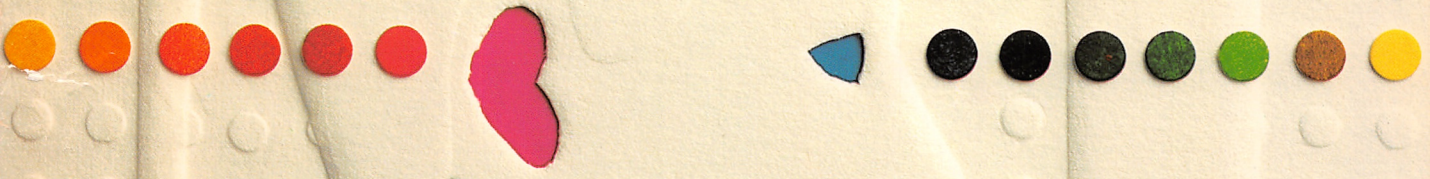
SOUND & GRAPHICS FOR THE IBM PCjr

James E. Kelley, Jr.,
author of *The IBM/PC Guide*



How to create spectacular shapes, 3-D pictures, animation, music and sound effects in the BASIC language

SOUND & GRAPHICS FOR THE IBM PCjr

SOUND & GRAPHICS FOR THE IBM PCjr

James E. Kelley, Jr.

A Banbury Book

Published by
Banbury Books, Inc.
353 West Lancaster Avenue
Wayne, Pennsylvania 19087

ISBN: 0-88693-068-5

First printing—October 1984
6 5 4 3 2 1

Printed in the United States of America

DISCLAIMER OF ALL WARRANTIES AND LIABILITIES

The author and publisher make no warranties, either expressed or implied, with respect to this book or with respect to the programs or the documentation contained herein, their quality, performance, merchantability, or fitness for any particular purpose. The author and the publisher shall not be liable for any incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or the use of materials in this book.

TABLE OF CONTENTS

PREFACE

Sound & Graphics for the IBM PCjr is dedicated to your having fun. It explores BASIC language features of the PCjr that are not available with the larger IBM PC.

FUN WITH SOUND

You know that sound had a great impact on the way movies developed. They became more exciting, more creative, more lifelike. The same thing happens when you add sound to computer graphics. The clacks, bleeps and whistles that accompany your robot monster across the video screen lend reality to the imaginary event. So it's important that you know how to generate sirens, motors, chatters and other noises that you can add to your programs.

But creating sound effects is just the beginning. You can also reproduce your favorite melodies and do experiments in harmony and musical composition. You can even play one of the parts as the PCjr gives a duo, trio or quartet performance. I'll show you how. If you're a nature lover, you can experiment with generating birdcalls.

FUN WITH GRAPHICS

PCjr has a special language for drawing, with which you'll learn to create any shape you like. You'll also learn how to move these shapes about on the video screen to do animation. Then you'll color them and enhance their movements with sounds and music.

WHAT HAVING FUN MEANS

To me, fun with a computer means "discovery"—trying endless permutations of a sound and graphics idea to see what interesting things they produce. This book will involve you in a seek-and-find process that shows you how to make your own discoveries.

WHAT YOU NEED TO KNOW

The book is written in conversational style for the novice. Of course, the more you already know about your IBM PCjr, the better. But for the most part, the text assumes you just took the computer out of the boxes it came in.

HOW THE MATERIAL IS INTEGRATED

The first two chapters explain the fundamentals of working with the PCjr—the things you absolutely need to know to get through the book. These beginning steps are biased toward our subject of sound and graphics.

The next couple of chapters are devoted exclusively to sound, beginning with synthesizing real and not-so-real effects, then progressing to music.

Next comes the more difficult topic of computer graphics, which includes integrating graphics with sound. This is a rich subject with many complications and special techniques. The math used in professional systems is overwhelming. This book will limit the technicalities to the most simple, but if the going gets slow at times, just hang in there. You'll soon learn all you need to know to get by, and you'll have fun while you're doing it.

The lion's share of every chapter involves stuff you need no specialized background to work with. However, there are a few places where some unexplained advanced BASIC material is involved. You'll be able to key this material into PCjr, but you'll have to try another book to get the background in BASIC for the PCjr. After you've gone through Chapters 1 and 2 of this book, you can start at Chapter 3 of my *IBM/PC Guide* (Dell/Banbury, 1983), which takes you through all the fundamental techniques of programming in BASIC. With minor exceptions, the material there is compatible with the PCjr.

MY PCjr

It may help you to know what equipment I used to write this book. I have a 128K PCjr with one disk drive, utilizing the Disk Operating System (DOS) 2.10, and Cartridge BASIC.

For monitors I have a SONY Trinitron TV and a LEEDEX Video 100 black and white video monitor attached. I wanted to see the effects produced on each monitor so that I could warn you if there were any unacceptable differences.

I also have an IBM Compact Printer to make printed records of my computing projects.

To see what cassette users experience, I attached an audio cassette recorder to the PCjr system.

Though I've tried to make the elementary aspects of this book work with Cassette BASIC only, most of PCjr's sound and graphics power resides in Cartridge BASIC. So if you don't have Cartridge BASIC and you want to do sound and graphics, you'd be well advised to buy it.

ACKNOWLEDGMENTS

I want to thank my son Timothy P. Kelley for helping me debug the manuscript. His efforts have enhanced the final product in many ways.

And special thanks to my daughter Genevieve C. Kelley for arranging and giving me permission to use one of her many musical compositions for the PCjr. I hope you'll enjoy listening to Jenny's "Summer Days/Summer Nights" as much as I do.

CHAPTER 1
WORKING ALL THE PARTS TOGETHER

Objectives:

It's important to find out how all the principal parts of your IBM PCjr relate to one another. By covering essential procedures that you must know to operate the equipment, you'll quickly learn how PCjr does things.

1. FOOLIN' AROUND

Sit down at your PCjr. You may already know the procedures this chapter describes, but why don't you read through it anyway, just to be sure.

CAUTION: If you haven't done so already, it would be a good idea to become familiar with your *Guide to Operations* manual, especially Section 6 on testing your equipment. You want to be sure the sound, color and other functions of your machine are working properly. In addition, if you have a disk drive, be sure to run the program *Exploring the IBM PCjr*, which is on the disk inserted in the cover of the *Guide to Operations*. Even if you haven't done all this yet, don't worry. You'll cover what you need to know as you go along.

Don't be discouraged if you must "hunt and peck" your way for a while. And don't be intimidated by all the mystery about computers. By the end of this chapter you'll be able to make all the principal parts of your system operate as if you were an old pro. As you read, remember to look up the references cited in the book. This practice will give you an overview of what you're doing, and help you learn to find your way in the manuals.

Powering Up

This book assumes that you have all the pieces of your PCjr hooked together correctly. Your dealer should have shown you how to do it.

Powering up is usually easy.

1. If you have a disk drive, disengage it by flipping the little lever on the front clockwise until it's horizontal.

2. If you have Cartridge BASIC, be sure the cartridge is removed from its slot. We're going to work with Cassette BASIC for a while.

3. Turn your TV power switch ON. Or, if you're using a video monitor with a separate power switch, turn it ON.

4. Locate PCjr's little red power switch on the left side of the System Unit at the top rear edge. Now flip PCjr's power ON!

CAVEAT: Some earlier personal computers had the nasty habit of destroying recorded information on diskettes if you turned power off

with the disk drive doors *closed*. This should not be a problem with the PCjr. However, I make it a habit to open the disk drive doors before turning power off, and I recommend the habit to you—just in case.

First Actions

When you turn the PCjr on, the IBM logo appears on the video screen. The PCjr then does a number of internal checks. The whole startup process should take fifteen to twenty seconds. If thirty seconds or more pass without some activity, turn the power switch OFF. Take your cues from the testing procedures in your *Guide to Operations*, Section 6.

CONTACT YOUR DEALER WHEN YOU CAN'T FIX THE PROBLEM YOURSELF!

When things go well, a message appears on the screen, identifying a version of Cassette BASIC, a copyright notice, the number of bytes of free memory (think of a byte as a letter or other typewriter character) and the "Ok" prompt for BASIC. Below the prompt is a blinking cursor (the underscore sign).

Using the TV or monitor controls, adjust the video picture to your satisfaction for color, contrast, brightness, and horizontal and vertical hold.

Showing up at the bottom of the screen is some other stuff, which relates to the special functions F1-F5. There are also functions F6-F10, but they are hidden from view when you power up. More about them later.

A Built-in Demo

You have probably already discovered the built-in *Keyboard Adventure* (see Section 2 of the *Guide to Operations*). The program runs when you power up with Cassette BASIC and then press the Esc(ape) key (in the upper left corner of the keyboard) before pressing any other key.

If you have not run this *Keyboard Adventure* before and you're new to computers, it would be worth spending the fifteen minutes it takes to run through it.

NOTE: The *Keyboard Adventure* is not accessible if you have Cartridge BASIC installed.

Positioning the Screen

If the right or left edge of the information on the screen extends beyond or comes too close to the border, try this method of adjusting the horizontal positioning:

Locate <Ctrl> (for Control key), the leftmost key on the third row. Also locate <Alt> (for Alternate Control key), just to the left of the space bar, bordered in blue on three sides. Got them?

Okay. At the lower right edge of the keyboard are two keys with white horizontal arrows above them, and <PgUp> and <PgDn> below. Now for the trick.

With your left hand, hold down <Ctrl> and <Alt>. Watch the screen as you strike either one of the two horizontal arrow keys. You should see the screen shift left or right, depending upon which arrow key you pressed.

The Keyboard Unit

Look at the Keyboard Unit for a couple of minutes. What do you see? You've already noticed that it's very much like the standard (QWERTY) typewriter keyboard, but with a few additional keys (the name QWERTY is derived from the first row of letters).

The principal character associated with each key is shown in white just above and to the left of each key. The associated upper-case characters are to the right in black. For some keys, there is a third character or function, which is indicated just below the key in a splash of color (e.g., the F1-F10 functions of the top row of the keyboard).

As you start to become oriented to the keyboard, you might want to use your *Guide to Operations* manual, Section 3, "The Keyboard," as you go through this discussion.

The Escape Key

Do you still have BASIC's "Ok" prompt and the blinking cursor on the screen? Press a few letter and number keys to see what happens.

Notice how the cursor moves along as you type, showing where the next character typed will appear.

On the left end of the keyboard, top row, next to the "1," is the Esc(ape) key. Press <Esc> to see what happens on the screen.

How about that! It erased your typing and placed the cursor at the left margin.

Do some more typing and erase it with <Esc>.

The Shift Keys

The Shift keys are in their usual typewriter keyboard positions with the bottom row of letters. In addition to being labeled "Shift," they are symbolized with an upward-pointing, hollowed-out arrow.

Hold down a Shift key and press some letters and numbers.

Capitals and special upper-case characters should appear on the screen. Try the other special characters on the keyboard with the Shift key (colon, quote, etc.). Just like a typewriter!

Type a zero. Now type a capital letter O next to it. Note how the zero has a slash through it so you won't confuse the two characters.

Also note that the number 1 (top left) and the lower-case l are distinct characters. If you're used to using the l for a 1 you'll have to break the habit.

The shift lock key is located just to the right of the space bar and is designated <CapsLock>.

Press <CapsLock>. Don't be surprised that it does not lock as on a typewriter. It's a toggle switch. Every time you strike it, you switch between capitals and small letters.

Press some letter and number keys for different settings of <CapsLock>.

Only letters appear as upper-case characters when <CapsLock> is in effect. To get the special upper-case characters, you must always use the Shift key.

Set <CapsLock> for upper-case letters. Hold <Shift> down and strike some letters. How about that! It "unshifts" them. This may confuse you for a while!

Backspace

The Backspace key is on the top row, right. It's labeled with an arrow pointing left and the word "Backspace." Press <Backspace> a couple of times and watch the cursor back up and erase characters you've already typed.

Start typing again to see what happens.

Don't forget! By hitting the Esc key (top left) you can erase the line where the cursor lies.

The Quick Brown Fox . . .

Now start typing again, just as if you were operating your typewriter. Don't worry about errors. If you must correct them, use <Backspace>. Type:

"Mistress Mary, quite contrary, jumped over the garden into the Wicker Works where Jack Benimble lit a flame deep in the Heart of Texas by the Old Corral as the Masked Man with the silver bullets rode off into the sunset . . ."

Don't stop in the middle. Keep typing so you can get used to the idea.

If you typed long enough, you saw the line of characters automatically wrap around to the beginning of the next line when eighty characters had been typed, but without hyphenating properly. (Hyphenating English words is a tough programming problem. At this point you can't do it with a computer, but need to consult a dictionary.)

<Enter> Starts Things Moving

Below the <Backspace> is a large key shaped like a backward L. This is the Enter key. It corresponds to, and operates somewhat like, the Carriage Return key on my IBM typewriter.

Press <Enger> to see what happens.

Did you get a "Syntax error" message? The PCjr doesn't like the material you typed. You get this message whenever a BASIC grammatical rule is violated. You'll learn them all eventually.

Try pressing <Enter> several times. Notice how the lines scroll upward as the cursor reaches the last line of the screen. (Can you figure out how many lines there are on the screen? How about columns?)

Now hold down any letter or number, in Shift or non-Shift mode. After a short delay it should start reproducing the character across the line and down the screen, for as long as you hold it there. This saves rapid pecking at keys when you want to generate strings of the same character.

Keep Tabs

There is a Tab key located just to the left of the Q key. In lower-case or non-Shift mode, the motion of the Tab is to the right.

Press <Tab> a few times. Note how it moves the cursor eight character positions each time.

The left-moving <Tab> (upper-case mode) is not functional unless you write special software to make it work.

Cleaning the Slate

By this time the video screen should be filled with your experimental activity.

Type: <Esc>	**(Cancel current line)**
Type: CLS <ENTER>	**(You may use lower-case letters too!)**

Whenever you want to clear the screen while in any BASIC mode you may type CLS. The cursor will always jump to the upper left-hand corner of the screen and give you the BASIC prompt ("Ok").

See Section 4 of your BASIC manual for more on CLS. Section 4 explains all the BASIC instructions. They are listed there in alphabetic order. Get in the habit of checking what I say about these commands by looking them up in Section 4. Eventually the BASIC manual must become your principal source of information on the BASIC commands and functions.

Put COLOR in Your Life

It's time to try a trick. First we need a screen setup command, so

Type: SCREEN 0,1 <ENTER> **(Switch to color text mode)**

Now for the trick.

Type: COLOR 0,15:CLS <ENTER>

Surprise! Now whatever you type will appear as dark on a light background.

Type: "Anything you want, but lots of it!"

To get back to light letters on a dark background

Type: COLOR 15,0:CLS <ENTER>

That's a relief. The light background is hard on the eyes.

By the way, the colon (:) is a symbol that lets you string BASIC commands together and execute them with just one <Enter> command. You could have performed each step separately. Try it.

Type: COLOR 0,15 <ENTER>
Type: CLS <ENTER>
Type: COLOR 15,0 <ENTER>
Type: CLS <ENTER>

We're going to use COLOR a lot, so start familiarizing yourself with the command in Section 4 of the BASIC manual. Now let's see what happens when we use some color.

Type: COLOR 15,1:CLS <ENTER>

With a color set, you should have high-intensity white letters (foreground) on a blue field (background). A black and white TV set or video monitor should show white letters on a dark, textured field.

In color text mode (i.e., when you've executed the SCREEN 0,1 command), there are sixteen colors you may use for foreground (the letters) and background (the field), as shown in Figure 1.1.

You can even put a border around the color text mode screen. Try this:

Type: COLOR 4,6,1:CLS <ENTER>

The last digit specifies the color of the border.

0 black	8 gray
1 blue	9 light blue
2 green	10 light green
3 cyan	11 light cyan
4 red	12 light red
5 magenta	13 light magenta
6 brown	14 yellow
7 (dirty) white	15 high-intensity white

Figure 1.1: Foreground and Background Colors for Color Text Mode

Try some experiments with COLOR, substituting different numeric values for foreground, background and border. That is,

Type: COLOR [frgrnd #],[bkgrnd #],[border #]:CLS <ENTER>

You'll have some fun with this exercise, and you'll discover some limitations (e.g., only 0-7 count for background) as well as lots of bilious color combinations. With black and white video you'll see the various textures you can get—a scale of grays.

Don't forget you have to do SCREEN 0,1 before COLOR will work in color text mode.

Another Way to Clear the Screen

Put some more junk on the screen and I'll show you another trick. First hold down the Ctrl key and strike the letter L. Voila! This way of clearing the screen is even simpler than CLS. I use it almost exclusively, but it's really of value only for manual (command level) operation. When you start writing BASIC programs, you'll normally use CLS to clear the screen.

Canceling COLOR

When you want to cancel color text mode, just

Type: SCREEN 0,0 <ENTER>

Go ahead, cancel COLOR right now.

2. USING A CASSETTE RECORDER WITH PCjr

If you do not intend to use an audio cassette player with your PCjr, skip to Section 3, which takes up the subject of using diskettes.

The rest of this book ignores the use of audio cassettes for storing programs and data in favor of diskettes and the Disk Operating System (DOS). But don't be put off. Most of the material here can be done using a cassette recorder, provided you also have Cartridge BASIC.

In this section you'll learn how to use a cassette recorder with PCjr, and how to interpret the rest of the book so you can adapt the material for tape storage.

100ssume your dealer has shown you how to attach a cassette recorder to the PCjr and carried you through the first few steps.

You've Got to Have the MOTOR Running

Are you still in Cassette BASIC? Put a blank audio cassette in your recorder. Try to make the tape move by depressing the PLAY switch. As you see, it doesn't work. Unlock the PLAY switch and try FAST FORWARD. That's no good either. RECORD and REWIND give the same negative results.

For your PCjr programs to control tape motion when one of these switches is depressed, they must be able to turn the motor on and off. Otherwise, the recorder will run off the end of the tape prematurely.

Try this. Depress the PLAY switch once more. Now, watch the tape spindles when you

Type: MOTOR 1 <ENTER>

Is the tape moving?

Type: MOTOR 0 <ENTER>

Is the tape stationary now?

As you see, you can switch tape movement on and off with the 1 and 0 tags to the MOTOR command, but you can also toggle the motor on or off without the 1 or 0.

Type: MOTOR <ENTER>

Now the tape should be moving. Unlock the PLAY switch to stop the tape motion, just as you would to stop music from playing. Now, depress REWIND. You see that with MOTOR active, the recorder's switches behave as you would normally expect.

Saving Stuff on Cassettes

To get past the leader of the tape, FAST FORWARD a little, and STOP, just as you would if you were going to start a recording session.

Now, toggle the MOTOR off, i.e.,

Type: MOTOR <ENTER>

Next, clear memory, i.e.,

Type: NEW <ENTER>

and type in a remark.

Type: 100 ' This remark defines the first file <ENTER>

To see that what you just typed is in memory,

Type: LIST <ENTER>

Let's SAVE this remark on the cassette. Depress the RECORD and PLAY switches just as you would to start recording. With the MOTOR off, the tape should remain stationary. Okay, next step.

Type: SAVE "FILE-1" <ENTER>

(Lower-case letters are okay too.) The SAVE manages the motor without your help.

When the file has been recorded, tape motion will cease and you'll have the "Ok" prompt on the screen.

Let's try it again. First change the file by substituting a new remark for the old one.

Type: 100 ' Here's a new remark for the second file <ENTER>

Check that it's in memory with LIST.

Proceed as before. Did you leave the RECORD and PLAY switches depressed? If not, depress them now.

Type: SAVE "FILE-2" <ENTER>

Once again the information is written on the cassette.

LOADing Files from Cassette

Now let's look at what we've recorded. Unlock the RECORD-PLAY switches. Then toggle MOTOR on, i.e.,

Type: MOTOR <ENTER>

Now, REWIND and STOP the recorder. Finally, toggle MOTOR off (i.e., type: MOTOR <ENTER> once more).

To verify that the computer read information from the cassette, clear memory with NEW, and test this action with LIST. Now we're ready to go. Depress PLAY and

Type: LOAD "FILE-1" <ENTER>

LOAD is like SAVE and controls the motor by itself.

When FILE-1 is located, you're given the message "FILE-1.B Found." The file is then read into memory. When all information has been read, tape motion ceases. (The .B is a filename extension indicating this is a BASIC file.)

You can check whether or not the file actually got into memory.

Type: LIST <ENTER>

If everything looks okay, LOAD the second file, i.e.,

Type: LOAD "FILE-2" <ENTER>

When the file is loaded, use LIST to check that FILE-2 was read correctly. Then unlock PLAY, and rewind the tape to the beginning.

Disk Commands You Can't Use with Cassettes

There are only a few common disk commands that you can't use with cassettes. These are:

FILES	List all the files on a diskette
KILL	Delete a file from a diskette
NAME	Change the name of a file on a diskette

Other exceptions will be rare, but be on your guard. If what I'm doing at a particular time doesn't seem to make sense for cassette use, don't try it.

Listing Files on a Cassette

You can easily find out what files are on a cassette—attempt to LOAD a nonexistent file. Is the cassette with which we've been experimenting rewound? If so, depress the PLAY switch. (If the tape starts to move, you know the MOTOR is active, which it shouldn't be.) Then,

Type: LOAD "NO-FILE" <ENTER>

As the LOAD command searches for NO-FILE, it prints on the screen the names of the files it finds, noting that it is skipping them. You can write the names down, or use the <Fn-Echo> command we'll cover later to get a printed list. You might also note the recorder's counter value so you can FAST FORWARD to that file before initiating the LOAD. It's the same strategy you use to find your favorite tune on a cassette.

Test Programs Cassette

As we go along I'll be telling you to SAVE programs on your *Test Programs* diskette. If you're using cassettes rather than diskettes, you'll use a *Test Programs* cassette, obviously.

Because of the limited capability of cassettes, your strategy has to be a little more involved. You should have two cassettes: one a working cassette, on which you develop a single program at a time, and the other an archive cassette, on which you store finished programs and files.

Here's how the process works: Position the archive tape by LOADing the last file on it into memory. Then remove the cassette without rewinding, and replace it with your working programs cassette. Next, LOAD the program to be archived from the working programs tape, and SAVE it on the archive tape. A word of caution—be careful, when recording data, that you don't destroy existing files. There are no good protection schemes, such as those that work for disk files, that will save you from your own mistakes. Have you figured out how to stop the search for the nonexistent NO-FILE? It's easy.

Type: <Fn-Break>

You'll get a "Device Timeout" error and the "Ok" prompt. Toggle MOTOR on and REWIND the cassette to return to familiar ground.

3. STARTING UP WITH DOS

I've taken a series of essential procedures from the PCjr reference manuals to give you a feel for the equipment. These operations are done so frequently that they will become as natural as shifting gears on your car. Burn them into your brain right from the start by trying them out as we cover each one. Two or three times in a row, in fact.

As you get a feel for what's happening, refer to your manuals for more details. And make notes! In particular, you should be sure to read the pamphlet entitled *Disk Operating System User's Guide*. The abbreviation DOS (Disk Operating System) is pronounced "doss."

You Need Cartridge BASIC

For what follows, you must load Cartridge BASIC. You needn't turn off PCjr's power to install the cartridge, though I'm chicken about such things and usually do. The effect of inserting or removing a cartridge while power is on is the same as turning power off and on again.

Start by holding the BASIC cartridge so you can read it, label side up. Then insert it into either one of the two slots below the disk drive. I usually use the left slot. You must use firm pressure. You'll feel the cartridge snap into place.

Read Section 7 of the *Guide to Operations* for more about cartridges.

Warm Start of DOS <Alt-Ctrl-Del>

If the PCjr's power is ON, proceed as follows:

1. Locate your DOS diskette. It comes inside the back cover of the Disk Operating System manual.

2. Put the DOS diskette in the disk drive. Then turn the disk access lever counterclockwise (pointing down).

Be sure to insert the diskette correctly—label side up, exposed area away from you. As you insert the diskette, the notch should be on the left. Push it in as far as it will go.

NOTE: If the diskette has an untabbed write-protect notch (your master copy should be notchless), put a metallic tab on it. Make it a general practice never to put an untabbed disk into a drive unless you fully expect to write on it. Even when you know what you're doing, you can accidentally write over important data and programs.

3. With your left hand, hold down the Alt and Ctrl keys on the lower left side of the keyboard. Then press the Del key on the lower right of the keyboard.

Go ahead! Try this "warm start" procedure. (The screen clears, and the disk drive whirs with its red light on.)

When things go well, you are told the current day and date. If you don't have a clock calendar in your PCjr, the day and date will be incorrect, probably Tue(sday), 1-01-1980 (January 1, 1980). If the date is the correct one, just press <Enter>. Otherwise you have the opportunity to answer the message: "Enter new date:".

Using numbers on the top row of the keyboard, type in the date in numeric form (e.g., Dec. 25, 1984 = 12-25-84).

To register the date inside the PCjr,

Type: <ENTER>

If, in typing the date, you make a mistake in the format, the PCjr will catch it and ask you to enter the date again.

Next you'll be told the current time in the form: [hour]:[minute]:[second]. The hour runs from 0 (midnight) to 23 (11:00 p.m.). Seconds are specified to two decimal places. Again, if you don't have a built-in clock, the current time is probably incorrect. You may type in a new time if you like, or simply press <Enter> to accept the one shown, even if incorrect. I don't worry about the seconds unit but type a time close to the minute shown on my watch (e.g., 13:15 for 1:15 p.m.).

NOTE: It's a good idea to enter the correct date and time when you power up. Whenever you save files and data on disk, the currently recorded date and time are stamped in the disk directory so you can tell when the file or data was generated.

If all goes well, you will see a copyright notice and DOS Version number, and a little prompt of the form "A>." Whenever you see this DOS (or just SYSTEM) prompt, the SYSTEM is waiting for you to give it a command of some sort.

The "Alt-Ctrl-Del" procedure is also called SYSTEM RESET.

Cold Start of DOS

Here's the way to begin when computer power is OFF. STOP!! DON'T TURN OFF THE COMPUTER! You'll have a chance to do a "cold start" later on. For now, just read about it.

1. Put the DOS diskette in the disk drive. Don't forget to turn down the latch.

2. Flip the *red* power switch to ON. If your monitor (or TV) has a separate power switch, turn it on too. I often forget this myself.

If PCjr is working properly, you will get the current date and the "Enter new date:" message. Proceed from here as with the warm start.

4. BASIC VERSUS DOS MODES

Perhaps at this point you're a little confused. You started out in Cassette BASIC mode, and now, all of a sudden, you're in DOS (or SYSTEM) mode. And then there's Cartridge BASIC.

The explanation, though perhaps a little long, is this:

For a modern computer to be used in any practical way, it has to be supplied with an operating system. Such a system makes it relatively easy to have all the computer units (keyboards, video screens, printers, disks, memory, communication channels, etc.) talk with each other in a well-ordered way.

An operating system is a special computer program written in machine language, but as far as we are concerned, it is part of the computer hardware. In fact, in the early days of computers, the func-

tions of operating systems were built into the hardware. Today that control hardware is eliminated, which contributes to the lowered prices of computers.

Your PCjr comes supplied with a built-in operating system called Cassette BASIC. As we played with the keyboard and controlled the screen, we made use of this operating system. Cassette BASIC also includes a "programming language" aspect, which most operating systems, including PCjr's DOS, do not have. CLS, for instance, is a BASIC programming language command.

Coincidentally, CLS also clears the screen while you have the DOS prompt (i.e., A>). Try typing CLS <Enter> with the DOS prompt showing. BASIC uses some of the functions of DOS, but in ways that are specified differently from those of DOS. It would be less confusing if they were the same. But design compromises intervened.

