATA 587,50,659,40,698,30
140 PRINT "Press the space bar to start."
150 S$=INKEY$:IF S$<>" " THEN 150
160 FOR PROB=1 TO 10
170 CLS:RANDOMIZE TIMER
180 FOR C=40 TO 120 STEP 20
190 LINE (0,C)-(620,C),1
200 NEXT C
210 LOCATE 20,10:PRINT "^  STEP UP";
220 LINE (75,152)-(75,160)
230 PRINT TAB(30);">  STAY THE SAME";
240 LINE (228,155)-(238,155)
250 PRINT TAB(60);"STEP DOWN"
260 LINE (452,150)-(452,158)
270 LINE (448,155)-(452,158)
280 LINE -(456,155)
290 N1=INT(9*RND)+1:N2=INT(9*RND)+1
300 CR=SGN(N2-N1)
310 Z=Y(N1)
320 LINE (160,Z)-(184,Z+20),3,BF
330 LINE (152,Z+3)-(192,Z+17),3,BF
340 LINE (144,Z+6)-(200,Z+14),3,BF
350 Z=Y(N2)
360 LINE (360,Z)-(384,Z+20),3,BF
370 LINE (352,Z+3)-(392,Z+17),3,BF
380 LINE (344,Z+6)-(400,Z+14),3,BF
390 SOUND 1300,2
400 A$=INKEY$:IF A$="" THEN 400
410 A=ASC(A$)
420 IF A<28 OR A>30 THEN 390
425 IF A=28 THEN R=1 ELSE IF A=29 THEN R=-1 ELSE R=0
430 IF CR=R THEN 460
440 SOUND 165,2:SOUND 131,2
450 GOTO 390
460 LOCATE 23,34:PRINT "CORRECT!"
```

```
470 SOUND F(N1),5:SOUND F(N2),5
480 FOR DELAY=1 TO 3000:NEXT DELAY
490 NEXT PROB
500 FOR P=1 TO 25
510 SOUND INT(400*RND)+400,1
520 NEXT P
530 LOCATE 23,1:PRINT "TRY AGAIN? (Y/N)"
540 A$=INKEY$:IF A$="Y" OR A$="y" THEN 160
550 IF A$<>"N" AND A$<>"n" THEN 540
560 CLS
570 END
```

Graphing Functions

Using the built-in functions and graphics commands to graph mathematical equations on the screen can help you understand mathematical concepts easily and quickly.

To graph simple equations by hand, you first set up an (x,y) coordinate system. You might have an equation such as Y=4*X, Y=X*X, or Y=SIN(X). You pick a value for x, solve for y, and plot the point (x,y). Continue this process for several points and you can see the graph of the function. All this manual calculation is tedious work, but fortunately, your computer can come to the rescue—by doing the repetitive work.

The following sample graphing programs define a function with DEF FN. The x values vary in a loop to get the y values. The results need to be scaled to look good on the screen. Rather than just plot the point (X<Y), the programs draw a LINE from y to the x-axis to show the graph a little better.

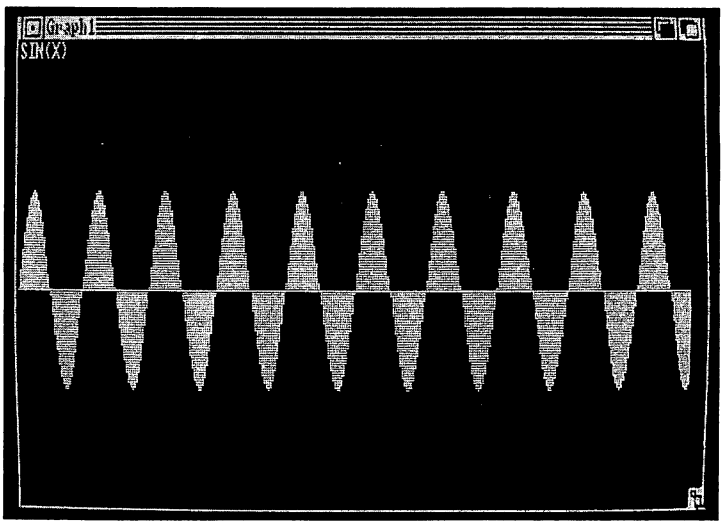
This first program graphs SIN(X). Line 30 uses DEF FN to define the function F(X) as SIN(X). Line 40 prints the title. To plot a different function, you can replace SIN(X) and change the title. This graph will use positive numbers only, so the left edge of the screen will be the y-axis. Line 50 uses the LINE command to draw the x-axis line across the screen and at the middle.

The loop in lines 60–110 picks an x value and evaluates the corresponding y value, which is a scaled value of FNF(X). The 100 is used to move the y value to the middle of the screen. Lines 80 and 90 make sure the graphed points don't go off the screen. We really don't need to worry about the limits for the SIN(X), but if you want to substitute a different function, it may need those limits.

Line 100 draws a line from the y value to the x-axis at the point for x. Line 60 increments the value for x by 0.2 in each

point evaluated. You can change the scaling factor of 40 in line 70 and the factor 10 in line 100 to get the size graph you want. You also can change the x values in line 60.

Figure 10-1. Graph of a Sine Wave



Program 10-3. Graphing SIN(X)

```
10 REM GRAPHING SIN(X)
20 CLS
30 DEF FNF(X)=SIN(X)
40 PRINT "SIN(X)"
50 LINE(0,100)-(640,100)
60 FOR X=0 TO 64 STEP .2
70 Y=100-(40*FNF(X))
80 IF Y<=0 THEN Y=0
90 IF Y>=199 THEN Y=199
100 LINE (X*10,Y)-(X*10,100)
110 NEXT X
120 END
```

SIN(X) is a built-in function, so line 30 could have been omitted and you could have simply used SIN(X) in line 70. However, the use of FNF(X) makes this program more general. To graph a different function, you only have to change the function definition in line 30 and change the label in line 40 (or delete line 40). Try these substitutions for line 30:

```
DEF FNF(X)=COS(X)
DEF FNF(X)=X/12
DEF FNF(X)=X*X/150
DEF FNF(X)=TAN(X)
DEF FNF(X)=LOG(X+1)
```

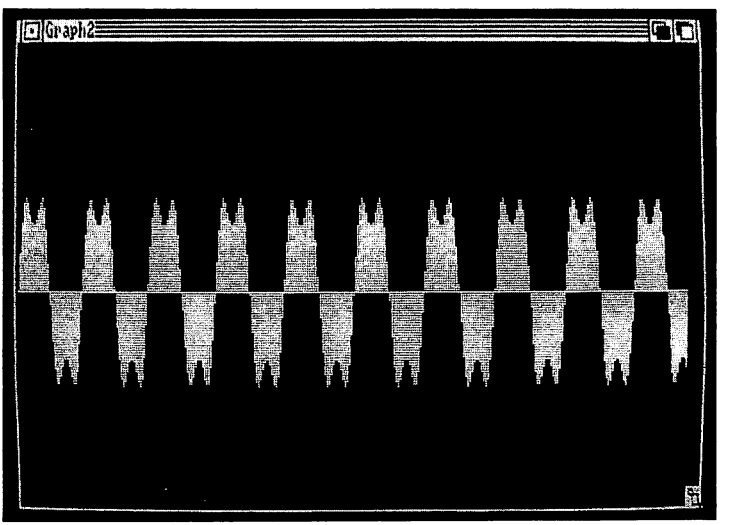
Go ahead and try some functions of your own. Remember, you may want to change some of the scaling factors or spread out your graph by changing the limit and step size in line 60.

Now let's try combining functions on the graph. You can use the same program, but change lines 40 and 70:

```
40 DEF FNG(X)=(1/3)*SIN(3*X)
70 Y=100-40*(FNF(X)+FNG(X))
```

The program will keep F(X) and add it to G(X). The complete listing should now look like that in Figure 10-2.

Figure 10-2. Combined Functions



Program 10-4. Combining Functions

```
10 REM COMBINING FUNCTIONS
20 CLS
30 DEF FNF(X)=SIN(X)
40 DEF FNG(X)=(1/3)*SIN(3*X)
50 LINE(0,100)-(640,100)
60 FOR X=0 TO 64 STEP .2
```

```
70 Y=100-40*(FNF(X)+FNG(X))
80 IF Y<=0 THEN Y=0
90 IF Y>=199 THEN Y=199
100 LINE (X*10,Y)-(X*10,100)
110 NEXT X
120 END
```

Program 10-4 adds the two functions and graphs the results. You can change to subtraction by changing the plus to a minus in line 70 or by making one of the defined functions negative.

Chapter 11

Educational Programming

Chapter 11

Educational Programming

Many people cite education as the main use for a computer in the home. While a computer will never take the place of a loving parent or schoolteacher, it can be a powerful learning tool for enhancing a child's education. In addition, color graphics and music can add a dimension of fun to the learning process.

The programs presented here are a beginning point—feel free to customize the programs for your own use. Perhaps you can change the music, the graphics, or the colors. Add names of your own students. I used a simple arpeggio for a correct answer; you can substitute your favorite tune. Change a drill to suit your own needs.

A Drill Program

There are many kinds of educational programs. Probably the most common type is the drill. "Simple Drill," Program 3-2, in the "Random Numbers" chapter illustrates the basic programming for a drill program using random questions. This structure can be used for any type of drill.

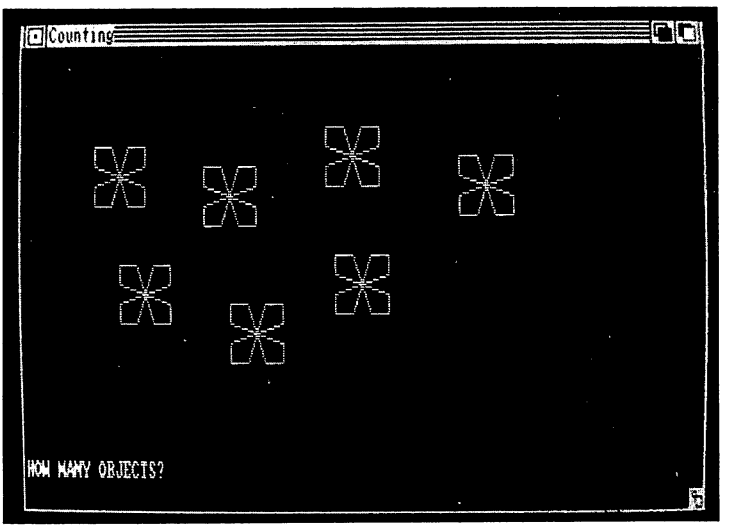
Program 11-1 is a counting drill for young children, consisting of ten problems. Up to seven shapes are displayed in random colors. A random number of objects—up to nine—appear on the screen. The child must count the objects and press the correct number. If the answer is incorrect, there is an *uh-oh* sound. The student must press the right number for the program to continue.

Point out to the child that the numbers are on the top row of keys (below the function keys) or on the number pad at the right side of the keyboard. This program can help children learn the concept of counting objects in a one-to-one relationship.

A(I) and B(I) are the coordinates for each of the nine objects. Line 160 chooses a random number, N, the number of objects from one to nine. Line 170 chooses a random number from 1 through 3 for the color. Line 180 chooses a random number, S, from 1 through 7 for the shape number. The shapes are in different subroutines, and line 210 is an ON-

GOSUB command to go to a particular subroutine depending on the value of S. The loop in lines 190–220 draws the right number of objects.

Figure 11-1. Counting Shapes



Program 11-1. Counting Shapes

```
10 REM COUNTING SHAPES
20 CLS
30 LOCATE 3,30
40 PRINT "** COUNTING SHAPES **
50 LOCATE 6,3:PRINT "You will see some shapes on the screen.
"
60 PRINT:PRINT "  How many shapes are there?"
70 PRINT:PRINT "  Press the correct number."
80 FOR J=1 TO 9:READ A(J),B(J):NEXT J
90 DATA 72,40,176,48,288,32,408,44
100 DATA 96,88,200,104,296,84,384,96,496,92
110 PRINT:PRINT:PRINT "  PRESS THE SPACE BAR TO START."
120 RANDOMIZE TIMER
130 A$=INKEY$:IF A$<>" " THEN 130
140 FOR P=1 TO 10
150 CLS:FL=0
160 N=INT(9*RND+1)
170 C=INT(3*RND+1)
180 S=INT(7*RND+1)
190 FOR L=1 TO N
200 X=A(L):Y=B(L)
210 ON S GOSUB 490,520,580,690,780,820,1040
220 NEXT L
230 LOCATE 22,1:PRINT "HOW MANY OBJECTS? ";
240 A$=INKEY$
```

```
250 IF A$="" THEN 240
260 IF A$<"1" OR A$>"9" THEN 250
270 PRINT A$
280 IF VAL(A$)=N THEN 370
290 SOUND 330,2:SOUND 262,2
300 FL=FL+1:IF FL<2 THEN LOCATE 22,19:PRINT " ":GOTO 230
310 FOR L=1 TO N
320 SOUND 1300,2:LOCATE B(L)/8,A(L)/8+1
330 PRINT L
340 NEXT L
350 FOR D=1 TO 4000:NEXT D
360 GOTO 400
370 SOUND 262,3:SOUND 330,3
380 SOUND 392,3:SOUND 523,6
390 FOR D=1 TO 2000:NEXT D
400 NEXT P
410 FOR J=1 TO 25
420 SOUND 900*RND+500,2
430 NEXT J
440 CLS
450 LOCATE 5,5:PRINT "TRY AGAIN?  (Y/N)"
460 A$=INKEY$:IF A$="Y" OR A$="y" THEN 140
470 IF A$="N" OR A$="n" THEN 1060 ELSE 460
480 REM SQUARE
490 LINE (X,Y)-(X+48,Y+24),C,BF
500 RETURN
510 REM TRIANGLE
520 LINE (X+24,Y)-(X+48,Y+24),C
530 LINE -(X,Y+24),C
540 LINE -(X+24,Y),C
560 RETURN
570 REM OCTAGON
580 LINE (X+16,Y)-(X+32,Y),C
590 LINE -(X+48,Y+8),C
600 LINE -(X+48,Y+16),C
610 LINE -(X+32,Y+24),C
620 LINE -(X+16,Y+24),C
630 LINE -(X,Y+16),C
640 LINE -(X,Y+8),C
650 LINE -(X+16,Y),C
670 RETURN
680 REM HEXAGON
690 LINE (X+16,Y)-(X+40,Y),C
700 LINE -(X+56,Y+12),C
710 LINE -(X+40,Y+24),C
720 LINE -(X+16,Y+24),C
730 LINE -(X,Y+12),C
740 LINE -(X+16,Y),C
760 RETURN
770 REM CROSS
780 LINE (X+16,Y)-(X+32,Y+24),C,BF
790 LINE (X,Y+8)-(X+48,Y+16),C,BF
800 RETURN
810 REM FLOWER
820 LINE (X,Y)-(X+16,Y),C
830 LINE -(X+24,Y+12),C
840 LINE -(X+32,Y),C
850 LINE -(X+48,Y),C
860 LINE -(X+48,Y+8),C
870 LINE -(X+24,Y+12),C
880 LINE -(X+48,Y+16),C
```

```
890 LINE -(X+48,Y+24),C
900 LINE -(X+32,Y+24),C
910 LINE -(X+24,Y+12),C
920 LINE -(X+16,Y+24),C
930 LINE -(X,Y+24),C
940 LINE -(X,Y+16),C
950 LINE -(X+24,Y+12),C
960 LINE -(X,Y+8),C
970 LINE -(X,Y),C
1020 RETURN
1030 REM RECTANGLE
1040 LINE (X,Y+6)-(X+48,Y+18),C,BF
1050 RETURN
1060 CLS
1070 END
```

Question-and-Answer Quizzes

Another type of drill is a question-and-answer quiz, such as a history quiz with dates corresponding to events. Any subject can be used with the basic programming idea, and any number of items or questions and answers may be used.

In Program 11-2, an event will be shown on the screen, and the student must type the year the event occurred and press RETURN. The dates and events are contained in the DATA statements. Line 20 defines N as the number of events.

Line 40 sets WIDTH 80 so the printing will be limited on the screen. You may use spaces in your events so the printing looks good and doesn't split words.

Line 100 READs the dates D(M) and events E$(M) from the DATA. Line 110 initializes the score, S, to be zero. S is incremented by one for each correct answer in line 220.

In line 230, the event, E$, is set to the null string, " ", so the event will not be chosen again.

If you use questions with words for answers rather than the numbers for dates, change the variable D to a string variable.

Program 11-2. History Trivia—Ontario

```
10 REM HISTORY TRIVIA--ONTARIO
20 N=18
30 DIM D(N),E$(N)
40 WIDTH 80:CLS
50 PRINT TAB(10);"** HISTORY TRIVIA:  ONTARIO **"
60 PRINT:PRINT:PRINT
70 PRINT "You will be given an event."
80 PRINT:PRINT "What year did it take place?"
90 PRINT:PRINT "Type the year and press <RETURN>."
100 FOR M=1 TO N:READ D(M),E$(M):NEXT M
110 S=0
```

```
120 PRINT:PRINT:PRINT "PRESS THE SPACE BAR TO START."
130 RANDOMIZE TIMER
140 A$=INKEY$:IF A$<>" " THEN 140
150 FOR T=1 TO 10
160 CLS:LOCATE 3,1
170 R=INT(N*RND)+1:IF E$(R)="" THEN 170
180 PRINT E$(R)
190 PRINT:INPUT "What was the year? ",Y
200 IF Y=D(R) THEN 220
210 PRINT:PRINT "The correct year was";D(R):GOTO 230
220 PRINT:PRINT "CORRECT!":S=S+1
230 E$(R)=""
240 PRINT:PRINT "PRESS <RETURN>."
250 A$=INKEY$
260 IF A$<>CHR$(13) THEN 250
270 NEXT T
280 CLS:PRINT
290 PRINT "Out of 10 events in this quiz, your score is";S
300 PRINT:PRINT:PRINT
310 GOTO 520
320 DATA 1615,Samuel de Champlain reached Lake Huron
330 DATA 1763,Ontario was ceded to Great Britain by the Trea
ty of Paris
340 DATA 1780,First British settlement on Niagara River
350 DATA 1784,Influx of United Empire Loyalists
360 DATA 1791,Separate government given to Upper Canada
370 DATA 1792,First session of Upper Canada Legislature at N
ewark
380 DATA 1812,War between U.S. and Great Britain
390 DATA 1827,King's College founded in Toronto
400 DATA 1837,William Lyon MacKenzie led rebellion at York
410 DATA 1840,Act of Union
420 DATA 1856,Toronto linked to Montreal by Grand Trunk Rail
way
430 DATA 1867,British North America Act; Canada became a nat
ion
440 DATA 1885,Canadian Pacific Railway completed to link Ont
ario and B.C.
450 DATA 1912,Ontario enlarged by Keewatin District of North
west Territories
460 DATA 1959,St. Lawrence Seaway completed from Lake Erie t
o Montreal
470 DATA 1962,Canada's first nuclear power plant at Rolphton
480 DATA 1964,Toronto's International Airport inaugurated
490 DATA 1978,Toronto PET Users Group organized
520 END
```

General-Purpose Multiple Choice

I have seen a number of programs written for multiple-choice tests. The computer is an ideal way to administer such tests because it can mix up the test questions so that each run is different. Each question has four possible answers, and the computer can keep track of the correct answer. However, all of the programs I have seen print the question, then the answers in the same order each time the program is run. Here is a general-purpose multiple-choice test (Program 11-3) that chooses

questions in a random order without repetition and also rearranges the possible answers in a random order.

I am including computer literacy questions here for an example. Again, you may use any number of questions. Line 300 sets up 20 questions for this quiz. Line 190 DIMensions variables for 30 questions. T$ is the question. A$ are the answers, and B is the correct answer. Lines 200–260 read in the questions and answers from the DATA from line 720 to the end. Note that the last DATA statement contains Z's to indicate the end as information is being read in.

S$ keeps track if a question is used. Line 230 sets each S$(I) equal to "A". As a question is used, S$ is set to " " in line 360. Lines 320–330 provide that if S$ is " ", then another question must be chosen.

Lines 370–390 define C(J) for the four answers for mixing up the order in which the answers are printed. Line 400 randomly chooses D for the correct answer. Line 410 defines AA$(D) to hold the correct answer. The C variable for the correct answer is set to zero so that it cannot be used in another position. Lines 430–490 mix up the order of the answers, making sure the correct answer is in the right position and each answer is used only once. Lines 500–530 print the four answers with the four possible choices A, B, C, and D.

Lines 540–580 receive the student's answer, making sure it is a letter from A through D (uppercase or lowercase), then print the choice. Line 590 checks to see if the key pressed is the correct choice. Line 600 prints the message for an incorrect answer and prints the correct answer. Line 620 prints CORRECT for a correct answer, then line 630 increments the score, SC. Lines 640–670 wait for the student to press RETURN before going to the next question. Lines 680–700 clear the screen and then print the score.

As you are typing the DATA statements, note that some of the lines have extra spaces. This is to adjust the printing for the WIDTH 77 screen so that words will not be split. If the question contains a comma as part of the printing, the question must be enclosed in quotation marks.

To make a test for a different topic, simply change the questions and answers in the DATA statements, making sure that you have enough questions for a complete quiz. The last DATA statement contains the ZZZ to signal the end.

Program 11-3. Multiple-Choice Test

```
100 REM MULTIPLE CHOICE TEST
110 CLS:WIDTH 80
120 PRINT "    ************************"
130 PRINT "    * MULTIPLE CHOICE TEST *"
140 PRINT "    ************************"
150 PRINT:PRINT:PRINT:PRINT
160 PRINT "TEST OF 20 QUESTIONS"
170 PRINT:PRINT "PRESS LETTER OF CORRECT"
180 PRINT "ANSWER FOR EACH QUESTION."
190 DIM T$(30),A$(30,4),B(30),S$(30),AA$(4)
200 I=1
210 READ T$(I),A$(I,1),A$(I,2),A$(I,3),A$(I,4),B(I)
220 IF T$(I)="ZZZ" THEN 260
230 S$(I)="A"
240 I=I+1
250 GOTO 210
260 I=I-1
270 PRINT:PRINT "PRESS <RETURN> TO START."
280 K$=INKEY$
290 IF K$<>CHR$(13) THEN 280
300 FOR P=1 TO 20
310 RANDOMIZE TIMER
320 X=INT(I*RND)+1
330 IF S$(X)="" THEN 320
340 CLS
350 PRINT T$(X):PRINT
360 PRINT:S$(X)=""
370 FOR J=1 TO 4
380 C(J)=1
390 NEXT J
400 D=INT(4*RND)+1
410 AA$(D)=A$(X,B(X))
420 C(B(X))=0
430 FOR J=1 TO 4
440 IF J=D THEN 490
450 E=INT(4*RND)+1
460 IF C(E)=0 THEN 450
470 AA$(J)=A$(X,E)
480 C(E)=0
490 NEXT J
500 FOR J=1 TO 4
510 PRINT CHR$(64+J);". ";AA$(J)
520 NEXT J
530 PRINT:PRINT
540 REM
550 K$=INKEY$
560 IF K$<"A" OR (K$>"D" AND K$<"a") OR K$>"d" THEN 550
570 IF ASC(K$)>68 THEN K$=CHR$(ASC(K$)-32)
580 PRINT K$:PRINT
590 IF ASC(K$)=64+D THEN 620
600 PRINT "NO, THE ANSWER IS ";CHR$(64+D);"."
610 GOTO 640
620 PRINT "CORRECT"
630 SC=SC+1
640 PRINT:PRINT "PRESS <RETURN>."
650 K$=INKEY$
660 IF K$<>CHR$(13) THEN 650
670 NEXT P
680 CLS
```

```
690 PRINT "OUT OF 20 QUESTIONS,"
700 PRINT "YOUR SCORE IS ";SC:PRINT:PRINT
710 GOTO 1360
720 DATA One of the major attractions of a computer is that
it
730 DATA has active involvement.,is expensive.,is a status s
ymbol.
740 DATA allows uninvolvement.,1
750 DATA A video game is best described as
760 DATA an expensive toy.,a special purpose computer.,a hom
e computer.,
an educational toy.,2
770 DATA The computer owes its flexibility to the fact that
it is
780 DATA small.,complicated.,programmable.,an electronic dev
ice.,3
790 DATA "Because a computer is programmable,"
800 DATA it can be used to perform only a limited number of
functions.
810 DATA it cannot be used for educational purposes.
820 DATA it cannot be used for entertainment.
830 DATA it can become a general purpose tool.,4
840 DATA The main advantage of a computer as opposed to othe
r calculating devices is  its
850 DATA cost.,size.,portability.,programmable nature.,4
860 DATA Books and manuals that accompany a computer-related
 product are
870 DATA software.,documentation.,data.,compu-forms.,2
880 DATA Visicalc is best described as
890 DATA a tutorial program.,an electronic spreadsheet.
900 DATA an educational program.,an entertainment program.,2
910 DATA All of the following are programming languages exce
pt
920 DATA BASIC.,Pascal.,Visicalc.,Logo.,3
930 DATA One of the major problems in acquiring computer lit
eracy is
940 DATA people need to be skilled in math to use computers.
950 DATA the computer is a very complicated machine.
960 DATA the field has its own lexicon or language.
970 DATA people need a background in logic and statistics.,3
980 DATA The parts of a computer are arranged in such a way
as to form
990 DATA a system.,a machine.,a subsystem.,an organization.,
1
1000 DATA The processing of data in a computer system result
s in the generation of
1010 DATA a program.,readouts.,information.,statistics.,3
1020 DATA "Basically, a computer is intended to produce"
1030 DATA information.,data.,statistics.,programs.,1
1040 DATA The basic function of a computer is to transform
1050 DATA programs into data.
1060 DATA data into programs.
1070 DATA information into data.
1080 DATA data into information.,4
1090 DATA "By using a _____, one may connect a computer to t
he telephone to permit       computer conferencing."
1100 DATA adapter,connector,conference link,modem,4
1110 DATA Intangibility is a major characteristic of
1120 DATA software.,the computer.,hardware.,magnetic disks.,
1
1130 DATA The use to which a computer is put is called
```

```
1140 DATA a program.,a routine.,an application.,a function.,
3
1150 DATA Inside the computer information is represented by
1160 DATA punched cards.,electronic signals.,magnetic tape.,
magnetic disks.,2
1170 DATA The on/off pattern that is used in the computer is
 the basis of the
1180 DATA circuit code.,binary code.,binomial code.,bidecima
l code.,2
1190 DATA "With telecommuting, information is most commonly
transmitted between          terminals"
1200 DATA by radio.,over telephone wires.
1210 DATA via satellite.,by television.,2
1220 DATA A computer program is an example of
1230 DATA hardware.,software.,firmware.,flexware.,2
1240 DATA The first electronic computer was
1250 DATA ENIAC.,ENID.,IBM MARK I.,IBM Cybernaught.,1
1260 DATA The computer is instructed or told what to do by
1270 DATA hardware.,firmware.,software.,smartware.,3
1280 DATA The most significant factor in purchasing a comput
er is
1290 DATA relative cost.,available software.,available hardw
are.,available firmware.,2
1300 DATA Which is the most common type of secondary storage
 currently used in          personal computers?
1310 DATA floppy disks,bubble memory,electric conductors,tun
nel junction memory,1
1320 DATA RAM is used as a measure of
1330 DATA primary storage capacity.,processing power.
1340 DATA processing speed.,word length.,1
1350 DATA ZZZ,Z,Z,Z,Z,0
1360 END
```

Homework Helper

Other valuable uses for the computer, both in class and at home, are in the areas of supplying example problems and answers as well as checking answers to homework problems.

Answering programs allow the student to indicate the type of problem and INPUT the numbers for the calculations; then the program prints the answer. Any type of problem with a formula may be made into this type of answering program. This type of program is helpful when there are many problems to solve that all use the same formula.

"Homework Helper—Factors," Program 11-4, is designed to help a student quickly check the answers to an assignment with problems involving factoring. It is written for problems encountered in the fourth, fifth, and sixth grades. The student should do the class assignment in the usual way, writing the problem down on paper and working it out step by step. This program may then be used to check the answers. The program has four main sections.

Find All Factors. The student enters a number, and all possible factors or divisors of that number are listed from largest to smallest. The list of factors includes the number itself and the number 1. The number to be factored must be greater than 1 and must be a whole number (integer). Example: All the factors of 18 are 18, 9, 6, 3, 2, and 1.

Find Prime Factors. Finding the prime factors is also called complete factorization or making a prime factor tree. The student enters a number, and all the prime factors of that number are listed from smallest to largest. Thse numbers multiplied together yield the original number. The student's answer does not have to list the factors in a certain order to be considered correct. This section lists all the factors necessary to obtain the given number. If only the prime factors of a number are desired, the student would still choose this option of the program, and the answer would consist of the list of factors without duplication of numbers. Example: All prime factors of 18 are 2, 3, 3. Prime factors without duplication would be 2 and 3.

Find Greatest Common Factor. The student enters two numbers. The computer lists the greatest common factor, which is the largest number that can be divided evenly into both the input numbers. If both numbers are prime or if they have no common factors, then the greatest common factor is 1. Example: The greatest common factor of 18 and 24 is 6.

Find Least Common Multiple. The student first indicates whether there are two numbers or three numbers, then inputs those numbers (this is adequate for fifth-grade or sixth-grade mathematics). The program will print the least common multiple, or the lowest number that all the given numbers may be divided into without remainders. This exercise is an introduction to finding least common denominators. Example: The least common multiple of 4 and 12 is 12. The least common multiple of 4, 6, and 5 is 60.

After each answer is given, the student has the option of trying another problem or going back to the main menu screen.

The INTeger function is used to help find the factors. In finding all the factors, numbers from 2 to the number are divided into the original number. If the quotient is equal to the INTeger of the quotient, that means the number may be divided evenly and the quotient is a factor. In finding the prime

factors, once a factor is found, the limit in the loop is changed. In finding the greatest common factor, numbers are tested as factors for both the input numbers.

Program 11-4. Homework Helper—Factors

```
10 REM FACTORS
20 CLS:WIDTH 80
30 PRINT TAB(20);STRING$(19,"*")
40 PRINT TAB(20);"* HOMEWORK HELPER *"
50 PRINT TAB(20);"*";SPC(17);"*"
60 PRINT TAB(20);"*     FACTORS     *"
70 PRINT TAB(20);STRING$(19,"*")
80 PRINT:PRINT:PRINT
90 PRINT TAB(10);"CHOOSE:  1  FIND ALL FACTORS"
100 PRINT TAB(19);"2  FIND PRIME FACTORS"
110 PRINT TAB(19);"3  FIND GREATEST COMMON FACTOR"
120 PRINT TAB(19);"4  FIND LEAST COMMON MULTIPLE"
130 PRINT TAB(19);"5  END PROGRAM"
140 C$=INKEY$
150 IF C$<"1" OR C$>"5" THEN 140
160 ON VAL(C$) GOTO 320,480,630,940,1350
170 PRINT:PRINT "CHOOSE:  1 ANOTHER PROBLEM"
180 PRINT TAB(10);"2 BACK TO MAIN MENU SCREEN"
190 C$=INKEY$
200 IF C$="2" THEN 20
210 IF C$<>"1" THEN 190
220 RETURN
230 PRINT:PRINT
240 INPUT "ENTER NUMBER TO BE FACTORED: ",N
250 IF N>1 THEN 270
260 PRINT:PRINT "NUMBER MUST BE GREATER THAN ONE.":PRINT:GOT
O 240
270 IF N=INT(N) THEN 290
280 PRINT:PRINT "NUMBER MUST BE A WHOLE NUMBER.":PRINT:GOTO
240
290 IF N<10001 THEN 310
300 PRINT:PRINT "PLEASE USE NUMBER LESS THAN 10000.":PRINT:G
OTO 240
310 RETURN
320 CLS
330 PRINT "GIVEN A NUMBER, FIND ALL ITS FACTORS."
340 GOSUB 230
350 PRINT:PRINT "FACTORS OF";N;"ARE"
360 PRINT N;
370 B=INT(N/2+1)
380 FOR C=2 TO B
390 IF N/C<>INT(N/C) THEN 430
400 B=N/C:PRINT B;
410 IF B=1 THEN 450
420 IF B=2 THEN 440
430 NEXT C
440 PRINT " 1"
450 PRINT
460 GOSUB 170
470 GOTO 320
480 CLS
490 PRINT "GIVEN A NUMBER, FIND THE PRIME FACTORS,"
500 PRINT "ALSO KNOWN AS COMPLETE FACTORIZATION"
```

```
510 GOSUB 230
520 PRINT:PRINT "THE PRIME FACTORS ARE:"
530 G=INT(N/2)
540 FOR M=2 TO G
550 IF N/M<>INT(N/M) THEN 570
560 N=N/M:G=N:PRINT M;:GOTO 540
570 NEXT M
580 IF N=1 THEN 600
590 PRINT N
600 PRINT
610 GOSUB 170
620 GOTO 480
630 CLS
640 PRINT "FIND THE GREATEST COMMON FACTOR OF TWO GIVEN NUMB
ERS."
650 PRINT
660 M=0:N=0
670 INPUT "FIRST NUMBER: ",M
680 IF M>1 THEN 700
690 PRINT:PRINT "ENTER A NUMBER GREATER THAN 1.":PRINT:GOTO
670
700 IF M<10000 THEN 720
710 PRINT:PRINT "MUST BE A NUMBER LESS THAN 10000.":PRINT:GO
TO 670
720 IF M=INT(M) THEN 740
730 PRINT:PRINT "A WHOLE NUMBER PLEASE.":PRINT:GOTO 670
740 PRINT:INPUT "SECOND NUMBER: ",N
750 IF N>1 THEN 770
760 PRINT:PRINT "ENTER A NUMBER GREATER THAN 1.":GOTO 740
770 IF N<10000 THEN 790
780 PRINT:PRINT "MUST BE A NUMBER LESS THAN 10000.":GOTO 740
790 IF N=INT(N) THEN 810
800 PRINT:PRINT "A WHOLE NUMBER PLEASE.":GOTO 740
810 PRINT:PRINT "GREATEST COMMON FACTOR = ";
820 IF M=N THEN G=M:GOTO 910
830 IF M<N THEN 850
840 SWAP M,N
850 FOR K=1 TO M
860 IF (M/K)<>INT(M/K) THEN 900
870 J=M/K
880 IF (N/J)<>INT(N/J) THEN 900
890 G=J:GOTO 910
900 NEXT K:G=1
910 PRINT G
920 GOSUB 170
930 GOTO 630
940 CLS
950 PRINT "FIND THE LEAST COMMON MULTIPLE OF TWO OR THREE NU
MBERS."
960 PRINT
970 PRINT:PRINT "HOW MANY NUMBERS--2 OR 3? ";
980 A$=INKEY$
990 IF A$<"2" OR A$>"3" THEN 980
1000 A=VAL(A$):PRINT A$
1010 FOR C=1 TO A:N(C)=0
1020 PRINT:PRINT "NUMBER";C;:INPUT N(C)
1030 IF N(C)>1 THEN 1050
1040 PRINT "NUMBER MUST BE GREATER THAN ONE.":GOTO 1020
1050 IF N(C)<1000 THEN 1070
1060 PRINT "NUMBER MUST BE LESS THAN 1000.":GOTO 1020
1070 IF N(C)=INT(N(C)) THEN 1090
```