As a BASIC programmer you will not use many of the facilities of DOS, except as implemented through BASIC. You will use DOS a great deal if you program in assembly language, Pascal, FORTRAN or "C."

The facilities of DOS that I cover here and in later chapters are essential to you for exploiting sound and graphics on PCjr.

5. LOOKING AT DISKETTE FILES WITH DOS

Snooping in a Directory (DOS Mode)

The DOS diskette has information written on it in what are called files. These files are named in a directory on the diskette. You can look at it if you like.

Type: DIR <ENTER> (You can type lower-case too!)

Whoops! Some of the information scrolled right off the top of the screen. I'll show you how to capture that lost information in a minute. First let's see what directory information shows up on the screen.

Each line corresponds to a file on the diskette. The first column gives the file's name, like "EDLIN," "MODE" or "PRINT." The second column is an extension of the file name, which may be used to classify files (e.g., .COM = a command file, .BAT = a batch file, .BAS = a program in BASIC; etc.). (See Chapter 2 of the *DOS User's Guide* for some details on file names.)

The third column tells how much space the file takes up (in bytes). The fourth and fifth columns give the date and time it was created. If you don't have an installed clock or if you don't type in the date and time at startup, the date-time stamp in the directory is going to be goofy.

You can have up to sixty-four file names in a directory.

For more on DIR, see Chapter 4 of the *DOS User's Guide*. (As we'll see, DIR is accomplished in BASIC with a command called FILES.)

Screen Freeze (Fn-Pause)

Now let's see what's at the beginning of the directory. Locate the "Fn" key (the Function key in the upper right corner). Got it? Now, locate the "Pause" key (the same as the Q key). You can *freeze* the screen (or suspend operations) while the directory is scrolling along by pressing the Fn key and then pressing the Pause key. You may need to practice a little to get the coordination down.

Try it now.

Type: DIR <ENTER>
Now quickly: <Fn-Pause>

Did you get it? If you missed, try again.

To unfreeze the screen, press any key (well, almost any). You can freeze and unfreeze the screen on one scrolling pass as many times as you wish.

This screen freeze function works in both BASIC and DOS modes.

Function-Break (Interrupting a Program)

Now locate the "Break" key—it's the B key. If you press <Fn> followed by <Break> while PCjr is operating, you interrupt program execution and return to DOS (or BASIC, if that's where control resides at the time).

Let's try <Fn-Break>. Read the directory (i.e., Type: DIR <ENTER>). As the directory scrolls by, perform <Fn-Break>. Note that when you do this operation you cannot start the directory scrolling again by hitting a key as you could with <Fn-Pause>.

6. SOME BASICS WITH BASIC

Down to BASIC from DOS

While the DOS prompt (i.e., A>) is showing,

Type: BASIC <ENTER>

Almost immediately the screen changes to a new copyright notice and announces a version of Cartridge BASIC and the number of free bytes.

Syntax and Other Errors

While using this book you will from time to time make typing errors. As you saw earlier, when you perform the <Enter> command, you sometimes generate a "Syntax error." If this happens, just retype the command correctly and go on.

You may also experience other errors that will at first puzzle you. Be sure to make a record in your notebook of the errors you encounter so you can begin developing a sense of their causes and eventually learn how to correct and avoid them.

More Snooping in a Directory (in BASIC Mode)

You.can also call up the diskette directory from BASIC (we did it in DOS with DIR). Try this.

Type: FILES <ENTER>

Well, this time the directory is arranged a bit differently, isn't it? (Probably so you can see all of it at once.) Only the filename and extension are given. You have to use DIR in DOS mode to get a file's size and date of last change.

LOAD a BASIC Program

Most files on the DOS diskette are system programs. However, there's a *Supplemental Programs* disk that comes with DOS, inside the back cover of the DOS manual. Remove the DOS diskette and insert the *Supplemental Programs* diskette in the disk drive. When you have the diskette in place,

Type: FILES <ENTER>

Scan the directory to see what's there. Do you see the BASIC file called BALL.BAS? Let's read it into the PCjr's memory.

Type: LOAD"BALL.BAS" <ENTER>

The disk drive will whir a bit as the program named BALL.BAS is LOADed in. When the LOAD is completed, an "Ok" is printed on the screen and PCjr awaits your next command.

LIST a BASIC Program to Screen

You can look at the program read into memory by transferring it to the screen this way:

Type: LIST <ENTER>

Wow! Look at it go. Too much information for one screenful. To catch the listing as it scrolls by, apply the screen freeze (Fn-Pause) procedure as you did earlier when the diskette directory scrolled by.

I don't want to stop now to read this program with you. There would be too many things to explain at this point. But you'll notice it's peppered with English words. So maybe there's hope that BASIC programming won't be so bad!

RUN a BASIC Program

Since the program BALL.BAS is in memory, let's RUN it to see what happens:

Type: RUN <ENTER>

As images come up on the screen you will be instructed what to do. When you tire of watching the ball bounce, hit <Esc> in the upper left section of the keyboard. The ball has to return to the left-hand wall before <Esc> is effective. When <Esc> executes, you are returned to BASIC command mode.

Wide and Narrow Screens

Let's try a little trick.

Type: WIDTH 80 <ENTER>

What happened? Does your screen look unusual? Let's see if we can bring up the program with the screen in this condition.

Type: LIST <ENTER>

There's the program BALL.BAS, which we LISTed before, but it looks much different now. The letters are narrowed, and practically unreadable on my TV screen. In fact, the left end of each line is hidden beyond the edge of the screen. But I can fix that by holding down the Alt and Ctrl keys and striking the Right arrow once or twice. (You remember that trick, don't you?)

Notice also that there are more characters across the screen now—eighty, in fact. As you gather, that's the whole point of WIDTH, to change the number of characters printed on a screen line. It's either forty or eighty.

On my black and white monitor, legibility is not a problem, because the monitor is designed to produce higher fidelity pictures than a TV. If you want to run certain kinds of applications on your PCjr that require an eighty-character screen (e.g., WordStar and other word processors), you may want to consider buying a video monitor.

By now you've no doubt guessed how to change the screen back to forty characters:

Type: WIDTH 40 <ENTER> (Be sure to put a space between H and 4)

Now LIST the program in memory again. Legibility should be much better.

Watch Your Spaces

Often PCjr is sensitive to the way you use spaces. In English, for example, "thick set" does not mean "thickset." If PCjr "spoke" English, it would probably class "thick set" as a syntax error. Similarly, "any body" and "anybody" conjure up different ideas.

In like manner, you need to be sensitive to PCjr's use of spaces to define the boundaries of the words and the character or number groups it can recognize.

Try this again:

Type: WIDTH80 <ENTER> (Omit the space)

Did you get a syntax error? There are cases where PCjr will do one thing if a space is present and another when it's not.

There is no general rule about spacing, but if you're sensitive to the need, you'll get the hang of it in no time.

LLIST a BASIC Program to Printer

Thus far, all the information you have seen on the screen has been allowed to disappear forever once you've finished with it. You can change that by using the printer, if you have one. I have the IBM PC Compact Printer, and that's the only one I'm going to refer to in this book. If you have a different printer, you'll have to adapt what follows to its particular characteristics.

Before we start, be sure the printer is attached correctly and paper is threaded through it properly. (See your printer manual for directions. It should have come in the box with the printer. You're supposed to insert the little manual into your *Guide to Operations* manual.)

DON'T FORGET! IF YOU HAVE TO MONKEY AROUND WITH CABLES, BE SURE ALL POWER IS OFF AND THE POWER CORD IS PULLED FROM THE WALL SOCKET.

If everything is all set, turn the power to the printer on. Then, using the paper feed button, adjust the paper to the position where you want print to begin.

Is BALL.BAS still in memory? Okay,

Type: LLIST <ENTER>

(That wasn't a stutter! There are two L's in this LLIST command.)

You should hear the printer start to print the BALL.BAS program as soon as you hit <Enter>. It will print about a foot of paper. When it is finished, the BASIC "Ok" will pop on the screen again.

You can stop LLIST with <Fn-Break>, or by turning the printer OFF.

Let's try an experiment while BALL.BAS is still in memory.

Type: WIDTH "LPT1:",20:LLIST <ENTER>

Stop the printing after a few lines by hitting <Fn-Break>.

If the printer goes on for a bit after you execute <Fn-Break>, don't worry. Just hit <Enter> again. With this you can now control the width of the text by specifying the printer (designated "LPT1:")and the line length (20 here) before calling for a LLIST.

To reset the printer to produce eighty-character lines,

Type: WIDTH "LPT1:",80:LLIST <ENTER>

Remember, you can stop with <Fn-Break>.

Program Function Keys

Now you'll learn some short cuts.

Type: KEY ON <ENTER>

Along the bottom of the screen are the words that pop up when you do a cold or warm start.

You should recognize at least three of those words: LIST, RUN and LOAD. They are functions you've just learned. There are some more. Once again,

Type: WIDTH 80 <ENTER>

Now you can see the full complement of ten program functions available.

Notice that the word KEY has the number 9 to its left. Try this:

Type: <Fn-9>

That is, push the Fn key followed by the 9 key. The word KEY should appear on the screen. Now,

Type: OFF <ENTER>

and the line of special function key prompts disappears. To bring it back,

Type: <Fn-9> ON <ENTER>

Now

Type: <Fn-1> <ENTER>

and BALL.BAS is listed to the screen again. (Stop with <Fn-Break>.)

Experiment with the other program function keys to see what happens. For example,

Type: <Fn-9> <Fn-1> <ENTER>

This combination does an inventory of the function keys for you.

The main purpose of these special function keys is to speed operation and reduce typing errors, but you have to memorize them first. In the meantime, why don't you make a printout of them you can keep handy. You can do this with "Little Sir Echo."

Little Sir Echo

Turn on the printer, and

Type: WIDTH 40 <ENTER> **(Restore screen to legibility)**

Type: <Fn-9> <Fn-1>

We're almost set up. Locate the "Echo" key—it's the E key. Then

Type: <Fn-Echo> <ENTER>

As the function key functions are listed on the screen, they are echoed to the printer. But when the operation is over, we've got to toggle the echoing off, so once again

Type: <Fn-Echo>

This echoing function also works while you're in DOS mode. Try it sometime when you'd like a printout of the disk's directory (when you do a DIR command).

7. A PAUSE THAT REFRESHES

Before I throw something new at you, let's practice some of what you've already learned. The *Supplemental Programs* diskette is still in the drive, isn't it?

Type: RUN"SAMPLES" <ENTER>

(Notice that you don't have to LOAD a program from disk before you can RUN it. And if you know its extension is .BAS, you needn't type the .BAS.)

See if you can operate SAMPLES by following the directions given on the screen. If you can't figure out the directions, make a note about your problem. It will be a helpful reminder to you later on when you write your own programs and must take the user's ignorance into account.

Now you're going to LLIST to your printer another one or two of the files on your *Supplemental Programs* diskette that you can load into memory using the LOAD command (it's not possible to LOAD a couple of the system programs there). You might pick the programs that interested you as you ran SAMPLES for later study. The general procedure is:

LOAD"name.ext" <ENTER>

LLIST <ENTER>

Be sure the printer is on.

As you learn more, study these programs to see what kinds of techniques the authors used to produce the effects you see on the screen. There is no better way to grow as a programmer than to read others' works.

8. SCREEN DUMP (Fn-PrtSc)

Here's a neat trick related to <Fn-Echo>. If there's a bunch of stuff on the screen, make sure your printer is on and ready to go, then locate the "PrtSc" key—actually the P key.

Type: <Fn-PrtSc>

That is, push the Fn key, then the PrtSc key. All the text on the screen is downloaded to the printer. Neat!

This function also works in DOS mode. It's a nice way to save tables, line diagrams and certain types of graphics you may construct on the screen. For instance, if you run the MORTGAGE option of SAMPLES, you can save the Monthly Mortgage Payment Comparisons table by hitting <Fn-PrtSc> while the table is showing on the screen. And you can do it while the program is operating. Operations are simply interrupted temporarily until printing is finished.

CAVEAT: <Fn-PrtSc> works only with the text-mode screen, so you won't be able to get printed copies of the graphics-mode screens on the IBM Compact Printer without a special program. However, if you have the equivalent of an Epson MX-80 printer, you can make <Fn-PrtSc> function in graphics modes this way: While in DOS, type GRAPHICS <Enter>. This action loads a special program into memory that makes PrtSc function properly to dump pictures to the compact printer. Some of the figures later in the book were printed this way, but using an IBM PC (rather than the PCjr) as host.

9. BACK TO DOS

You may have forgotten by now just how we arrived where we are. Look back to the procedure "Down to BASIC from DOS." We can reverse our steps somewhat.

Type: SYSTEM <ENTER>

The screen clears, Drive A whirs—and there's that DOS prompt again (i.e., "A>").

10. SECURITY OF YOUR DISKETTES

Backup Diskettes—Their Importance

The DOS diskette is rather important to you. Have you thought what you'd do if you put a crease in it while inserting it in the drive?

Suppose that while you're demonstrating the PCjr, your best friend drops hot cigarette ashes on it. What happens if the cat or dog plays with it or you put it on a hot radiator mixed up with some papers? There are endless ways you can destroy your diskette and the information on it, one of which is exposing it to a magnetic field, so please keep magnets away! This includes ringing telephones and working electric motors.

If you lose the DOS diskette, you may be only temporarily inconvenienced. Your dealer will probably let you copy his if you do something dumb to destroy your only copy. Or IBM may sell you a replacement at a nominal fee.

Suppose, however, that it's not a system diskette you've lost but your own programs and data that you invested many hours or dollars to assemble. Tragic, but fortunately avoidable if you maintain backup copies of your programs and data.

Making a backup copy is a two-step process using DOS. First you format a blank diskette. Then you copy the original onto it.

Format a Blank Diskette

If you have been following me right along, you have DOS loaded and its cursor blinking away. (If not, load DOS with the procedure given earlier.)

The formatting procedure for a single disk drive amounts to this:

1. Insert the DOS diskette in the disk drive and shift into DOS mode (i.e., if you don't have the DOS prompt, type SYSTEM if you are in BASIC mode, or do a warm or cold start).

2. Initiate the FORMAT.COM program on the DOS diskette, i.e.,

Type: FORMAT /S <ENTER>

You should get the message "Insert new diskette for Drive A: and strike any key when ready." Your single disk is called drive A:. If you had two disk drives, the second would be called B:.

NOTE: The /S option in the FORMAT command permits us to make a "bootable" diskette. That is, we'll be able to start up DOS with it. We won't have quite as much room on the diskette for other programs, but we won't have to change diskettes all the time.

3. Remove the DOS diskette from the drive and insert a blank diskette, or one that you're willing to erase and reformat.

4. Now, with the blank diskette in place, hit the Space or Enter key. (I usually hit one of these two keys for operations of this type, but you can hit any key on the keyboard.) The PCjr tells you it's "formatting." In about a minute it tells you "Format complete" and gives you some statistics on how disk space is being used, and how much space is available.

5. PCjr also asks whether you want to format another diskette. If you type Y (yes), you're back to inserting another blank (step 3). If you type N (no), you're back to DOS command mode with the DOS prompt. For now,

Type: N

Look at what's on the disk, i.e.,

Type: DIR <ENTER>

There's just a DOS system program called COMMAND.COM and a couple of others that are hidden, which are not important for us. You can verify that the formatted diskette will "boot" the system by trying a warm start.

Type: <Ctrl-Alt-Del>

6. Remove the formatted diskette and set it aside. (If you use a felt marker and don't press hard, you might mark a little F on the diskette label to indicate that it's "formatted.")

Try this formatting procedure youself. Proceed to format all your blank diskettes (or at least some of them).

You are going to make some backup copies shortly, and you'll need a reserve of formatted diskettes. In fact, you should never be without a freshly formatted diskette. You may run out of disk space in the middle of a program and need a pristine formatted disk.

See Chapter 4 of the *DOS User's Guide* for more on the FORMAT command.

Label Your Test Programs Diskette

At this time prepare a label that says *Test Programs* and put it on one of the newly formatted diskettes. To avoid creasing or denting the diskette, write the label before attaching it.

You will be using your Test Programs diskette shortly to store working programs and other data.

Copy a Diskette

Now you'll make some backup diskettes.

1. Put the DOS diskette in the disk drive and

Type: DISKCOPY <ENTER>

2. You're asked to "Insert source diskette in drive A:"; "Strike any key when ready." The source diskette is the one you want to copy. You will copy the DOS diskette this first time, so leave it in the drive.

3. When you're ready,

Type: <ENTER>

You'll get a message like "Copying 9 sectors per track, 2 side(s)." The DISKCOPY program is simply filling up memory with the information on the diskette. When memory is full or the end of the data is reached, you're told, "Insert target diskette in drive A:"; "Strike any key when ready." The target disk is the blank diskette onto which you will copy the source.

4. Okay, remove the source diskette and insert the target diskette in the drive. Then,

Type: <ENTER>

The data in memory is written on the target disk. If there's any more data on the source disk to be copied, you're told to "Insert the source diskette in drive A:"; "Strike any key when ready."

And so it goes until all the data on the source diskette has been copied to the target diskette. After about three passes you're told, "Copy complete."

If you get an error message saying that the target disk may be unusable, try the process again with a different blank disk.

5. PCjr also asks if you want to repeat the copy operation with another target diskette, or with new source and target diskettes. If you type Y (yes), PCjr picks up at step 4. If you type N (no), PCjr exits to DOS command mode.

Type: N

Test to see if the directory contains the programs you're expecting to see there.

Type: DIR <ENTER>

6. Remove the target diskette and label it appropriately. (This first time it's *DOS—Working Copy*.) Also place a metallic write-protect tab over the write-protect notch.

Making Backup Copies

Now try the whole DISKCOPY process again on your own, using the *Supplemental Programs* diskette. And don't forget to label and tab the diskette. Do you have any more systems or applications diskettes for which you have but one copy right now? If they have untabbed write-protect notches, tab them before you put them into a disk drive.

Make backup copies right now, then take your originals and put them in a box or envelope. Store them in a place that is out of harm's way, dust free, and neither too hot nor too cold.

From now on, do all your day-to-day operations with the working copies.

NOTE: When the information on a diskette is especially valuable, you may want to consider holding two or more backups in reserve, stored in different locations, of course!

11. POWERING OFF

It may seem a little silly to talk about turning OFF your IBM PCjr. It looks so like a TV that you'd think it only takes the flip of a switch. But there are some things you should consider.

Save Programs and Data on Diskettes First

In contrast to a TV, you can't flip PCjr back ON and find yourself in the program you just left. When you shut down, you lose what's on the screen and in memory (RAM). Always ask: What will I lose if I turn the power off?

Think before you act. It may save you hours of reconstructing lost information. Be sure to save needed information on diskettes before shutting down.

Disconnect the Wall Plugs

You know that heavy boxlike unit in the middle of PCjr's power cord? It's a transformer. Perhaps you can hear it moaning. It's using electricity even when the power switch is off. Disconnect it from the wall outlet when you shut down operations.

It's also a good idea to disconnect your computer when a lightning storm is in the area. If your power line is hit—not a very probable event—your computer and other equipment could be severely damaged. I've got my wife and kids trained to check the computer room when there are signs of a storm.

Don't Forget to Turn OFF Your Other Units

It is so easy to be in a hurry, to flip the power switch and walk away. If your monitor doesn't have a prominent POWER ON light, you can easily forget and let it run all night.

The same may apply to your printer or other equipment, unless they have noisy fans that invite you to turn them off at the earliest possible moment.

Avoid Turning PCjr ON and OFF Too Often

It's not a good idea to turn your PCjr ON and OFF frequently. Heating up and cooling down tends to fatigue the electronic components and shorten their life.

If you've been running the PCjr all morning and expect to use it after lunch, leave it ON. But be sure to save what you've been working on. Someone may fool around with the equipment while you're gone, power might be lost, or any number of unfortunate things might happen.

12. REVIEW YOUR PROGRESS

Congratulations! You have made it through Chaper 1 unscathed. You have accomplished quite a lot in just a few hours. Take a few moments to review your progress, and you'll see for yourself just how far you've come.

Now look at Figure 1.2 at the end of the chapter. It summarizes the key commands you should commit to memory. If you're unsure of any, look back in the chapter to where they are covered. You should also make a habit of looking them up in the BASIC and DOS manuals.

I hope you've been keeping good notes. Very soon now the number of details you must deal with will begin to seem overwhelming, and you will need your notes to find your way back and forth over the ground we cover so you won't become completely lost. And don't avoid recording errors you experience and data that will help isolate their causes when you analyze them.

Test Yourself

Before continuing to the next chapter, you should test yourself. You should know much of the detail behind the chapter summary in Figure 1.1. You should also be able to handle these activities:

(1) Power up ("boot") your PCjr. (Do both a cold and a warm start.)

(2) Print the directory of the Supplemental Programs diskette.

(3) Switch back and forth between DOS and Disk BASIC command modes. (BASIC and SYSTEM.)

(4) Print the directory of the DOS diskette on the screen while in Disk BASIC mode.

(5) Run a program stored on the DOS diskette (say, SAMPLES).

(6) Print a BASIC program on the printer (LLIST). Why don't you try LIST,"LPT1:" while you're at it?

(7) Format a blank diskette.

(8) Copy a diskette.

(9) Print the directory of a diskette on the printer while in DOS mode.

BASIC COMMAND	FUNCTION
CLS	Clear screen
COLON (:)	Separate BASIC commands on same line
COLOR 0,15	Print dark letters on light background
COLOR 15,0	Print light letters on dark background
COLOR 1,15	Print light letters on blue background
Ctrl-L	Clear screen
ENTER (<—')	Carriage return; enter commands/data
Esc(ape)	Delete line at cursor
FILES	Read diskette directory to screen
KEY (ON, OFF)	Toggle the screen prompts
LIST	List program memory to screen
LLIST	List program memory to printer
LOAD	Load program on diskette or cassette to program memory
LPT1	Device name of printer
MOTOR	Activate/deactivate cassette recorder motor
NEW	Clear program memory
RUN	Execute program in program memory
SAVE	Write program file on cassette
SCREEN 0,1	Switch to color text mode
SYSTEM	Transfer command control to DOS
Tab (—>|)	Tab advance
WIDTH	Change printing width of screen or printer

DOS COMMAND	
BASIC	Transfer command control to Disk BASIC
Ctrl-Num Lock	Freeze screen
Ctrl-Break	Stop current operation
DIR	Read diskette directory
DISKCOPY	Copy a diskette
Esc(ape)	Delete line at cursor
FORMAT	Format a blank diskette
Fn-Break	Stop program
Fn-Echo	Echo screen output to printer
Fn-Pause	Freeze screen
Fn-PrtSc	Dump screen to printer
MISCELLANEOUS	
Alt-Ctrl-Del	Warm start; system reset
Default drive	Drive selected when not specified

Figure 1.2: Summary of Key Commands in Chapter 1

CHAPTER 2
JUNIOR, WATCH YOUR BACKTALK

Objectives:

In this chapter you'll enhance your "hands-on" skills and learn how to edit text on the screen with the BASIC Editor. You'll also gain a pretty good idea of how to manipulate the principal controls of the PCjr and its standard units, and you'll learn a little about sound and graphics programming in BASIC.

1. SAY SOMETHING, PCjr!

One key to learning about your IBM PCjr is to *experiment freely*. As long as you don't monkey around with the electronics, or plug in cables incorrectly, or abuse the equipment physically, there is little chance you will hurt your PCjr. Don't be afraid to try ideas just to see what happens. You will learn things that are never very clear in any book on the subject.

Below are some experiments you can try on your IBM PCjr just to get into the mood, but develop a habit of continuing experiments on your own.

First put the *Test Programs* diskette into the disk drive and do a cold or warm start. If you have doubts about how to do this, refer back to Chapter 1. When you see the DOS prompt A>,

Type: BASIC <ENTER>

You're now ready to write or run programs in Cartridge BASIC. Try some commands performed with the standard keys on the keyboard.

Type: PRINT"Hi there!" <ENTER>

The computer should print "Hi there!" just below. The PRINT command is the principal instruction for directing the computer to display the information between quotes on the screen, or—using a slightly modified form of the command—on the printer. It is also used to record information on diskettes.

Try typing the PRINT statement incorrectly, e.g.,

Type: PIRNT"Can't you spell?" <ENTER>

You should get a "Syntax error" at this point. This simply means that, as far as BASIC is concerned, you typed nonsense.

Type: PRINT Here is what happens when you forget both quotation marks <ENTER>

The PCjr looks at an unquoted string of characters and thinks the words are names of variables. It assigns each the value zero. Try these cases:

```
PRINT This is no better" <ENTER>
PRINT "Now we're closer to the rule. <ENTER>
PRINT "But it screws up in this case:PRINT"Right!" <ENTER>
```

NOTE: In the last example the colon (:) lets you put more than one command on a single line, but since the closing quote on the first literal is missing, the colon is not effective. Compare the result with this case.

```
PRINT"But it doesn't screw up now":PRINT"Understand?"
  <ENTER>
```

By now you have the idea that you must <Enter> after typing a command if you want the PCjr to do anything. If the text of this book isn't explicit about this action at times, don't sit there waiting for something to happen. Hit <Enter>!

NOTE: Here's a shortcut that will make your work a little easier. Instead of typing the five-letter word PRINT, type a question mark (?). You get the same result.

2. GETTING SOME WORK DONE

The PRINT command also lets you use the PCjr as a calculator. Try these commands!

```
PRINT "3+2.1"
PRINT 3+2.1
```

In the first case the numerals and plus sign are treated as part of a sentence, as a "literal." When the quotation marks are removed, PCjr

treats the expression as numbers subject to the operation of addition, and proceeds to perform the indicated arithmetic.

The PCjr can do these elementary operations on numbers:

ADDITION: Specified by a plus sign (+) as above

SUBTRACTION: Specified by the usual minus sign (-)

MULTIPLICATION: The PCjr follows the general computer convention of the asterisk (*) for this operation. Try this example:

PRINT 4*5

DIVISION: Specified by the slash (/). Try this case:

PRINT 81/9

Can you guess in advance what will happen when you try this example?

PRINT 10/0

Make a note of the result. If you accidentally do this, you'll want to be able to recognize the problem.

In this last case PCjr even tells you approximately how big it thinks "infinity" is, namely, 1.701412E+38, which is nearly 2 followed by thirty-eight zeros.

Occasionally you'll generate some funny-looking numbers like that one. Don't let them hang you up in confusion. For now ignore them and go on. I'll tell you about them when and if it's relevant. (If you must know right now what this "scientific" notation means, read about floating-point numbers in Chapter 3 of the BASIC manual.)

You'll also see some funny arithmetic results that will make you suspect PCjr's arithmetic capability. Try this example:

PRINT 15*(1/15) (Thought you'd get one, didn't you?)

The problems PCjr has with arithmetic stem from rounding numbers to a specific number of digits, or from converting a number from binary to decimal representation. To deal with this issue adequately is very complex and, for the moment, unimportant.

EXPONENTIATION: This is multiplying a number by itself a given number of times. For example, 3*3*3*3*3 is represented as 3^5, the circumflex (^) denoting the superscripting operation one sees in algebra books for "raising to a power." Do the following example and check it by successive multiplications:

```
PRINT 3^7
```

Don't be concerned! If you don't use exponents in your work, you don't have to use them with PCjr.

OTHER OPERATIONS: You can also do integer division, modulus arithmetic, and elementary circular functions right on the screen as we have just been doing. I won't introduce these advanced topics. If you know about them already, you don't need me to explain them to you. If you're not familiar with them now, you'll probably never have need for them. (For the definitions of these operations and intrinsic functions, see the BASIC manual.)