```
1080 PRINT "A WHOLE NUMBER PLEASE.":GOTO 1020
1090 NEXT C
1100 IF A$="3" THEN 1180
1110 IF N(1)<>N(2) THEN 1130
1120 L=N(1):GOTO 1320
1130 IF N(1)<N(2) THEN 1150
1140 SWAP N(1),N(2)
1150 FOR C=1 TO N(1)
1160 IF C*N(2)/N(1)=INT(C*N(2)/N(1)) THEN L=C*N(2):GOTO 1320
1170 NEXT C:L=N(1)*N(2):GOTO 1320
1180 S=0
1190 FOR C=1 TO 2
1200 IF N(C)<=N(C+1) THEN 1220
1210 SWAP N(C),N(C+1):S=1
1220 NEXT C:IF S=1 THEN 1180
1230 FOR C=1 TO N(2)
1240 F=C*N(3)
1250 IF (F/N(1)=INT(F/N(1))) AND (F/N(2)=INT(F/N(2))) THEN L
=F:GOTO 1320
1260 NEXT C
1270 M=N(2)*N(3)
1280 FOR C=1 TO N(1)
1290 F=C*M
1300 IF F/N(1)=INT(F/N(1)) THEN L=F:GOTO 1320
1310 NEXT C:L=M*N(1)
1320 PRINT:PRINT "LEAST COMMON MULTIPLE IS ";L
1330 GOSUB 170
1340 GOTO 940
1350 CLS
1360 END
```

Interaction with the Student

Tutorial programs actually teach the student as the program is used. The main idea for tutorial programs as computer-aided instruction is that a student can work at his or her own pace. The computer introduces topics as the student is ready for them. Often, the program advances only when the student masters a certain level, and the program may offer remediation if necessary. The student has instant feedback through interaction with the computer.

The typing program and the algebra program in Chapter 14, "Sample Programs," are examples of tutorials. "Notes" in the chapter on music may also be considered a tutorial with a drill. "Locating Points" presented here is a tutorial for beginning coordinate geometry at the elementary level.

This program teaches the student how a rectangular coordinate system works. First, a random example point with a given x coordinate and y coordinate is shown on a rectangular coordinate system. The student may press Y (yes) to see another example point or N (no) to continue the program.

If the student presses N, the screen clears and a random point is shown. The student must identify the point by first pressing the number for the x coordinate and then the number for the y coordinate. If the answer is incorrect, the correct answer is shown and the student must do another problem. If the answer is correct, the student has the option of trying another problem of the same kind or continuing the program.

The next section of this tutorial shows the grid with a point at (0,0). A random set of coordinates is chosen and printed. The student must locate the point by moving the dot on the grid. Arrow keys are used to move the dot. When the student has moved the point to the desired position, he or she presses the RETURN key. The point changes color and is checked. If the answer is incorrect, the student is shown the right location and given another problem. If the answer is correct, the student has the option of having another of the same type of problem, starting the program over, or ending the program.

The grid is printed several times, so it is placed in a subroutine in lines 50–140. The COLOR command changes the color of the printing.

Line 250 converts the x and y values chosen to ROW and COLumn variables for use in the LOCATE commands. The subroutine in lines 290–360 changes the colors of the lines to show a location first using a COLOR statement, then printing the lines. Printing from right to left for the line is done by using a FOR-NEXT loop with a negative STEP size, lines 330–340.

Outline of Locating Points Program

Lines	Explanation
40	Branches past subroutines.
50–140	Subroutine to clear screen and draw grid.
150–190	Subroutine to wait for student to press RETURN.
200–260	Subroutine to randomly choose x and y and determine the ROW and COL position.
270–360	Subroutine to play *uh-oh* for incorrect answer and to draw lines showing correct position.
370–390	Subroutine to play arpeggio for correct answer.
400	Clears screen.
420	Prints title.

Lines	Explanation
430–470	Define strings for graphic characters used in printing grid.
480–550	Print title screen with information.
560–630	Print first screen showing example point.
640–670	Present option for another example or to continue and branch.
680–760	Print problem and ask student for coordinates.
770–840	Receive student's answers.
850–900	If answer is incorrect, show and print correct answer; do another problem.
910–1000	Print instructions.
1010–1020	Ask student to locate point.
1030–1190	Move point as student presses arrow keys.
1200–1205	When student presses RETURN, change color of point.
1210–1240	If answer is incorrect, show correct point and do another problem.
1250	If answer is correct, plays an arpeggio.
1260–1320	Print options and branch.
1330–1340	Clear screen and end.

Program 11-5. Locating Points

```
10 REM LOCATING POINTS
40 GOTO 400
50 REM GRID
60 CLS:COLOR 1:PRINT
70 FOR M=6 TO 1 STEP -1
80 PRINT TAB(5);RIGHT$(STR$(M),1);L1$
90 PRINT TAB(6);U1$:PRINT TAB(6);U1$
100 NEXT M
110 PRINT TAB(5);"0";L1$
120 PRINT TAB(6);"0   1   2   3   4   5   6   7   8   9"
140 RETURN
150 LOCATE 22,2:PRINT "PRESS <RETURN>."
160 BEEP
170 E$=INKEY$
180 IF E$<>CHR$(13) THEN 170
190 RETURN
200 REM CHOOSE X AND Y
210 X=INT(10*RND)
220 Y=INT(7*RND)
230 X$=RIGHT$(STR$(X),1)
240 Y$=RIGHT$(STR$(Y),1)
250 ROW=20-3*Y:COL=6+4*X
260 RETURN
270 REM INCORRECT
280 SOUND 250,5:SOUND 200,5
290 REM DRAW STRIPES
300 COLOR 3
310 FOR M=21-Y*3 TO 20
320 LOCATE M,COL:PRINT U$:NEXT M
```

```
330 FOR M=3+4*X TO 6 STEP -1
340 LOCATE ROW,M:PRINT L$:NEXT M
350 LOCATE ROW,COL:PRINT "+";:COLOR 1
360 RETURN
370 SOUND 262,3:SOUND 330,3
380 SOUND 392,3:SOUND 523,6
390 RETURN
400 CLS:WIDTH 80
420 PRINT TAB(20);"** LOCATING POINTS **"
430 C$="+":S$=" "
440 L$=C$+"---"
450 U$="|":U2$=U$+"   "
460 L1$=L$:U1$=U2$
470 FOR M=1 TO 9:L1$=L1$+L$:U1$=U1$+U2$:NEXT M
480 PRINT:PRINT
490 PRINT "   This program discusses coordinate geometry,"
500 PRINT:PRINT "   or locating points using x- and y-coordi
nates."
510 PRINT
520 PRINT "   Any point can be defined on the grid by specif
ying"
530 PRINT:PRINT "   an x distance and a y distance from the
origin."
540 RANDOMIZE TIMER
550 GOSUB 150
560 GOSUB 60
570 X=INT(5*RND)+1:Y=INT(4*RND)+1
580 GOSUB 230
590 GOSUB 300
600 PRINT " (";X$;",";Y$;")"
610 COLOR 2:LOCATE 10,50
620 PRINT "A point has an x-coordinate"
630 PRINT TAB(50);"and a y-coordinate."
640 PRINT:PRINT TAB(50);"Want another example? (Y/N)";
660 E$=INKEY$:IF E$="Y" OR E$="y" THEN 560
670 IF E$<>"N" AND E$<>"n" THEN 660
680 GOSUB 60
690 GOSUB 210
700 COLOR 3
710 LOCATE ROW,COL:PRINT "*"
720 COLOR 2
730 LOCATE 10,50:PRINT "What are the coordinates?"
740 PRINT:PRINT TAB(50);"(?,?)"
750 COLOR 3
760 LOCATE 12,51:PRINT "?"
770 E$=INKEY$:IF E$<"0" OR E$>"9" THEN 770
780 LOCATE 12,51
790 PRINT RIGHT$(E$,1);
810 LOCATE 12,53:PRINT "?"
820 F$=INKEY$:IF F$<"0" OR F$>"9" THEN 820
830 LOCATE 12,53
840 PRINT RIGHT$(F$,1);
850 IF VAL(E$)=X AND VAL(F$)=Y THEN GOSUB 370:GOTO 910
860 GOSUB 280
870 LOCATE 13,50:PRINT "(";X$;",";Y$;")"
880 COLOR 1:LOCATE 15,50:PRINT "PRESS <RETURN>."
890 GOSUB 170
900 GOTO 680
910 COLOR 1:LOCATE 15,49
920 PRINT "PRESS 1 FOR SAME TYPE PROBLEM"
```

```
930 PRINT TAB(55);"2 TO CONTINUE PROGRAM"
940 E$=INKEY$:IF E$="1" THEN 680
950 IF E$<>"2" THEN 940
960 CLS
970 PRINT "You will be given the coordinates."
980 PRINT:PRINT "Use the arrow keys to position the point,"
990 PRINT:PRINT "then press <return>."
1000 GOSUB 150
1010 GOSUB 60:GOSUB 210
1020 LOCATE 10,50:PRINT "LOCATE (";X$;",";Y$;")"
1030 TR=20:TC=6
1040 COLOR 3:LOCATE TR,TC:PRINT "*"
1050 E$=INKEY$:IF E$="" THEN 1050
1060 IF E$=CHR$(13) THEN 1200
1090 IF ASC(E$)=30 THEN DC=4:DR=0:GOTO 1140
1100 IF ASC(E$)=28 THEN DC=0:DR=-3:GOTO 1140
1110 IF ASC(E$)=31 THEN DC=-4:DR=0:GOTO 1140
1120 IF ASC(E$)<>29 THEN 1050
1130 DC=0:DR=3
1140 COLOR 1:LOCATE TR,TC:PRINT C$
1150 TR=TR+DR:IF TR>20 THEN TR=20
1160 IF TR<2 THEN TR=2
1170 TC=TC+DC:IF TC>42 THEN TC=42
1180 IF TC<6 THEN TC=6
1190 GOTO 1040
1200 COLOR 1:LOCATE TR,TC:PRINT "*"
1205 COLOR 3:LOCATE TR,TC:PRINT "*"
1210 IF TR=ROW AND TC=COL THEN 1250
1220 GOSUB 280
1230 LOCATE 15,50:PRINT "PRESS <RETURN>"
1240 GOSUB 170:GOTO 1010
1250 GOSUB 370
1260 COLOR 1:LOCATE 17,49
1270 PRINT "PRESS 1 FOR SAME TYPE PROBLEM"
1280 PRINT TAB(55);"2 START PROGRAM OVER"
1290 PRINT TAB(55);"3 END PROGRAM"
1300 E$=INKEY$
1310 IF E$<"1" OR E$>"3" THEN 1300
1320 ON VAL(E$) GOTO 1010,400,1330
1330 CLS
1340 END
```

Educational Games

Another type of educational program is a game. The student can have fun while learning or reviewing concepts.

"Grid" is a game to practice the concept of coordinates learned in the Locating Points program. An object is hidden somewhere on the grid. The objective is to find the hidden point in as few guesses as possible. Guesses are made by specifying coordinates, first pressing the x coordinate, then the y coordinate. The point guessed is shown.

If the answer is not the position of the hidden point, there is an *uh-oh* sound and a hint is printed. An arrow on the point shows the direction of the hidden point.

The score of the number of guesses is shown in the upper right corner of the screen. After the student has successfully found the point, the option to try again is presented.

You can make a better theme by using different graphics—perhaps a wumpus in a hidden cave or a bomb to be detonated in a hotel or a black hole in space.

Printing the grid and the logic for printing the points are similar to the Locating Points program. To print the hint arrows, IF-THEN statements are used comparing the student's answer to the coordinates of the hidden point.

Program 11-6. Grid

```
10 REM GRID
40 C$="+":S$=" ":D$="*"
50 L$=C$+"---"
60 U$="|":U2$=U$+"   "
70 L1$=L$:U1$=U2$
80 FOR M=1 TO 9:L1$=L1$+L$:U1$=U1$+U2$:NEXT M
90 CLS
110 PRINT:PRINT TAB(26);"** GRID **"
120 PRINT:PRINT:PRINT
130 PRINT "   Find the hidden point on the grid."
140 PRINT:PRINT
150 PRINT "   Specify an x-coordinate then a y-coordinate."
160 PRINT:PRINT
170 PRINT "   If the point you chose is incorrect, you will
be given a hint."
180 PRINT:PRINT
190 PRINT "   Your score is shown in the upper right corner
of the screen."
210 PRINT:PRINT:PRINT
220 PRINT TAB(10);"PRESS THE SPACE BAR TO BEGIN."
230 RANDOMIZE TIMER
240 A$=INKEY$:IF A$<>" " THEN 240
250 CLS:COLOR 2:PRINT
260 FOR M=6 TO 1 STEP -1
270 PRINT TAB(5);RIGHT$(STR$(M),1);L1$
280 PRINT TAB(6);U1$:PRINT TAB(6);U1$
290 NEXT M
300 PRINT TAB(5);"0";L1$
310 PRINT TAB(6);"0   1   2   3   4   5   6   7   8   9"
320 COLOR 1
330 X=INT(10*RND)
340 Y=INT(7*RND)
350 X$=RIGHT$(STR$(X),1)
360 Y$=RIGHT$(STR$(Y),1)
370 ROW=20-3*Y:COL=6+4*X
380 LOCATE 1,65:PRINT "SCORE"
390 SC=0
400 SC=SC+1:IF SC>99 THEN 690
410 COLOR 1:LOCATE 2,67:PRINT SC
420 LOCATE 10,50:PRINT "(?,?)"
430 COLOR 3:LOCATE 10,51:PRINT "?"
450 E$=INKEY$:IF E$<"0" OR E$>"9" THEN 450
460 LOCATE 10,51
```

```
470 PRINT RIGHT$(E$,1)
490 LOCATE 10,53:PRINT "?"
500 F$=INKEY$:IF F$<"0" OR F$>"6" THEN 500
510 LOCATE 10,53
520 PRINT RIGHT$(F$,1)
530 E=VAL(E$):F=VAL(F$)
540 EX=6+4*E:FY=20-3*F:LOCATE FY,EX
550 PRINT D$
560 IF E=X AND F=Y THEN 650
570 BEEP
580 IF E=X THEN 610
590 IF E>X THEN H$=CHR$(60) ELSE H$=CHR$(62)
600 GOTO 630
610 IF F>Y THEN H$=CHR$(68) ELSE H$=CHR$(94)
630 LOCATE FY,EX:PRINT H$
640 GOTO 400
650 COLOR 1:LOCATE FY,EX:PRINT D$
660 SOUND 262,3:SOUND 330,3
670 SOUND 392,3:SOUND 523,6
680 GOTO 720
690 COLOR 1:LOCATE 15,50
700 PRINT "SORRY, YOU LOST."
710 PRINT TAB(50);"THE POINT IS AT (";X$;",";Y$;")."
720 COLOR 1:LOCATE 18,50
730 PRINT "PLAY AGAIN? (Y/N)"
740 A$=INKEY$:IF A$="Y" OR A$="y" THEN 90
750 IF A$<>"N" AND A$<>"n" THEN 740
760 CLS
770 END
```

Text Simulations

Simulations and creativity programs probably offer the best use of a computer in education, but these programs are more difficult to write in BASIC. However, it is possible to write simple simulation games in BASIC using text instead of complicated graphics sequences.

"Flight Schedule" fits in the category of a text simulation. The student may make choices and the program continues depending on those choices. The student starts at Seattle, Washington, but is given a choice of a destination in the eastern part of the United States. This program is designed to help students learn to read and interpret flight schedules.

Three airlines are listed with some of their destination cities. The student may select one of the airlines. The flights chosen must be direct flights. The portion of the schedule for a particular airline going to the destination is shown, and the student must answer comprehensive questions about the schedule. All questions have multiple-choice answers. Most of the answers include an explanation after the student's response.

The flight schedules for this program are modified excerpts from actual airlines, although the names of the airlines have been changed. All flights are theoretically possible.

This program is mainly composed of PRINT statements with branching. You can enhance the program by adding more destination cities, giving a choice of origination cities, and allowing more connecting flights.

The PRINT statements to display the schedules are in subroutines because they are printed several times. The subroutines use line labels rather than line numbers. Numbers are used, however, on some of the other lines that need referencing. E$ is always used for the student's answer, and INKEY$ is used rather than INPUT to prevent scrolling of the schedule.

Program 11-7. Flight Schedule

```
10  REM FLIGHT SCHEDULE
20  CLS:PRINT
    PRINT TAB(20);"** FLIGHT SCHEDULE **"
    PRINT:PRINT:PRINT
    PRINT "Your object is to fly from the west coast to the"
    PRINT:PRINT "east coast area.  Plan the trip."
       D1$(1)="New York City"
       D1$(2)="Greensboro, N.C."
    PRINT:PRINT:PRINT
    PRINT "Your originating city is SEATTLE, WASHINGTON."
    PRINT:PRINT:GOSUB PRESSKEY

30  CLS:PRINT:PRINT
    PRINT "You will leave from the Seattle/Tacoma Airport."
    PRINT:PRINT "What is your destination?":PRINT
    PRINT "  1  ";D1$(1)
    PRINT "  2  ";D1$(2)
    SOUND 1300,2
40     E$=INKEY$:IF E$<"1" OR E$>"2" THEN 40
    DD=VAL(E$)
    PRINT:PRINT
    PRINT "Your destination city is ";D1$(DD):PRINT:PRINT
    GOSUB PRESSKEY

50  GOSUB AIRLINES
    IF AL<>2 THEN 70
    PRINT:PRINT "Sorry, Beeline does not have direct flights
"
60  PRINT "to ";D1$(DD)
    GOSUB PRESSKEY:GOTO 50
70  IF DD=2 AND AL=3 THEN GOTO UNIVERSAL.G
    IF AL=1 AND DD=1 THEN GOTO AIRWEST.NY
    IF AL=3 AND DD=1 THEN GOTO UNIVERSAL.NY
    PRINT:PRINT "Sorry, Airwest does not have direct flights
":GOTO 60

REM Seattle to New York, Airwest
AIRWEST.NY:
    GOSUB AA1
```

```
    PRINT "What time do you want to leave?":PRINT
    PRINT "1   7:00 a.m.":PRINT "2   3:15 p.m."
80      E$=INKEY$:IF E$<>"1" AND E$<>"2" THEN 80
        E=VAL(E$)
    GOSUB AA2
    PRINT "If you leave at ";L$(E)","
    PRINT "what time will you arrive at New York?"
    PRINT "1   7:00 a.m.":PRINT "2   4:35 p.m."
    PRINT "3   3:30 p.m.":PRINT "4  12:35 a.m."
    PRINT "5   7:00 p.m."
90      E$=INKEY$:IF E$<"1" OR E$>"5" THEN 90
    IF E=1 AND E$="2" THEN PRINT:PRINT "YES.":GOTO 100
    IF E=2 AND E$="4" THEN PRINT:PRINT "YES.":GOTO 100
    PRINT:PRINT "NO."
100 GOSUB ARRIVE
    GOSUB AA2
    PRINT "Is your ";L$(E);" flight nonstop?  Y/N"
110 E$=INKEY$:IF E$="Y" OR E$="y" THEN 120
    IF E$<>"N" AND E$<>"n" THEN 110 ELSE PRINT "CORRECT."
120 PRINT:PRINT "The STOPS column indicates there is one sto
p."
    PRINT "Where is it?":PRINT
    PRINT "1  Houston":PRINT "2  Greensboro":PRINT "3  Denve
r"
    PRINT "4  Salt Lake City":PRINT "5  Chicago":PRINT
130 E$=INKEY$:IF E$<"1" OR E$>"5" THEN 130
    IF E$="4" THEN PRINT "CORRECT."
    PRINT "SLC stands for Salt Lake City."
    GOSUB PRESSKEY:GOSUB AA2
    PRINT "Meal symbols:"
    PRINT "B-Breakfast  L-Lunch  D-Dinner  S-Snack"
    PRINT "M-More than one meal appropriate to time."
    PRINT:PRINT "Do you get a meal on your ";L$(E);" flight?
"
    PRINT "  1  Yes, breakfast":PRINT "  2  Yes, lunch"
    PRINT "  3  Yes, dinner":PRINT "  4  Yes, more than one"
    PRINT "  5  No":PRINT
140 E$=INKEY$:IF E$<"1" OR E$>"5" THEN 140
    IF E=1 THEN 150
    IF E$="3" THEN 160
    PRINT "The MEAL column indicates 'D' meaning dinner.":GO
TO 170
150 IF E$="4" THEN 160
    PRINT "The MEAL column indicates 'M' meaning more than o
ne meal."
    GOTO 170
160 PRINT "CORRECT.":PRINT
170 GOSUB PRESSKEY:GOSUB AA2
    PRINT "You chose to leave at ";L$(E);"."
    PRINT "What is the flight number of your first flight?"
    PRINT "  A  353":PRINT "  B  566"
    PRINT "  C  720":PRINT "  D  700"
    PRINT "  E  435":PRINT
180 E$=INKEY$:IF E$<"A" OR (E$>"E" AND E$<"a") OR E$>"e" THE
N 180
    IF E=1 AND (E$="A" OR E$="a") THEN PRINT "Correct; Fligh
t No. 353":GOTO 190
    IF E=2 AND (E$="C" OR E$="c") THEN PRINT "Correct; Fligh
t No. 720":GOTO 190
    PRINT "The first number in the FLIGHT column is the flig
ht"
```

```
    PRINT "number for the first part of the trip."
190 GOSUB PRESSKEY:GOTO PLANE

REM Seattle-Greensboro, Universal
UNIVERSAL.G:
    GOSUB UA1
    PRINT "What time do you want to leave?":PRINT
    FOR T=1 TO 4:PRINT T;SPC(3);L$(T):NEXT T
200 E$=INKEY$:IF E$<"1" OR E$>"4" THEN 200
    E=VAL(E$):GOSUB UA2
    PRINT "If you leave at ";L$(E);", what time will you arr
ive"
    PRINT "at Greensboro?":PRINT
    PRINT "A   6:35 p.m.":PRINT "B  10:36 p.m.":PRINT "C   9
:51 a.m."
    PRINT "D  10:35 a.m.":PRINT "E   1:40 p.m.":PRINT
210 E$=INKEY$:IF E$<"A" OR (E$>"E" AND E$<"a") OR E$>"e" THE
N 210
       IF E=1 AND (E$="A" OR E$="a") THEN PRINT "YES."
       IF E=2 AND (E$="B" OR E$="b") THEN PRINT "YES."
       IF E=3 AND (E$="B" OR E$="b") THEN PRINT "YES."
       IF E=4 AND (E$="C" OR E$="c") THEN PRINT "YES."
    GOSUB ARRIVE:GOSUB UA2
    PRINT "If you wanted to arrive at Greensboro and wait th
e"
    PRINT "minimum time for a noon meeting, which would be t
he best?"
    PRINT "Leave at:"
    FOR T=1 TO 4:PRINT T;SPC(3);L$(T):NEXT T
220 E$=INKEY$:IF E$<"1" OR E$>"4" THEN 220
    IF E$="4" THEN PRINT:PRINT "YES."
    PRINT:PRINT "The 12:20 a.m. flight arrives before noon."
    GOSUB PRESSKEY:GOSUB UA2
    PRINT "The VIA column indicates you have a connecting fl
ight."
    PRINT "Where do you stop?"
    PRINT "  1  Las Vegas":PRINT "  2  Salt Lake City"
    PRINT "  3  Denver":PRINT "  4  Chicago"
    PRINT "  5  Cincinnati":PRINT "  6  Houston"
230 E$=INKEY$:IF E$<"1" OR E$>"6" THEN 230
    IF E$="4" THEN PRINT:PRINT "CORRECT."
    PRINT:PRINT "CHI stands for Chicago."
    GOSUB PRESSKEY:GOSUB UA2
    PRINT "You chose the ";L$(E);" flight."
    PRINT "What is the beginning flight number?"
    PRINT " A  140":PRINT " B  144":PRINT " C  150"
    PRINT " D  902":PRINT " E  745":PRINT " F  408"
240 E$=INKEY$:IF E$<"A" OR (E$>"F" AND E$<"a") OR E$>"f" THE
N 240
    IF E=ASC(E$)-64 OR E=ASC(E$)-96 THEN PRINT:PRINT "CORREC
T."
    PRINT:PRINT "The first number in the FLIGHT column is ";
F$(E)"."
    GOSUB PRESSKEY:GOSUB UA2
    PRINT "Two of the flights actually combine in Chicago."
    PRINT "What is their final flight number?"
    PRINT " A  884":PRINT " B  144":PRINT " C  492"
    PRINT " D  150":PRINT " E  217":PRINT " F  951"
250 E$=INKEY$:IF E$<"A" OR (E$>"F" AND E$<"a") OR E$>"f" THE
N 250
    IF E$="C" OR E$="c" THEN PRINT:PRINT "CORRECT."
```

```
    PRINT:PRINT "Flights 144 and 150 both join Flight 492."
    GOSUB PRESSKEY:GOSUB UA2
    GOSUB SERVICE
    GOTO PLANE

UNIVERSAL.NY:
    GOSUB UA3
    PRINT "What time do you want to leave?":PRINT
    FOR T=1 TO 5:PRINT T;SPC(3);L$(T):NEXT T
260 E$=INKEY$:IF E$<"1" OR E$>"5" THEN 260
    E=VAL(E$)
    GOSUB UA4
    PRINT "If you leave at ";L$(E);", what time will you arr
ive in New York?"
    FOR T=1 TO 5:PRINT CHR$(64+T);SPC(3);A$(T):NEXT T
270 E$=INKEY$:IF E$<"A" OR (E$>"E" AND E$<"a") OR E$>"e" THE
N 270
    IF E=ASC(E$)-64 OR E=ASC(E$)-96 THEN PRINT:PRINT "CORREC
T."
    PRINT:PRINT "The second column has the corresponding arr
ival time."
    PRINT "It is ";A$(E)
    GOSUB PRESSKEY:GOSUB UA4
    PRINT "You chose the ";L$(E);" flight."
    PRINT "Where will you land?":PRINT
    PRINT "  1  Newark Airport"
    PRINT "  2  Kennedy Airport":PRINT
280 E$=INKEY$:IF E$<"1" OR E$>"2" THEN 280
    IF (E=3 OR E=5) AND E$="2" THEN PRINT "Yes, ";:GOTO 290
    IF E$="2" THEN PRINT "No, ";:GOTO 290
    IF E=3 OR E=5 THEN PRINT "No, ";:GOTO 290
    PRINT "Yes, ";
290 PRINT "E indicates Newark and J indicates Kennedy."
    GOSUB PRESSKEY:GOSUB UA4
    PRINT "You chose the ";L$(E);" flight."
    PRINT "What is your beginning flight number?"
    FOR T=1 TO 5:PRINT CHR$(64+T);"  ";FL(T):NEXT T
300 E$=INKEY$:IF E$<"A" OR (E$>"E" AND E$<"a") OR E$>"e" THE
N 300
    IF E=ASC(E$)-64 OR E=ASC(E$)-96 THEN PRINT:PRINT "CORREC
T."
    PRINT "Your flight number is";FL(E);"."
    GOSUB PRESSKEY:GOSUB UA4
    PRINT:PRINT "If you wanted a non-stop flight and wanted
to"
    RANDOMIZE TIMER:A=INT(2*RND+1)
    W$(1)="before":W$(2)="after"
    PRINT "leave ";W$(A);" noon, what would be your flight n
umber?"
    FOR T=1 TO 5:PRINT CHR$(64+T);"  ";FL(T):NEXT T
310 E$=INKEY$:IF E$<"A" OR (E$>"E" AND E$<"a") OR E$>"e" THE
N 310
    IF A=1 AND (E$="C" OR E$="c") THEN PRINT:PRINT "CORRECT.
"
    IF A=2 AND (E$="E" OR E$="e") THEN PRINT:PRINT "CORRECT.
"
    PRINT:PRINT "0 indicates no stops.  Flight 40 leaves bef
ore"
    PRINT "noon, and Flight 18 after."
    GOSUB PRESSKEY:GOSUB UA4
```

```
    PRINT:PRINT "If you want to meet a friend at the Staplet
on"
    PRINT "International Airport in Denver, what time do you
 leave Seattle?"
    FOR T=1 TO 5:PRINT CHR$(64+T);"  ";L$(T):NEXT T
320 E$=INKEY$:IF E$<"A" OR (E$>"E" AND E$<"a") OR E$>"e" THE
N 320
    IF E$="A" OR E$="a" THEN PRINT:PRINT "CORRECT."
    PRINT:PRINT "The last column indicates the first flight
stops in Denver."
    GOSUB PRESSKEY:GOSUB UA4
    PRINT "You chose the ";L$(E);" flight."
    PRINT "Does it make any stops?"
    PRINT:PRINT "1  Yes, one in Denver."
    PRINT "2  Yes, one in Chicago."
    PRINT "3  Yes, in Denver and in Chicago."
    PRINT "4  Yes, in Salt Lake City."
    PRINT "5  No, it is non-stop.":PRINT
330 E$=INKEY$:IF E$<"1" OR E$>"5" THEN 330
    IF E=1 AND E$="1" THEN PRINT "CORRECT.":GOTO 340
    IF E=1 THEN PRINT "NO.":GOTO 340
    IF E=3 AND E$="5" THEN PRINT "CORRECT.":GOTO 340
    IF E=5 AND E$="5" THEN PRINT "CORRECT.":GOTO 340
    IF E=3 OR E=5 THEN PRINT "NO.":GOTO 340
    IF E$="2" THEN PRINT "CORRECT.":GOTO 340
    PRINT "NO."
340 PRINT "The last column indicates 0 for nonstop,"
    PRINT "or a city abbreviation for a stop."
    GOSUB PRESSKEY:GOSUB UA4
    GOSUB SERVICE
    GOTO PLANE
    STOP

PRESSKEY:
    PRINT:PRINT "Press <RETURN>";
2   RE$=INKEY$:IF RE$="" THEN GOTO 2
    IF RE$<>CHR$(13) THEN 2
    RETURN

AIRLINES:
    CLS:PRINT "Seattle/Tacoma"
    PRINT:PRINT "** AIRWEST AIRLINES **";TAB(45);"** BEELINE
 **"
    PRINT "   To Chicago";TAB(48);"To Honolulu"
    PRINT "   To Dallas/Ft. Worth";TAB(48);"To Las Vegas, Ne
v."
    PRINT "   To Denver":PRINT "   To Houston"
    PRINT "   To Las Vegas, Nev.":PRINT "   To New York"
    PRINT "   To Washington, D.C."
    PRINT:PRINT "** UNIVERSAL AIRWAYS **"
    PRINT "   To Chicago":PRINT "   To Dallas/Ft. Worth"
    PRINT "   To Greensboro/High Point/Winston-Salem"
    PRINT "   To Miami, Fl.":PRINT "   To New York/Newark"
    PRINT "   To Toronto, Ont."
    PRINT:PRINT "CHOOSE AN AIRLINES:  A, B, OR U";
3   E$=INKEY$:IF E$="" THEN 3
    IF E$<>"A" AND E$<>"a" AND E$<>"B" AND E$<>"b" AND E$<>"
U" AND E$<>"u" THEN 3
    IF E$="A" OR E$="a" THEN AL=1
    IF E$="B" OR E$="b" THEN AL=2
    IF E$="U" OR E$="u" THEN AL=3
    CLS:RETURN
```

```
ARRIVE:
    PRINT:PRINT "The first column is when you leave."
    PRINT "The second column is when you arrive."
    PRINT "You will arrive at ";A$(E)
    GOSUB PRESSKEY
    RETURN

SERVICE:
    PRINT "SERVICE:  X Meal";TAB(30);"S Snacks";TAB(50);"% C
ocktails"
    PRINT TAB(11);"@ Movie";TAB(30);"a Audio";TAB(50);"& Sho
rt subject"
    IF Q=2 THEN 6
    PRINT:PRINT "Will you get a meal on your flight?"
    PRINT "  Y--Yes     N--No"
4   E$=INKEY$:IF E$="Y" OR E$="y" THEN PRINT:PRINT "CORRECT.
":GOTO 5
    IF E$<>"N" AND E$<>"n" THEN 4
5   PRINT:PRINT "The X symbol indicates a meal.":GOSUB PRESS
KEY
6   RETURN

AA1:
    L$(1)="7:00 a.m.":L$(2)="3:15 p.m."
    A$(1)="4:35 p.m.":A$(2)="12:35 a.m."
AA2:
    CLS:PRINT "** AIRWEST AIRLINES **":PRINT
    PRINT "LEAVE      ARRIVE       FLIGHT      STOPS    CNCT
  MEAL"
    PRINT "Seattle/Tacoma to New York K-Kennedy":PRINT
    PRINT "7:00a       4:35p K     353/566      1      SLC
    M"
    PRINT "3:15p      12:35a K     720/700      1      SLC
    D"
    PRINT:PRINT:PRINT
    RETURN

UA1:
    L$(1)="7:45 a.m.":L$(2)="10:35 a.m.":L$(3)="1:10 p.m.":L
$(4)="12:20 a.m."
    A$(1)="6:35 p.m.":A$(2)="10:36 p.m.":A$(3)=A$(2):A$(4)="
9:51 a.m."
    F$(1)="140":F$(2)="144":F$(3)="150":F$(4)="902"
    F2$(1)="884":F2$(2)="782":F2$(3)="492":F2$(4)="644"