Though I've used integer numbers for the most part in these examples, you can do the arithmetic operations with decimal fractions—you know, decimal points, and dollar-cents numbers. Try a few cases of your own making for each arithmetic operation. When the numbers get too big or too small, they are represented in scientific notation.

BASIC performs arithmetic in a definite order—it's complicated. The answer you get depends on your use of parentheses to group terms. Try the following examples, recording in your notebook the variety of answers obtained:

```
PRINT 3+4/2
PRINT (3+4)/2
PRINT 4/2+3
PRINT 4/(2+3)
PRINT  2*(3+4*5-6)
PRINT  2*3+4*5-6
PRINT  2*(3+4)*5-6
PRINT 2*(3+4)*(5-6)
```

Do you get the answers you expect?

Invent some more complex problems of your own to test how parentheses are handled. Make notes of any unexpected results you get. *It's important to understand what happens to arithmetic results as you do or don't use parentheses to group terms.*

3. DIRECT AND INDIRECT MODES

Making Your Point

By combining the PRINT and COLOR commands, you can produce some fun and useful results. First let's set up the screen to color text mode.

Type: SCREEN 0,1 <ENTER>

Okay, now try this long-winded combination of commands. (Don't be concerned if the line wraps around when you reach forty characters. But be careful, and watch out for the semicolons [;].)

```
PRINT"To ";:COLOR 0,4:PRINT" EMPHASIZE, ";:COLOR
  7,0:PRINT" use COLOR"
```

Don't forget the <ENTER>.

If you translate this series of commands into English, here is what it says:

- At the line of the cursor, print the characters "To " and stay on the same line (the semicolon keeps you there).
- Change the screen output to print dark characters on a red background.
- Print "EMPHASIZE" and leave the cursor where it is.
- Change the screen output back to light characters on a dark background.
- Print " use COLOR" and move the cursor to the next line. (No semicolon this time.)

Repeat the above example, this time omitting all semicolons. Try using the cursor controls to delete the semicolons in the line you just typed. This approach can save work at times.

What happens is this: The semicolon prevents PCjr from automatically moving the cursor to the left margin (carriage return) and to the next line (line feed). In this way you may generate a string of characters on the same line using a series of PRINT commands. Make a note of this! It's an important fact, but easily forgotten.

Try substituting other color combinations for foreground and background. Remember the table of colors in Chapter 1. Look up the COLOR command in the BASIC manual and begin getting familiar with it.

Play It Again, Sam, and Again, and Again . . .

Now try this, being careful as you type the commands. Use <Backspace> to correct errors.

Type: SCREEN 0,1:COLOR 1,15 <ENTER>
Type: CLS:FOR COUNT = 1 TO 500:PRINT"[*]";:NEXT COUNT <ENTER>

Here's what happens, command by command.

• SCREEN 0,1 sets up color text mode. The screen should already have been in this mode, but don't take chances.

• COLOR 1,15 sets up to print white letters on a blue field.

• CLS clears the screen and moves the cursor to the upper left-hand corner.

• FOR sends a "repeat" command to the PCjr to do five hundred times everything that is sandwiched between it and the NEXT statement.

• The tally is kept in the variable called "COUNT." That is, COUNT starts at 1. On each repetition it is increased by 1 until it exceeds the limit of 500. Then the process stops.

• Meanwhile, the PRINT is repeated over and over. When a line is filled, printing "wraps around" to continue at the beginning of the next line.

Try running the last example for different values of "COUNT," e.g., 20, 100, 1000. Also try different combinations of characters in the PRINT statement—as many characters as you like. See what kinds of designs you can generate.

Understand What You're Doing

There is a temptation to run through examples without stopping to think about what you've done. Resist it, if you can, and stop to consider what's going on. What does each command do? Translate them in sequence into English as I did earlier. Remember, *You must be able to read a program before you'll be able to write one. If you have trouble understanding a piece of BASIC programming, use PCjr to help.*

• Try isolating and executing just a few of the commands in a sequence so you can find out what they do and the state of PCjr when it finishes them.

• Successively add more commands in the sequence until you've figured out just what the whole sequence does.

This little process, elaborated upon, is what I do all the time to check out parts of my own and others' programs. Get in the habit of doing it. PCjr itself is the final word on what it can and cannot do—not what's written in books.

Turning Off Text Mode Color

Just as we selected to work with color on the text mode screen with the command SCREEN 0,1, we can select to work without color.

Type: SCREEN Zero,Zero <ENTER>

Now type some stuff on the screen, or execute some commands. As you see, you're back to white letters on black.

PCjr Remembers Your Commands

By this time you must be getting tired of keying almost the same thing each time to try out different ideas. Although the exercise helps you become familiar with the keyboard, we'll cover some ways to reduce the amount of typing you do. Thus far you have been using the PCjr in what is variously called COMMAND Mode, DIRECT Mode or IMMEDIATE Mode. That is, you execute commands directly from the keyboard whenever you strike <Enter>.

The Command Mode provides a handy facility. It is not available on many larger computers. Among other things, it makes it easy to test various combinations of instructions when you do not quite understand what they will do.

Now do this experiment.

Type: 50 FOR COUNT=1 TO 500:PRINT"Your Message Goes Here!";:NEXT COUNT:END

(Don't forget <Enter>. And don't forget the semicolon [;].)

This combination of commands is very much like what you did just a few minutes ago. (END means just that. It prevents PCjr from executing commands beyond it. Of course, there aren't any here.)

Nothing obvious should have happened when you pressed <Enter>, unless you made a syntax or other error.

Type: GOTO 50 <ENTER>

That's more like it! Type "GOTO 50" again. And again. You can do it all day because the statement you typed so many times is now stored in PCjr's memory.

You can check the status of this stored statement.

Type: LIST 50 <ENTER>

Try typing LIST alone to see what happens. Maybe there are other things in the memory you didn't know were there.

When you execute commands that are stored in the memory, you are said to be operating in EXECUTE Mode, INDIRECT Mode or DEFERRED Mode.

The number that labels the command or program statement is called a line number. Line numbers can be any integers from 0 to 65529. Try these cases:

Type: -10 GOTO 50 <ENTER>
70000 GOTO 50 <ENTER>

4. THE IMPERTURBABLE SCRATCH PAD

Clearing Program Memory

Now that you know one way to store information in PCjr's program memory, let's exploit your knowledge in a practical way. First, since you know how to put things into the program memory, here's how to erase it:

Type: NEW <ENTER>

NEW will always clear out the program memory. Check me by typing

LIST <ENTER>

Voila! Nothing there! NEW is normally used when you've finished running a BASIC program and want to key in a new program on a clean slate.

Keying a Program

Well, that's just what we want to do at this juncture. Here's a program for you to type into program memory. Although it isn't indicated, remember that you have to type <Enter> at the end of each numbered program line.

If you make an error as you are typing a line, use <Backspace> to correct it. If you catch an error after you've ENTERed the line, forget about it and go on to the next line. We'll pick up the loose ends later.

Carefully now, type exactly what you see.

```
100 ' LETTERS.BAS
110 ' A silly program for learning editing
120 PLAY"MF" ' PLAY SOUND IN BACKGROUND
130 SCRN 3 ' SELECT LOW RES GRAPHICS
140 CLS ' CLEAR SCREEN
150 FOR I=1 TO 155 ' COUNT COLORS
160 P=35 ' INITIAL VALUES
165 Q=120
170 FOR K=1 TO 30 ' COUNT REPETITIONS
180 R=1000*RND+40 ' RANDOM FREQUENCY
190 SOUND R,1,6,0 ' GENERATE SOUND
200 NEXT ' GO REPEAT SOUND
210 PRINT STRING$(20,63+I); ' DISPLAY LETTERS
220 NEXT I ' INCREASE COLOR COUNT & REPEAT
230 GOTO 150 ' START OVER
```

CAVEAT: Don't try to RUN this little program yet. You might just screw up the screen and not be able to recover without starting from scratch with a warm start.

Get "Hardcopy" Output

As you've probably gathered, if you're to do some editing on this program, there must be something wrong with it. Indeed, there is. Not only are there the problems already included in the program, but there's a good chance you made some errors as you keyed in the program as well.

The first thing to do is get some "hardcopy"—a printout of the program. (Remember? Just turn on your printer and Type: LLIST

<Enter>.) Of course, if you don't have a printer you'll have to LIST the program to the screen and make notes.

Now compare the printout with the listing above, character by character, and mark any deviations you find.

Substituting Stuff

In line 120, I intended to play sounds in the background but inadvertently set PLAY to foreground mode. Let's fix this item first.

Type: <Ctrl-L> **(Clear screen)**
Type: LIST <ENTER>

The program is on the screen, and the blinking cursor is below the "Ok" prompt near the bottom, right? Using the UP arrow (or Home) key, off to the center right, move the cursor up to line 120.

Next, using the right arrow (or PgDn) key, move the cursor to the right until it's under the F in the MF string between the quotation marks. When you're there,

Type: B <ENTER>

NOTE: It is essential to press <Enter> for the change in the program to register.

Adding Stuff

There's a more serious error in the next line, line 130. For some reason, I abbreviated SCREEN. We've got to put the EE's into the command.

The <Enter> that registered the fix at line 120 should have put the cursor at the beginning of line 130. In any case, put the cursor there with the arrow keys (cursor movement keys, to be precise).

Now move the cursor under the N in SCRN. Locate the Ins(ert) key on the bottom row, just to the right of <CapsLock>.

Watch the cursor as you quickly press and release the <Ins>. Did you see the cursor change from an underscore to a block that is half the height of a letter? Press <Ins> again and you're returned to the blinking underscore. <Ins> is a toggle key.

Well, we've still got the change to make, so press <Ins> again. Now for the correction. Watch the cursor as you

Type: EE

Isn't that neat? Don't forget to register the correction—type: <Enter>. Notice that hitting <Enter> cancels the insert mode and returns the cursor to a blinking underscore.

Deleting Stuff

Oh dear! My fingers stuttered on the 5 when I typed line 150. I've got the color count running to 155 instead of to 15.

Move the cursor with the arrow keys until it's under one of the 5's in the 155 of line 150. Are you there?

Now locate the Del key on the bottom row, just to the right of the Ins key. Got it?

Be sure you don't use a heavy hand on the next operation or you'll wipe out a bunch of characters. Okay, watch the cursor as you quickly press and release <Del>.

If the extra 5 didn't disappear, you may have been too light with your touch. Try again.

And don't forget to register all corrections with <Enter>.

Scrap the Trash

Good grief! Where did lines 160 and 165 come from? The variables P and Q have no place in this program. We'll DELETE them.

Type: DELETE 160-165 <ENTER>

Let's check that the deletion took place correctly. Once more,

Type: <Ctrl-L>
Type: LIST <ENTER>

Inserting New Lines

I'll bet that my eye skipped line 160 when I typed in this program. Ah, yes! Here's the missing piece in my hand-scrawled copy.

Type: 160 COLOR I,16-I ' SET FORE- & BACKGROUND
<ENTER>

LIST the program to the screen once more to see if the line was inserted correctly.

Notice how PCjr keeps all the program lines in correct sequence.

The EDIT Command

Sometimes it's not convenient to make changes in a screen LISTing of your program as we've been doing. If the corrections are widely separated, as when you have a long program, there's an easier way.

Type: <Ctrl-L>
Type: EDIT 210 <ENTER>

There you have line 210 all ready to be corrected. In this case we want to change the number 20 inside the STRING$ command to a 100.

Move the cursor right to the 2 in 20. Then,

Type: <Del>	**(Cancel the 2)**
Type: <Ins>	**(Switch to Insert mode)**
Type: 10 <ENTER>	**(Make the insert)**

LIST line 210 to verify that the corrections were made correctly. That is,

Type: LIST 210 <ENTER>

In summary, here is how the corrections you made should look on the screen:

```
120 PLAY"MB" ' PLAY SOUND IN BACKGROUND
130 SCREEN 3 ' SELECT LOW RES GRAPHICS
150 FOR I=1 TO 15 ' COUNT COLORS
160 COLOR I,16-I ' SET FORE- & BACKGROUND
210 PRINT STRING$(100,63+I); ' DISPLAY LETTERS
```

And, of course, line 165 should no longer be with us.

Fix Your Own Boo-boos

If you introduced typing errors as you keyed in the program, fix them now. It's just a matter of inserting, deleting or substituting characters or program lines.

Don't run the program yet!

5. SAVE YOUR PROGRAM ON DISK

A Little Insurance

CAVEAT: Make it a general rule when you've typed in a new program, or made extensive or complicated changes in one, to SAVE it on disk before you try RUNning it.

It's possible for you to make an error in writing or correcting a program that will erase program memory or lock you out of the computer. In either case all your hard work is down the drain. If you had SAVEd the program first, you'd at least have a copy on disk that you could work with to locate and fix the problem.

With this in mind, let's SAVE LETTERS.BAS on disk. *Test Programs* is still in the disk drive, isn't it? Okay,

Type: SAVE "LETTERS.BAS" <ENTER>

(You can omit the .BAS in this case if you like, but remember that you can use a filename extension in the SAVE command. You can use any three acceptable characters you like.)

Did the disk drive whir? Okay, check that the file LETTERS.BAS was recorded.

Type: FILES <ENTER>

What would you do to bring LETTERS.BAS back into program memory if you lost power, or inadvertently typed NEW <Enter>, or something else unfortunate? You'd do a LOAD "LETTERS.BAS" command.

It's Okay to RUN Now

You've been kept on the string long enough. You can RUN the program now.

Type: <Fn-2> (Execute RUN command)

It really is a silly program, but it does show you a fair sampling of PCjr's colors—in combination, too. Or, if you have only a black and white set, you see the different textures and shades you can get.

If it still didn't work for you, you had better get a LLISTing on your printer and check the copy against the book. You may still have a keying error in the program.

If you can't find your problem, don't be too concerned at this point. The program has served its purpose as a vehicle to learn about the BASIC Editor. Later on, when you know more, your present difficulties will become obvious to you.

If LETTERS.BAS Works for You...

Don't go turning off power or pulling plugs to stop the horrid noise. Just

Type: <Fn-Break>

Notice that the letters look bigger than before. They're twice as wide, in fact. Count them. There are only twenty characters to the line.

To return to the forty-character text screen,

Type: SCREEN 0 <ENTER>

Make a LLISTing of LETTERS.BAS anyway so you'll have a copy to mark on and study.

The BASIC Program Editor

When you were keying and correcting LETTERS.BAS, you used the BASIC Editor. It is very important that you develop a ready skill with it. The Editor will save you much agonizing typing effort.

Read about the BASIC Editor in Chapter 2 of the BASIC manual. Then experiment with the commands, using LETTERS.BAS until you feel you're gaining some mastery.

A summary of the Editor commands is given in Figure 2.1. There may be a couple we haven't used explicitly, so be sure to try every one of them.

6. DISCOVERING THE SIMPLE AND USEFUL

I want to keep emphasizing the importance of your experimenting with PCjr. Discoveries are fun! And if you're persistent, you'll receive valuable rewards for your efforts. Let's discover something interesting and useful right now.

Ctrl-Letter Combinations

You'll have noticed that some of the commands involve the joint operation of two or more keys—like our clearing the screen with <Ctrl-L>. Let's find out if there are other combinations that do something useful. Try this.

Type: <Ctrl-G> (Hold Ctrl and strike G)

How about that! A beep! Incidentally, you can get the same sound at least two other ways:

Type: BEEP <ENTER>
Type: PRINT CHR$(7) <ENTER>

These are not important at the moment.

Do you have a lot of junk on the screen? If not, put some there and do this:

Type: <Ctrl-Fn-Home>

That is, hold down the Ctrl key. Then successively strike the Fn and Home (or UP arrow) keys, in that order. The screen should clear—same as CLS and <Ctrl-L>.

Now maybe you are beginning to see the potential. Find out what all the letter keys do when you strike them while pressing <Ctrl>.

Make a table on a clean sheet of paper. The first column is the alphabet. The second column is titled "Ctrl Key" (leave space for more columns). In the second column write down what happens for each letter struck in combination with Ctrl.

NOTE: If nothing seems to happen, put material on the screen, or move the cursor to the middle of the screen before the experiment. Good hunting!

Alt-Letter Combinations

After you've tried the Ctrl-letter combinations on your own, try this experiment. Label a third column "Alt." Try pressing the letters as you hold the Alt key down. Record your observations as before.

Check your discoveries by comparing your table with Figure 2.2. Give yourself a high score if you discovered that <Ctrl-6> is "up arrow" and <Ctrl-minus> is "down arrow."

7. PCjr KNOWS ITS ABC'S—AND THEN SOME

Here's an Alphabet for You

Here's something new to try. Clear program memory (Type: NEW) and key the following little BASIC program.

```
100 ' ASCII.BAS
110 SCREEN 0,1 ' Color text mode screen
120 COLOR 15,1 ' White letters on blue
130 CLS ' Clear screen
140 FOR ASCII.CODE=0 TO 255 ' Set count range
150 IF ASCII.CODE=12 THEN 170 ' Avoid clear screen code
160 PRINT CHR$(ASCII.CODE);" "; ' Display character
170 NEXT ASCII.CODE ' Repeat for next character
180 END
```

NOTE: You don't need to type the remarks to the right of the apostrophe ('). They are there to help you understand what's going on.

As soon as you've finished typing, LIST the program to the screen. Examine it carefully, character by character. Did you make a mistake? Use your BASIC Editor skills to correct any errors, then SAVE this program.

Type: SAVE "ASCII",A <ENTER>

Did you notice? Another new wrinkle. What's that A for? Well, since we're talking about ASCII (pronounced "ask-key") stuff, we might as well SAVE this file as an ASCII file instead of as a binary file (whatever that is; things'll clear up as we go along).

So now we can RUN ASCII.BAS.

Type: RUN <ENTER> (or <Fn-2>, right?)

This little program should bring PCjr's basic repertoire of characters up on the screen. You should have heard a beep too, as it displayed the characters.

Open your BASIC reference manual to Appendix G and you will see a table of these two hundred fifty-six characters. Compare them

with the contents of the screen. There are some irregularities in the way the printing is laid out, because not all the codes of the table have printing characters (carriage return, line feed, etc.).

Look once more at the program and see if you understand it. The commands are executed in ascending order by line number unless some command (e.g., the "IF") causes a change in this sequence.

LLIST a copy of the program and translate it into English.

Here are some hints. Lines 110 and 120 set up the color text screen as we've done before. Line 130 clears the screen. Line 140 sets up a FOR . . . NEXT loop (completed in line 170). When the counter, ASCII.CODE, reaches 255, execution continues to line 180, where the program terminates. The "meat" of the program occurs in lines 150 and 160.

Line 150 introduces a new idea of considerable importance, as it gives PCjr the ability to make decisions. It says that if the counter, ASCII.CODE, is equal to 12, then skip to line 170, avoiding the print statement in line 160.

Why would I want to skip 12? Does Appendix G of the BASIC manual give you an idea? What if you tried this experiment, using PCjr to help you understand?

Type: PRINT CHR$(12) <ENTER>

Of course, you know this clears the screen. If we left it in the program, the first few characters PCjr printed would have been wiped out when it got to 12.

What about line 160? The new thing there is the command CHR$ (for character). But you've deduced already that it simply generates the characters in the table of Appendix G opposite the number specified in parentheses. Try a few cases and check them with that table.

Type: PRINT CHR$(21) <ENTER>
Type: PRINT CHR$(65) <ENTER>
Type: PRINT CHR$(97) <ENTER>
Type: PRINT CHR$(273) <ENTER>

What's All This ASCII Business Anyway?

ASCII stands for American Standard Code for Information Interchange. It was set up by people in the communications field to ensure the information compatibility of different equipment manufacturers.

Your PCjr is considered a communication device. And indeed it may be used as a terminal to communicate over telephone lines (if you install some additional equipment). Most printers are also classed as communication devices, and use the ASCII character codes.

The ASCII coding system covers only the characters up to 127. IBM has doubled that number for the PCjr.

Are you curious to know how your printer will treat IBM's extended character set? Turn your printer ON and we'll find out. Using the EDIT command, or any other editing approach that does the job, change line 160 to read

```
160 LPRINT CHR$(ASCII.CODE);" ";
```

Watch the semicolons.

We also need an additional instruction at the end of the program to print what remains in the printer's buffer (the place where the printer stores information waiting to be printed).

Type: `175 LPRINT ' Empty printer buffer`

Now turn on the printer and

Type: RUN <ENTER>

Almost any time you want to print on the printer instead of the screen, put L in front of PRINT—but not always! (See the BASIC manual for more on LPRINT.)

Use ASCII.BAS to experiment with the text color combinations, especially if you have only a black and white set. Some of the combinations are unreadable, even on the forty-character screen.

Decisions, Decisions

You will have noticed that ASCII codes 7-13, 28-31 and 255 print no character on the screen. The little program you typed can be changed to avoid them. The IF . . . THEN is the key. Type the following commands:

```
145 IF ASCII.CODE>=7 AND ASCII.CODE<=13 THEN 170
148 IF ASCII.CODE>=28 AND ASCII.CODE<=31 THEN 170
150 IF ASCII.CODE=255 THEN 170
```

Don't forget to key <Enter> after each statement.

At this point, LIST the program to see if you typed the lines correctly. Everything okay?

RUN the program and watch the screen.

Well, do I have to tell you everything? You didn't delete the L in front of PRINT in line 160. How can you print the characters on the screen? And what about line 175? Make the changes and try again!

Let's look at what you did. Line 145 simply translates, "IF the count, ASCII.CODE, is ever 7 or more, and at the same time 13 or less (i.e., count is in the range 7-13), THEN skip to line 170 (i.e., bypass the printing)." Compare this translation with line 145 to be sure it makes sense to you. The "AND" is a logical operator called "conjunction." But you know that from English grammar!

Look again at Appendix G in the BASIC manual to verify that the ranges of numbers that we are to skip are correct. (Of course you know that < means less than and > means greater than.)

Line 148 is similar to line 145. Line 150 has not changed its form. You might have written

```
145 IF ASCII.CODE>6 AND ASCII.CODE<14 THEN 170
```

instead. It gives the same results. Why?

You'll see as we go along that there are still other ways to write lines 145-150 using the IF . . . THEN . . . ELSE combination. You'll see more as we go along. (For more on IF . . . THEN, read about it in Chapter 4 of the BASIC manual.)

More on ASCII

For almost all the ASCII characters associated with keys on the keyboard, there is an alternate representation in BASIC. If you put a letter or character between quotes, it's the same as writing CHR$( ) with the appropriate number between the parentheses. Try these cases:

```
PRINT CHR$(35),"#"
PRINT CHR$(65),"A"
PRINT CHR$(97),"a"
PRINT CHR$(126),"~"
```

In each case the character should print twice, demonstrating equivalence. (Notice I used a comma [,] instead of a semicolon [;] this time. Try one or two cases with a semicolon to see what happens. *Don't forget the result. Make a note.)*

There is an important exception to the equivalences noted. Try to put a quote (") between quotes. That is,

Type: PRINT CHR$(34),""""

Only one quote should appear. BASIC uses quotation marks to delineate literals (strings of characters). It does not permit (double) quotes to appear in the literal. If you want to PRINT a (double) quote, you must use CHR$(34).

8. JUST FOR FUN

Zounds! What Beeps and Clicks Come from Yonder Computer?

Here's something to play with.

Type: SOUND 523.25,30

How about that! It's not quite "Met" quality, but it approximates C above middle C on a piano. If you change the first number, the frequency in hertz (vibrations per second), you get different notes. If you change the second number (the duration in units of 18.2 per second), you make the tone longer or shorter.

To my ear, increasing the frequency by forty gives about a half-tone step on the piano. Try this one to play a kind of chromatic scale:

Type: SOUND OFF:BEEP ON <ENTER>
Type: FOR FREQ=500 TO 7500 STEP 40:
SOUND FREQ,10:NEXT FREQ

That clears the wax from your ears! Notice that the STEP in the FOR...NEXT sandwich is 40 hertz, the change in frequency from note to note.

Try the example again, but change the duration from 10 to 1. Use your BASIC Editor commands to move the cursor up to the FOR statement and make the change right on the screen. Saves typing!

Now change the 1 to .1 and try again. Keep making the duration smaller and smaller. You can't go below 0.0015.

Using your screen editing skills, reverse the frequencies and put a minus sign in front of the 40 so the line looks like this:

```
FOR FREQ=7500 TO 500 STEP -40:SOUND FREQ,1:
  NEXT FREQ
```

Translate this program example into English.

Random Sounds

Here's one I like:

```
Type: FOR COUNT=0 TO 500:SOUND 5000*RND(1)
      +50,1:NEXT COUNT
```

The new idea here is RND(1). This is a built-in BASIC function that generates random numbers between 0 and 1.

```
Type: FOR COUNT=1 TO 10:PRINT RND(1):NEXT COUNT
  <ENTER>
```

So you see, every time you do RND(1), a new number between 0 and 1 is generated.

In the SOUND example, I added 50 hertz to the calculated frequency to avoid an error in case RND(1) became zero. The smallest frequency allowed is 37 hertz. See the BASIC manual for more on RND.

Do you really understand what is happening in this example, how the random number makes the tone jump all over the place? Translate the program into English.

Controlling SOUND from the Keyboard

Here's a neat trick that comes up over and over again in programming. Our objective is to set up a little conversation with PCjr that goes something like this.

PCjr: Here's a sound. How do you like it?
You: Not too well. Make it a little higher.
PCjr: Okay, how about this?
You: No, a little higher yet.
PCjr: Now is it okay? You're hard to satisfy.
You: Too high now. Down a little.
PCjr: Dear me! There's no pleasing you! What about this frequency?

And so on.

The core of this imagined conversation is a set of program steps like this (clear program memory with NEW and start typing):

```
100 ' KBCTRL.BAS
110 ' Control SOUND from the Keyboard
120 F=500 ' Initial frequency
220 SOUND F,.8 ' Play sound
```

So far the program just plays a frequency of 500 hertz for 0.8 clock ticks. The trick comes with the use of the BASIC command INKEY$ in the following instruction:

```
130 A$=INKEY$ ' Get character from keyboard
```

Whenever you strike a key, its code is generated and transferred to a typing buffer of fifteen characters. When the INKEY$ command is executed, the earliest character you typed is pulled from the buffer and placed in a pigeonhole labeled A$. (We could have called it something else, like CHAR$, for character. The $ says the information is an ASCII character or string.) If the buffer is empty, A$ is a string with no characters.

So now that we have a character, what do we do with it? Well, if we don't like the sound, let's type U or u to tell PCjr to make it a higher frequency, or D or d to make it lower. PCjr has to know whether the typed character is U or D. We use the IF command to inform it.

```
140 ' Test for upper or lower case U
150 IF A$="U" OR A$="u" THEN F=F+10
160 ' Test for upper or lower case D
170 IF A$="D" OR A$="d" THEN F=F-10
230 GOTO 130 ' Repeat sound
```

The key instructions are in lines 150 and 170. If A$ is U, the frequency is increased by 10 hertz; if it's D, the frequency is decreased by 10 hertz. After the change, the program jumps back to 130 with a GOTO to see whether we typed another character.

Try RUNning the program.

There are a couple of other things worth doing to make the program operate more smoothly. First we should make sure the frequency F does not go outside the legal range 37 to 32767 hertz. We need a couple of IF's.

```
200 IF F<37 THEN F=37 ' Stay at lower sound limit
210 IF F>32767 THEN F=32767 ' Stay at upper sound limit
```

We also need a way to stop the program gracefully. Suppose we test for the ASCII code for the Esc(ape) key like this.

```
180 ' Test for Escape, to quit
190 IF A$=CHR$(27) THEN END
```

Finally, we can get some visual idea of the frequencies being sounded by adding a statement to print the current value of F.

```
225 PRINT F
```

Don't forget to SAVE and LLIST KBCTRL.BAS for future reference.