UA2:
    CLS:PRINT "** UNIVERSAL AIRLINES **":PRINT
    PRINT " LEAVE     ARRIVE     FLIGHT      SERVICE      VIA
"
    PRINT "Seattle/Tacoma"
    PRINT "   TO Greensboro/High Point/Winston-Salem":PRINT
    PRINT " 7:45a      6:35p     140/884     o @ % X      Chi
"
    PRINT "10:35a     10:36p     144/492     o @ % X      Chi
"
    PRINT " 1:10p     10:36p     150/492     * @ % X      Chi
"
    PRINT "12:20a      9:51a     902/217     # @ % X      Chi
"
    PRINT:PRINT
    RETURN
```

```
UA3:
    L$(1)="7:00a.m.":L$(2)="7:45 a.m.":L$(3)="8:05 a.m."
    L$(4)="10:35 a.m.":L$(5)="1:00 p.m.":A$(1)="4:52 p.m."
    A$(2)="5:20 p.m.":A$(3)="4:00 p.m.":A$(4)="8:12 p.m."
    A$(5)="8:55 p.m.":FL(1)=752:FL(2)=140:FL(3)=40
    FL(4)=144:FL(5)=18

UA4:
    CLS:PRINT "** UNIVERSAL AIRLINES **":PRINT
    PRINT " LEAVE     ARRIVE       FLIGHT       SERVICE      V
IA"
    PRINT "Seattle/Tacoma"
    PRINT "  TO New York, N.Y.   E=Newark   J=Kennedy"
    PRINT " 7:00a     4:52p(E)     752/694     o @ % X      D
en"
    PRINT " 7:45a     5:20p(E)     140/122     o @ % X      C
hi"
    PRINT " 8:05a     4:00p(J)      40         # @ % X
0 "
    PRINT "10:35a     8:12p(E)     144/104     > @ % X      C
hi"
    PRINT " 1:00p     8:55p(J)      18         o @ % X
0
    PRINT
    RETURN

PLANE: CLS
    PRINT "HAVE A NICE FLIGHT!"
    RESTORE PICTURE
    PICTURE:
       DATA 212,88,480,136,6
       DATA 528,160,528,164,520,166
       DATA 484,164,416,152,260,126
       DATA 396,118,420,66,2
       DATA 408,63,346,109
       DATA 356,142,156,174,2
       DATA 136,170,296,131
       DATA 196,107,156,88,7
       DATA 156,85,246,100,228,96
       DATA 156,57,128,53,112,52,164,86
       DATA 164,61,170,40,2
       DATA 160,36,128,53
       DATA 134,68,80,74,3
       DATA 64,74,56,72,120,56
       DATA 316,98,310,100,5
       DATA 310,103,318,102,316,98
       DATA 252,88,250,94
       DATA 256,121,260,122,12
       DATA 260,126,250,127,250,126
       DATA 252,123,256,121,224,112
       DATA 188,106,188,110,186,112
       DATA 180,112,220,123,254,127
    FOR J=1 TO 8
       READ X1,Y1,X2,Y2,N
       LINE (X1,Y1)-(X2,Y2)
       FOR T=1 TO N
          READ X,Y:LINE -(X,Y)
       NEXT T
    NEXT J
    LOCATE 22,1
END
```

Chapter 12

Miscellaneous Techniques

Chapter 12

Miscellaneous Techniques

Sorting

One of the functions of a computer is to organize data, and there are many sort routines which take your raw data and arrange it in ascending or descending order. For example, you may want to sort a list of people by birth date, or you may want to alphabetize a list of names. Here are four BASIC algorithms for sorting (Programs 12-1 through 12-4).

In these examples, 50 random numbers are printed, sorted in ascending order, and then printed. The array A is DIMensioned in line 20 for the 50 numbers. WIDTH 77 is used to print the numbers on the screen. Line 30 uses RANDOMIZE TIMER to randomize the numbers, and line 40 chooses and prints the 50 random numbers.

SWAP is used to switch numbers. SWAP *a,b* will put the value of number *a* into *b*, and the value that was in *b* will become *a*.

These algorithms sort in ascending order. If you need descending order, change the less-than (<) signs to greater-than (>) signs. If you need to use strings rather than numbers, put dollar signs on all the variable names that contain items to be sorted—use A$ instead of A.

Bubble Sort

The bubble sort, or simple interchange sort, is commonly used because it is easy to understand. The program compares each number to the next number and exchanges numbers where necessary. If even one switch has been made during a pass through all the numbers, the loop of comparisons starts over. The number of passes through the loop depends on how many items were out of order. This sort is better for lists that are not much out of order or that haven't very many items. It can be quite slow for long lists of very mixed up items.

Program 12-1. The Bubble Sort

```
10 REM BUBBLE SORT
20 DIM A(50):WIDTH 77
30 RANDOMIZE TIMER
40 FOR I=1 TO 50:A(I)=INT(100*RND):PRINT A(I);:NEXT I:PRINT:
PRINT
50 L=49
60 S=0:FOR I=1 TO L:IF A(I)<=A(I+1) THEN 80
70 SWAP A(I),A(I+1):S=1:L=I
80  NEXT I
90  IF S=1 THEN 60
100 FOR I=1 TO 50:PRINT A(I);:NEXT I
110 END
```

Shell Sort

The shell sort is considerably faster than the bubble sort because the number of comparisons that need to be made is reduced. In an array of *N* numbers, first determine B so that 2 to the B power is less than *N*, and *N* is less than 2 to the B+1 power. Then initialize B as 2 to the B−1 power. The loop varies counter I from 1 to *N*−B. First, check if A(I) is less than or equal to A(I+B). If so, increment I and continue comparisons. If not, exchange A(I) and A(I+B) and change the subscript. When I reaches the value of *N*, reduce B by a factor of 2 and start the loop again. When B equals 0, the sort is complete.

Program 12-2. The Shell Sort

```
10 REM SHELL SORT
20 DIM A(50):WIDTH 77
30 RANDOMIZE TIMER
40 FOR I=1 TO 50:A(I)=INT(100*RND):PRINT A(I);:NEXT I:PRINT:
PRINT
50 B=1
60 B=2*B:IF B<=50 THEN 60
70 B=INT(B/2):IF B=0 THEN 120
80 FOR I=1 TO 50-B:C=I
90 D=C+B:IF A(C)<=A(D) THEN 110
100 SWAP A(C),A(D):C=C-B:IF C>0 THEN 90
110 NEXT I:GOTO 70
120 FOR I=1 TO 50:PRINT A(I);:NEXT
130 END
```

Maximum and Minimum Sort

The maximum and minimum sort passes through all the numbers and puts the smallest number at one end and the largest number at the other end. Each pass finds the next smallest and next largest numbers and puts them toward the appro-

priate ends. The numbers are filled in from the ends toward the center in the array of sorted numbers.

Program 12-3. Maximum and Minimum Sort

```
10 REM MAXIMUM AND MINIMUM SORT
20 DIM A(50):WIDTH 77
30 RANDOMIZE TIMER
40 FOR I=1 TO 50:A(I)=INT(100*RND):PRINT A(I);:NEXT I:PRINT:
PRINT
50 N=50:S=1
60 L=A(S):J=S:U=L:K=S
70 FOR I=S TO N
80 IF A(I)>U THEN U=A(I):K=I
90 IF A(I)<L THEN L=A(I):J=I
100 NEXT I
110 IF J=N THEN J=K
120 SWAP A(N),A(K):N=N-1
130 SWAP A(S),A(J):S=S+1
140 IF N>S THEN 60
150 FOR I=1 TO 50:PRINT A(I);:NEXT
160 END
```

Quick Sort

The quick sort has become popular because it is one of the fastest sorting procedures in BASIC. I have translated it for Amiga BASIC here.

Program 12-4. Quick Sort

```
10 REM QUICK SORT
20 DIM A(50):WIDTH 77
30 RANDOMIZE TIMER
40 FOR I=1 TO 50:A(I)=INT(100*RND):PRINT A(I);:NEXT I:PRINT:
PRINT
50 N=50:S(1)=1:S(2)=N:T=1
60 IF T=0 THEN 180
70 T=T-1:I=2*T:L=S(I+1):M=S(I+2):X=A(L):J=L:K=M+1
80 K=K-1:IF K=J THEN 140
90 IF X<=A(K) THEN 80
100 A(J)=A(K)
110 J=J+1:IF K=J THEN 140
120 IF X>=A(J) THEN 110
130 A(K)=A(J):GOTO 80
140 A(J)=X:IF M-J<2 THEN 160
150 I=2*T:S(I+1)=J+1:S(I+2)=M:T=T+1
160 IF K-L<2 THEN 60
170 I=2*T:S(I+1)=L:S(I+2)=K-1:T=T+1:GOTO 60
180 FOR I=1 TO N:PRINT A(I);:NEXT I
190 END
```

Dealing Cards

How do you choose random items without repetition? One way is to have all the items in an array and set a flag when an element is chosen. For example, in the "States and Capitals" program in Chapter 14, the S$ array contains the names of the states. When a state has been named correctly, the particular S$ element is set to the null string, " ", so that it won't be chosen again.

This next sample program, Program 12-5, illustrates how you can simulate dealing cards from a deck without replacing cards. I've just printed the number or name of the card with its suit, but in an actual game you would use graphics.

Line 20 DIMensions an array C(13,4) to hold cards for the 13 possible numbers (1 through 10 plus jack, queen, and king) and the four possible suits. Lines 30–40 define S$ strings for the names of the four suits. Line 50 clears the screen and then starts the loop for dealing five cards.

Line 60 uses RANDOMIZE TIMER to make sure different numbers are chosen each time the program is run. Line 70 chooses a random NUMBER from 1 through 13. Line 80 chooses a random SUIT from 1 through 4. Line 90 makes sure the card hasn't been chosen previously. Originally, all the C elements will be 0. Line 150 sets the card chosen to 1 so that it cannot be chosen again. For example, if NUMBER is 5 and SUIT is 3, C(5,3) is the card chosen. Lines 100–140 print the card, and line 160 goes to the next card.

Lines 170–200 print the options. The first option will deal five more cards from the same deck; that is, cards are not replaced. Option 2 replaces the five cards and deals from a whole new deck.

Lines 240–250 keep track of how many cards have been dealt if cards are not replaced. After 50 cards, a new deck is needed. Lines 280–320 reset the C array so that all values are zero and all cards may then be chosen.

Program 12-5. Dealing Cards

```
10 REM CARDS
20 DIM C(13,4)
30 S$(1)="HEART":S$(2)="CLUB"
40 S$(3)="DIAMOND":S$(4)="SPADE"
50 CLS:FOR DEAL=1 TO 5
60 RANDOMIZE TIMER
70 NUMBER=INT(13*RND)+1
80 SUIT=INT(4*RND)+1
```

```
90 IF C(NUMBER,SUIT)=1 THEN 70
100 PRINT
110 IF NUMBER=11 THEN PRINT "JACK";:GOTO 140
120 IF NUMBER=12 THEN PRINT "QUEEN";:GOTO 140
130 IF NUMBER=13 THEN PRINT "KING"; ELSE PRINT NUMBER;
140 PRINT TAB(10);S$(SUIT)
150 C(NUMBER,SUIT)=1
160 NEXT DEAL
170 PRINT:PRINT
180 PRINT "PRESS 1  DEAL FIVE MORE"
190 PRINT "       2  DEAL FROM FULL DECK"
200 PRINT "       3  END PROGRAM"
210 K$=INKEY$
220 IF K$<"1" OR K$>"3" THEN 210
230 ON VAL(K$) GOTO 240,280,340
240 T=T+1
250 IF T<10 THEN 50
260 PRINT:PRINT "OUT OF CARDS; STARTING OVER"
270 FOR DELAY=1 TO 2000:NEXT DELAY
280 FOR J=1 TO 13
290 FOR K=1 TO 4
300 C(J,K)=0
310 NEXT K
320 NEXT J
330 GOTO 50
340 END
```

Using a Timing Device

You may wish to use a timing device in a program to time how long it takes to respond to a question or to perform an act for a certain amount of time in a game. TIMER is similar to a clock and can return a number that can be used as a number of seconds. For example, go to the Output screen and type PRINT TIMER and press RETURN. A number is printed, such as 43525. Wait a few seconds and PRINT TIMER again. This time the number might be 43537. The difference between the two numbers is the amount of time that elapsed between the two commands. In this case, the difference is 43537 − 43525 = 12. This number is in seconds.

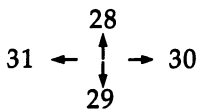
Here is a short program (Program 12-6) that illustrates the use of TIMER. The program will time how long it takes you to type in a message. Line 50 will BEEP to signal the start of the timing. Line 60 sets the variable T1 to TIMER. Line 70 is INPUT to receive your typing. When you press RETURN, line 80 sets the variable T2 to the new value of TIMER. Line 100 prints the length of time, which is the difference between T2 and T1.

Program 12-6. Timing

```
10 REM TIMING
20 PRINT "TYPE IN A MESSAGE THEN PRESS <RETURN>."
30 PRINT "START AT THE TONE."
40 FOR DELAY=1 TO 2000:NEXT DELAY
50 BEEP
60 T1=TIMER
70 INPUT MSG$
80 T2=TIMER
90 PRINT:PRINT
100 PRINT "THE TIME WAS";T2-T1;"SECONDS."
110 END
```

Arrow Keys

There are several ways you can use the arrow keys in a program. One way is to use INKEY$ to scan the keyboard and see if a key is pressed. The ASCII codes of the arrow keys are



You can use IF-THEN statements to test the ASCII value of the keys to see which direction is pressed. "Stepping," Program 10-2, in Chapter 10 uses the arrow keys to receive the answers.

The following program illustrates a use of the arrow keys. "Doodler" is a drawing program. A point starts at the middle of the screen with x and y coordinates 310 and 90. PSET draws the point on the screen (line 30). Line 40 initializes the color C.

Line 50 uses K$=INKEY$ to see if a key is pressed. Lines 60–80 check whether the key pressed is the space bar, and if so, the color number is changed. The color may be from 0 through 4.

Lines 90–120 check to see whether the arrow keys have been pressed. If an arrow key is pressed, the appropriate x or y coordinate is changed. Lines 130–160 check the border conditions for the point. If it is at an edge, the point will "wrap" to the opposite edge. Line 170 uses PSET to draw the new point. Line 180 returns to line 50 for the next keypress.

Program 12-7. Doodler

```
10 REM DOODLE
12 CLS
14 PRINT "** DOODLE **"
```

```
16 PRINT:PRINT "Use the arrow keys to move."
18 PRINT:PRINT "Press the space bar to change colors."
20 PRINT:PRINT:PRINT
22 PRINT "PRESS <RETURN> TO START DRAWING."
24 E$=INKEY$
26 IF E$<>CHR$(13) THEN 24
28 CLS
30 X=310:Y=90:PSET (X,Y)
40 C=0
50 K$=INKEY$:IF K$="" THEN 50
60 IF K$<>" " THEN 90
70 C=C+1:IF C>3 THEN C=0
80 COLOR C:GOTO 170
90 IF ASC(K$)=31 THEN X=X-1:GOTO 130
100 IF ASC(K$)=30 THEN X=X+1:GOTO 130
110 IF ASC(K$)=28 THEN Y=Y-1:GOTO 130
120 IF ASC(K$)=29 THEN Y=Y+1 ELSE 50
130 IF X<0 THEN X=615
140 IF X>615 THEN X=0
150 IF Y<0 THEN Y=185
160 IF Y>185 THEN Y=0
170 PSET (X,Y)
180 GOTO 50
190 END
```

Spelling Practice

School children all over the nation seem to have weekly spelling tests. Drill, practice, repetition—another use for a computer. Program 12-8, "Spelling Flash Cards," is designed to computerize spelling flash cards.

It is designed for up to 30 words, but if you have more, change the DIM statement in line 20 and the limit in line 150.

Put your own spelling words in the DATA statements in lines 630–680, making sure that the words are separated by commas. After the last spelling word, put a comma and the @ symbol (typed with SHIFT-2). Save the program, then you can use it all week.

The program presents the words in a random order. A word is printed on the screen momentarily, then erased. The student must type the word and press RETURN. To see the word again, press the space bar. If the word is missed twice, the word is shown and will then appear again later in the quiz. The quiz continues until all the words have been spelled correctly. Asterisks at the bottom of the screen indicate correctly spelled words.

If you want to have the word flashed on the screen a different length of time, change the limit in line 300. You may want to add speech to have the computer say the word rather

than flash it on the screen. Speech does take time to experiment on the pronunciations.

Outline of Spelling Flash Cards Program

Lines	Explanation
20	DIMensions W$ for 30 words.
30–120	Print title and instructions.
130–150	READ in spelling words from DATA. The last data item must be @.
160	Initializes the number of words, N, and column, C, for printing the asterisks for correct words.
190	Waits for student to press the space bar to start.
230	Clears screen.
240	Performs quiz for N number of words.
250	Initializes T for times word is missed.
260	Randomly chooses a word that has not previously been spelled correctly.
270	Clears previous printing.
280–310	Print word, delay, then clear word.
320–410	Receive student's spelling.
420–510	If word is incorrect, go back for another try; if word is missed twice, print word, then wait for student to press RETURN.
520–570	If word is correct, print an asterisk, play an arpeggio, set W$ to " " so that it cannot be chosen again, and go to next word.
580–620	Clear screen; play tune of random notes.
630–680	DATA containing spelling words.
690	Ends.

Program 12-8. Spelling Flash Cards

```
10 REM SPELLING PRACTICE
20 DIM W$(30)
30 CLS
40 PRINT TAB(18);"**********************"
50 PRINT TAB(18);"* SPELLING PRACTICE *"
60 PRINT TAB(18);"**********************"
70 PRINT:PRINT
80 PRINT "You will see a spelling word flash on the screen."
90 PRINT
100 PRINT "When it disappears you type the word then press <
RETURN>."
110 PRINT
120 PRINT "If you need to see the word again, press the spac
e bar."
130 T=1:RESTORE
140 READ W$(T):IF W$(T)="@" THEN 160
```

```
150 T=T+1:IF T<31 THEN 140
160 N=T-1:C=0
170 PRINT:PRINT:PRINT
180 PRINT "PRESS THE SPACE BAR TO START."
190 S$=INKEY$:IF S$<>" " THEN 190
230 CLS:RANDOMIZE TIMER
240 FOR P=1 TO N
250 T=0
260 R=INT(N*RND+1):IF W$(R)="" THEN 260
270 LOCATE 5,6:PRINT SPACE$(20)
280 BEEP:LOCATE 5,6:PRINT W$(R)
290 L=LEN(W$(R))
300 FOR DELAY=1 TO 3000:NEXT DELAY
310 LOCATE 5,6:PRINT SPACE$(20)
320 LOCATE 5,4:PRINT "> ";
330 B$=""
340 FOR J=1 TO L+5
350 E$=INKEY$:IF E$="" THEN 350
360 IF ASC(E$)=13 THEN 420
370 IF E$=" " THEN 270
380 E$=UCASE$(E$)
390 IF E$<"A" OR E$>"Z" THEN 350
400 PRINT E$;:B$=B$+E$
410 NEXT J
420 IF B$=W$(R) THEN 520
430 SOUND 330,2:SOUND 262,2
440 T=T+1:IF T<2 THEN 270
450 PRINT:PRINT TAB(6);W$(R)
460 PRINT:PRINT "PRESS <RETURN> TO CONTINUE."
470 E$=INKEY$
480 IF E$<>CHR$(13) THEN 470
490 LOCATE 6,6:PRINT SPACE$(20)
500 PRINT:PRINT SPACE$(27)
510 GOTO 250
520 C=C+1:LOCATE 22,C*2+10
530 PRINT "*"
540 SOUND 262,2:SOUND 330,2
550 SOUND 392,2:SOUND 523,6
560 W$(R)=""
570 NEXT P
580 CLS
590 FOR T=1 TO 30
600 SOUND 500+500*RND,2
610 NEXT T
620 GOTO 690
630 DATA BEAUTIFUL,FIR,SKIRT,CIRCLE
640 DATA SQUIRREL,DOCTOR,BEYOND,CLOSET
650 DATA CONNECT,CONCERN,COSTUME,PROMISE
660 DATA PRODUCTS,PROBABLY,POPULAR
670 DATA HORIZONTAL,MUSICIAN
680 DATA ELECTRICITY,ADDITION,@
690 END
```

Chapter 13

Debugging

Chapter 13

Debugging

It can be frustrating to spend hours typing in a program, then have it not run properly. Before you heave the keyboard at the monitor or call yourself or the author all sorts of names, here are some suggestions on how to diagnose and solve programming problems.

Syntax Errors

The easiest problems to correct are syntax errors. When you run the program, it will stop if there is a syntax error. The List window will appear with the program listing. The line containing the error will be outlined in orange, and an error message will appear at the top of the screen. Press the mouse arrow at the OK box by the error message, then click the cursor at the offending line to correct it.

The most likely syntax error is a mistyped BASIC word or another symbol where there should be a colon, semicolon, or comma. Make sure parentheses are in the right places and matched. Quotation marks may also need to be checked. Check also that a zero has not been mistaken for the letter *O* or vice versa. I avoid using the letter *O* as a variable name. You might also watch out for the letter *I* and the number 1 and the letter *B* and the number 8. FOR statements must be matched with NEXT statements.

Line Number Errors

It's a little more difficult to pinpoint errors where the program seems to be running improperly. Refer to the line-by-line explanation to try to pinpoint the lines that could be causing the problem. LIST those lines to check for typing errors.

Check line numbers in branching statements such as GOTO, ON-GOTO, ON-GOSUB, GOSUB, and IF-THEN statements. Be sure to type the line numbers exactly as they are shown in the listing. One little number can cause the computer to branch to a wrong statement and thus act strangely.

Check Variable Values

Anytime the computer stops with an error, you can print out values for variables to see what they are at that point. For example, if you get an error message and the statement is LOCATE R,C, you can type PRINT R,C in the Output window and press RETURN to see the present values for those variables. If they are incorrect, you can refer to previous lines to see how those variables were calculated. See where the variables were defined, then follow through the logic to see where they could have been altered incorrectly.

Watch Those DATA Statements

Judging from my mail, the most common place for errors to occur in a program is in DATA statements, especially if there are lots of items with commas. Even if the error message refers to a different line, the DATA statement may be the real cause for the error. Although DATA statements can be much longer, I tried to keep them short, so they would be easier to type in.

You need to compare the corresponding READ statement with the DATA statements to make sure items are read in the right order. The error could be in a previous DATA statement if data has been read in previously.

If there is a RESTORE statement, make sure the line number is typed correctly. In DATA statements, type the numbers carefully, making sure all commas are exactly as shown in the listing. Do not end a DATA statement with a comma. It is possible that DATA lists contain commas with nothing between them. This indicates that the value read would be the null string, or " ". If you have several commas in the list, be sure you have the right number of commas.

Keep in mind that when the program stops with an error you can print the values of any variables. If you are reading data in a loop, you can print the loop counter to see how far into the DATA statements you have successfully read. You can also print the variable(s) being read to see what the last acceptable value was, then use that information to pinpoint a typing error in a DATA statement.

The listings in this book were taken directly from the computer to try to avoid typesetting errors. Although we hope the listings are error-free, the possibility of errors does exist.

After you have checked and rechecked and still haven't found the problem, you may be tempted to write the author for help. If you do write, please be specific about the type of error that occurred, and be sure to specify the program title, the name of the book, and what type of computer you are using.

Chapter 14

Sample Programs

Chapter 14

Sample Programs

You could use a calculator to calculate a mathematical formula, and it would probably be faster for one problem than the computer. However, if you have to solve many problems using the same formula, the computer can simplify the task. Program 14-1 illustrates that concept by using the formula to find a monthly payment when you borrow a certain amount of money.

The user enters an amount borrowed, the number of years for the loan, and the yearly interest rate. Prompts are given for these numbers. The computer then calculates and prints what the monthly payment would be.

After each INPUT statement, the number entered is checked to see whether it is within reasonable bounds for the formula. The formula is calculated in line 240 and is rounded to the nearest cent in line 250.

Program 14-1. Loan Payments

```
10 REM LOAN PAYMENTS
20 CLS
30 PRINT "** LOAN PAYMENTS **"
40 PRINT
50 PRINT "YOU WANT TO BORROW A CERTAIN"
60 PRINT "AMOUNT OF MONEY."
70 PRINT:PRINT "IF INTEREST IS COMPOUNDED,"
80 PRINT "WHAT IS THE MONTHLY PAYMENT?":PRINT
90 INPUT "AMOUNT BORROWED";P
100 IF P>0 THEN 130
110 PRINT "ENTER AMOUNT MORE THAN ZERO PLEASE"
120 PRINT:GOTO 80
130 PRINT
140 INPUT "HOW MANY YEARS";Y
150 IF Y>0 THEN 180
160 PRINT "MUST BE MORE THAN ZERO"
170 GOTO 130
180 N=12*Y
190 PRINT
200 PRINT "WHAT IS THE INTEREST RATE"
210 INPUT "IN PERCENT";I
220 I=I/1200
230 F=(1+I)^N
240 M=P*(I*F/(F-1))
250 M=(INT(100*(M+.005)))/100
260 PRINT
270 PRINT "MONTHLY PAYMENT =";M
280 PRINT
290 END
```

Adverbs

Here is a method for printing random sentences. The computer "makes up" a sentence, and the user must find the adverb in the sentence and type it. There is a quiz of ten sentences.

The words are read in from data in lines 90–120. Arrays are used to hold the words. Line 80 reads in the words. The words in the DATA statements are in the order of an article or modifier, a noun for the subject, a verb, and an adverb. The arrays for the words are A$, B$, C$, and D$, respectively, and each array has ten words.

Lines 190–200 choose one of the ten words in each array to be used for the sentence. Line 210 chooses a random number J from 1 through 3 which will determine how the sentence is written. Line 220 uses ON-GOTO to branch to the printing procedure. The first possibility (line 230) is A$, B$, C$, D$. The second possibility (line 250) is A$, B$, D$, C$. The third possibility (line 270) is D$, A$, B$, C$. D$ will always be the adverb.

Line 290 receives the user's answer of what the adverb is and is called V$. Line 300 checks the answer. Line 310 prints the correct word if the user's answer was incorrect. Lines 330–340 print a message and increment the score for a correct answer. Lines 350–370 wait for the user to press RETURN before continuing to the next sentence.

Line 400 prints the score for ten sentences, then lines 410–450 present the option to try again and branch appropriately.

Program 14-2. Adverbs

```
10 REM ADVERBS
20 CLS:PRINT
30 PRINT TAB(30);"***********"
40 PRINT TAB(30);"* ADVERBS *"
50 PRINT TAB(30);"***********"
60 PRINT:PRINT:PRINT "You will be shown a sentence."
70 PRINT:PRINT "Type the adverb then press <RETURN>."
80 FOR C=0 TO 9:READ A$(C),B$(C),C$(C),D$(C):NEXT C
90 DATA THE,CAT,CRAWLED,QUICKLY,A,DOG,JUMPED,QUIETLY,MY,DEER
,RAN,HAPPILY
100 DATA YOUR,COW,LOPED,SLYLY,HIS,FOX,WIGGLED,SLOWLY,HER,WOL
F,GALLOPED
110 DATA JOYFULLY,ITS,BOY,SPED,RAPIDLY,OUR,GIRL,CREEPED,SILE
NTLY
120 DATA THAT,BUG,HURRIED,CALMLY,ONE,BEAR,MOVED,SWIFTLY
130 PRINT:PRINT:PRINT "Press the space bar to start."
```

```
140 E$=INKEY$:IF E$<>" " THEN 140
150 RANDOMIZE TIMER
160 SCORE=0
170 FOR T=1 TO 10
180 CLS:PRINT:PRINT
190 A=INT(10*RND):B=INT(10*RND)
200 C=INT(10*RND):D=INT(10*RND)
210 J=INT(3*RND)+1
220 ON J GOTO 230,250,270
230 PRINT A$(A);" ";B$(B);" ";C$(C);" ";D$(D);"."
240 GOTO 280
250 PRINT A$(A);" ";B$(B);" ";D$(D);" ";C$(C);"."
260 GOTO 280
270 PRINT D$(D);" ";A$(A);" ";B$(B);" ";C$(C);"."
280 PRINT:PRINT
290 INPUT "ADVERB:  ",V$:V$=UCASE$(V$)
300 IF V$=D$(D) THEN 330
310 PRINT:PRINT "The adverb is ";D$(D)
320 GOTO 350
330 PRINT:PRINT "CORRECT!"
340 SCORE=SCORE+1
350 PRINT:PRINT "Press <RETURN>."
360 E$=INKEY$
370 IF E$<>CHR$(13) THEN 360
380 NEXT T
390 CLS
400 PRINT "Your score is";SCORE;"right out of 10 sentences."
410 PRINT:PRINT:PRINT "Try again?  (Y/N)"
420 E$=INKEY$
430 IF E$="Y" OR E$="y" THEN 160
440 IF E$<>"N" AND E$<>"n" THEN 420
450 PRINT "NO":PRINT:PRINT:PRINT
460 END
```

States and Capitals

One of the most common drill programs is testing a student on the names of states and capitals. Here is a version for the Amiga. First, a map of the continental United States is drawn, then in random order a state is outlined. The user must type the name of the state. (Make sure the CAPS LOCK key is on.) If the state named is not correct, the user has a second chance. If it is incorrect twice, the state name is given and that state will appear again later in the quiz.

When the state is named correctly, the user will be asked for the capital. Again, there are two chances to get the correct answer before the correct capital is given. If the state and capital are named correctly, that state will not appear again in the quiz. A score is kept by keeping track of the number of guesses for both the state and the capital. Since there are 50 states with their 50 capitals, a perfect score would be 100. The computer will print the number of guesses it takes to go through all the states.

Line 20 is a DIMension statement for the state array S$ and the capital array C$. Lines 30–100 clear the screen and print the title and instructions.

Line 110 reads in from data the state and the capital for the 50 states. Lines 120–280 contain the states with their capitals. Lines 290–300 wait for the user to press the space bar to continue the program.

Line 310 uses RANDOMIZE TIMER so that states will be chosen randomly. Line 320 initializes G for the number of guesses. Line 330 is the beginning of the FOR-NEXT loop for the 50 states. Line 340 initializes F for a flag for incorrect answers.

Line 350 clears the screen for each new problem. Line 360 RESTOREs the data starting at line 370 for drawing the map. The DATA statements in lines 370–490 contain x and y coordinates for drawing the map using the LINE command. The drawing is in lines 500–530.

Line 540 chooses a random number R from 1 to 50. If the state was previously named correctly, S$(R) will have been set to the null string, " ", so if the random number R points to a null string, another number must be chosen. Lines 550–580 use ON-GOSUB to go to a subroutine that will RESTORE the proper data for drawing that state. The subroutines are named by state abbreviations rather than line numbers.

Lines 590–620 draw the state. The DATA statements for each state have an (x1,y1) and (x2,y2) set of coordinates for the LINE command, then a number C indicating how many LINE commands will be needed, and then pairs of numbers for the coordinates for the rest of the lines to complete drawing the state.

Lines 630–870 receive the answers and check them. Lines 880–920 print the number of guesses. Lines 930–950 play a tune of random tones.

If you have trouble with this program, the most likely place for errors is in typing the numbers in the DATA statements. Keep in mind that most of the numbers are pairs of x and y coordinates. Make sure there are no extra commas and no commas at the ends of the lines. The subroutines are labeled with state abbreviations to help you know which DATA statements correspond to which state if you do have an error.

Program 14-3. States and Capitals

```
10 REM STATES AND CAPITALS
20 DIM S$(50),C$(50)
30 CLS
40 LOCATE 2,22:PRINT "** UNITED STATES **"
50 LOCATE 5,6:PRINT "You will see an outline of a state."
60 LOCATE 7,6:PRINT "Type the name of the state, then its ca
pital city."
70 LOCATE 9,6:PRINT "If you get the state and capital correc
t,"
80 LOCATE 11,11:PRINT "it will not appear again."
90 LOCATE 13,6:PRINT "The quiz consists of all 50 states in
a random order."
100 LOCATE 15,6:PRINT "PLEASE MAKE SURE CAPS LOCK IS ON."