So What Can You Use SOUND For?

People who are into games can make all sorts of pops, clicks and buzzes from a simple mechanism like SOUND. I'll show you some of these tricks as we go along.

But beyond games, there are more serious uses for sounds that you should know about. I frequently use the sound-making facilities to warn me when I should watch out for an error or when I should perform some routine task such as changing disks or starting the printer.

Audio signals can prove to be a great convenience. Suppose that while you're running one of your programs you want to work at something else across the room. It's nice to write your program so you can know what's going on when it's running without having to walk over and check. Suppose you periodically put out a sound pattern that indicates, "I'm okay, just operating fine. But *keep your hands off*!" A

different pattern might indicate a change in the phase of the program's operation. A third pattern might say, "I need a new diskette mounted." A fourth might say, "I'm ready to start printing reports. Turn ON the printer." A fifth might say, "I've got an error that you should attend to."

Here's an example you can build upon:

```
100 ' ENDPROG.BAS—END OF PROGRAM SIGNAL ('NBC')
110 SEC=20:NUM.TICKS=875*SEC
120 SOUND 650,7
130 FOR COUNT=1 TO 100:NEXT COUNT:' PAUSE
140 SOUND 1100,7
150 FOR COUNT=1 TO 100:NEXT COUNT:' ANOTHER PAUSE
160 SOUND 880,8
170 FOR TICK=1 TO NUM.TICKS:NEXT ' WAIT 20 SECONDS
180 GOTO 120:' REPEAT SIGNAL
```

Key this program into program memory. Test it. You stop it with a <Fn-Break>. The twenty-second delay between sounds is a long time, so be patient.

Then SAVE ENDPROG.BAS on your *Test Programs* diskette—as an ASCII file.

There is a useful idea embedded in this little program. Lines 110 and 170 give you a method for setting up timing delays of specific duration. These lines form a clock that beats at a certain number of ticks per second. The clock rate is slower when you include the counter's name in the NEXT command. When you omit the name, as in line 170, the clock rate is faster, about 875 beats per second. Notice how this rate is incorporated in the count limit, NUM.TICKS, defined in line 110.

Set up an experiment to test whether the constant is really 875 or an amount larger or smaller. (Hint: Do a BEEP at the start and at the end of the FOR...NEXT with SECS=10, and measure the time with your watch. Your reward is to prove I'm wrong and can't measure time very well.)

Use your imagination. Sounds can enhance your computer's operations. (For more on SOUND, see the BASIC manual.)

So You Want to Be in Pictures!

Talking about movies, let's make one. No sense in thinking small. We'll make a De Mille special on the grand scale—and with a sound track to boot.

First, let's do a time-lapse version of the creation of the universe. (How's that for grand scale?) But what theory shall we use? The "Big Bang"? The "Steady State"? How about using our own theory? Call it the "Random Position" theory.

We'll start with empty space. Then we'll put a star here, and a star there, and on and on until we fill up all of space. And as we set the stars aglow, we'll fill space with "space" music.

Roll 'em! First we need empty space of appropriate color. (Type NEW to clear memory.)

```
110 SCREEN 0,1:COLOR 15,1
120 CLS:KEY OFF
```

And we want to turn off the cursor so it won't spoil the effect.

```
125 LOCATE ,,0 ' Turn off cursor
```

Don't omit those two commas.

Then we need a random position in space—a random row and column on the screen.

```
130 ROW=22*RND(1)+1 ' Pick random row
140 COL=39*RND(1)+1 ' Pick random column
```

We have to put a "star" in that random location.

```
150 LOCATE ROW,COL ' Move cursor to row,col
160 PRINT ".";' Place star at cursor's position
```

Next we need sound effects—a magic "space" sound. Here's some "brittle" music.

```
170 MAGIC=5000*RND(1)+50 ' Calc sound frequency
180 SOUND MAGIC,3 ' Play MAGIC sound
```

Finally, we have to repeat the process for another star.

```
190 GOTO 130 ' repeat CREATION
```

But let's not forget the opening title.

100 ' CREATION.BAS

There, that's just fine. Cut and print! (SAVE it on your Test Programs diskette.)

If you're ready, let's see how our "creation" scene will look in the theater. RUN it! Stop CREATION.BAS with <Fn'Break>.

Make a LLISTing of CREATION.BAS and write a translation of it into English.

The only new idea here is the LOCATE command, which moves the cursor to the row and column positions calculated in line 150, where the star is printed. Don't forget to turn the cursor back on.

Type: LOCATE ,,1 <ENTER>

Why don't you try substituting other characters for the dot in line 160 of CREATION. Experimenting broadens your perspective and opens up all sorts of possibilities.

Drawing Simple Shapes

The BASIC commands we've used thus far are simple but powerful enough to illustrate the programming process in a useful way. The capability of printing any ASCII character on the screen at a specific or a random position can be exploited for a variety of effects.

Suppose you wanted to draw a small balsam fir on the screen—a Christmas tree. How would you do it?

We don't have to be elaborate. A simple triangular shape will do. We might draw it in outline form, using combinations of the hyphen, period, underscore or other linear characters.

When you're thinking about something to put on the screen, it's always a good idea to look at a table of ASCII characters. It might suggest shapes to try that you hadn't thought of. (Open to Appendix G in your BASIC manual.)

To generate the outline of the triangle, you would have to tell the PCjr to start at a specific point and to generate a sequence of positions along the perimeter of the shape.

But what about making a "solid" triangle instead?

Look at ASCII character 219. It gives us a character with which to paint in a specific color.

Now comes the strategy of drawing. We'll exploit the FOR...NEXT command, which makes it easy to print a series of characters along a given line on the screen. We'll use the LOCATE command to pick the starting positions. We can start at the top and go to the base, or vice versa.

Let's start at the bottom and print the tree in the middle of the screen. The trunk will form a center line in column 20. Suppose the base of the tree is on line 20. Then the length of this base, in numbers of characters, must be such that the tree comes to a point, more or less, in line 1. How big is the base?

Suppose we cut one character off each end of each successive pair of layers as we build the tree. To come to a point in line 1, there must be ten characters on each side of the center line (in column 20) in line 20.

Let's write the program. (Clear memory with NEW.) We'll start by setting down the usual preliminaries—program name, color text screen, etc. Start typing in lines. (Don't forget to clear memory first—NEW.)

```
100 ' XMASTREE.BAS
110 SCREEN 0,1
120 KEY OFF
130 ' DRAW TREE
```

Then we'll color the tree green and clear the screen.

```
140 COLOR 2,0:CLS ' Color tree green
```

Starting in row 20 we have to build successive pairs of layers until we get to row 1. We can do this operation with a FOR...NEXT sandwich.

```
150 FOR ROW=20 TO 1 STEP -2
250 NEXT ROW
```

With a STEP of -2 we express that we're to build two layers of the tree each time the instructions between lines 150 and 250 are executed. Granted, we don't know just what these instructions are yet. Notice I've left lots of room to sandwich these layer-generating instructions inside the FOR...NEXT.

As we conceived the drawing process, we noted that in lines 20 and 19 we'd need ten characters on the left and ten on the right of

column 20. In lines 18 and 17 we'd need nine. And so on. Therefore, to generate one layer, we sandwich a PRINT statement in a FOR...NEXT of the following sort.

```
155 ' Generate first layer of pair
160 FOR COL=20-((ROW-1))/2 TO 20+((ROW-1))/2
190 NEXT COL
```

Notice how we have the column position limits in the FOR...NEXT as functions of the row position. As the row number decreases, the length of the layer decreases.

However, there are two layers to print for the same column limits. So reduce the row by one and repeat the same layer-generating instructions.

```
195 ' Generate second layer of pair
200 NXROW=ROW-1 ' Reduce screen row by one
210 FOR COL=20-((ROW-1))/2 TO 20+((ROW-1))/2
240 NEXT COL
```

To actually print the characters in a layer, we must position the cursor and print the first layer of the pair:

```
170 LOCATE ROW,COL ' Position cursor
180 PRINT CHR$(219); ' Place character of 1st layer
```

The second layer is generated similarly, but at the next row, NXROW, this way:

```
220 LOCATE NXROW,COL ' Position cursor
230 PRINT CHR$(219); ' Place character of 2nd layer
```

Let's put a red base on the Christmas tree this way. First the color.

```
260 ' DRAW BASE OF TREE
270 COLOR 4,0 ' Color base red
```

Then we'll put a splash of red under the center of the tree's bottom with a string of ASCII character 219—the same one we've already used.

```
280 LOCATE 21,19
290 PRINT STRING$(3,CHR$(219));
```

We're nearly finished. Only three more things to do. First, we don't want to have BASIC's "Ok" prompt on the screen when the program stops. It'll spoil the view. This is easy to do with the following simple closed loop:

```
320 GOTO 320 ' Avoid printing BASIC prompt
```

This means you have to do a <Fn-Break> to get back to BASIC when you tire of viewing the tree.

Next, we want to keep the cursor from messing up the view. So we'll turn it off this way:

```
310 LOCATE ,,0 ' Turn cursor off
```

Finally, we want to leave the screen colors in something that's legible when you finally quit. Here's one possibility.

```
300 COLOR 15,1 ' Change color to white on blue
```

Now we're finished. SAVE the XMASTREE.BAS. Then go ahead and RUN it to see if it works.

Now that you have a basic tree, there are other details you might add, such as ornaments or lights (e.g., try ASCII 1-5). You can make the ornaments blink brightly (use foreground numbers for the COLOR command in the range 16 to 31). You could put a star on top, or make a tinkling sound as ornaments appear (SOUND FREQ,1), or play carols. The possibilities are endless.

9. TAKING STOCK

These first two chapters have introduced you to the principal manual skills you need to operate the PCjr, as well as some first ideas about programming in BASIC in general, and sound and graphics in particular. Before continuing, you should reinforce your learning experience. Here are a few things to do:

Do a Summary Check

(1) Take another look at the BASIC Editor commands, summarized in Figure 2.1. Are you sure you know them? Why don't you just try each one once more.

(2) Turn to Figure 2.3. It summarizes, in alphabetical order, the BASIC commands introduced in this chapter. Do you know what each one does? What role it plays? Refer back to the chapter to reinforce your memory, and look up the commands in the BASIC manual.

Read down the list, stopping at each entry. Shut your eyes and visualize the command's function. Do you remember the examples in which it was used? If you have trouble, look it up in your manuals and trace back in the chapter to where and how it was used.

(3) Turn back to Figure 1.1, where the commands introduced in Chapter 1 are summarized. Repeat the detailed review process.

The Proof of the Pudding

Here's where you really find out whether you know what's going on. If you have trouble, get PCjr to help. Experiment with ideas. Delve deeply into the table of ASCII codes (Appendix G in your BASIC manual).

(1) Write an elaboration of the CREATION program of your own devising. Consider making stars blink. Try other characters besides the asterisk for the effects they produce. How could you make some of the stars "die" with age? (Hint: print random blanks accompanied by a high frequency sound.) Some ideas for elaboration are given in the DEMOPROG.BAS file on your *BASIC Utilities* diskette.

(2) If you haven't done so, elaborate on the Christmas tree program. Try this: Generate the tree from the top down, with sounds proceeding from high to low frequency. Then, on a second pass, generate the lights from the bottom up with sounds proceeding from low to high frequency. Finally, put a blinking star on top. (Hint: You can overprint material on the screen. The last character printed stays there.)

(3) Write and test a program to:

(a) Put a border around the screen. (Use FOR...NEXT and LOCATE.)

(b) Put a tick-tack-toe game board on the screen.

(c) Draw diagonals on the screen.

(d) Draw a small box anywhere on the screen (a poor man's "O").

(e) Draw a small X anywhere on the screen.
(Here are the graphic ingredients for playing tick-tack-toe with PCjr.)

(f) Draw a "solid" rectangle. (Adapt the Christmas tree program.)

(4) Write and test a program to:

(a) Draw a triangle (just the outline) with horizontal base and equal vertical sides (isosceles triangle).

(b) Draw the same triangle upside down.

(c) Combine the two triangles to form a Star of David.

(d) Draw a "solid" Star of David and decorate it.

(5) Write a program to draw an unending sequence of solid rectangles of random size and position. Fill the rectangles with random characters. It's okay for successive rectangles to overlap. Generate some beeps and squawks to go with the rectangle generation process. (This problem may be a little hard for you, so don't worry if you can't do it.)

WARNING: You'll learn very little if you perform the programmed actions without understanding. Read the explanatory paragraphs with concentration, answer all the questions, and do all the exercises. They are there to teach you.

EDITOR COMMAND	FUNCTION
Ctrl-ENTER	Line feed & carriage return; line NOT ENTER(ed) in program memory
Ctrl-Pause	Freeze output to screen
Ctrl —>	Move cursor one word to right
Ctrl <—	Move cursor one word to left
Ctrl-Fn-End	Delete from cursor to end of line
Ctrl-Fn-Home	Clear screen, cursor to upper left
Del	Delete character at cursor
Down arrow	Cursor down
Fn-Break	Stop program
Fn-End	Move cursor to end of line
Fn-Home	Move cursor to upper left of screen
ENTER	Line feed & carriage return; line is ENTER(ed) in program memory
Esc(ape)	Delete line of cursor
Ins	Insert character at cursor
Left arrow (<—)	Cursor left
Right arrow (—>)	Cursor right
Up arrow	Cursor up
Tab (—> I)	Tab 8 spaces right

Figure 2.1: Summary of BASIC Screen Editor Commands for PCjr

CHARACTER	with CONTROL KEY	with ALT KEY
A	Smiling face	AUTO
B	Left one word, Ctrl-LEFT arrow	BSAVE
C	Move cursor to start of next line	COLOR
D	Diamond	DELETE
E	Clear to end of line, Ctrl-Fn-End	ELSE
F	Right one word, Ctrl-RIGHT arrow	FOR
G	Beep	GOTO
H	Delete character to left of cursor	HEX$
I	Tab right 8 spaces	INPUT
J	Move character to start of next line	J
K	Fn-Home	KEY
L	Clear screen	LOCATE
M	ENTER	MOTOR
N	Fn-End	NEXT
O	Star	OPEN
P	Right arrow	PRINT
Q	Left arrow	Q
R	Ins(ert)	RUN
S	Double exclamation	SCREEN
T	Paragraph sign	THEN
U	Item sign	USING
V	Heavy underscore	VAL
W	Up & down pointer	WIDTH
X	Up arrow	XOR
Y	Down arrow	Y
Z	Right arrow	Z
6	Move up cursor	-
- (hyphen)	Move down cursor	-

Figure 2.2: BASIC Command Generation and Other Cursor Controls for PCjr

BASIC COMMANDS	FUNCTION
Alt-Letter	Generate BASIC reserved words
Arithmetic Operations	Add, subtract, multiply, divide, raise to power
ASC	Convert character to ASCII code
BEEP	Make "beep" sound
CHR$	Convert ASCII code to character
CLS	Clear screen
COLOR 15,1	Change screen foreground and background colors
DELETE	Delete lines from program memory
EDIT	LIST line, leaving cursor ready for editing line
END	Terminate program
FOR...NEXT	Repeat a sequence of commands
GOTO	Take next command from line specified
IF...THEN	Make decision on program flow
INKEY$	Get typed character from input buffer
KEY ON/OFF	Turn function key prompts ON/OFF
KILL	Delete a diskette file or program
LIST	List program memory to screen
LLIST	List program memory to printer
LOAD	Transfer program from diskette to program memory
LOCATE	Position cursor on screen
LPRINT	Print information on printer (LPT1:)

NAME...AS...	Rename a diskette file or program
NEW	Erase program memory
PRINT	Print information on the screen
RND	Generate random number between 0 & 1
SAVE	Transfer BASIC program from program memory to diskette
SCREEN 0,1	Set up color text mode
SOUND	Play a sound frequency of specific duration

Figure 2.3: Summary of Key Commands in Chapter 2

CHAPTER 3
THE SOUNDS AROUND US

Objective:

This chapter will let you experiment with natural, and not so natural sounds—everything from clicks, bleeps and screams to musical birdcalls.

1. SOME NEEDED BACKGROUND

You have already discovered some simple ways to generate sounds on PCjr. They were used to enhance the impact of generated pictures, or to signal the current state of the computer. SOUND was the BASIC command used for this purpose.

There's lots more fun to be had with SOUND. Here you'll discover ways to imitate the sounds you hear every day. The same methods of imitation can be used to invent new sound patterns for the creatures and machines of your imagination.

To succeed in this venture, however, you must come to an understanding of the fundamentals of sound.

What Is Sound Anyway?

Basically, sound is a vibration of matter. The shattering of your best table glass, dropped onto the kitchen floor by a slippery-handed child, is transmitted, via a compression wave in the air, to your ear. There the eardrum vibrates in sympathy—if not sympathetically—and transmits an electromechanical signal to the inner ear and on to the brain, where it is compared with a memory data bank of stored noises, classified, and identified.

As you can see, real-life sounds are very complex affairs. But in its simplest components, a sound consists of a constant-rate vibration (its frequency or pitch), at a certain energy level (its volume, amplitude or loudness), for a period of time (its duration).

The Ranges of Sound

The frequency of a sound is normally measured in beats or cycles per second. This unit of measure is called a *hertz*, after the great German scientist Heinrich Hertz (1857-1894). At low frequencies, less than fifteen hertz (beats per second), you can only sense the vibration as a shake or tremor. And if the amplitude is too small, say under two one-hundredths of an inch, you might not even feel it.

For frequencies between fifteen and thirty hertz, the tremor may be sensed with accompanying clicks, rumbles and other noises. The rumbles of major earthquakes are so deep that you're not sure you've felt or heard them.

Somewhere just short of about thirty hertz is the beginning of perceived sound vibrations. Musical sounds extend to about 15,000 hertz, the upper limit of audibility. However, the eighty-eight notes of my

piano run the narrower range from low A (27.5 hertz), three octaves below middle C, to high C (4,186 hertz), four octaves above middle C. Some pianos will play two or three notes lower than my Steinway.

Figure 3.1 gives you an idea of the ranges of sound emitted by various common sources.

SOUND SOURCE	RANGE	(IN HERTZ)
Pipe organ	10	– 8,000
Hi-fi system	15	– 30,000
Green frog	50	– 8,000
Clarinet	75	– 1,800
Man	85	– 1,100
Kettle drum	95	– 180
Lion	110	– 1,080
Trumpet	190	– 990
Violin	200	– 2,650
Horse	320	– 3,040
Dog	452	– 1,080
Cat	760	– 1,520
Falcon	1,500	– 4,500
Robin	2,000	– 8,000
Grasshopper	7,000	– 100,000
Porpoise	7,000	– 120,000
Bat	10,000	– 120,000
Silent dog whistle	12,000	– 14,000

Figure 3.1: Approximate Ranges of Sounds Emitted from Common Sources

Picking Your Speaker

There are two sound-generator chips in PCjr. One is a single-note generator with a range from 37 hertz to 32,767 hertz. Above about 15,000 hertz, only your dog or cat will hear what's going on.

Do you have the BASIC prompt showing? Try this:

Type: SOUND OFF:BEEP OFF <ENTER> (Direct sound to internal speaker only)

Type: SOUND 37,8 <ENTER> (The Bronx Cheer)

The other chip generates sounds from 110 hertz (the A note one octave below middle C on the piano) to 32,767 hertz. Try this:

Type: SOUND ON:BEEP OFF <ENTER> (Direct sound to external speaker only)
Type: SOUND 37,8 <ENTER>

Notice you didn't get the Bronx Cheer, but something more pleasing, namely the A of 110 hertz. In fact, every frequency from 37 to 110 hertz will produce 110 hertz from this second chip. One way to verify this is to play 110 hertz on the internal speaker and compare it with the notes played on the external speaker:

Type: SOUND OFF:BEEP OFF <ENTER> (Direct sound to internal speaker only)
Type: SOUND 110,8 <ENTER>

What's the highest frequency you can hear? Try both the internal and the external speaker. You may get different answers. (Don't try too hard to find the highest hertz value you can hear. You'll have to raise the volume to do so, and you might hurt your ears.)

Not only can you switch sound outputs from the internal to the external speaker, you can play through both speakers simultaneously. Try this:

Type: SOUND OFF:BEEP ON <ENTER> (Use both speakers)
Type: BEEP <ENTER>

You cancel playing through both speakers by executing BEEP OFF.

What happens to the sound when you do SOUND ON:BEEP ON?

2. GENERATING INTERESTING SOUND PATTERNS

Now that you know some fundamentals, it's time to get down to cases and apply what you know.

My first approach to generating sound patterns was partly logical but mostly a random search for interesting noises. I'd think of a sound

I'd like to make and mentally analyze its component frequencies, durations and counterpoints—using data like that of Figure 3.1 to set a frequency range. Then I'd write a little program to produce the sound I wanted. Of course, when I tried the program, it wouldn't even come close to my intention, but the generated noise was often interesting and sometimes like some other familiar sound. So I'd play around with the program, and eventually I got a feel for how to generate various tone and noise patterns.

I heartily recommend that you do this type of noodling yourself. To get you started, and to stimulate your imagination, I have presented a small library of sound patterns in Figure 3.2 as a series of subroutines. Key the subroutine library into program memory and then SAVE it on your *Test Programs* diskette as an ASCII file. You'll then have it handy to incorporate into other programs you write.

When you have finished keying Figure 3.2 and have SAVEd it, try the various sound patterns. To call a particular sound pattern, the Ambulance, for example,

Type: GOSUB 2330 <ENTER>

For other sound patterns, substitute in the GOSUB command the line number of the remark statement that describes the pattern.

NOTE: These routines were written for the internal speaker and may not work as designed for the external speaker. Try all the sound patterns for each speaker separately, and note when the effect is acceptable and when it is not.

When you tire of listening to the sound pattern, type <Esc> to return to BASIC command mode. Be patient. Some of the subroutines may have to complete a cycle before they detect the <Esc>.

As you listen to a sound pattern, read the program lines that generate it so you can learn to associate the sound with the instructions that produce it.

```
2000 '  SOUNDLIB.BAS
2010 ' Subroutines for common or imaginative sounds
2020 ' to which the reader is to add his or her own.
2030 '
2040 ' TRUCKS PASSING ON THE HIGHWAY
2050 SOUND ON
2060 VEL=88:FREQ=110
2070 F=(1800/(1800-VEL))*FREQ
2080 FOR V=0 TO 15 STEP .01
2090 SOUND F,.06,V
2100 NEXT V
2110 F=(1800/(1800+VEL))*FREQ
2120 FOR V=15 TO 0 STEP -.01
2130 SOUND F,.06,V
2140 NEXT V
2150 A$=INKEY$:IF A$=CHR$(27) THEN SOUND
       OFF:RETURN ELSE 2060
2160 '
2170 ' EXTRATERRESTRIAL MACHINERY
2180 FOR I=600 TO 1400 STEP 250
2190 SOUND I,.55
2200 NEXT I
2210 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2180
2220 '
2230 ' BRITTLE PATTERNS
2240 R=5000*RND(1)+500
2250 SOUND R,1
2260 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2240
2270 '
2280 ' EXERCISES
2290 FOR K=100 TO 1000 STEP 100:SOUND K,2:NEXT K
2300 FOR K=1000 TO 100 STEP -50:SOUND K,1:NEXT K
2310 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
     ELSE 2290
```

```
2320 '
2330 ' AMBULANCE
2340 SOUND 700,7
2350 SOUND 800,7
2360 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2340
2370 '
2380 ' IMPATIENT BOY FRIEND
2390 FOR K=1 TO
       3:SOUND 400,4:SOUND 32767,1:NEXT K
2400 FOR K=1 TO 2500:NEXT:' DELAY
2410 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2390
2420 '
2430 ' THE CLOCK
2440 SOUND 550,3:SOUND 100,0:SOUND 440,4:SOUND
       880,0
2450 FOR I=1 TO 600:NEXT I
2460 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2440
2470 '
2480 ' THE CUCKCOO
2490 SOUND 900,3
2500 SOUND 725,4
2510 SOUND 32767,3
2520 FOR I=1 TO 500:NEXT I
2530 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2490
2540 '
2550 ' END OF PROGRAM SIGNAL
2560 SEC=20:T=1120*SEC
2570 SOUND 650,7
2580 FOR I=1 TO 100:NEXT
2590 SOUND 1100,7
2600 FOR I=1 TO 100:NEXT
2610 SOUND 880,8
```

```
2620 FOR I=1 TO T
2630 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
2640 NEXT I:GOTO 2570
2650 '
2660 ' BUCK ROGERS RAY GUN BLASTER
2670 FOR K=1 TO 5
2680 FOR I=200 TO 1600 STEP 120
2690 SOUND I,.19
2700 NEXT
2710 NEXT K
2720 FOR I=1 TO 875:NEXT
2730 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2670
2740 '
2750 ' WWI AIRPLANE ENGINE
2760 FOR I=40 TO 80
2770 SOUND I,.03
2780 NEXT I
2790 A$=INKEY$:IF A$=CHR$(27) THEN RETURN
       ELSE 2760
```

Figure 3.2: SOUNDLIB.BAS—A Beginning Library of Sounds and Noises.

3. A MORE SCIENTIFIC APPROACH

The Frequency-Time Diagram

You can have lots of fun trying all sorts of combinations of SOUND, but for our purposes here, a more organized approach might be helpful.

Have you ever seen the analysis of a birdcall on a TV nature show? The naturalist feeds a tape recording of the sound into a spectrum analyzer, which plots the various pitches that make up the sound onto a time scale. The result looks like a series of painted strokes of various lengths. Let's see if we can use this idea of a sound spectrum to discover the constituent parts of noises and everyday sounds.

Figure 3.3 shows a sawtoothed curve, one of the simplest examples of a frequency-time diagram. The vertical axis represents the sound's pitch (or frequency). The horizontal axis represents time. The leftmost slash on the diagram illustrates a repeating sound pattern that starts at 1,000 hertz and rises to 4,000 hertz over a one-second period. Then there's a one-second delay before the rising sound is repeated.

Let's find out what this diagram sounds like when it's programmed for PCjr. As we go along, see how many familiar sounds you can think of that have this simple sawtoothed character. You'll use your list later to test the program you're about to develop.

Programming the Frequency-Time Diagram

Do you have a clean program memory? If not,

Type: NEW <ENTER>

Now you'll need a SOUND command to create whatever pitch you want.

```
160 SOUND X,T,12,0
```

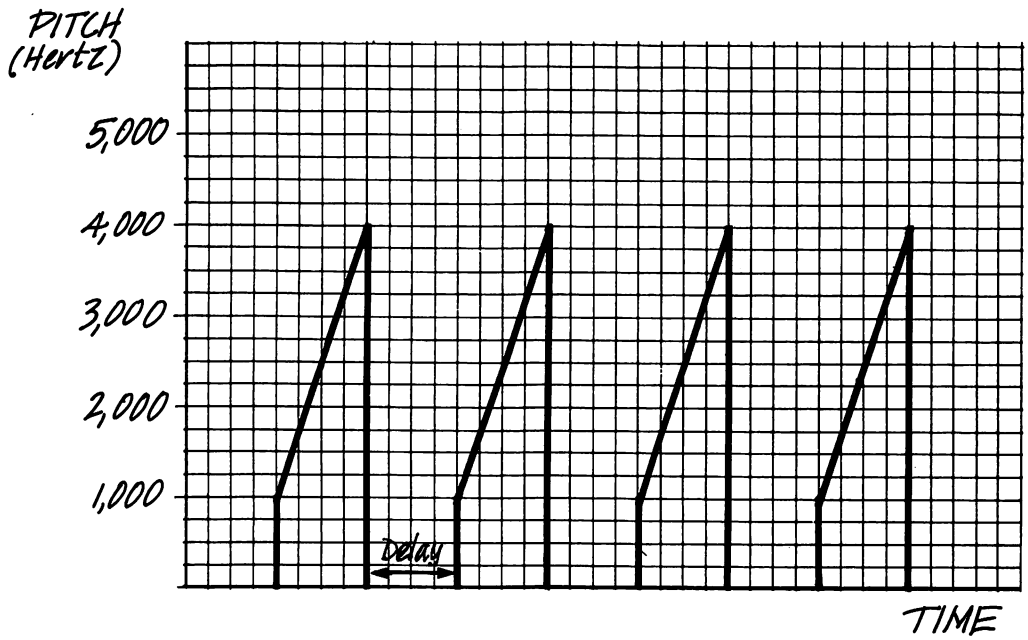


Figure 3.3: Repeated Single-Stroke Sound Pattern

This instruction plays a pitch of X hertz (unspecified yet), for a duration of T computer clock times (unspecified yet), at loudness 12, through voice channel 0.