110 FOR C=1 TO 50:READ S$(C),C$(C):NEXT C
120 DATA ALABAMA,MONTGOMERY,ALASKA,JUNEAU,ARIZONA,PHOENIX,AR
KANSAS
130 DATA LITTLE ROCK,CALIFORNIA,SACRAMENTO,COLORADO,DENVER
140 DATA CONNECTICUT,HARTFORD,DELAWARE,DOVER,FLORIDA,TALLAHA
SSEE
150 DATA GEORGIA,ATLANTA,HAWAII,HONOLULU,IDAHO,BOISE,ILLINOI
S
160 DATA SPRINGFIELD,INDIANA,INDIANAPOLIS,IOWA,DES MOINES,KA
NSAS
170 DATA TOPEKA,KENTUCKY,FRANKFORT,LOUISIANA,BATON ROUGE,MAI
NE
180 DATA AUGUSTA,MARYLAND,ANNAPOLIS,MASSACHUSETTS,BOSTON,MIC
HIGAN
190 DATA LANSING,MINNESOTA,ST. PAUL,MISSISSIPPI,JACKSON,MISS
OURI
200 DATA JEFFERSON CITY,MONTANA,HELENA,NEBRASKA,LINCOLN,NEVA
DA
210 DATA CARSON CITY,NEW HAMPSHIRE,CONCORD,NEW JERSEY,TRENTO
N
220 DATA NEW MEXICO,SANTA FE,NEW YORK,ALBANY,NORTH CAROLINA,
RALEIGH
230 DATA NORTH DAKOTA,BISMARCK,OHIO,COLUMBUS,OKLAHOMA,OKLAHO
MA CITY
240 DATA OREGON,SALEM,PENNSYLVANIA,HARRISBURG,RHODE ISLAND,P
ROVIDENCE
250 DATA SOUTH CAROLINA,COLUMBIA,SOUTH DAKOTA,PIERRE,TENNESS
EE
260 DATA NASHVILLE,TEXAS,AUSTIN,UTAH,SALT LAKE CITY,VERMONT
270 DATA MONTPELIER,VIRGINIA,RICHMOND,WASHINGTON,OLYMPIA
280 DATA WEST VIRGINIA,CHARLESTON,WISCONSIN,MADISON,WYOMING,
CHEYENNE
290 LOCATE 22,14:PRINT "Press the space bar to start."
300 A$=INKEY$:IF A$<>" " THEN 300
310 RANDOMIZE TIMER
320 G=0
330 FOR N=1 TO 50
340 F=0
350 CLS
    REM DRAW MAP
360 RESTORE 370
370 DATA 280,32,356,32,356,28,364,28,360,32,398,35,380,42,39
0,42
```

```
380 DATA 410,37,406,40,420,43,432,40,444,43,436,43,448,46,44
6,53
390 DATA 448,54,454,51,460,56,456,64,466,63,486,54,486,53,50
2,48
400 DATA 512,40,536,36,540,28,542,22,546,23,555,21,566,31,56
8,30
410 DATA 568,33,550,40,548,46,555,51,558,51,558,49,558,51,55
5,51
420 DATA 528,61,532,63,526,79,532,85,528,91,492,110,492,118,
512,135
430 DATA 512,142,508,145,496,140,486,133,484,126,472,123,460
,125
440 DATA 454,123,420,126,420,129,424,131,404,131,404,129,396
,128
450 DATA 392,130,380,130,344,138,344,147,350,149,332,148,324
,144
460 DATA 326,142,306,130,296,130,288,133,278,130,272,123,268
,123
470 DATA 260,119,246,118,246,120,216,118,186,110,186,108,166
,108
480 DATA 164,104,158,101,140,95,128,80,134,76,128,76,124,72,
120,62
490 DATA 126,54,128,50,142,35,144,24,156,26,160,23
500 LINE (160,23)-(224,28)
510 FOR T=1 TO 97
520 READ X,Y:LINE -(X,Y)
530 NEXT T
    REM DRAW STATE
540 R=INT(50*RND+1):IF S$(R)="" THEN 540
550 IF R>25 THEN 580
560 ON R GOSUB ALA,ALS,AZ,AK,CA,COL,CON,DE,FL,GA,HA,ID,IL,IN
,IO,KA,KY,LA,ME,MD,MAS,MICH,MINN,MISS,MO
570 GOTO 590
580 ON R-25 GOSUB MONT,NEB,NEV,NH,NJ,NM,NY,NC,ND,OH,OKL,ORE,
PA,RI,SC,SD,TN,TX,UT,VT,VIR,WA,WV,WI,WY
590 READ X1,Y1,X2,Y2,C:LINE(X1,Y1)-(X2,Y2)
600 FOR J=1 TO C
610 READ X,Y:LINE -(X,Y)
620 NEXT J
630 SOUND 1300,2:G=G+1
640 LOCATE 21,2:INPUT "STATE";SS$:SS$=UCASE$(SS$)
650 IF SS$=S$(R) THEN 740
660 SOUND 330,2:SOUND 262,2
670 F=F+1:IF F=2 THEN 700
680 LINE (0,160)-(400,199),0,BF
690 GOTO 630
700 PRINT " The state is ";S$(R)
710 PRINT:PRINT " Press the space bar to continue.";
720 A$=INKEY$
730 IF A$=" " THEN 340 ELSE 720
740 SOUND 262,2:SOUND 330,2
750 SOUND 392,2:SOUND 523,4:F2=0
760 G=G+1:LOCATE 22,2:INPUT "CAPITAL";CC$:CC$=UCASE$(CC$)
770 IF CC$=C$(R) THEN 840
780 SOUND 330,2:SOUND 262,2
790 F2=F2+1:IF F2=2 THEN 820
800 LINE (0,168)-(400,199),0,BF
810 GOTO 760
820 PRINT " The capital is ";C$(R)
830 GOTO 710
```

```
840 SOUND 262,2:SOUND 330,2
850 SOUND 392,2:SOUND 523,4
860 S$(R)="":FOR D=1 TO 2000:NEXT D
870 NEXT N
880 CLS
890 PRINT:PRINT "A perfect score is 100."
900 PRINT:PRINT
910 PRINT "You had";G;"guesses."
920 PRINT:PRINT:PRINT
930 FOR C=1 TO 40
940 SOUND 500*RND+300,1
950 NEXT C
960 GOTO 2740

ALA: RESTORE A1
A1:  DATA 426,124,426,99,5,448,99
     DATA 456,112,458,119,436,119,438,123
     RETURN
ALS: RESTORE A2
A2:  DATA 80,26,76,5,26,48,1,28,6,38,12,22,12,24,16,34,16
     DATA 34,18,20,21,20,26,26,24,24,29,38,31,32,35,0,40
     DATA 32,36,48,32,56,25,52,30,66,27,76,29,84,29
     DATA 106,40,106,36,114,39,96,28,88,29
     LOCATE 2,8:PRINT "?"
     RETURN
AZ: RESTORE A3
A3:  DATA 236,119,242,91,7,200,88,196,93,192,92
     DATA 192,102,188,106,190,108,186,110
     RETURN
AK: RESTORE A4
A4:  DATA 368,94,406,94,9,406,96,412,96
     DATA 410,100,400,107,400,112,374,112
     DATA 374,110,368,110,368,94
     RETURN
CA: RESTORE C1
C1:  DATA 128,55,160,58,6,152,73,190,98
     DATA 192,102,186,109,166,108,164,104
     RETURN
COL: RESTORE C2
C2:  DATA 248,72,304,72,3,304,92,248,92,248,72
     RETURN
CON: RESTORE C3
C3:  DATA 534,58,532,52,2,546,51,546,55
     LOCATE 8,69:PRINT "\"
     LOCATE 9,70:PRINT "?"
     RETURN
DE: RESTORE D1
D1:  DATA 520,67,524,72,2,528,72,522,67
     LOCATE 9,70:PRINT "<?"
     RETURN
FL: RESTORE F1
F1:  DATA 438,123,434,119,6,456,119,456,120
     DATA 484,120,486,121,486,117,492,117
     RETURN
GA: RESTORE G1
G1:  DATA 490,110,468,97,8,448,97,456,111,454,115,460,119,48
4,119
     DATA 486,121,486,117,492,117
     RETURN
HA: RESTORE H1
```

```
H1:  DATA 56,80,60,80,4,64,84,54,86,50,81,54,80
     LOCATE 10,2:PRINT "O o ."
     LOCATE 9,9:PRINT "?"
     RETURN
ID: RESTORE I1
I1:  DATA 200,27,194,40,11,198,41,188,48,184,56,232,57,236,5
0
     DATA 220,51,216,45,212,46,214,40,206,36,208,27
     RETURN
IL: RESTORE I2
I2:  DATA 418,62,396,62,10,400,66,390,74,398,82
     DATA 402,80,402,84,414,90,424,86
     DATA 426,80,424,66,418,62
     RETURN
IN: RESTORE I3
I3:  DATA 446,64,428,64,7,422,66,426,79
     DATA 424,85,438,83,442,79,446,78,446,64
     RETURN
IO: RESTORE I4
I4:  DATA 390,58,350,58,7,348,63,356,74
     DATA 390,74,394,69,400,66,390,61,390,58
     RETURN
KA: RESTORE K1
K1:  DATA 360,77,304,77,4,304,92
     DATA 368,92,366,82,360,77
     RETURN
KY: RESTORE K2
K2:  DATA 402,93,402,90,10,424,85,438,83
     DATA 442,79,460,80,464,83,474,86
     DATA 464,91,412,91,412,93,402,93
     RETURN
LA: RESTORE L1
L1:  DATA 418,125,416,122,7,398,122
     DATA 404,116,400,111,376,111
     DATA 376,117,380,123,378,129
     RETURN
ME: RESTORE M1
M1:  DATA 536,34,548,45,0
     LOCATE 4,74:PRINT "?"
     RETURN
MD: RESTORE M2
M2:  DATA 530,72,522,72,7,526,67,490,69
     DATA 490,72,504,70,512,72,512,74,522,75
     RETURN
MAS: RESTORE M3
M3:  DATA 548,48,532,48,3,532,52,548,52,548,54
     RETURN
MICH: RESTORE M4
M4:  DATA 392,42,410,45,8,416,48,418,45
     DATA 436,44,424,51,424,56
     DATA 430,60,428,64,456,64
     RETURN
MINN: RESTORE M5
M5:  DATA 342,32,346,47,6,342,48,348,59
     DATA 392,59,378,52,376,47,380,42
     RETURN
MISS: RESTORE M6
M6:  DATA 428,124,426,100,7,406,100
     DATA 400,107,400,111,404,115
     DATA 398,122,416,122,418,126
     RETURN
```

```
MO: RESTORE M7
M7:  DATA 392,74,356,74,11,366,82,368,95
     DATA 406,95,406,97,410,97,416,92
     DATA 416,90,402,84,402,80,398,82,392,74
     RETURN
MONT: RESTORE M8
M8:  DATA 208,27,206,36,7,214,40,212,46
     DATA 216,45,220,52,236,50,290,50,290,32
     RETURN
NEB: RESTORE N1
N1:  DATA 288,62,288,72,6,304,72,304,78
     DATA 360,78,350,64,332,62,288,62
     RETURN
NEV: RESTORE N2
N2:  DATA 204,61,156,60,5,156,74,192,96
     DATA 196,92,202,94,204,61
     RETURN
NH: RESTORE N3
N3:  DATA 536,34,548,45,2,534,48,536,36
     LOCATE 6,72:PRINT "<?"
     RETURN
NJ: RESTORE N4
N4:  DATA 534,58,522,58,4,522,62,526,64
     DATA 520,67,530,69
     LOCATE 9,70:PRINT "?"
     RETURN
NM: RESTORE N5
N5:  DATA 238,120,240,91,4,296,91
     DATA 296,117,260,117,260,119
     RETURN
NY: RESTORE N6
N6:  DATA 476,59,516,55,6,532,59,528,60
     DATA 548,56,532,59,530,47,524,38
     LOCATE 7,65:PRINT "?"
     RETURN
NC: RESTORE N7
N7:  DATA 526,83,478,87,4,460,96,488,94
     DATA 502,95,512,98
     RETURN
ND: RESTORE N8
N8:  DATA 290,32,290,47,2,346,47,342,32
     RETURN
OH: RESTORE O1
O1:  DATA 456,63,446,63,4,446,76,468,79
     DATA 480,71,480,58
     RETURN
OKL: RESTORE O2
O2:  DATA 294,92,368,92,5,372,108,320,106
     DATA 320,96,294,96,294,92
     RETURN
ORE: RESTORE O3
O3:  DATA 144,33,152,38,5,200,41,188,48
     DATA 190,50,184,60,128,55
     RETURN
PA: RESTORE P1
P1:  DATA 480,59,480,69,6,522,67,526,63
     DATA 524,62,524,57,516,56,480,59
     RETURN
RI: RESTORE R1
R1:  DATA 548,54,548,50,2,542,50,542,55
     LOCATE 8,70:PRINT "\"
```

```
     LOCATE 9,71:PRINT "?"
     RETURN
SC: RESTORE S1
S1:  DATA 490,110,468,97,4,488,94
     DATA 492,96,502,95,512,99
     RETURN
SD: RESTORE S2
S2:  DATA 344,47,288,47,5,288,61,334,61
     DATA 350,63,348,62,344,47
     RETURN
TN: RESTORE T1
T1:  DATA 424,90,480,90,5,456,98,404,98
     DATA 412,92,424,92,424,90
     RETURN
TX: RESTORE T2
T2:  DATA 260,118,296,118,9,296,95,320,95
     DATA 320,106,340,108,366,108,374,110
     DATA 376,118,380,122,378,130
     RETURN
UT  RESTORE UI
U1:  DATA 232,63,204,63,5,204,90,248,90
     DATA 248,71,232,71,232,63
     RETURN
VT: RESTORE V1
V1:  DATA 536,36,534,49,2,528,49,524,38
     RETURN
VIR: RESTORE V2
V2:  DATA 524,75,512,74,9,512,72,504,70
     DATA 496,76,492,76,488,82,480,84
     DATA 472,84,462,88,528,84
     RETURN
WA: RESTORE W1
W1:  DATA 200,26,196,40,2,152,38,144,33
     RETURN
WV: RESTORE W2
W2:  DATA 492,70,482,70,13,480,68,480,72,468,78,470,82
     DATA 478,84,488,82,492,76,496,77
     DATA 500,71,506,71,506,69,492,72,492,70
     RETURN
WI: RESTORE W3
W3:  DATA 380,42,376,47,11,378,53,390,57
     DATA 390,61,396,63,418,63,416,55
     DATA 420,47,412,51,414,47,408,45,392,42
     RETURN
WY: RESTORE W4
W4:  DATA 288,52,224,52,3,224,71,288,71,288,52
     RETURN
2740 END
```

Typing

A common use for a home computer is to learn touch typing, or keyboarding. The computer is ideal for learning to type because the keyboard is like the keys on a typewriter, and the computer can give immediate feedback. Typing tutorials also are useful if you like to program, because you will want to learn to type efficiently.

This program is called "Type1" because it is only the first unit of a possible typing course. It is a tutorial for the "home position" keys only. Subsequent programs could teach the rest of the keys in a progressive order. Additional programs could present drills to improve typing skills after the key positions and fingering are learned.

Outlines of the hands are drawn by using LINE commands. The DATA statements containing coordinates are in subroutines in lines 270–460. The subroutine in lines 210–260 reads the data and draws the lines.

Lines 570–580 contain data for the keys with their corresponding column position and a frequency for the SOUND command when that key is indicated. Line 590 reads in this information for the eight home row keys. The LOCATE command is used to place the cursor in the proper position for printing.

Lines 900–910 contain phrases that can be typed by using only the home keys. The quiz randomly chooses from these phrases, and the student must type five phrases correctly to complete the quiz.

Program 14-4. Type1

```
10 REM TYPE1
20 CLS
50 PRINT TAB(25);"T Y P E - E T T E"
60 PRINT:PRINT TAB(30);"UNIT 1"
70 PRINT:PRINT TAB(27);"HOME POSITION"
80 PRINT:PRINT:PRINT
90 PRINT "CAPS LOCK SHOULD BE ON"
100 GOTO 470
120 PRINT:PRINT "PRESS ANY KEY TO CONTINUE."
140 E$=INKEY$:IF E$="" THEN R=RND:GOTO 140
150 RETURN
160 LOCATE 10,A(J):PRINT L$(J)
170 SOUND F(J),3
180 E$=INKEY$:IF E$<>L$(J) THEN 180
190 LOCATE 10,A(J):PRINT " "
200 RETURN
210 READ X1,Y1,X2,Y2,N
220 LINE(X1,Y1)-(X2,Y2),3
230 FOR I=1 TO N
240 READ X2,Y2:LINE -(X2,Y2),3
250 NEXT I
260 RETURN
270 REM LEFT HAND
280 RESTORE 300
300 DATA 170,199,230,160,36,230,158
305 DATA 224,156,208,156,188,162,160,171
310 DATA 152,171,148,166,152,155,182,112
315 DATA 184,104,180,99,172,98,156,111
320 DATA 142,129,130,139,142,92,136,88
```

```
325 DATA 128,86,120,88,112,100,96,138
330 DATA 94,93,92,89,86,87,80,88
335 DATA 72,92,70,97,64,114,62,139
340 DATA 60,142,48,111,40,108,32,108
345 DATA 26,111,28,128,24,199
350 GOSUB 210
360 RETURN
370 REM RIGHT HAND
380 RESTORE 390
390 DATA 416,199,370,156,26,372,151
395 DATA 382,150,392,152,432,166
400 DATA 422,136,408,107,412,100,418,98
405 DATA 430,101,456,142,454,92,464,87
410 DATA 476,87,484,92,496,140,516,98
415 DATA 528,92,534,92,544,96,534,144
420 DATA 566,112,576,110,582,115,574,131
425 DATA 554,163,552,199
430 GOSUB 210
460 RETURN
470 PRINT:PRINT
480 PRINT "This unit will teach you the 'home' position of t
ouch typing."
500 PRINT:PRINT "As you learn to type, your fingers will res
t lightly"
510 PRINT "on these 'home' keys."
520 PRINT:PRINT "You will gradually learn to type other lett
ers, but your fingers"
530 PRINT "should always return to the home position."
540 PRINT:PRINT
560 RESTORE 570
570 DATA A,6,262,S,12,292,D,18,330,F,24,349
580 DATA J,53,392,K,60,440,L,68,494,;,74,523
590 FOR M=1 TO 8:READ L$(M),A(M),F(M):NEXT M
600 GOSUB 120
610 CLS
620 GOSUB 280:GOSUB 380
630 LINE (192,136)-(400,147),1,BF
640 PRINT "Place your fingers on the keys as shown."
650 PRINT
660 PRINT "Your right thumb will press the space bar."
670 FOR M=1 TO 8
680 LOCATE 10,A(M):PRINT L$(M)
690 SOUND F(M),3
700 NEXT M
710 LOCATE 4,2:GOSUB 120
720 LINE (0,0)-(600,80),0,BF
730 LOCATE 1,4
740 PRINT "Type each letter as it appears."
760 FOR T=1 TO 3:FOR J=1 TO 8
770 GOSUB 160
780 NEXT J:NEXT T
790 FOR T=1 TO 30
800 J=INT(8*RND+1):IF J=K THEN 800
810 K=J
820 GOSUB 160
830 NEXT T
840 LOCATE 1,4:PRINT "CHOOSE:  1  Try again";SPACE$(12)
850 PRINT TAB(13);"2  Continue program"
860 E$=INKEY$:IF E$="1" THEN 720
870 IF E$<>"2" THEN 860
880 CLS
```

```
890 RESTORE 900
900 DATA A SAD LAD;,A FAD;,ASK A LAD;,A SAD FAD,A LAD ASKS D
AD
910 DATA ALFALFA,ALAS A SAD DAD,DAD ASKS A LAD,ASK DAD
920 FOR T=1 TO 9:READ P$(T):NEXT
930 PRINT "Use your right little finger to press <RETURN>."
940 PRINT
950 PRINT "Type the phrase shown, then press <RETURN>."
960 PRINT
970 PRINT "You must type five phrases correctly to end the d
rill."
1050 GOSUB 120
1060 FOR T=1 TO 5
1070 CLS
1080 FOR X=80 TO 480 STEP 40
1090 LINE (X,62)-(X+24,74),1,BF
1100 NEXT X
1110 COLOR 2,1
1120 LOCATE 9,12:PRINT "A";SPC(4);"S";SPC(4);"D";SPC(4);"F";
1130 LOCATE 9,42:PRINT "J";SPC(4);"K";SPC(4);"L";SPC(4);";";
1140 COLOR 1,0
1150 J=INT(9*RND+1)
1160 IF P$(J)="" THEN 1150
1170 LOCATE 14,30:PRINT P$(J)
1180 COLOR 2,3:LOCATE 15,28:INPUT B$:COLOR 1,0
1190 IF B$=P$(J) THEN 1230
1200 SOUND 330,2:SOUND 262,2
1210 GOSUB 120
1220 GOTO 1070
1230 SOUND 262,3:SOUND 330,3
1240 SOUND 392,3:SOUND 523,6
1250 P$(J)=""
1260 NEXT T
1270 PRINT:PRINT
1280 PRINT "CHOOSE:  1  Practice letters"
1290 PRINT TAB(10);"2  Practice words"
1300 PRINT TAB(10);"3  End program"
1310 A$=INKEY$:IF A$="1" THEN 610
1320 IF A$="2" THEN 880
1330 IF A$<>"3" THEN 1310
1340 CLS
1350 END
```

Algebra: Binomial Multiplication

While there is a lot of educational software available for younger children, material for older students is not so readily available. This algebra program offers something for an older student. I have written this program for other computers and it has been so popular that I am including it here. Binomial multiplication is just one concept in algebra, but this program can give you ideas so that you can write programs for other topics in algebra.

This program is for practicing multiplication of two binomials. First, an example problem is shown. It is printed

with lines 120–220. Lines 240–570 show the multiplication in general form.

Lines 580–860 present the first problem for the student. It contains only positive numbers, and the numeric factors are random numbers A and B from 1 through 3 (so the results will always be one-digit numbers). Lines 890–990 print another example problem, and lines 1000–1010 call a subroutine to present a problem with positive numbers and coefficients for all factors.

Lines 1030–1100 print a screen about using positive and negative numbers. Line 1110 calls the subroutine to present a problem which can contain positive and negative numbers.

Lines 1130–1280 print more information.

Lines 1300–1350 are the subroutine to wait for the student to press RETURN before continuing the program. Lines 1360–1370 are the subroutine to play the *uh-oh* tones for an incorrect response. Lines 1380–1390 play the arpeggio for a correct response.

Lines 1400–1450 are the subroutine to receive an answer for a number. A question mark is blinked in the position specified by ROW,C.

Lines 1460–1530 present the option to have another similar problem or to continue the program. If any part of the problem is answered incorrectly, another problem is presented. If the problem is correct, then the program proceeds to this subroutine and the student may choose whether to practice more or to go on.

Lines 1540–2050 contain the subroutine to print a problem with random numbers. T may equal 1 or 2. If T is 1, all the numbers are positive. A, B, D, and E are the coefficients and factors to be multiplied. F is a flag to indicate an incorrect answer. The SGN function is used to determine the sign of a product or sum.

Lines 2060–2170 are the subroutine to get the plus or minus sign on problems that contain positive and negative numbers.

Lines 2180–2270 position the numbers in answers depending on whether they are one digit or two digits.

Program 14-5. Algebra

```
10 REM ALGEBRA
20 CLS
30 RANDOMIZE TIMER
```

```
70 PRINT TAB(20);"BINOMIAL MULTIPLICATION"
80 PRINT:PRINT "This program discusses multiplication of two
 binomials,"
90 PRINT
100 PRINT "such as (x+5) times (x+4)."
110 GOSUB 1300
120 PRINT "Compare algebraic multiplication to numeric multi
plication:"
130 PRINT:PRINT
140 PRINT TAB(10);"1 2";TAB(60);"x + 2"
150 PRINT
160 PRINT TAB(10);"1 3";TAB(60);"x + 3"
170 PRINT TAB(8);"_____";TAB(58);"________"
180 PRINT
190 PRINT TAB(10);"3 6";TAB(59);"3x + 6"
195 PRINT TAB(55);"2"
200 PRINT TAB(8);"1 2";TAB(54);"x  + 2x"
210 PRINT TAB(7);"_____";TAB(54);"__________"
215 PRINT TAB(55);"2"
220 PRINT TAB(8);"1 5 6";TAB(54);"x  + 5x + 6"
230 GOSUB 1300
240 CC=3
250 PRINT "IN GENERAL,
260 PRINT:PRINT
270 PRINT TAB(40);"x  +  ";
280 COLOR 3,0:PRINT "a"
290 COLOR 1,0:PRINT:PRINT TAB(40);"x  +  ";
300 COLOR 3,0:PRINT "b"
310 COLOR 1,0:PRINT TAB(39);"________"
320 PRINT:COLOR 3,0
330 PRINT TAB(39);"b";
340 COLOR 1,0:PRINT "x  + ";
350 COLOR 3,0:PRINT "ab"
360 COLOR 1,0:PRINT TAB(32);"2"
370 PRINT TAB(31);"x  +    ";
380 COLOR 3,0:PRINT "a";
390 COLOR 1,0:PRINT "x"
400 PRINT TAB(31);"________________"
405 PRINT TAB(32);"2"
410 PRINT TAB(31);"x  + ";
420 COLOR 3,0:PRINT "(a+b)";
430 COLOR 1,0:PRINT "x + ";
440 COLOR 3,0:PRINT "ab"
450 COLOR 1,0
460 PRINT:PRINT
470 PRINT "The first term is x * x"
480 PRINT "The last term is ";
490 COLOR 3,0:PRINT "a";
500 COLOR 1,0:PRINT " * ";
510 COLOR 3,0:PRINT "b"
520 COLOR 1,0:PRINT "The middle term combines ";
530 COLOR 3,0:PRINT "a";
540 COLOR 1,0:PRINT " and ";
550 COLOR 3,0:PRINT "b";
560 COLOR 1,0:PRINT " multiplied by x"
570 GOSUB 1300
580 PRINT "NOW YOU MULTIPLY:"
590 A=INT(3*RND)+1:B=INT(3*RND)+1:F=0
600 PRINT:PRINT TAB(50);"x +";A
610 PRINT:PRINT TAB(50);"x +";B
620 PRINT TAB(49);"_____"
```

```
630 PRINT
640 PRINT B;"TIMES TOP ROW";TAB(49);"?x + ?"
650 ROW=8:C=49:GOSUB 1400
660 IF VAL(K$)=B THEN 680
670 GOSUB 1360:GOTO 650
680 C=54:GOSUB 1400
690 IF VAL(K$)=B*A THEN 710
700 GOSUB 1360:GOTO 680
710 PRINT:PRINT TAB(45);"2"
720 PRINT " x TIMES TOP ROW";TAB(44);"x  + ?x"
730 ROW=10:C=49:GOSUB 1400
740 IF VAL(K$)=A THEN 760
750 GOSUB 1360:GOTO 730
760 PRINT:PRINT TAB(44);STRING$(11,"_")
770 PRINT TAB(45);"2"
780 PRINT " ADD";TAB(44);"x  + ?x + ?"
790 ROW=13:GOSUB 1400
800 IF VAL(K$)=A+B THEN 820
810 GOSUB 1360:GOTO 790
820 C=54:GOSUB 1400
830 IF VAL(K$)=A*B THEN 850
840 GOSUB 1360:GOTO 820
850 IF F=0 THEN 870
860 GOSUB 1300:GOTO 580
870 GOSUB 1460
880 IF K$="1" THEN 580
890 PRINT "There may be coefficients of the first term,"
900 PRINT "but the rules don't change."
910 PRINT:PRINT "For example,"
920 PRINT:PRINT TAB(30);"2y + 5"
930 PRINT:PRINT TAB(30);"3y + 1"
940 PRINT TAB(30);"_____"
950 PRINT:PRINT TAB(30);"2y + 5"
955 PRINT TAB(25);"2"
960 PRINT TAB(23);"6y  + 15y "
970 PRINT TAB(23);STRING$(13,"_")
975 PRINT TAB(25);"2"
980 PRINT TAB(23);"6y  + 17y + 5"
990 GOSUB 1300
1000 T=1:SD=1:SD$="+":SE=1:SE$="+"
1010 GOSUB 1590
1020 IF K$="1" THEN 1010
1030 PRINT "Binomials may contain + or - numbers."
1040 PRINT:PRINT "Multiply the numbers as usual, and"
1050 PRINT:PRINT "remember the rules for the signs."
1060 PRINT:PRINT TAB(10);"+ * + = +"
1070 PRINT:PRINT TAB(10);"+ * - = -"
1080 PRINT:PRINT TAB(10);"- * + = -"
1090 PRINT:PRINT TAB(10);"- * - = +"
1100 GOSUB 1300
1110 T=2:GOSUB 1550
1120 IF K$="1" THEN 1110
1130 PRINT "There may be cases when the middle term becomes
zero"
1140 PRINT
1150 PRINT "so you do not need to specify the middle term."
1160 PRINT:PRINT
1170 PRINT "   x  + 3";TAB(40);"2y  + 2"
1180 PRINT:PRINT "   x  - 3";TAB(40);"4y  - 4"
1190 PRINT "_____";TAB(40);"_____"
1195 PRINT:PRINT TAB(5);"2";TAB(42);"2"
```

```
1200 PRINT "    x  - 9";TAB(40);"8y  - 8"
1210 GOSUB 1300
1220 PRINT "Other multiplication problems include + or -"
1230 PRINT:PRINT "numbers in the first term"
1240 PRINT:PRINT "and/or alphabetic characters as coefficien
ts"
1250 PRINT:PRINT "for either term."
1260 PRINT:PRINT
1270 PRINT "This completes this unit of instruction.":PRINT:
PRINT
1280 GOTO 2280
1290 STOP
1300 LOCATE 23,1:PRINT "PRESS <RETURN>.";
1310 E$=INKEY$
1320 IF E$="" THEN 1310
1330 IF ASC(E$)<>13 THEN 1310
1340 CLS
1350 RETURN
1360 SOUND 330,2:SOUND 262,2:F=1
1370 RETURN
1380 SOUND 262,2:SOUND 330,2:SOUND 392,2:SOUND 523,4
1390 RETURN
1400 SOUND 1300,2
1410 K$=INKEY$:IF K$<>"" THEN 1440
1420 LOCATE ROW,C:PRINT "?";
1430 LOCATE ROW,C:PRINT " ";:GOTO 1410
1440 LOCATE ROW,C:PRINT K$;
1450 RETURN
1460 LOCATE 22,1
1470 PRINT "CHOOSE:  1  ANOTHER PROBLEM"
1480 PRINT TAB(10);"2  CONTINUE PROGRAM"
1490 SOUND 1300,2
1500 K$=INKEY$
1510 IF K$<>"1" AND K$<>"2" THEN 1500
1520 CLS
1530 RETURN
1540 IF T=1 THEN 1590
1550 SD=(-1)^(INT(2*RND)+1)
1560 IF SD=1 THEN SD$="+" ELSE SD$="-"
1570 SE=(-1)^(INT(2*RND)+1)
1580 IF SE=1 THEN SE$="+" ELSE SE$="-"
1590 CLS
1600 A=INT(7*RND)+1:B=INT(7*RND)+1
1610 D=INT(7*RND)+1:E=INT(7*RND)+1:F=0
1620 IF A=B AND D=E THEN 1600
1630 IF (A*E*SE=(-1)*B*D*SD) THEN 1600
1640 A$=RIGHT$(STR$(A),1)
1650 B$=RIGHT$(STR$(B),1)
1660 X$=CHR$(87+INT(4*RND)+32)
1670 PRINT "MULTIPLY":PRINT
1680 PRINT TAB(29);A$;X$;" ";SD$;" ";D
1690 PRINT:PRINT TAB(29);B$;X$;" ";SE$;" ";E
1700 PRINT TAB(28);"_______":PRINT
1710 PRINT SE$;RIGHT$(STR$(E),1);" * TOP ROW";TAB(30);X$;" +
"
1720 ROW=8:IF T=1 THEN 1740
1730 C=26:S$=SE$:GOSUB 2060
1740 CC=27:P=A*E:GOSUB 2180
1750 IF T=1 THEN 1790
1760 C=32:SS=SGN(SE*SD)
1770 IF SS=1 THEN S$="+" ELSE S$="-"
```

```
1780 GOSUB 2060
1790 CC=33:P=D*E:GOSUB 2180
1800 PRINT:PRINT TAB(24);"2"
1810 PRINT " "B$;X$;" * TOP ROW";TAB(23);X$;"  +   ";X$
1820 ROW=ROW+2:CC=20:P=A*B:GOSUB 2180
1830 IF T=1 THEN 1850
1840 C=26:S$=SD$:GOSUB 2060
1850 CC=27:P=B*D:GOSUB 2180
1860 PRINT TAB(21);STRING$(15,"_")
1870 PRINT TAB(24);"2"
1880 PRINT " ADD";TAB(23);X$;"  +   ";X$;" +"
1890 ROW=ROW+3:CC=20:P=A*B:GOSUB 2180
1900 M=A*E*SE+B*D*SD
1910 IF T=1 THEN 1950
1920 C=26:SS=SGN(M)
1930 IF SS=1 THEN S$="+" ELSE S$="-"
1940 GOSUB 2060
1950 CC=27:P=ABS(M):GOSUB 2180
1960 IF T=1 THEN 2000
1970 C=32:SS=SGN(SE*SD)
1980 IF SS=1 THEN S$="+" ELSE S$="-"
1990 GOSUB 2060
2000 CC=33:P=D*E:GOSUB 2180
2010 GOSUB 1380
2020 IF F=1 THEN GOSUB 1300:GOTO 1540
2030 GOSUB 1460
2040 IF K$="1" THEN 1540
2050 RETURN
2060 SOUND 1300,2
2070 K$=INKEY$:IF K$<>"" THEN 2110
2080 LOCATE ROW,C:PRINT "+";
2090 LOCATE ROW,C:PRINT "-";
2100 GOTO 2070
2110 LOCATE ROW,C:PRINT K$;
2120 IF K$=S$ THEN 2160
2130 IF S$="+" AND K$="=" THEN 2160
2140 IF S$="-" AND K$="_" THEN 2160
2150 GOSUB 1360
2160 LOCATE ROW,C:PRINT S$
2170 RETURN
2180 L=LEN(STR$(P))-1
2190 IF L=1 THEN CC=CC+1
2200 C$="":FOR T=1 TO L
2210 C=CC+T:GOSUB 1400
2220 C$=C$+K$
2230 NEXT T
2240 IF VAL(C$)=P THEN 2270
2250 GOSUB 1360
2260 FOR T=1 TO L:LOCATE ROW,CC+L:PRINT " ";:NEXT T:GOTO 220
0
2270 RETURN
2280 END
```

Index

To order your copy of *Elementary Amiga BASIC Disk*, call our toll-free US order line: 1-800-346-6767 (in NY 212-887-8525) or send your prepaid order to:

Elementary Amiga BASIC Disk
COMPUTE! Publications
P.O. Box 5038
F.D.R. Station
New York, NY 10150

Send _____ copies of *128 LADS Disk* at $15.95 per copy.

All orders must be prepaid (check, charge, or money order). NC residents add 5% sales tax. NY residents add 8.25% sales tax.

Subtotal $_____

Shipping and Handling: $2.00/disk $_____

Sales tax (if applicable) $_____

Total payment enclosed $_____

□ Payment enclosed
□ Charge □ Visa □ MasterCard □ American Express

Acct. No. _____ Exp. Date _____ (Required)

Name _____

Address _____

City _____ State _____ Zip _____

Please allow 4-5 weeks for delivery.

Amiga BASIC

More than just a programmer's manual, *Elementary Amiga BASIC* is a complete primer in the fundamentals of BASIC programming on the Amiga. Commands and techniques are demonstrated with plenty of practical example programs you can type in and use, or modify to meet your own needs.

All the main features of this powerful 16-bit computer are explored here, including how to:

- Create your own screens and windows
- Design and display menus
- Compose and play music
- Format and display text
- Create colorful graphics
- Produce a variety of sound effects
- Manipulate sprites and animated objects
- Program the computer to speak in different voices and accents—even to speak a foreign language.

Whether you are just starting in programming or have worked with BASIC on another computer, you will find all the commands, statements, and techniques you will need for effective programming on the Amiga.

All the programs in this book are available on a companion disk. See the coupon in the back for details.

ISBN 0-87455-041-6