You know what pitch is—the highness or lowness of a tone. But what about a duration of T computer clock times? The explanation is simple. PCjr has a clock inside. There are 18.2 clock times per second. This means that one computer clock time is about 0.055 of a second. So if we put T=1, pitch X will sound for 0.055 second.

The loudness or volume number controls the decibel level of the sound. A value of 0 turns the sound off. The highest value permitted is 15.

The music chip used with the external speaker permits you to play up to three voices together. They are numbered 0, 1 and 2.

The next step is to specify the pitch of the sound. One value won't do, of course. According to Figure 3.3, the pitch ranges from 1,000 to 4,000 hertz. Let's call the lower pitch limit A and the upper pitch limit B. Then all you need do is sandwich the sound statement in a FOR...NEXT loop this way:

```
150 FOR X=A TO B STEP S
170 NEXT
```

LIST the growing program so you can see the sandwich.

Although the figure shows the pitch varying continuously, we'll move along by steps of length S.

NOTE: I've omitted the index (i.e., pitch) variable X from the NEXT command to make the loop operate faster.

To see whether what you have so far works correctly, set up the variables A, B, S and T manually for this test. Try these values:

```
Type: A=1000:B=4000:S=150:T=.5 <ENTER>
Type: GOTO 150 <ENTER>
```

The sound should have zipped along from low to high pitch. If it didn't, check your typing. And if that's not the problem, you may have accidentally turned the sound off. Type: SOUND OFF:BEEP OFF <Enter> and try again.

Marking Time

The next step is to put in the delay before repeating the upward-soaring noise.

```
190 SOUND 32767,18.2*S2,,0
```

Keep LISTing as lines are added so you appreciate the relations among commands and the thought process involved in building a program.

With a pitch of 32767, you won't hear a sound at all. This is equivalent to a soundless delay of the specified duration. Notice that seconds of delay, S2, are converted to clock ticks by multiplying by 18.2. There's no need for volume, but the delay should occur in voice channel 0. Thus, the extra comma is needed to tell PCjr that 0 is to be interpreted as "voice," not "volume."

Closing the Loop

Next, we've got to repeat the sound pattern by starting again at line 150.

```
200 GOTO 150
```

Let's see if it works. Just proceed as before.

Type: A=1000:B=4000:S=150:T=.5 <ENTER>
Type: GOTO 150 <ENTER>

Setup Preliminaries

You're almost finished. You've already defined A and B, the lower and upper pitch limits, and S2, the duration of the delay in seconds. Call the duration of the sound itself S1 (in seconds). Suppose you divide this sound interval into N parts. Then you can define all the data needed to reproduce the sound of Figure 3.3 in a line of coding like this:

```
110 A=1000:B=4000:S1=1:S2=1:N=50
```

The number of clock ticks, T, for the SOUND command is then calculated from the value of S1 this way:

```
120 T=18.2*S1/N
```

And the interval step of the FOR...NEXT loop is simply

```
130 S=(B-A)/N
```

Finally, a couple of technicalities.

```
140 SOUND ON:BEEP OFF:PLAY "MB"
```

Of course, you've already learned that SOUND ON:BEEP OFF directs sound to the external speaker. PLAY, which we'll use extensively later, is a special command that tells PCjr to run sound in the Background Mode. That is, it permits the program to go ahead and do its thing while the sounds play in the background from notes placed in a sound buffer (a special part of the computer's memory set aside for that purpose).

Don't Leave Pieces Out

Before we go any further, there's an error we must take care of. The SOUND statement will not work for a duration of under 0.015 clock ticks, so if we specify a zero delay (i.e., S2=0) the program won't work. Let's test for this case.

```
180 IF 18.2*S2<.015 THEN 150
```

Finishing the Job

Now label it:

```
100 ' SINGLE STROKE SOUND PATTERN
```

And save it.

Type: SAVE"1-STROKE.BAS" <ENTER>

Does the program work?

Type: RUN <ENTER>.

If it doesn't work, check your typing against the text above. And if you make corrections, don't forget to SAVE the program again.

For ready reference, make a printout of 1-STROKE.BAS with the LLIST command.

Discovering the Potential of 1-Stroke

As we've gone along, have you thought about what kinds of familiar sounds might be imitated by a one-stroke sound pattern? To imitate them with our program you must guess at A, B, S1, S2 and N. Then you plug these guesses into line 110 and RUN. If your guess is not acceptable, you've got to fiddle with the values until you find the best imitation.

To get you started, here are a few examples you might try. Just plug the indicated values into line 110 of 1-STROKE.BAS.

	A	B	S1	S2	N
Whooper Siren	800	1000	1.0	0	50
Busy Signal	50	100	.3	.8	10
Telephone Ring	500	505	.5	2.0	120
Hungry Chick	3500	2000	.2	.8	15

Try running these, as well as those you make up yourself, in the foreground. That is, substitute "MF" (Foreground Mode) for "MB" (Background Mode) in the PLAY instructions of line 140. Note the different effects. You may be able to exploit them later on.

CAVEAT: If you find that your changes in A, B, etc. are not effective, you probably forgot to type <Enter> to register the changes before you moved to another line to do RUN or the equivalent.

4. MULTIPLE-STROKE VOICE PRINTS

As you discovered pretty quickly, the program 1-STROKE.BAS has limited possibilities. The next obvious step is to work with voice prints of two or more different strokes, separated by time delays—including zero delay, of course. Figure 3.4 shows a possible pattern for two strokes.

The Police Siren

The easiest way to program Figure 3.4 is to run 1-STROKE.BAS twice in tandem, but with different values of A, B, S1, etc. Actually, the

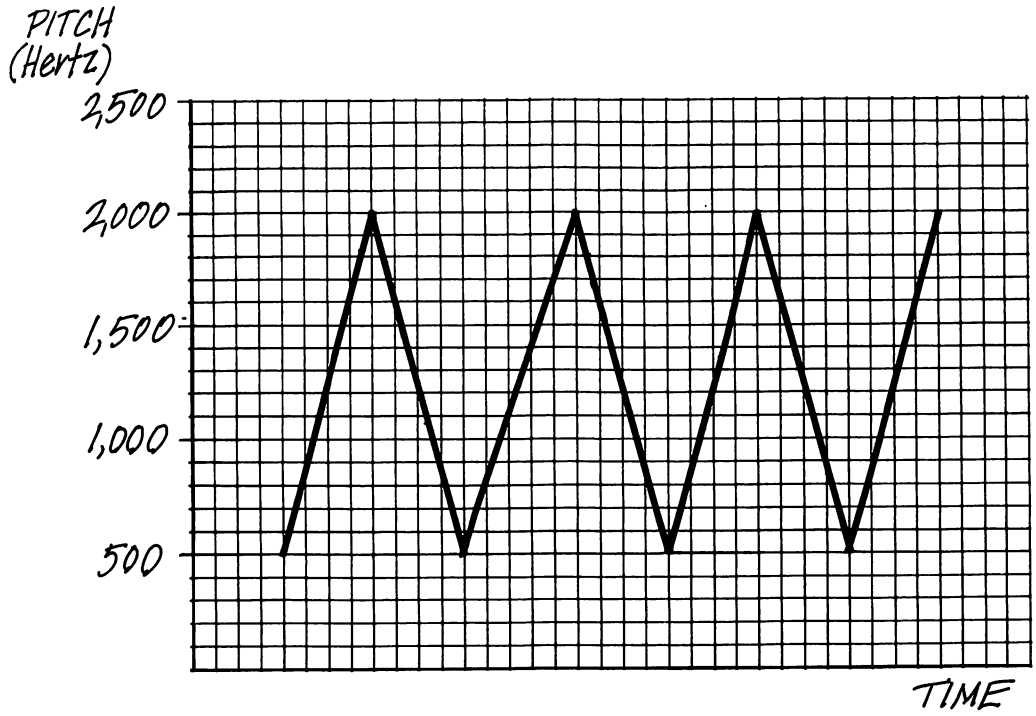


Figure 3.4: Repeated Double-Stroke Sound Pattern (Police Siren)

program-building process is more like merging 1-STROKE.BAS with itself, because you want to put the setup commands together up front. Figure 3.5 shows how to do the merge. The data used here emulates a police siren.

You can see by comparing 1-STROKE.BAS with 2-STROKE.BAS that an index number, 1 or 2, has been appended to each variable so that you'll know to which stroke it applies. For example, A1 and A2 are the lower pitch limits of stroke 1 and stroke 2, respectively.

Notice, too, that I've introduced new variables, V1 and V2, to represent the volumes of the strokes. Varying volume is important if you want one of the sound strokes to dominate.

A different scheme has also been used to create time delays, namely the FOR...NEXT loops of lines 230 and 280. A count of 875 is

```
100 ' DOUBLE STROKE SOUND PATTERN
110 ' INITIALIZE
120 A1=500:B1=2000:S11=.8:S21=0:N1=40:V1=12
130 A2=2000:B2=500:S12=.8:S22=0:N2=40:V2=12
140 T1=18.2*S11/N1
150 T2=18.2*S12/N2
160 S1=(B1-A1)/N1
170 S2=(B2-A2)/N2
180 SOUND ON:BEEP OFF:PLAY "mb"
190 ' FIRST STROKE
200 FOR X=A1 TO B1 STEP S1
210 SOUND X,T1,V1,0
220 NEXT
230 FOR I=1 TO 875*S21:NEXT
240 ' SECOND STROKE
250 FOR X=A2 TO B2 STEP S2
260 SOUND X,T2,V2,0
270 NEXT
280 FOR I=1 TO 875*S22:NEXT ' Pause before repeat
290 GOTO 200 ' Repeat cycle
```

Figure 3.5: 2-STROKE.BAS

close to one second on PCjr. So multiplying 875 by S21 and S22 (delay in seconds) does the trick. This programming device works for a delay of zero without need for the test introduced in line 180 of 1-STROKE.BAS.

Applying 2-STROKE.BAS

The police siren was easy to conceive. However, you have to concentrate a little harder now to analyze a two-stroke voice print. There are so many variables. One way to develop skill in generating voice prints is this:

Take a piece of paper and make columns headed A1, B1, S11,..., N1, A2, B2,..., N2—all the variables involved. Now select values for

these variables and plug them into 2-STROKE.BAS. Listen to the sound produced and classify it using common noise words: click, burp, wail, chatter, rumble, buzz, chirp, warble, etc. Your early experiments should involve large changes in the variables so that you're not stuck trying to distinguish between sounds that are too similar.

As soon as you've developed ten to twelve different sounds, you'll begin to notice that particular types occur for given combinations of the variables—e.g., high short with low long pitch, short with long play interval, medium with high volume, many with few interval counts (i.e., N1 versus N2). Notice that when durations of individual sounds are relatively long, there is a loss of continuity in the sound pattern.

To get you started, here are a few examples to try.

Bloppy Sucking Sound

This is a funky boat's engine, like the one on the African Queen. Or perhaps a mad scientist's lab equipment in operation. Be sure to do it on the external speaker.

```
120 A1=400:B1=5000:S11=.1:S21=0:N1=3:V1=4
130 A2=50:B2=999:S12=.05:S22=0:N2=9:V2=12
```

CAVEAT: As you key these examples, take care that your eye doesn't jump to the wrong line as you're typing. Lay a straightedge just under the line being typed as a guide.

Crickets

Here's how you might simulate the sounds of a hot summer's night.

```
120 A1=1800:B1=1810:S11=.1:S21=0:N1=20:V1=12
130 A2=1500:B2=1510:S12=.1:S22=0:N2=30:V2=6
```

Bomb Drop—Sort Of!

This poor man's Stuka dive bomber could do with improvement. What can you do with it?

```
120 A1=6000:B1=1500:S11=1.8:S21=1.5:N1=100:V1=12
130 A2=110:B2=125:S12=.1:S22=1:N2=30:V2=12
```

Frogs

Try these parameters:

```
120 A1=50:B1=40:S11=.1:S21=.3:N1=15:V1=15
130 A2=40:B2=50:S12=.1:S22=.3:N2=15:V2=15
```

It doesn't sound much like frogs yet, but the reason isn't in the parameters. The two sound chips have different effects on the sound equipment. The TV speaker seems to pick up a click when the multi-voice chip is activated and deactivated.

Turn SOUND OFF in line 180, so that the program is played through the internal speaker, and try it.

Now try all the previous sounds with this single-voice chip, that is, with SOUND OFF, to hear what happens.

Has it occurred to you that it may be appropriate to play part of a sound pattern through one chip and another part through the other chip? Just use SOUND ON and SOUND OFF to do the switching back and forth.

5. BIRDCALLS

The Chestnut-sided Warbler

Of course, you now see how you might expand 2-STROKE.BAS to three or more strokes and develop all sorts of sound pattern combinations. There's one variation on the 4-stroke generator I'd like to show you. It's really an N-stroke generator, but the middle N-1 strokes are all the same.

Figure 3.6 illustrates the case. It depicts one of the calls of the Chestnut-sided Warbler.

Figure 3.7 is a program to generate the sound patterns corresponding to Figure 3.6:

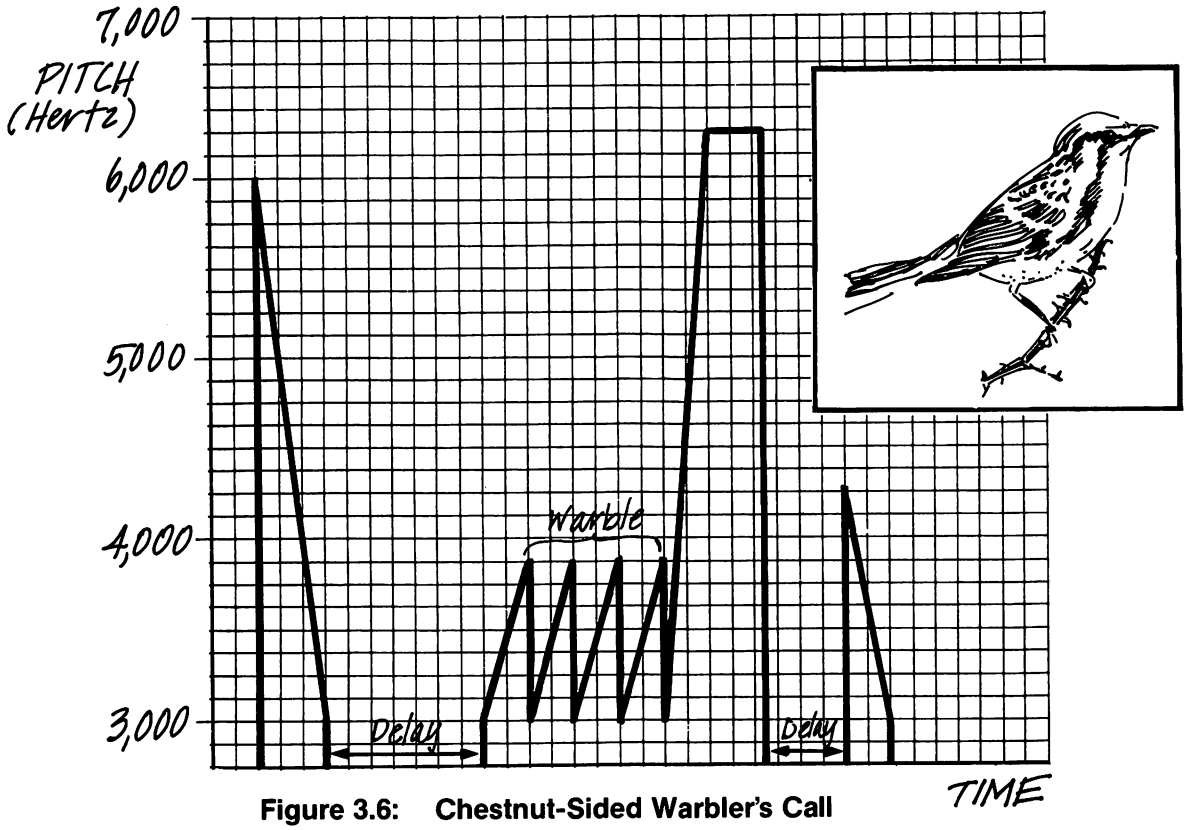


Figure 3.6: Chestnut-Sided Warbler's Call

```
100 ' WARBLER.BAS, CHESTNUT-SIDED
110 ' initialize
120 SOUND OFF:BEEP OFF:PLAY "mb" ' Select
      internal speaker
130 A1=6000:B1=3000:S11=.2:S21=.9:N1=15 '
      Define notes
140 A2=3000:B2=3800:S12=.1:S22=0:N2=6
150 A3=3000:B3=6200:S13=.05:S23=.5:N3=9
160 A4=4200:B4=3000:S14=.1:S24=2:N4=5
170 T1=18.2*S11/N1 ' Define note durations
180 T2=18.2*S12/N2
190 T3=18.2*S13/N3
```

```
200 T4=18.2*S13/N4
205 T5=.3
210 S1=(B1-A1)/N1 ' Define steps
220 S2=(B2-A2)/N2
230 S3=(B3-A3)/N3
240 S4=(B4-A4)/N4
250 ' first stroke
260 FOR X=A1 TO B1 STEP S1
270 SOUND X,T1,12,0
280 NEXT
290 FOR I=1 TO 875*S21:NEXT ' Delay
300 ' second stroke
310 FOR K=1 TO 4 ' Create warble of second stroke
320 FOR X=A2 TO B2 STEP S2
330 SOUND X,T2,8,0
340 NEXT
350 NEXT K
360 FOR I=1 TO 875*S22:NEXT ' Delay
370 ' third stroke
380 FOR X=A3 TO B3 STEP S3
390 SOUND X,T3,10,0
400 NEXT
405 SOUND B3,T5,10,0 ' Sustain B3 slightly
410 FOR I=1 TO 875*S23:NEXT ' Delay
420 ' fourth stroke
430 FOR X=A4 TO B4 STEP S4
440 SOUND X,T4,6,0
450 NEXT
460 FOR X=1 TO 875*S24:NEXT ' Pause before repeat
470 GOTO 260 ' Go repeat call
```

Figure 3.7: WARBLER.BAS

Notice that the principal deviation from the general program design of 2-STROKE.BAS is the embedding of the second stroke in a FOR...NEXT loop, starting at line 310. In this way the stroke may be repeated a number of times. Try modifying this program so that varying volumes may be used.

Does a change in volume make any difference when you use the internal speaker as we've done here?

The Love Call of the Flicker

The same principles we've been learning can be simplified to some extent by leaving out lines of coding that are redundant—e.g., zero delays between strokes. Then you may not feel so constrained to fit in additional sounds, sustaining the first and/or last sound of a stroke.

Let's do an example by emulating the love call of the Yellow-shafted Flicker, modeled in Figure 3.8.

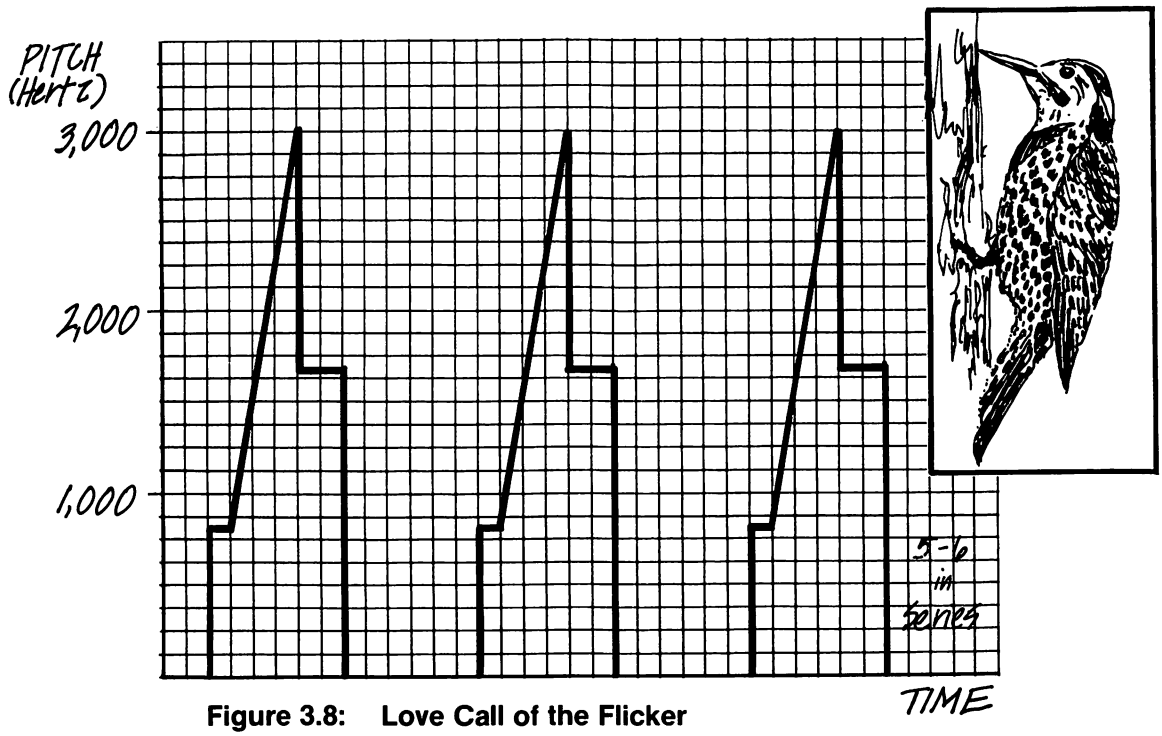


Figure 3.8: Love Call of the Flicker

The program corresponding to Figure 3.8 is given in Figure 3.9.

```
100 ' FLICKER.BAS—COURTSHIP CALL
110 SOUND OFF:BEEP OFF:PLAY "MB"
120 A=900:B=3000:C=1400
130 S=525:T1=.05:T2=.05:T3=.1:D=.02
140 FOR K=1 TO 6 ' Create repetition of pattern
150 SOUND A,T1 ' Sustain first note
160 FOR I=A TO B STEP S ' Play first stroke
170 SOUND I,T2
180 NEXT
190 SOUND C,T3 ' Sustain third note
200 FOR I=1 TO 875*D:NEXT ' Delay between stroke
    patterns
210 NEXT K
220 FOR I=1 TO 875*2:NEXT ' 2-sec delay between
    calls
230 GOTO 140 ' Go repeat call
```

Figure 3.9: FLICKER.BAS

The Wood Pewee's Call

There's another example, a rather musical one, I'd like to share with you. It's the Wood Pewee's call, diagrammed in Figure 3.10.

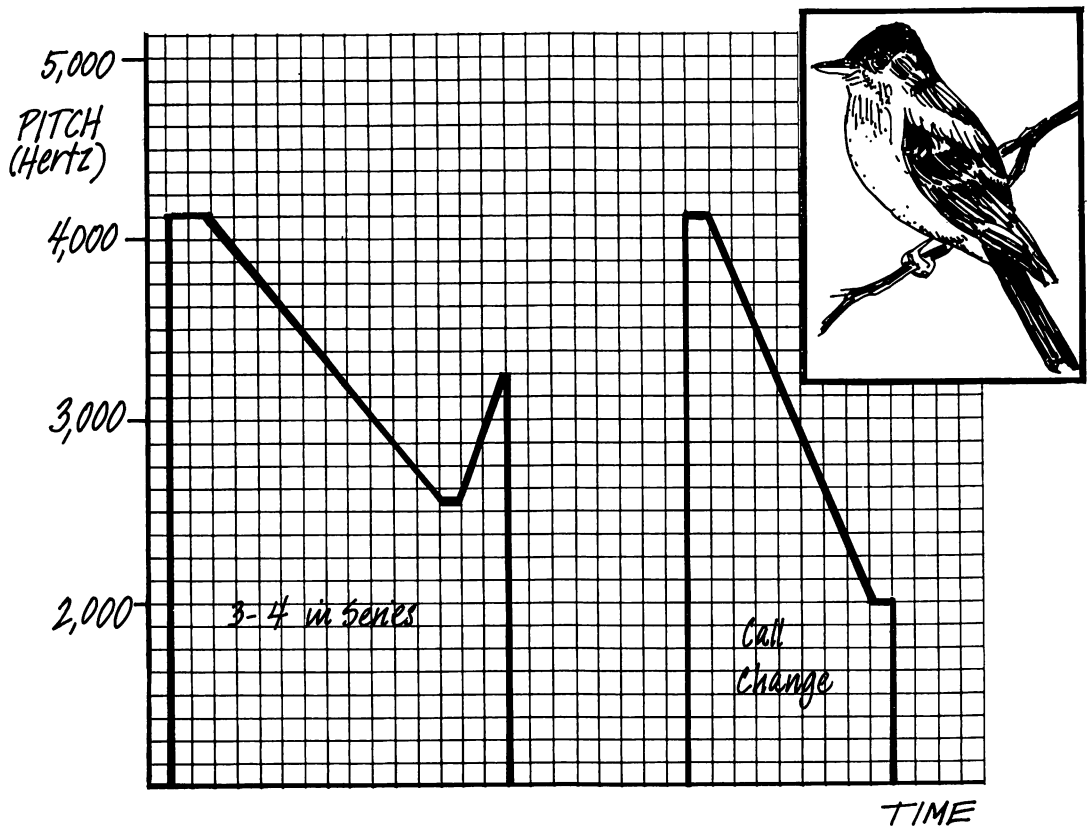


Figure 3.10: The Wood Pewee's Call

The program corresponding to Figure 3.10 is given in Figure 3.11. As you read through the program, notice how it is built from the same principles we've used already, though it's perhaps a little more involved than the others.

```
100 ' WOODPWEE.BAS
110 A=4100:B=2600:C=3400:E=2000
120 T1=1:T2=.4:T3=.6:T4=.1:T5=.3:T6=.2:T7=.2
130 S1=-100:S2=50:S3=-100
140 D0=.8:D1=.3:D2=2.5:V=6
150 SOUND OFF:BEEP OFF:PLAY "MB"
160 FOR K=1 TO 3 ' Repeat first sound pattern
170 SOUND A,T1,V ' Sustain first note
180 FOR I=A TO B STEP S1 ' Do first stroke
190 SOUND I,T2,V
200 NEXT
210 SOUND B,T3,V ' Sustain second note
220 FOR I=B TO C STEP S2 ' Do second stroke
230 SOUND C,T4,V
240 NEXT
245 FOR I=1 TO 875*D0:NEXT
249 NEXT K
250 FOR I=1 TO 875*D1:NEXT ' Delay before second
    pattern
260 SOUND A,T5,V ' Sustain starting note of stroke
270 FOR I=A TO E STEP S3 ' Do stroke
280 SOUND I,T6,V
290 NEXT
300 SOUND E,T7,V ' Sustain ending note
310 FOR I=1 TO 875*D2:NEXT ' Delay before repeating
320 GOTO 160 ' Repeat call
```

Figure 3.11: WOODPWEE.BAS

Sources of Birdcalls

There's no guarantee that the birdcalls simulated here sound like the real thing. I've never heard the birds in question. If realism is what you require, you'll have to make the comparison yourself.

I was guided by a wonderful little book by Aretas A. Saunders, *A Guide to Bird Songs* (Doubleday, 1951). You should try to borrow it from the library. It's full of diagrams with the same flavor as Figures 3.6, 3.8 and 3.10. You can program them just as we've done here.

Saunders gives the approximate pitch of the key note in each call, and a phonetic rendition. This is generally enough information to approximate the frequencies involved.

Some bird identification books, such as *Birds of North America*, by Robbins, Bruun and Zim (Golden Press, 1966), contain audiospectrograms, or sonagrams, of many of the birdcalls. These frequency-time diagrams were developed from Saunders' work. However, they were generated by scientific instruments and contain all the overtones and sound variations the birds produce. These voice prints can be quite difficult to interpret for writing a simple computer program. What does one include, exclude? In fact, most overtones shown in these sonagrams are drowned out by the principal tone and you can't even hear them. So until you develop an interpretive skill, you'll probably be more successful using Saunders' little book.

6. WHAT DO YOU KNOW NOW?

Are you following along okay? So far the strategy has been not to introduce too many programming ideas, but to exploit simple, ever so powerful, FOR...NEXT loops. The concentration here has been on sound patterns themselves, and how by changing frequency ranges and durations you can mimic things.

The more you practice making sound patterns, the better you'll become at it. So why don't you go back through the chapter once more and review each example. Ask yourself how you might improve the different sounds and try out your ideas. Then ask yourself what other sound patterns might be generated if you changed a frequency range here or a duration there in the example. Be sure to record your results for future reference.

Also be sure you understand the BASIC commands used. Figure 3.12 summarizes those used in the chapter. Go down the list and check your recollection of their functions. When you're not sure, look them up in the BASIC manual, and check back to where they're used in the chapter.

BASIC COMMAND	FUNCTION
FOR...NEXT	Repeat a sequence of commands
GOSUB . . . RETURN	Call and return from subroutine
GOTO	Take next command from line indicated
INKEY$	Read character from type-ahead buffer
PLAY "mb"	Play sounds in background
RND	Generate random number between 0 & 1
SAVE filespec,A	SAVE program on disk in ASCII format
SOUND ON:BEEP OFF	Direct sound to external speaker only
SOUND OFF:BEEP ON	Direct sound to both speakers
SOUND OFF:BEEP OFF	Direct sound to internal speaker only
SOUND ON:BEEP ON	Turn off both speakers
SOUND	Play pitch of specified duration & volume

Figure 3.12: Summary of Commands in Chapter 3

CHAPTER 4
EUTERPEAN MUSINGS

Objective:

This chapter presents PCjr's easy-to-learn musical language.

1. THE FUNDAMENTALS OF PCjr MUSIC-MAKING

Scales and Stuff

All music, whether it be classical, jazz, bluegrass, rock, sacred or whatever, is organized around fundamental musical structures called scales. Whether you realize it or not, you're probably already familiar with one of the most basic scales. It's called the diatonic major, and you give a rendition of it every time you sing, "Do, re, mi, fa, sol, la, ti, do." "Diatonic" simply means the tones that correspond to the white keys on a piano.

PCjr also knows this scale. Here it is in the key of C:

Type: SOUND ON:BEEP OFF <ENTER>
Type: PLAY "O3CDEFGAB>C" <ENTER>

That's a letter O for "octave" (eight tones of progressing degrees) in the PLAY command, not the digit zero. Don't get them confused as we go along.

The O3 selects the octave beginning at middle C (a note that's close to the middle of the piano keyboard and that occupies the space between the treble staff and bass staff on a sheet of music). The eight notes of this scale, called the C Major scale, are named C D E F G A B C. As you can see, the eighth, or octave, note of this scale has the same name as the first, and this is true of the first and last notes of every scale. To keep PCjr from getting confused, the > tells it to move up to the C in the next octave, O4, rather than jump back to the C note at the beginning.

NOTE: The eighth note of any octave is also the first note of the next octave. All octaves on PCjr start at C.

Changing Octaves

The O, or octave, command lets you select the octave in which you want to play the lettered notes. There are seven octaves on PCjr, numbered 0 to 6. You can see from Figure 4.1 where some of them fall on the piano keyboard. But try out the diatonic scale in some of the other octaves so that you can hear the differences. For example,

Type: PLAY "O1CDEFGAB>C" <ENTER>

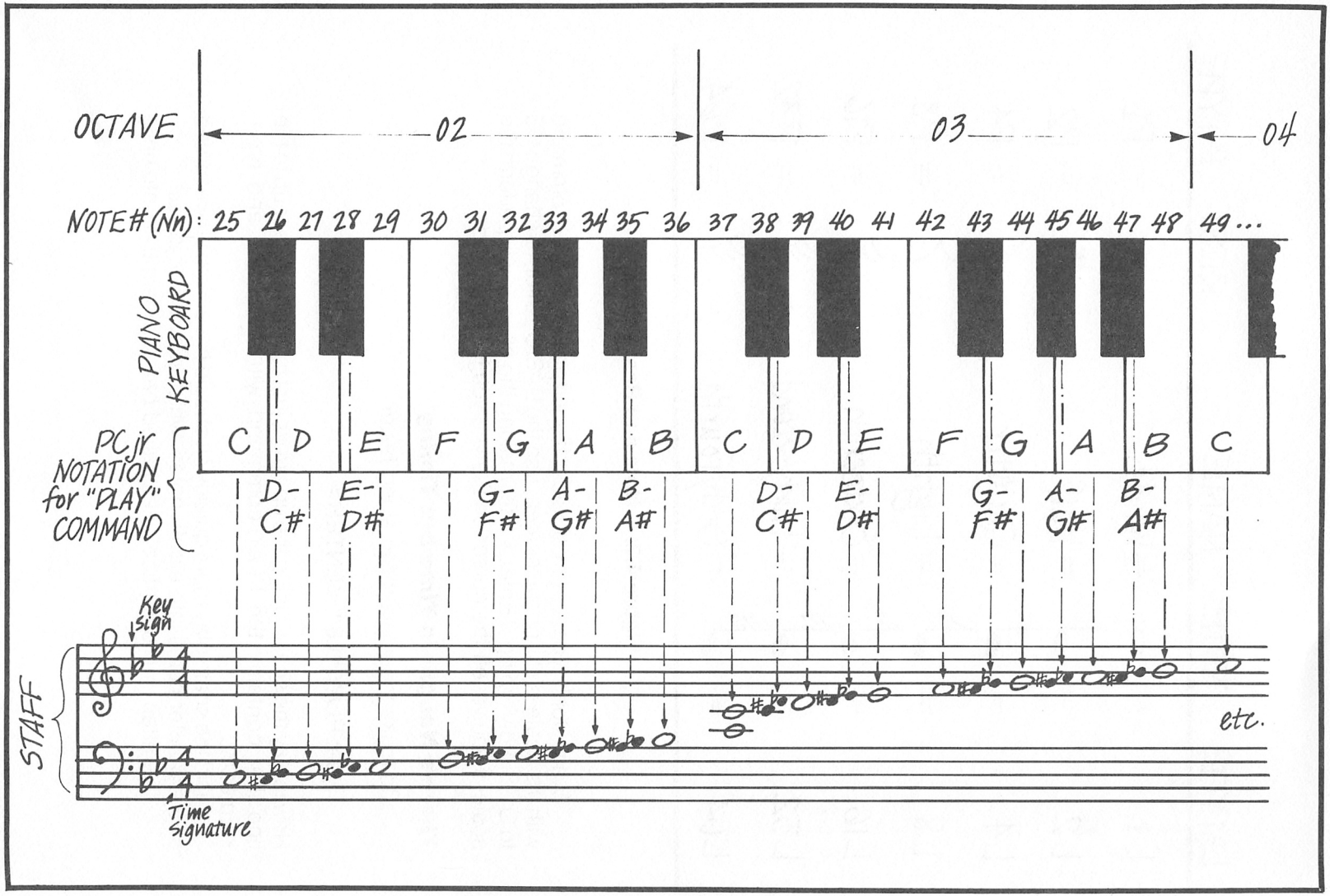


Figure 4.1: PC jr's Music Staff

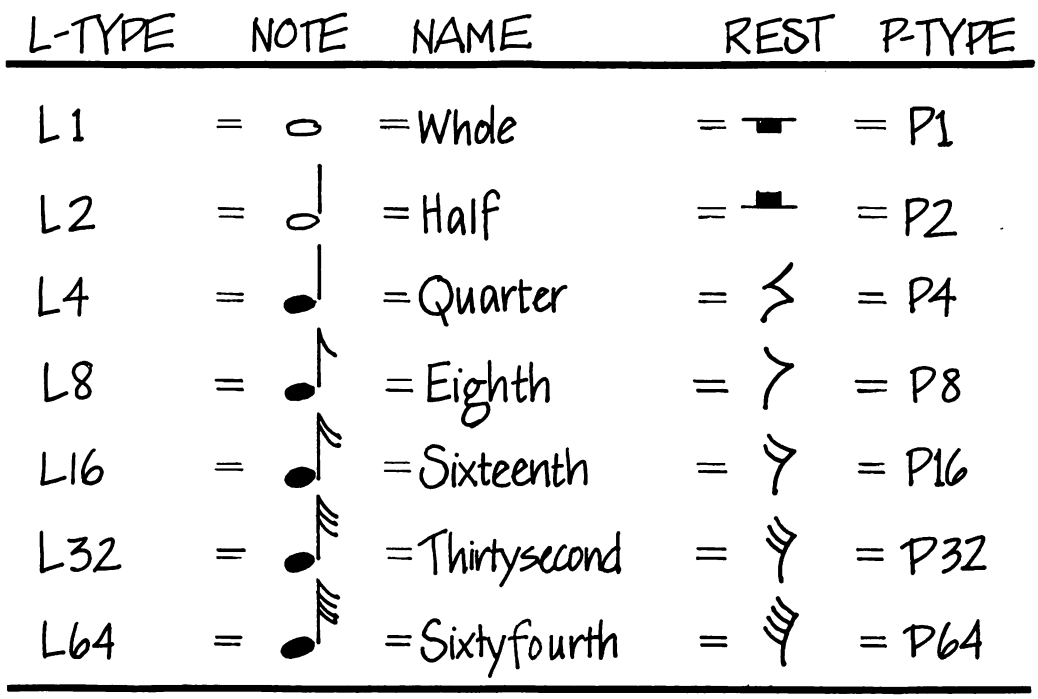


Figure 4.2: Note and Rest Values

Wait! Don't type it from scratch. If your previous PLAY command with the scale is still on the screen, use the BASIC editor, to change O3 to O1. Simply typing <Enter> after making the change will perform the scale, if you've left the cursor on the line you just changed.

There's Naught in Time but Timing

To make the scale move along faster,

Type: PLAY "L8" <ENTER>

Now replay the scale. Using the up arrow, just move the cursor up to the line that contains the PLAY command with the scale and type: <Enter>.

Did the scale play more quickly? It should have. With L8 we turned all the quarter notes into eighth notes, which play twice as fast.

In written music, vertical lines called *bars* divide music into units

of time called *measures*. At the beginning of a musical composition a regular number of beats is assigned to each measure by the *time signature*, which consists of two numbers, one written over the other. The most common signature, 4/4, indicates by the top number that there are four beats assigned to each measure, and by the bottom number that a quarter (1/4) note is the type that will signify one beat. Thus, a measure in 4/4 time can contain four quarter notes or eight eighth notes (two 1/8 notes equal one 1/4 note, just as the fractions 1/8 + 1/8 = 1/4). Types of notes can also be combined so that you might fill one of these measures with two quarter notes and four eighth notes. So long as the total of their time values (i.e., beat) equals four, or their fractional values (1/4 + 1/4 +1/8 + 1/8 +1/8 + 1/8) equal one, you're legal.

Figure 4.2 shows the standard musical notes and indicates how you specify them with the L, or note length, command.

Plug some different note lengths into the diatonic scale we constructed earlier to hear what happens to the playing speed. Hold it! Here's a short cut:

Type: PLAY "L16O3CDEFGAB>C" <ENTER>

Then, simply change the 16 to a 32 or a 1 or whatever. As a matter of fact, you can substitute any integer between 1 and 64 into the note length command. You can get some pretty weird timing combinations this way. Mostly you'll avoid using this facility unless you want to reproduce some kinds of jazz and other modern music, which steal a little time here and add it there in ways difficult to write on the musical staff.

Sharps and Flats

Another musical scale, the *minor* scale, renders a different musical effect from the major. Listen for yourself.

Type: PLAY "L4O3CDE-FGA-B>C" <ENTER>

The minus sign (-) tells the computer to flat the lettered note preceding it. Two notes, however, C and F cannot be flatted in PCjr's musical language, though they can be meaningfully represented in normal musical notation. Try to flat C and F to see what happens.

Another scale of interest is the *whole-tone scale*, much exploited by Debussy and other modern composers.

Type: PLAY "O3L4CDEF#G#A#>CDC<A#G#F#EDC" <ENTER>

The sharp sign (#) tells PCjr to play the lettered note as a sharp (a tone that is halfway up to the next note in the scale). However, B and E may not be sharped. You'll get an "Illegal function call" message. If you want B# or E#, play C or F, respectively, instead. You'll get exactly the tones you want. NOTE: You may use the plus sign (+) instead of #.

Sharps and flats correspond to the black keys on the piano.

In the above example, you saw how to change from octave O3 to octave O4 by using the greater-than sign (>), and back again using the less-than sign (<). If you want to jump up two octaves, use two >'s in succession before playing the next note. Try the whole-tone scale in several combinations of octaves and tempos (i.e., speeds).

Figure 4.1 shows the possible note letter combinations you can use with PCjr, and where they are located on the piano and on the music staff.

Dotted Notes

In written music, you can extend the time value of a note by one-half simply by writing a dot (.) just to its right. You can do the same in PCjr's musical vocabulary. Try this funky pentatonic (five-tone) scale, in which every other note is dotted.

Type: PLAY "O3L4CD.EG.A>C.DE.GA." <ENTER>

Sort of gives a syncopated feel, doesn't it?

Rests

Sometimes the most attractive thing about a piece of music is the way the composer uses silence. Silence is denoted in written music by the *rest*. On PCjr the P, or pause, command is used to perform a rest. Try this example:

Type: PLAY "O3L8FP8>C<FE-P8B-E-D-P8A-D-C" <ENTER>

Now let's speed it up a bit by changing the tempo. To make the change

Type: PLAY "T230" <ENTER>

Now move the cursor up to the line of the previous PLAY command and

Type: <ENTER>

That moves along now. To slow it down, change the tempo to something like T80 and try the example again. You may vary the tempo from 32 to 255 quarter notes per minute. The default value is 120.

Look at the table of tempos given in Chapter 4 of the BASIC manual, under the SOUND command. It gives the standard music designations of tempo and the quarter-note-per-minute (beats) equivalent ranges.

The T command can be incorporated anywhere in a string of PCjr music commands to change the tempo as the piece progresses.

Volume

Another important aspect of musical performance is dynamics—how the *forte* (loud) and *piano* (soft) of the piece interplay. Of course, you can control the overall sound level by using the TV's volume control knob. But to vary the sound level within a piece, you need PCjr's V (volume) command.

Try this example. But before you do, turn the TV volume down. You don't want to hurt your ears. You can gradually raise the TV's volume to enhance the contrasts.

Type: PLAY"V12AV4AV8A" <ENTER>

You say you didn't hear any difference? Well, you need to have SOUND ON, to activate the external speaker, for the volume control to be effective.

The volume number may range from 0 (silence) to 15 (pull out all stops). The default value when you start is 8.

2. PCjr—THE MELLOW FELLOW

By now you should have a sense that you don't need to be conservatory-trained to get PCjr to make music, but the more you know, the better. It is desirable that you at least become familiar with the rudiments of reading music notation. Much of what you'll want to do musically with PCjr amounts to transferring notes from printed music to PCjr's music notation. But even if you only know how to play a tune on the piano with one finger, you should still be able to transfer what you can play to PCjr's music notation and then replay it on the computer.

If you don't have a piano, we will soon fix you up with a little substitute so that you won't have that excuse to hide behind.

For now, let's give the music transcription business a go.

Find the Starting Note

Here's a tune everybody can pick out. Hum "Three Blind Mice." Now go to the piano and locate the opening three notes—to which you sing the words "three blind mice." (It's okay to use a guitar, or flute, or whatever.)

Have you got the keys? Okay, look at Figure 4.1 to identify them. These notes orient you for the rest of the tune.

I don't know what your starting note is, but I started with E just above middle C, so my first three notes are E, D, C. As you know by looking at Figure 4.1, they belong to the O3 octave. Of course, these same three notes are echoed in the next phrase, a repeat of "three blind mice."

Define Your Basic Note Value

In the third phrase, "See how they run," there's an extra syllable to deal with, which gets us right to the question of timing and note value.

The first two phrases of the tune count off this way (count by tapping your foot, one tap per beat):

Notes	E D C -	
Words	Three blind mice (pause)	
Count	DUM – DUM – DUM – silent beat	(repeat)

Each phrase has four beats, so we'll try using a quarter note (four to the bar) as a basic unit. Looking at Figure 4.2, we see that the quarter note is defined by L4.

Let Your Ear Guide You

We've got enough data to try something on PCjr now.

Type: PLAY "O3 L4E D C" <ENTER>

Separating notes by spaces makes it easier to read music phrases, but it's not essential to do so.

What we've just done is fine for the first phrase. But remember, there's a pause before it's repeated.

Golden Moments of Silence

The pause has the same duration as the syllables, which are all quarter notes, so we need a quarter rest. Look in Figure 4.2 again. You'll see that the quarter-note rest, or pause, is specified by P4. Now move the cursor up to the PLAY command and make the addition:

Type: PLAY "O3 L4E D C P4" <ENTER>

We're getting there.

Save the Correct Answer

Now that we've got something working, we should save it in program memory so we don't have to key it in and debug it all over again. Don't clear the screen, but clear program memory with NEW, then move the cursor up to the P in PLAY. Carefully,

Type: <Ins> (The Insert key)
Type: 120 <ENTER>

Repeat this operation to save the second phrase, which echoes the first; that is, change the 120 line number on the screen to 130 and hit ENTER. You should now have the first two phrases of "Three Blind Mice" in memory. To play it, we should select a speaker.

Type: 110 SOUND ON:BEEP OFF

Type RUN, and it should play for you. LIST, and you'll see the two PLAY commands.

Splitting Note Values

Next we have "See how they run" to deal with. First, on what note does the phrase start? It's higher than E, so fish along up the scale on the piano till you find it. Since I started on E, this next phrase starts on G for me.

The count is a little more complex this time. Here's how I see it:

Notes	G F F E -	
Words	See how they run (pause)	
Count	DUM – DUM-di – DUM – silent beat	(repeat)

The "see" and the "run" still have the feel of the quarter notes in the first phrase, but the "DUM-di" business is two shorter beats that must add up to a quarter note. Let's try two eighth notes for them. We're still in octave O3, and we still need a quarter-note rest at the end of the phrase.

Type: PLAY "O3 L4G L8F F L4E P4" <ENTER>

That's close, but it doesn't quite capture the rhythm. The first F, the one on the "DUM," should be held slightly longer than the one on the "di."

Let's steal a little time from the second F and add it to the first. This stealing is done by "dotting" the first F and cutting the time of the second in half. So where you see F F in the PLAY command, substitute F. L16F using the cursor controls. To play the command, just hit <Enter>.

Save this command in program memory too. Move the cursor up to the P in PLAY, strike the Insert key, and

Type: 140 <ENTER>

The line should look like this:

140 PLAY"O3 L4G L8F. L16F L4E P4"

As before, duplicate this line by changing the 140 line number on the screen to 150 and hit <Enter>. A RUN at this point plays the first four phrases (actually measures or bars) of "Three Blind Mice." Try it.

Tougher Stuff

The next phrase is tougher yet. Here's how I gauge it.

```
Notes    G  C  C  B  A  B   C   C   G          G
Words  They all ran af-ter the farm-er's wife, who
Count  di – DUM-di – DUM-de-di – DUM-di – DUM – di
```

If a "DUM-di" corresponds to a dotted eighth and a sixteenth (e.g., L8C.L16C), we've got something of a count mismatch. The first word, "They," must be part of the previous phrase, in the same way that "who" is the last word in the current phrase. We'll have to fix line 150.

Type: EDIT 150 <ENTER>

Move the cursor right under the last 4 in the P4 at the end of the line. Append these new music characters:

Type: 8.L16G" <ENTER>

That'll take care of the "They" by converting the quarter rest to a dotted eighth rest and adding the sixteenth note of value G.

The phrase also has another problem. The words "after the" have three syllables, which must be fit into the time range of a quarter note—a so-called *triplet*. How do you divide a quarter note into three equal parts? You've got to be good at fractions or ratios to solve this kind of problem: 1 is to 4 (the quarter) as 3 (the triplet) is to X (the unknown). I get X = 12. We've got to make the B A B triplet consist of twelfth notes.

This creates a problem. Twelfth notes don't exist in Figure 4.2. But don't worry. PCjr has the answer, L12. Write the phrase this way:

```
PLAY "O4L8C. L16C <L12BAB >L8C. L16C <L8G. L16G"
  <ENTER>
```

How these lines correspond to music notation is shown in Figure 4.3. From all this discussion you can see that playing music on the PCjr is largely a copy operation—transcribing from notes on the piano or sheet music.

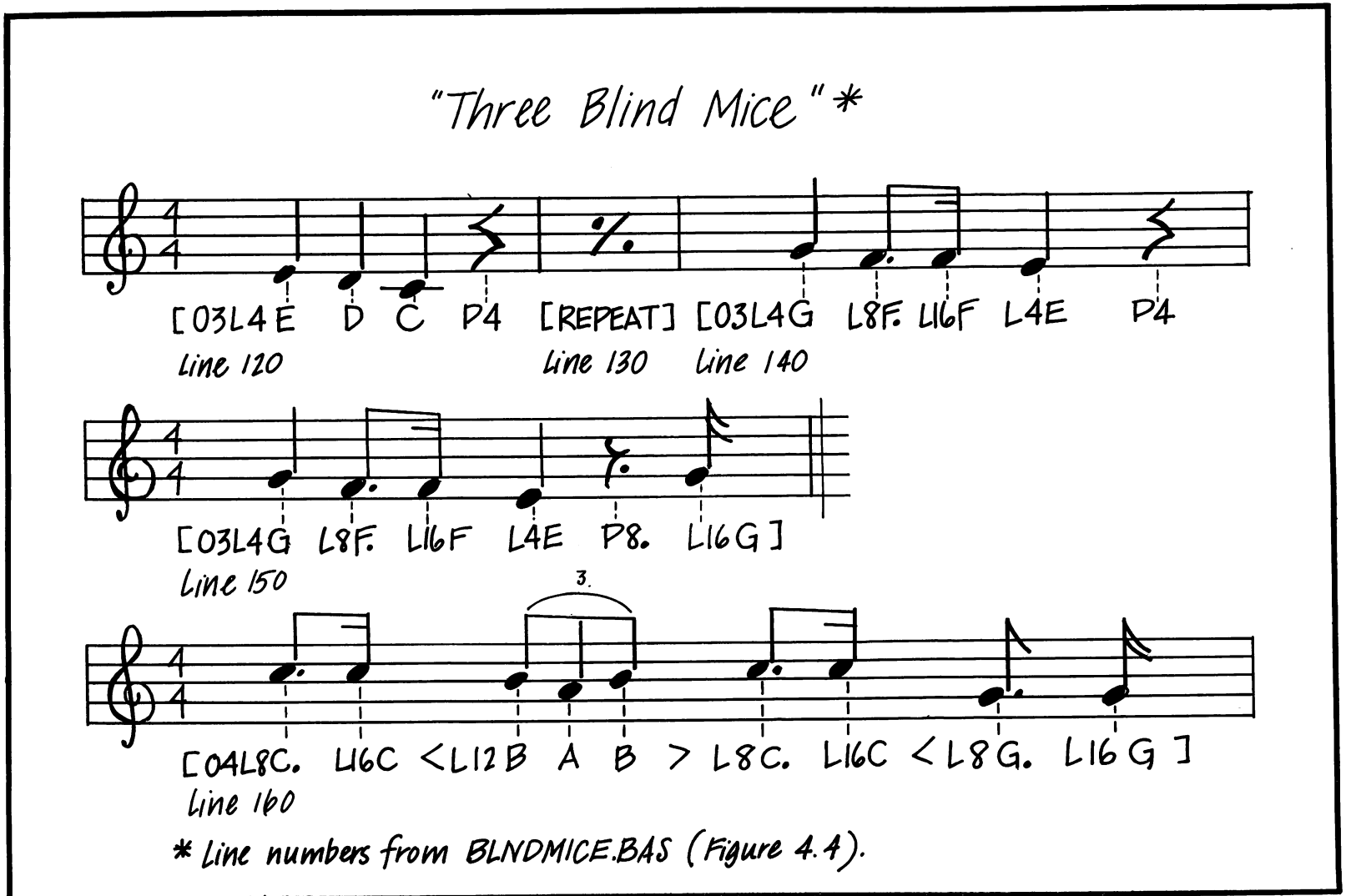


Figure 4.3: Transcribing Music Notation to PC jr Music Language

At this point you know all you need to know to finish "Three Blind Mice" by yourself. But Figure 4.4 gives the whole tune so you can check yourself against my try.

```
100 ' BLNDMICE.BAS
110 SOUND OFF:BEEP ON ' Internal speaker
120 PLAY"O3L4E D C P4"
130 PLAY"O3L4E D C P4"
140 PLAY"O3L4G L8F. L16F L4E P4"
150 PLAY"O3L4G L8F. L16F L4E P8. L16G"
160 PLAY"O4L8C. L16C <L12BAB >L8C. L16C <L8G.
      L16G"
170 PLAY"O4L8C. L16C <L12BAB >L8C. L16C <L8G.
      L16G"
180 PLAY"O4L8C. L16C <L12BAB >L8C. L16C <L8G.
      L16F"
190 PLAY"O3L4E D C P4"
```

Figure 4.4: BLNDMICE.BAS

NOTE: Observe that I make a habit of writing PLAY commands by the measure or bar. You don't have to, of course, but it seems easier to correct errors and to modify arrangements when working in units of one measure.

NOTE: I also consider a measure to be independent of its immediate predecessor. So I tend to begin by specifying the octave anew as well as the starting note length. If I'm concerned about dynamics, I'll include a V command too. Tempo usually remains constant for a section of the piece.

Jump Right into It!

The place to start, even if you're fairly competent in music, is with simple tunes. Then, when you're used to the way PCjr wants you to transcribe a tune, you can branch out.

Right now why don't you try to transcribe for PCjr some simple tunes like "Frère Jacques"; "Twinkle, Twinkle, Little Star"; "Row, Row,

Row Your Boat"; "How Dry I Am"; "Yankee Doodle"; "London Bridge Is Falling Down"; or "Mary Had a Little Lamb." You should begin with unsyncopated tunes that stay within a one-octave range.

Don't be concerned if you don't get them correct the first time around, even if you know how to read music. I have to hear each measure and juggle the music character strings until I get them right. I expect you'll have to do the same.

To get you launched, I've done "Twinkle, Twinkle" in Figure 4.5. You'll note I probably got "Twinkle, Twinkle", "Baa, Baa, Black Sheep" and the "A-B-Cs," which all go to the melody, a little mixed up in my rendition.

```
100 ' TWINKLE.BAS
105 SOUND ON:BEEP OFF
110 PLAY "O3L4C C G G A A L2G"
120 PLAY "O3L4F F E E L8D D D D L2C"
130 PLAY "O3L4G G F F E E L2D"
140 PLAY "O3L8G G L4G F F E E L2D"
150 PLAY "O3L4C C G G A A L2G"
160 PLAY "O3L4F F E E D D L2C"
```

Figure 4.5: TWINKLE.BAS—"Twinkle, Twinkle, Little Star"

3. WHEN YOU DON'T HAVE THE MUSIC

Well, how much hacking did you end up doing? Transcribing gets kind of confusing, doesn't it? All those letters and numbers. You may find that using lower-case letters instead of capitals, so that letters and numbers don't run together, will make for easier reading of the music strings.

However, I think it's the number of things you've got to consider at about the same time that is the really confusing factor. To deal with this problem, I've found it best to develop a tune at the keyboard with a layered approach using well-defined steps.

First, get the notes of the tune correct, regardless of timing. Next, iron out the timing. Third, handle the nuances—legato, staccato. Finally, monkey with the dynamics (volume and tempo). (I don't think this layered approach is necessary when you're transcribing directly from written music. All the information needed is right in front of you on the music paper.)

First Layer—the Notes

Let's try an example. The transcription of "Bicycle Built for Two" (Daisy) in Figure 4.6 is what I ended up with at the end of the first step—the sequence of notes of the tune, divided into what I thought were measures. Type DAISY1.BAS into memory so we can go to the next step.

```
100 ' DAISY1.BAS
105 SOUND ON:BEEP OFF
110 PLAY "T80 O3G E C <G"
120 PLAY "O2A B >C <A >C <G"
130 PLAY "O3D G E C"
140 PLAY "O2A B >C D E D E"
150 PLAY "O3F E D G E D C D"
160 PLAY "O3E C <A >C <A G G"
170 PLAY "O3C E D <G >C E"
180 PLAY "O3D E F G E C D <G >C"
```

Figure 4.6: DAISY1.BAS—Sequence of Notes Only

Second Layer—the Timing

DAISY1.BAS sure is Dullsville—no character at all. It even drags along because of the T80 I put in the first phrase (line 110) to more easily identify errors as it's playing. So let's introduce the timing.

Although the default is the quarter note, put an L4 between the 3 and the G of line 110 for the correct timing of the first phrase.

Okay, what do you do with line 120? Well, the initial AB>C, corresponding to the words "Give me your," is a triplet, right? You can just feel three notes on one beat. We had this case in BLNDMICE.BAS, you'll remember, and handled it with L12. So place L12 in line 120, just before the AB>C sequence.

The next word, "an-swer," corresponds to the A>C sequence of notes. This is a "DUM-di" rhythm corresponding to a dotted eighth and sixteenth note. So you replace A>C by L8A.>L16C, right?

Hold it! That's not a "DUM-di" rhythm. It's sort of a "DUM-a- di." There's a triplet there, but only two notes. The "an-" of "answer" goes with the "DUM-a" count. The "-swer" goes with the "di." So how do we handle this case? Just write L12AAC.

There's another case like this one in line 140. So beware!

The last word of line 120, "do!", corresponds to the <G. This is at least a quarter note. But, on humming the tune, you can feel it for two beats. So make it a half note and put an L2 in front of the G in line 120.

Well, you've got the idea.

Fill in the rest of the timing commands for DAISY1.BAS. I ended up with the expanded transcription shown in Figure 4.7. Check me to see if I made an error.

```
100 ' DAISY2.BAS
105 SOUND ON:BEEP OFF
110 PLAY "T80 O3L4G E C <G"
120 PLAY "O2L12A B >C <L12AA>C <L4G P4"
130 PLAY "O3L4D G E C"
140 PLAY "O2L12A B >C L12DDE L4D P8. L16E"
150 PLAY "O3L12FED L8G. L16E L8D C P8. L16D"
160 PLAY "O3L8E. L16C <L8A. >L16C <L8A G P8.
      L16G"
170 PLAY "O3L8C. L16E L8D. <L16G >L8C. L16E"
180 PLAY "O3L12DEF L12G E C L8D. <L16G >L2C"
```

Figure 4.7: DAISY2.BAS—Notes and Timing

Legato and Normal Modes

You've RUN DAISY2.BAS haven't you? Not bad except for the double A's and D's in lines 120 and 140, respectively, right?

Maybe you haven't thought about it, but when you PLAY two or more notes of the same pitch in sequence, you hear distinct notes. Try this:

Type: PLAY "L4AAAA" <ENTER>

Although these are quarter notes, they sound for only 7/8 the time indicated by the L command. This is the "normal" mode of sounding notes.

Now try this:

Type: PLAY "MLL4AAAA" <ENTER>

Just use the editor to stick "ML" in front of the L4.

You should get a continuous sounding of A this time. It played the notes legato (smoothly), passing from one note to the next without pause. Of course, in normal music you'd write the four connected A's as a whole note, not as a slurred sequence. It's when you have a sequence of different notes that you want to play smoothly that you'd write a slur. Hear how the following cases contrast:

Type: PLAY "MNL4CDEFG" <ENTER>
Type: PLAY "MLL4CDEFG" <ENTER>

The MN set normal mode—7/8 the note timing.

Now to apply all this to DAISY2.BAS. You've observed, I'm sure, that the way we have the tune at this point, the notes seem overly clipped in places. The triplet in line 120 is a case in point. Suppose we play this triplet "smoothly"—in legato mode—so that the three notes flow. But we want to avoid extending the flow to subsequent notes. We can accomplish this goal by inserting ML before the first A, and MN before the first C in line 120. Line 120 would then read

120 PLAY "O2L12MLAB>MNC<L12AA>C<L4GP4"

Another use for legato mode occurs in line 120, to tie the double A together as a single note. Insert ML before the first of these two A's. We have a choice of placing the MN, to change back to normal mode, in front of the second A or in front of the following C. In the second case we'd be treating the triplet as legato, not just tying the A's together—though that happens as a consequence.

Try your hand at phrasing DAISY2.BAS to make it play more smoothly.

Figure 4.8 shows how I went about phrasing the tune. Notice I've speeded it up a bit by setting the tempo to 100 in line 110.

I make no claim that my version is the best one. I'm hardly a music editor, and you may well find improvements. Notice that I tied dotted eighth notes to following sixteenths using the legato mode. There are lots of examples in the last few lines of the tune. I think the ties add a little more body. If you don't like this idea, feel free to change it.

```
100 ' DAISY3.BAS
105 SOUND ON:BEEP OFF
110 PLAY "T100O3L4GEC<G"
120 PLAY "O2MLL12AB>MNC<L12MLAMNA>
      C<MLL4GP4"
130 PLAY "O3MNL4DGEC"
140 PLAY "O2MLL12AB>MNCL12MLDMNDEMLL
      4DP8.MNL16E"
150 PLAY "O3MLL12FEMNDL8MLG.MNL16EL
      8DMLCP8.MNL16D"
160 PLAY "O3MLL8E.L16MNC<MLL8A.>MNL16C<
      L8AMLGP8.MNL16G"
170 PLAY "O3MLL8C.MNL16EL8MLD.<L16MNG>
      L8MLC.L16MNE"
180 PLAY "O3MLL12DEMNFMLL12GEMNCMLL8D.
      <L16MNG>L2C"
```

Figure 4.8: DAISY3.BAS—With Nuances

Last Layer—Dynamics

The last step is to add interest by varying the dynamics of the tune, principally variations in volume—forte, piano and all that.

I don't have any general rules. Mostly it's a matter of taste. But one approach for a tune like "Daisy" is to carry the vocal lilt over into the computer presentation. Play the downbeat a little louder than the other notes—i.e., accent it.

Suppose you play these accents at V8. Then you may want to play unaccented notes of higher pitch than the last accented one at say V6, and those below it at V4. The reason for the difference is that higher pitches are generally quieter than lower pitches for the same energy

input. So you'd want to play the lower notes at a somewhat softer volume to even things out. But there's lots of leeway for you to choose—almost too much.

Try your hand at building dynamics into DAISY3.BAS.

Contrast your solution with the one I did in Figure 4.9. Note the addition of line 105 to be sure the external speaker is used while we're controlling volume.

You can see from the complexity of Figure 4.9 why I suggested you develop your transcriptions in layers.

```
100 ' DAISY4.BAS
105 SOUND ON:BEEP OFF
110 PLAY "T100O3L4V8GV4EV8C<V4G"
120 PLAY "O2MLL12V8AV6B>MNC<L12MLV8
      AMNA>V4C<MLL4V8GP4"
130 PLAY "O3MNL4V8DV6GV8EV6C"
140 PLAY "O2MLL12V8AV6B>MNCL12MLV8DMNDV6
      EMLL4V8DP8.MNL16V6E"
150 PLAY "O3MLL12V8FV6EMNDL8MLV8G.MNL16V6EL8V8
      DMLV6CP8.MNL16D"
160 PLAY "O3MLL8V8E.L16MNV6C<MLL8A.>MNL16C
      <L8V8AMLV6GP8.MNL16G"
170 PLAY "O3MLL8V8C.MNL16V6EL8MLV8D.
      <L16MNV6G>L8MLV8C.L16MNV6E"
180 PLAY "O3MLL12V8DV4EMNFMLL12V8GV6
      EMNCMLL8V8D.<L16MNV6G>L2V8C"
```

Figure 4.9: DAISY4.BAS—With Dynamics Added

Staccato Mode

There's another note-quality mode, called staccato and designated MS, by which each note sounds at 3/4 the time indicated by the L command. Remember, normal mode sounds at 7/8 the L value. When I hear the word "staccato," David Rose's "Holiday for String" starts playing in my brain, with all the pizzicato (plucked) notes of the first part.

Try these experiments to hear how this mode differs from the normal mode.

Type: SOUND ON:BEEP OFF <ENTER>
Type: PLAY "O3MSL1AAAA","O3MNL1BBBB" <ENTER>

Now I know what's played when they do those civil-defense tests on the radio!

Did you hear the A cut off before the B? Now move the cursor up to the PLAY command and change the L1 to L2—in both places mind you—and hit <Enter>. You can still hear the cutoff, right?

Successively raise the time value of the A and the B in the L commands by one unit until you can't hear the cutoff any more. This should give you some feel for staccato. Applying staccato to rapidly moving passages can be a waste of effort—you may not hear the difference.

I cast about for a simple tune to illustrate staccato contrasted with legato and came up with one of my favorites, Bach's Gavotte in G. It's illustrated in Figure 4.10.

```
100 ' GAVOTTE.BAS—Gavotte in G by Bach
110 SOUND ON:BEEP OFF
120 PLAY "T160O4MSL4V8BV9G"
130 PLAY "O4MLL4V10DL8V8EV9F#MSL4GV10E"
140 PLAY "O3MLL2V9B>MSL4V8EV9C"
150 PLAY "O3MLL4V10AL8V8B>CD<V9B>C<V10A
160 PLAY "O4L8V9C<BAV8G>MSL4BV9G"
170 PLAY "O4L4MNV10EL8V8MLEF#MSL4GV9E"
180 PLAY "O4MLL4C#L8V7DEV8F#DV9GV10E"
190 PLAY "O4L8F#V11DL4AL8V10GF#V9EV8F#"
200 PLAY "O4L2V7MND"
210 GOTO 120
```

Figure 4.10: GAVOTTE.BAS—Gavotte in G by Bach

4. TUNE YOUR GUITAR

There are other fun things you can do with PCjr's musical capabilities, such as writing a program to tune your guitar—or, with some modifications, any instrument.

The Program

The program is listed in Figure 4.11. If you don't have a piano or a pitch pipe, you may find it to be of great value. In any case, it gives us a chance to further explore PCjr's music language.

Type in and SAVE TUNEGTAR.BAS and we'll try it.

```
100 ' TUNEGTAR.BAS
110 GOTO 330 ' Set up screen instructions
120 SEC=2 ' Delay between note soundings
130 SOUND ON:BEEP OFF ' External speaker
140 PLAY"L2" ' Quarter note intervals
150 A$=INKEY$:IF A$="" THEN 150 ' Get keyboard input
160 NOTE=INSTR("EADGBeadgb654321Qq",A$) '
      Decode key selected
170 IF NOTE=0 THEN BEEP:GOTO 150 ' BEEP on
      undefined key
175 ' Branch to play note selected
180 ON NOTE GOTO 190,220,230,240,250,190,220,230,240,
      250,260,220,230,240,250,270,280,280
190 PLAY"O1E" ' Special case
200 FOR I=1 TO 875*SEC:NEXT
210 GOTO 270
220 PLAY"O1A":GOTO 300
230 PLAY"O2D":GOTO 300
240 PLAY"O2G":GOTO 300
250 PLAY"O2B":GOTO 300
260 PLAY"O1E":GOTO 300
270 PLAY"O3E":GOTO 300
280 PLAY"O1EA>DGB>E" ' Play all strings at end
290 PRINT:PRINT"FINISHED TUNING":END
300 FOR I=1 TO 875*SEC:NEXT ' Between soundings
      delay
```

```
310 A$=INKEY$:IF A$=CHR$(27) THEN 150 '
      Cancel note
320 GOTO 180 ' Repeat note selected
330 ' Set up screen instructions
340 KEY OFF:CLS
350 PRINT:PRINT:PRINT" THE PCjr, HOT-ZOOTY
      GUITAR TUNER"
360 PRINT:PRINT TAB(10);"STRING   NOTE/KEY"
370 PRINT:PRINT TAB(10);" 1   E"
380 PRINT TAB(10);" 2   B"
390 PRINT TAB(10);" 3   G"
400 PRINT TAB(10);" 4   D"
410 PRINT TAB(10);" 5   A"
420 PRINT TAB(10);" 6   E (bass)"
430 PRINT:PRINT"Press numbered or lettered keys to"
440 PRINT"sound note"
450 PRINT:PRINT"Press <Escape> to cancel note"
460 PRINT:PRINT"Press Q to quit program"
470 GOTO 120 ' Start up mainline of program
```

Figure 4.11: TUNEGTAR.BAS

To tune your guitar with TUNEGTAR.BAS, select the string you want to adjust, and press the corresponding key on PCjr's keyboard. Listen to the note and start adjusting the string until the string sounds the note PCjr is playing. The note will sound over and over until you strike the Escape key.

That's all there is to it. When you want to quit, type Q.

One peculiarity of the program is that if you select E with the lettered key, you get the notes for both the first and the sixth strings in succession. The reason: there's only one letter E on the keyboard, and I didn't want to favor either the first or the sixth string.

Programming Notes

Notice that the program jumped down to line 330 right off to set up the operating instructions on the screen. These are straightforward PRINT instructions. Then you're sent back to the mainline (i.e., principal part) of the program at line 120.

At line 150, PCjr looks for input from the keyboard—one of the letters or numbers defined on the screen. The one you pick is decoded in line 160 by the INSTR(ing) command. The letter and number possibilities are shown between quotes. Notice that both upper and lower-case letters are permitted. The INSTR command puts the position of the match with the ASCII character in A$ into the pigeonhole called NOTE. If there's no match, a zero is placed in NOTE.

In 170, NOTE is tested for zero. If it is, you're BEEPed at and sent back to 150 to key in something else—a legitimate key this time.

If NOTE has a legitimate value, it is paired with a line number in 180 where the desired note is PLAYed. For instance, if you typed A, then NOTE = 2, and you'd be sent to line 220. Line 220 then PLAYs the A in octave O1 and jumps to line 300. The situation is similar for the other notes of the open-string guitar.

At line 300 a two second delay occurs, giving you time to hit the Escape key to cancel the note. <Esc> is detected in line 310. If you don't press it, you're sent back to 180 to repeat the note. Otherwise you're sent to 150 to make another selection.

When you press Q to quit, a 17 or 18 is placed in NOTE, depending on whether you hit a capital or lower-case Q. Then line 180 sends you to line 280, where all six notes are sounded just before the program ENDs.

NOTE: It's important for you to study this listing if you're not very familiar with programming. Though it may seem a little long and complex, it is really very simple. It just repeats a simple pattern over and over for each of six possible notes. A great many interesting things can be done by repeating patterns with slight variations. And you'll realize that you can write such programs yourself, once you recognize that *long* doesn't necessarily mean *complex*.

5. LITTLE PIANO

Let's turn PCjr into a musical instrument that can be played like a recorder, ocarina or other instrument. The exercise will provide an additional dimension to programming sounds and music, and you'll end up with something to have fun with. You might even use it to help in making transcriptions to PCjr music language when you don't have a piano or written music available.

The Keyboard

First we'll need a keyboard, so let's adapt PCjr's for the purpose. We'll make the second to top row correspond to the white keys on the piano, with the letter T as middle C. The top row will correspond to the black keys.

Figure 4.12 shows the note values inside the key blocks, and the letter values above the blocks. You'll notice that not all keys on the top row have musical functions for the assignment.

Retrieving a Note

The first thing we have to figure out is how to translate the striking of a key on the top two rows into a sounded note. Recall that we can read the keyboard with the INKEY$ command. So here's a start:

```
100 ' LITTLE PIANO
160 A$=INKEY$:IF A$="" THEN 160 ' Get note
```

This instruction step tells PCjr to keep looking at the keyboard until something is put in the type-ahead buffer. The funny string of two double quotes represents "nothing." That is, the IF asks if there is anything in the type-ahead buffer. If there's "nothing," it returns to line 160 and asks again.

Eventually something will be put in the type-ahead buffer and the problem of determining which key was struck will present itself. The special BASIC instruction called INSTR(ing) can help us solve the problem.

```
260 ' Decode typed note
270 NOTE=INSTR("Q2W3E4RT6Y7UI9O0P-[]",A$)
```

Compare the list of characters between the quotation marks in line 270 with the key assignments in Figure 4.12. You'll notice they're all the keys to which notes have been assigned in ascending musical sequence—except for the E of the Tab key.

INSTR(ing) compares this sequence of characters with the character in A$. If it finds a match it sets NOTE equal to the position value of the character (Q = 1, 2 = 2, W = 3, etc.). If there is no match, as there would be if you typed A or > or some other unassigned character, INSTR(ing) sets NOTE to zero. In this way you can figure out what note to play. Well, almost.

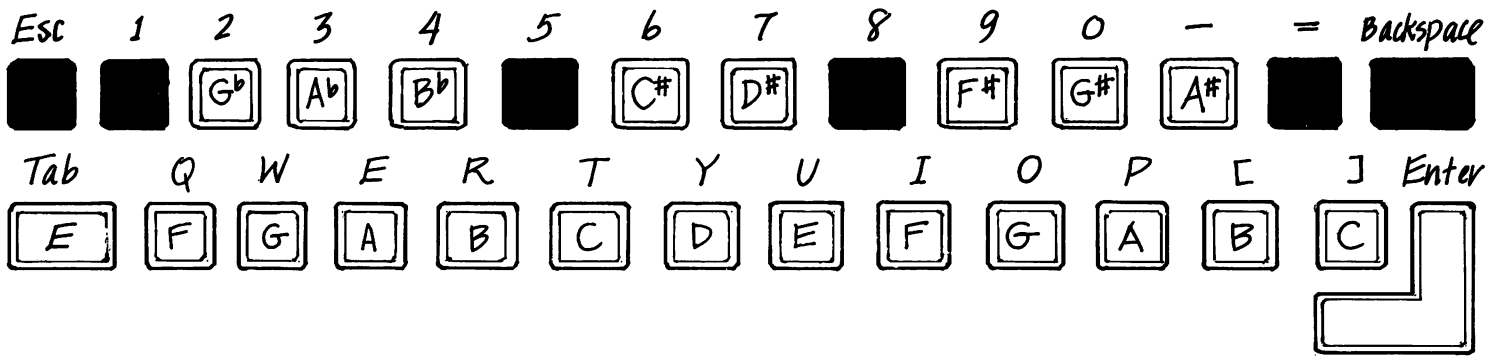


Figure 4.12: Assignment of Notes to PCjr's Keyboard

Include <Tab> in the sequence of defined notes. <Tab> must be generated by CHR$ since it is not a printed character. It's equal to CHR$(9). Add it at the beginning of the character string this way:

```
270 NOTE=INSTR(CHR$(9)+"Q2W3E4RT6Y7UI9O0P-[]",A$)
```

If NOTE turns out to be zero (i.e., an undefined key was struck), we want to get another note. This is done as follows:

```
280 ' NOTE = 0 if undefined key struck
300 IF NOTE=0 THEN 160 ' Get new note
```

Playing NOTE

Since the note E of the Tab key corresponds to the 14th playable note in PCjr's repertoire while <Tab> is the first character in the sequence (making NOTE = 1), we have to add an offset of 13:

```
310 NOTE=NOTE+13 ' Add offset
```

Now finally, we can play the note and return for another.

```
320 PLAY "N=NOTE;" ' Play current note
330 GOTO 160 ' Get next note
```

Taking Care of Lower-case Letters Too

Notice in line 270 that if you type a lower-case letter, q for instance, NOTE will be set to zero. So if you want the program to work, you must turn <CapsLock> on. But that's not the way to write a program. All we have to do is repeat the essence of line 270, but with lower-case characters instead. Like this:

```
290 IF NOTE=0THEN NOTE=INSTR(CHR$(9)+
    "q2w3e4rt6y7ui9o0p-[]",A$)
```

Setting Initial Defaults

It's a good idea to explicitly set the initial conditions you want in a program. You can't always be sure of the state of the computer when you make entries. Here are some we might use for our Little Piano:

```
110 V=8:T=120 ' Default volume & tempo
120 SOUND ON:BEEP OFF 'Select external speaker
130 PLAY "MFL8T=t;V=v;" ' Initialize
```

Watch out for the semicolons.

All we're doing here is selecting the external speaker, setting the initial tempo and volume, and specifying that the notes will be considered eighth notes played in the foreground.

Changing Volume and Tempo as You Go Along

Not all tunes can be played acceptably with these tempo and volume defaults. If you try the program the way it is now, notes are sustained too long for fast tunes. Also, different sections of a tune should be played at different volumes.

We can define additional keys to signal PCjr to change the defaults. For instance, if we type a capital V, we can have PCjr raise the volume one unit (remember the range is 0 to 15). Then by typing lower-case v we can have the volume reduced one unit.

Here are the program steps:

```
210 ' Capital V increases volume
220 IF A$="V" THEN V=V+1:IF V>15 THEN V=15
230 ' Lower-case V decreases volume
240 IF A$="v" THEN V=V-1:IF V<0 THEN V=0
```

Remember, A$ in line 220 has a character from the type-ahead buffer. If it's a V then V is increased by 1. But if V exceeds 15, the upper limit of volume allowed, then V is set to this maximum. The same situation, but reversed, occurs in line 240 if a lower-case v is typed in.

We can handle changes in tempo in a similar way, but we can't use T for "tempo," because T is already assigned. How about S for "speed"? Now you can change tempo with these lines of coding:

```
170 ' Capital S increases tempo
180 IF A$="S" THEN T=T+20:IF T>255 THEN T=255
190 ' Lower-case S decreases tempo
200 IF A$="s" THEN T=T-20:IF T<32 THEN T=32
```

When the defaults have been reset, they have to be executed in a PLAY command to make them effective.

```
250 PLAY "T=t;V=v;" ' Set current tempo & volume
```

Some Visual Reminders

It's always a good idea to give users some hints about how to use your program. For instance, the screen could be cleared and they could be instructed where to find a note, say middle C. Like this:

```
140 CLS:LOCATE 10,9:PRINT"LETTER 'T' = MIDDLE 'C'"
150 LOCATE 13,11:PRINT"YOU GO FROM THERE!"
```

You might also want to tell the user that the keys on the top row of the computer are the black keys of the piano. And how about saying that you can change the volume and tempo with V and S. Try your hand at adding these extra instructions to what's written on the screen.

Don't forget to SAVE this little program. I saved it under the name LILPIANO.BAS.

Try Out the Piano

Here's a tune to try on the Little Piano (Figure 4.13). Just press the keys indicated by the letters and other characters indicated. The dot (.) at the beginning is a rest, so don't strike any key there. Play the notes at equal time value in groups of three (triplets) to get the rhythmic feel. Do you recognize the tune? It's a famous chorale by Bach.

Don't forget that you can speed up tempo with capital S and increase volume with capital V. Use lower-case s and lower-case v to reverse these actions.

```
. TYUOI IPOO][
]OUTYUIOPOIU
YUTRTYWRYIUY
UTYUOI IPOO][
]PITYUEOIUYT
WTRTUO ]
```

Figure 4.13: "Jesu, Joy of Man's Desiring" (Bach), Arranged for LILPIANO.BAS

6. RUN THROUGH IT ONCE MORE

You should have a pretty good handle on playing music on PCjr by now, but there's still more to the subject. We'll take it up again after we've had a chance to learn some graphics and to use what we know to enhance our picture-making.

Before you go on, you ought to go back through the chapter and review the material. Also, bring your notes up to date. If the material begins to get overwhelming, you'll be glad you have notes to bail you out.

As you review the chapter, also be sure you understand the BASIC commands used to develop the examples. Figure 4.14 lists those used in the chapter. Look over the list, and reinforce your growing habit of looking up problem commands in the BASIC manual. I know it's tough, but you've got to become familiar with that manual for the long haul, and now's the time to do it.

BASIC COMMAND	FUNCTION
BEEP ON/OFF	Speaker control with SOUND ON/OFF
CLS	Clear screen
FOR...NEXT	Repeat a sequence of commands
GOTO	Take next command from line specified
IF...THEN	Make decision on program flow
INKEY$	Read character from type-ahead buffer
INSTR	Position of first occurrence of one string in another
KEY OFF	Toggle screen prompts off
LIST	List program memory to the screen
LOCATE	Position cursor on screen for printing text
ON...GOTO	Computed GOTO multiple line numbers

PLAY	Sound the notes and execute the commands in a music command string
PRINT	Print text on the screen
SOUND	Play a pitch of specific duration
SOUND ON/OFF	Speaker control with BEEP ON/OFF

Figure 4.14: Summary of Commands in Chapter 4

CHAPTER 5
ADVENTURES WITH EUCLID

Objective:

This chapter explores graphics generation, using dots, lines, space and color.

1. THE VIDEO SCREEN—OUR NEW DRAWING PAPER

You've solved puzzles with match sticks, haven't you? And drawn stick figures and boxes and all sorts of shapes using straight lines, and then shaded them in with the side of your pencil or with a color crayon?

You're about to do the same with PCjr, the only difference being that your pencils and crayons and colored construction paper will be the video screen.

A Review of Text Mode

You know all this by now, but let's do the experiment anyway. Try the little program in Figure 5.1. It simply fills the text screen with digits.

```
100 ' Fill the screen in text mode
110 KEY OFF:CLS ' Fresh screen
120 SCREEN 0,1 ' Set color text mode
130 COLOR 15,1 ' White char on blue
140 FOR COL=0 TO 39 ' Count off cols
150 FOR ROW=0 TO 23 ' Count off rows
160 I=COL MOD 10 ' "I" goes 0 to 9 over and over
170 IF ROW=23 AND COL=39 THEN 210 ' Don't print
    last digit
180 LOCATE ROW+1,COL+1:PRINT CHR$(I+48); '
    Print digit
190 NEXT ROW ' Do next digit
200 NEXT COL ' Do next col
210 GOTO 210 ' Don't mess screen with Ok
```

Figure 5.1: Fill Text Mode Screen with Digits

How this program works should be pretty clear to you. A couple of technical points however: Line 160 converts the value of COL, which may range from 0 to 39, to a digit from 0 to 9. The MOD 10 tells PCjr to divide COL by 10, and to save only the remainder. So if COL = 8 or 18 or 28 or 38, for instance, you get I = 8.

An explanation for line 170 is also in order. Every time you print a character in the last column of a line, the cursor jumps to the first column of the next line. What happens when it's the last column of the last line? You've seen it a hundred times by now. The whole screen scrolls upward. Line 170 prevents printing in this last space and thereby avoids scrolling.

A last point. The program prints down a column instead of across a row because we force it to do so with the LOCATE command in line 180. As you've seen, LOCATE permits you to put a character any place on the screen.

The weird character formula printed in line 180, namely CHR$(I+48), is the equivalent of the digit I, but converted from numeric form to an ASCII character. (Remember Appendix G of the BASIC manual?) I did this conversion to avoid printing the spaces that PCjr generates when you print numeric quantities.

You might try substituting PRINT I in line 180 to see what happens. It generates quite different results.

Medium-Resolution Graphics

Our real interest is not the text mode screen. We're going to do a lot of stuff on the medium-resolution graphics screen, so we need to understand how it functions in relation to the text mode screen.

Compare the program of Figure 5.1 with that of Figure 5.2, line by line. Then make the appropriate editing changes in the first to produce the second. RUN it to see what happens.

```
100 ' Fill medium-resolution graphics screen
110 KEY OFF:CLS ' Fresh screen
120 SCREEN 1,0 ' Set med res graphics
130 COLOR 1,1 ' Palette 1 on blue
140 FOR COL=0 TO 299 ' Count off cols
150 FOR ROW=0 TO 199 ' Count off rows
160 ' No equivalent required
170 ' No equivalent required
180 PSET (COL,ROW),3 ' Place dot
190 NEXT ROW ' Do next dot
200 NEXT COL ' Do next col
210 GOTO 210 ' Don't mess screen with Ok
```

Figure 5.2: Fill Medium-Resolution Screen with Dots

Since this program takes a while to run, you might cheat a little. After you've seen what the program is doing, verify that it ends correctly by changing line 140 to read

```
140 FOR COL=295 TO 299 ' Count off cols
```

I use that trick all the time in situations like this. It saves waiting for the whole process to be completed. But don't forget to change the modified line back to its correct form.

In the program of Figure 5.1, we don't have the same technicalities to deal with as before. The key differences are these: The COLOR command in 130 is interpreted differently now. It is the first digit that follows COLOR, the 1 in this case, that defines the background (field) color, not the second digit as with color text mode.

The second digit of COLOR represents one of two palettes (i.e., sets) of four colors you can pick for the foreground—the color of the crayon you'll draw with. Figure 5.3 gives the table of default colors you can use with each palette. As we'll see later, you can change these defaults.

COLOR #	PALETTE 0	PALETTE 1
0	None	None
1	Green	Cyan
2	Red	Magenta
3	Brown	White

Figure 5.3: Default Palette Colors

The number of colored objects you can place on the screen is considerably larger now—three hundred across a row, two hundred down a column. You can see this in lines 140 and 150.

Finally, instead of using LOCATE to determine where to place something, in graphics mode you use PSET.

NOTE: Actually you can use LOCATE to put ASCII characters on the graphics screen, just as if it were a text mode screen.

Note that in the PSET command, the COL comes before the ROW specification. They're in the opposite order from the LOCATE command. The number 3 specifies the color of the dot to be placed—white in this case since we're using palette 1. (If you just want to move the

graphics equivalent of the cursor to a given position on the screen to specify where some following graphics command is to start, use color 0.)

2. DRAWING WITH LINES OF LIGHT

Clear memory (NEW), and set up the medium-resolution graphics screen for some experiments.

Type: KEY OFF:SCREEN 1,0:COLOR 1,1 <ENTER>

Screen Coordinates

The position of a point on the screen, specified by the row and column in which it lies, is called the point's coordinates and is designated by the form: (column, row). You've probably guessed from Figure 5.2 that position (0,0) on the screen is in the upper left corner, and that position (299,199) is in the lower right. Just to get a feel for this arrangement, try these commands and watch where the dot is placed on the screen.

Type: PSET (50,100),3 <ENTER>
Type: PSET (200,10),3 <ENTER>
Type: PSET (20,180),3 <ENTER>
Type: PSET (150,100),3 <ENTER>

Try a few more cases of your own invention. Clear the screen if you get confused about where the dot is being drawn.

Erase one of the points you have put on the screen by changing the color attribute to 0. That is, try something like this.

Type: PSET (20,180),3 <ENTER>
Type: PSET (20,180),0 <ENTER>

Do you know what happens to the dots you place on the screen if you scroll off the bottom? What are their new coordinates?

Now that you know how to put dots on the medium-resolution screen, here's a problem for you to solve:

Translate CREATION.BAS from text mode to medium-resolution mode. (You merely substitute a few commands.)

The Shortest Distance Between Two Points

Although you can use PSET to draw a line or dots between two coordinates on the screen, there's an easier way to do it.

Type: <Ctrl-L>
Type: LINE (20,50)-(200,120),3 <ENTER>

You can erase the line by changing the attribute color, the 3 at the end of the LINE command, to 0. Try it. Just move the cursor up to the typed LINE command, change 3 to 0, and hit <Enter>.

A Pile of Hay

Let's fill the screen with random lines using the little program of Figure 5.4.

```
100 ' RANDOM LINES
110 KEY OFF:SCREEN 1,0:COLOR 1,1
120 CLS ' Clear screen
130 FOR I=1 TO 200 ' Count off lines
140 X1=300*RND:Y1=200*RND ' Beginning coordinate
    (X1,Y1)
150 X2=300*RND:Y2=200*RND ' Ending coordinate
    (X2,Y2)
160 LINE (X1,Y1)-(X2,Y2),3 ' Generate line
170 NEXT ' Go do another line
```

Figure 5.4: HAY.BAS—The Pile of Hay

The result looks something like a pile of hay—pretty disorganized.

A Random Walk

Suppose we make a little change or two. We'll use PSET to set a starting point.

Type: 125 PSET (150,100),3

Then we'll use another property of LINE. If you just specify the ending coordinate in the command, the line will start at the last coordinate specified. So

Type: 160 LINE -(X2,Y2),3 ' Generate line to (X2,Y2)

RUN the program with these changes to see what happens. You still get a pile of hay, but it's one long straw this time—sometimes called a "random walk." Don't worry about line 140. It's redundant. You can eliminate it if you like. You'll get a different-looking result now because you'll generate a different sequence of random numbers.

3. GEOMETRY CAN BE FUN

There's not much you can do with a pile of hay. A little more organization is needed. Try some simple types of organization—drawing common geometric shapes, overlapping them on one another, and coloring them.

The Triangle

Figure 5.5 is a program called TRIANGLE.FUN. It draws an equilateral triangle centered on the medium-resolution screen. Type it in, RUN it, and SAVE it. We'll use it as a springboard.

```
100 ' TRIANGLE.FUN
110 KEY OFF:SCREEN 1,0:COLOR 1,1
120 CLS ' Clear screen
130 PI=3.141593:FR=PI/180
140 T=0:R=30:X0=150:Y0=100
200 ' Calculate vertices
210 X1=R*COS(FR*T):Y1=R*SIN(FR*T)
220 X2=R*COS(FR*(T+120)):Y2=R*SIN(FR*(T+120))
230 X3=R*COS(FR*(T+240)):Y3=R*SIN(FR*(T+240))
240 ' Draw the edges
```

```
250 PSET(X1+X0,Y1+Y0),0
260 LINE-(X2+X0,Y2+Y0),3
270 LINE-(X3+X0,Y3+Y0),3
280 LINE-(X1+X0,Y1+Y0),3
```

Figure 5.5: TRIANGLE.FUN

Don't be too concerned if you don't understand the formulas in lines 210 to 230. Basically they're equations for circles of radius R centered at origin. But you don't need to know this to enjoy the effects produced. In a later chapter we'll get away from lines and circles and purely geometric figures, and we'll actually draw on the screen. Then you won't have to worry about math. But, frankly, to invent your own geometric designs as we're doing here, you've got to know some trigonometry and analytic geometry.

NOTE: As we go along, consider saving the program at its current development stage for future reference. You may want to demonstrate some of the patterns. Select punchy names for the files so you can remember what they do.

Rotating the Triangle

The variable T controls the orientation of the triangle. Watch what happens when we put T in a FOR...NEXT loop with these lines:

```
190 FOR T=0 TO 360 STEP 20
300 NEXT T
```

Actually, because of the symmetry, you can get the same effect faster by taking T from 0 to 100 instead of to 360, but for present purposes leave it at 360.

What happens when you change the step size to 60? Try it and you'll get a familiar figure.

Controlling the Triangle's Size

The radius R controls the size of the triangle. Insert these lines and see what happens.

```
180 FOR R=30 TO 90 STEP 30
310 NEXT R
```

Already you see that a variety of interesting patterns will result just from rotating the figure and changing its size.

Size and Orientation Together

But suppose we do both simultaneously. Make these changes:

```
180 FOR R=10 TO 100 STEP 10
190 T=T+5
```
Type: DELETE 300 <ENTER>

That makes matters more interesting. We can do the figure from out to in too. Try this change:

```
180 FOR R=100 TO 10 STEP -10
```

By drawing from the outside in, we can readily PAINT the figures to get a fairly complex color pattern. Add these lines and try them:

```
170 CLR=2 ' Initial palette attribute (color)
300 PAINT (X0,Y0),CLR,3 ' Fill figure with color
305 CLR=((CLR+1) MOD 3) ' Select next palette attribute
```

The PAINT command starts filling the screen with the palette color (attribute) placed in the variable CLR. It will fill the whole screen unless there's a boundary of the color (attribute) in the next parameter position, the 3 in the present case.

Play around with size and orientation to see what you can discover on your own. For instance, what happens when you use a larger or smaller step in line 180, or a larger range of R? Or what happens when you make the change in T in line 190 larger or smaller?

Don't forget to SAVE the results you like. But use different filenames so you won't erase any files with similar names already recorded on your *Test Programs* disk.

Changing the Center of the Triangle

ReLOAD TRIANGLE.FUN and we'll do some different things. That is,

Type: LOAD "TRIANGLE.FUN" <ENTER>

You need the filename extension in this case.

You can get interesting effects if you change the center of the figure. And these can merit an "ooh" or "ah" when combined with changes in R and T too.

Here are some basic changes to move the triangle to another center without removing the screen's center as a point of control. Type them into TRIANGLE.FUN and RUN them.

```
160 R1=60:T1=0
165 NX0=150:NY0=100 ' New screen center
175 FOR T1=0 TO 360 STEP 60 ' Change angle of center
180 X0=R1*COS(FR*T1)+NX0 ' Calc figure center
185 Y0=R1*SIN(FR*T1)+NY0
300 NEXT T1
```

As you see, these changes place identical triangles in a circle of about radius 60.

Try changing the STEP in line 175 to 10, and other values. But put the STEP back to 60 when you're finished experimenting.

We can get a more symmetric arrangement of the triangles by rotating them. With the STEP in 175 at 60 degrees, add the following line:

```
190 T=T+60 ' Rotate figure
```

Looks like someone cut slices out of a hexagonal (six-sided) pie. (This motif could be the basis for an explosion in an animated story.)

Let's put the pie together by controlling R, the size of the figures.

Later on, change R in line 140 to the value 60, and reRUN. Then you'll want to try other combinations of these parameters too. But for now leave the program as it stands. There are more steps to take with it.

Designs That Snake Along

You can generate an interesting class of designs in which the center of the figure changes a little from step to step and the orientation and size also change along the way. To illustrate, make these additions and RUN.

```
145 R=60
170 FOR NX0=210 TO 110 STEP -10
310 NEXT NX0
```

This pattern starts off looking like a jungle gym but ends up looking like a weave of some sort. Let's see if we can't get something a little less static.

How about putting all the figures on a horizontal line by making R1=0? That is, insert this line and RUN.

```
172 R1=0
```

Not very interesting. Another weave. You'll get lots of these.

Let's rotate the triangle. Make this change and RUN.

```
190 T=T+15
```

Sort of looks like a cocoon or a caterpillar. But still not very interesting.

You don't really need lines 175 and 300. They don't apply right now and only delay building the figure.

Type: DELETE 175 <ENTER>
Type: DELETE 300 <ENTER>

Let's make the triangle smaller as it moves along and give it a little less rotation.

```
173 R=R-5
190 T=T+5
```

Try RUNning now.

As you see, this makes a slightly twisted tunnel effect. But what happens if the steps move along slowly? Try this change:

```
170 FOR NX0=210 TO 110 STEP -2
```

That twists much more drastically.

Try this, though it seems a little odd: Make the radius R of the triangle vary as the sine of the angle T.

```
173 R=70*SIN(FR*T)
```

But before you RUN, lower the STEP length in line 170 from -2 to -1.5.

Interesting, no?

You can go on like this for hours with rather simple programs. It's like eating salted peanuts as you look greedily for the next effect.

CAVEAT: It's a bit of a bother, but you should save your programs at the current settings with great frequency and with imaginative filenames. Otherwise, you'll be sorry when you try to repeat some figure you'd like to show off. You'll never remember the settings. You can always trash the examples you don't want later on.

4. FUN WITH BOXES

There's another fun aspect of the LINE command. Clear the screen and try this.

Type: LINE (100,100)-(50,50),3,B <ENTER>

Here's a property of LINE that's quite useful. Using your screen-editing skills, put an F (for "fill") just to the right of the B (for "box," of course), and hit <Enter>. Naturally, the color attribute should not be 0. Try the command with attributes 1 and 2.

Figure 5.6 gives some surprising results when you vary its parameters. Type it in and RUN it to see what it does.

```
100 ' BOX SURPRISE
110 KEY OFF:SCREEN 1,0:COLOR 1,1
120 CLS
122 RF=3.14159/180 ' Radians per degree
124 X0=150:Y0=100 ' Center of circle
125 PSET (X0,Y0),0 ' Move to center
130 FOR ANGLE=0 TO 360 STEP 90 ' Count off angles
135 ' Calculate coordinates of point on circumference
140 X=100*SIN(RF*ANGLE)+X0
150 Y=100*COS(RF*ANGLE)+Y0
160 LINE (X0,Y0)-(X,Y),3,B ' Plot box
170 NEXT ' Do next box
```

Figure 5.6: BOX.BAS—Box Surprise

For all the apparent complexity of this program, it's dull, isn't it? Just a couple of crossed lines.

Well, don't be too sure. Look at the STEP portion of line 130. This command counts off angles around the center of a circle in steps of 90 degrees. That's not many times through the FOR...NEXT loop, considering there are 360 degrees in a circle. So what happens if we make this step smaller? Let's find out.

Use the EDIT command to bring up line 130 on the screen, and use the cursor controls to change the 90 to a 60. Then RUN the program again.

Not much more interesting, but be patient. On successive tries, change the STEP in line 130 to the following values and RUN: 45, 30, 20, 10, and 2 degrees.

Aha! That's more like it.

Before we leave this program, let's make one more change. Modify line 160 so it reads:

```
160 LINE -(X,Y),3,B ' Plot box
```

This relieves the center of the circle from being a corner for each box. Now the program will use a corner of the last box plotted for the new box.

Starting with a STEP of 90 degrees in line 130, successively reduce the STEP through the values we used earlier to see what happens.

Whenever you have a little program that you or someone else has developed for a particular graphics purpose, always play with it as we have here. You will probably discover something of interest that you didn't expect, and that you may be able to exploit at some future time. Graphics programming is an art form—a grab bag of tricks. You've got to experiment continually to develop and retain the skill.

5. AN ASIDE ON RENUM

You'll have noticed that the line numbers of Figure 5.6 don't move along evenly as in earlier examples. The fact is I generated this program from the one of Figure 5.4 by adding and modifying a few lines. If

I'd made too many additions, I'd have run out of line numbers and could have had some difficulty organizing the program.

There's a neat editing feature of BASIC for PCjr that permits you to RENUMber all the lines. While Figure 5.6 is still in memory try this:

Type: RENUM 1000 <ENTER>
Type: LIST <ENTER>

All the lines have been renumbered starting at 1000 and stepping along by 10.

But what if you had fifty lines to add between two existing lines? How can we make enough room? Try this:

Type: RENUM 500,,100 <ENTER>
Type: LIST <ENTER>

Now you have plenty of room, and when you've made the additions, you can straighten out the line numbers with something like RENUM 100.

Read about RENUM in your BASIC manual.

6. MOIRE PATTERNS

Have your eyes ever played tricks on you when you've looked through a window with overlapping screens, or driven past tennis courts with high fences? You may have seen the same effect produced by overlapping bamboo blinds. If you're into textiles, you're certainly familiar with the moire patterns of silk and other glossy fabrics. The effect comes from the parallel weave.

This shimmering effect can be produced by almost any pair of superimposed sets of parallel lines that are slightly misaligned. To illustrate, let's try some experiments with PCjr.

Effects of Different Periods

Figure 5.7 gives a simple program for drawing parallel lines that we can use as a point of departure. Key it into memory and RUN it.

```
100 ' MOIRE PATTERN FUN
110 KEY OFF:SCREEN 1,0:COLOR 6,1
120 CLS
130 CLR1=3:CLR2=2
140 S1=3:S2=3
150 FOR I=0 TO 199 STEP S1
160 LINE (0,I)-(299,I),CLR1
170 NEXT
180 FOR I=1 TO 199 STEP S2
190 LINE (0,I)-(299,I),CLR2
200 NEXT
```

Figure 5.7: MOIREFUN.BAS—Moire Pattern Fun

NOTE: If you're using a black and white TV or video monitor, you may have trouble with these experiments. Some lines show up ever so faintly on black and white.

You'll see from lines 140, 150 and 180 that the two sets of parallel lines have a period of three units. That is, the parallel lines in each set are three units apart. Change the period by making S2=4 in line 140, and RUN again.

How about that! Every fourth white line has disappeared. Of course, the magenta line has hidden it, producing what are called "beats." Try S2=5 to see what happens.

Now the beats are farther apart. But you can easily see why.

Effects of Misalignment

Let's reset the program to equal periods, that is, set S2=3 in line 140. But let's make this second set of parallel lines intersect the first set at a small angle. We can do this by modifying line 190 so that each line intersects the right edge of the screen at a lower point than its intercept does the left edge. Change line 190 to read this way:

```
190 LINE (0,I)-(299,I+15),CLR2
```

Where the magenta lines hide the white lines, we get a dark vertical bar. These are beats of another type.

What do you suppose will happen if we increase the angle of intersection even more, that is, move the intercept on the right edge even farther down? Make this change in line 190:

```
190 LINE (0,I)-(299,I+99),CLR2
```

Ah! Just increases the number of beats.

Joint Effects

Now let's also change the periodicity. Change S2 in line 140 to 5 again, and RUN. Quite a texture, eh?

Now that you have the idea, do some of your own experiments with this program. You can make finer-grained or coarser-grained textures, mix colors by changing the background and the palette, or whatever. Remember, experiment freely.

NOTE: To make the texture fill the whole screen, you have to start the second set of parallel lines off the top of the screen. For instance, try this substitution for line 180:

```
180 FOR I=-99 TO 199 STEP S2
```

7. ARTFUL EXPLOITATION OF MOIRE EFFECTS

There are some simple ways you can make beats swirl around and form interesting patterns. To get things rolling, try MOIRE.BAS of Figure 5.8. It builds on what we've done. But instead of working with parallel lines, we pick a random point on the screen and draw lines from it to the edge of the screen. We let the spacing of the intercepts at the edge of the screen be equal.

```
100 ' MOIRE.BAS
110 KEY OFF:SCREEN 1,0:COLOR 1,1
120 CLR=3
130 S=3
140 CLS
150 X0=300*RND
160 Y0=200*RND
170 ' Top quadrant
180 FOR I=0 TO 299 STEP S
190 LINE (X0,Y0)-(I,0),CLR
200 NEXT
210 ' Right quadrant
220 FOR I=0 TO 199 STEP S
230 LINE (X0,Y0)-(299,I),CLR
240 NEXT
250 ' Bottom quadrant
260 FOR I=0 TO 299 STEP S
270 LINE (X0,Y0)-(299-I,199),CLR
280 NEXT
290 ' Left quadrant
300 FOR I=0 TO 199 STEP S
310 LINE (X0,Y0)-(0,199-I),CLR
320 NEXT
330 FOR I=1 TO 1500:NEXT
340 GOTO 140
```

Figure 5.8: MOIRE.BAS

The essentials of MOIRE.BAS consist of four similar parts. Each part draws lines from the random point (X0,Y0) to one of the four screen edges. Only every S-th point is selected as an intercept. S is set in line 130 and used as a STEP in lines 180, 220, 260 and 300. The results become coarser as S increases.

The program is set up to repeat with a new random point after a short pause (line 330) while you admire the result. Stop the program with <Fn-Break>.

Compare the FOR...NEXT loops of Figures 5.8 and 5.7 to be sure you understand how the effects differ.

A Moire Carpet

One undesirable aspect of MOIRE.BAS is the seams it produces when it moves from one edge to the next. The problem is the abrupt change in angle that occurs at these boundaries. One way to deal with this problem is to use an angle generator such as one used to generate circles or ellipses.

Figure 5.9 gives a little subroutine generating an ellipse out of lines drawn from its center to its circumference. Type it into memory so we can work with it.

```
1000 ' Ellipse of lines from its center
1010 FOR T=0 TO 360 STEP S ' Step around the ellipse
1020 ' Ellipse equations
1030 X=A*R*SIN(P*T)+X0
1040 Y=B*R*COS(P*T)+Y0
1050 ' draw line from center to circumference
1060 LINE (X0,Y0)-(X,Y),C
1070 NEXT ' Repeat for next angle
1080 RETURN ' Go back to mainline
```

Figure 5.9: Ellipse of Lines from Its Center

There are a few things we've got to set up to use this subroutine, such as the following:

```
110 KEY OFF:CLS
120 SCREEN 1,0 ' Med res graphics
130 COLOR 1,1 ' Blue field, palette 1
140 P=3.14159/180 ' Radians per degree
```

You've seen the screen setup commands before. Line 140 defines a constant that will convert degrees into radians for the—ugh— trigonometric functions you must use. One of the nice qualities of all

subroutines is that you can use them over and over again. To get different effects, you must change the values for some of the constants the subroutine uses. Here are some we can try with the present case.

```
150 ' Draw first ellipse
160 A=6:B=5 ' Aspect ratio for circle
170 S=1 ' Step size in degrees
180 X0=150:Y0=100 ' Center
190 R=50 ' Essentially the radius
```

If you try to draw a circle on your video screen using standard formulas you are likely to get an ellipse rather than a circle, because the vertical and horizontal units of space on the video screen are not equal. As a result, you must play around with the constants A and B in Figure 5.9 to find the combination that works for you. For my color TV, the units A=6 and B=5 seem to work best, as you see in line 160.

If you are to use the subroutine, you must also specify the angular STEP size (S in line 170) for angling the lines, the coordinates X0 and Y0 (line 180) where the figure will be centered, the radius R (line 190), and the palette color, C, to use. Oh, where's C? We'll define it at the same time we call the subroutine.

```
200 C=3:GOSUB 1010 ' Set color & go draw
```

But before we try this program, we'd better put in the statement

```
210 END
```

to halt the program. Okay, SAVE this program as CARPET.BAS, and RUN it!

As you can see, the result is similar to MOIRE.BAS but without the seams. All you need do is define the center, (X0,Y0), randomly, and then recycle the process after a short pause. That is an exercise you might try later.

Using the Ellipse Subroutine

Right now let's play a little with the ellipse-generating subroutine. Don't type anything into program memory for a bit—you know, don't use line numbers.

You've noticed that the radius, R, is so large that the circumference of the circle is off the screen. Try this:

Type: <Ctrl-L>
Type: R=5:gosub 1000 <ENTER>

There, you can see the edge of the circle. We can put another circle of different radius at another point this way. Wait! Don't clear the screen!

Type: R=7:X0=180:Y0=175:gosub 1000 <ENTER>

My Phantom Spider

You might go on this way for as long as you like, plotting overlapping circles of various sizes. But I've got another idea based on what we did earlier with moire patterns. Type in these lines along with what's already in memory:

```
2000 ' PHANTOM SPIDER
2010 SCREEN 1,0:COLOR 1,1
2020 P=3.14159/180 ' Radians per degree
2030 CLS:A=8:B=5:C=3 ' Define ellipse
2040 R=50:S=3 ' Of large radius
2050 X0=150:Y0=100 ' In center of screen
2060 GOSUB 1000 ' Go draw it
2070 X0=165:Y0=100 ' Offset it slightly
2080 GOSUB 1000 ' And redraw it
```

When you're ready,

Type: RUN 2000 <ENTER>

Notice how the beats form a series of curves extending from the center of the screen. If you're into inkblots you may see the big spider, formed from the beats, sitting in the center of its web. Step back from the screen if you have trouble picturing it.

Don't forget to SAVE the program, perhaps with the name SPIDER.BAS.

The Throbbing Eye

Let's try one more experiment with the ellipse subroutine of Figure 5.9. As a starting point, we'll use the moire carpet pattern we generated earlier. The instructions to generate it should still be in memory, lines 110 to 200. What we'll do is superimpose a small ellipse on the center of the carpet with these instructions.

```
100 ' THROBBING EYE
210 ' Draw eye—second ellipse
220 A=8:B=5 ' Less circular aspect
230 C=2:R=10:S=4:GOSUB 1010
```

NOTE: If you don't type in this line 210, be sure to DELETE the END statement in the existing line 210.

Now give it a throbbing action.

```
235 ' Introduce the throb
240 FOR I=0 TO 7:COLOR I,1:FOR K=1 TO
    500:NEXT:NEXT:GOTO 240
```

That's it. SAVE it—I call it THROB.BAS—and RUN it.

Here's a series of experiments you might try. Insert a line in the "ellipse of lines" subroutine to change the color of the line drawn on each cycle of the FOR...NEXT loop. For instance, try a line like this:

```
1015 C=2+((C+1) MOD 2)
```

Or this variant of the same thing:

```
1015 C=1+((C+1) MOD 3)
```

Always keep experimenting. And go back to earlier examples to try new tricks on them too. You might discover something great.

8. HUMANIZING MATHEMATICAL FORMS

All that we're doing with graphics in this chapter is based on mathematical forms—lines, boxes, circles, ellipses and so on. There's a certain slickness about such constructions that makes them look as if

they were drawn by a robot, which of course they were. But you can take advantage of the compactness and preciseness of math, and yet mar the slickness a little to fool the humanist.

One way is to introduce random variations in the mathematical forms so they appear to have been drawn by an unsteady eye. Another technique is to eliminate some detail. Human artists tend to paint the "soul" of the subject, the essence, not the unimportant trifles. Let's try it ourselves.

Fans

Figure 5.10 builds on the techniques we've worked with to splash fanlike structures on the screen. On my black and white screen, some of the pictures are reminiscent of frosted or cut glass designs. Type the program and RUN it.

```
100 ' FANS
110 CLS:KEY OFF
120 SCREEN 1,0
130 COLOR 1,1
140 FOR I=1 TO 5 ' Count off 6 splashes
150 ' Define one end of first line
160 X1=INT(RND*320):Y1=INT(RND*200)
170 ' Define the other end of first line
180 X2=INT(RND*320):Y2=INT(RND*200)
190 FOR J=1 TO 55 ' Count off lines
200 ' Vary one end of line randomly
210 Y2=Y2-1-INT(RND*5):X2=X2-1-INT(RND*5)
220 LINE (X1,Y1)-(X2,Y2),3 ' Draw line
230 NEXT ' Do next line
240 NEXT ' Do next splash
250 FOR K=1 TO 2000:NEXT ' Pause to admire picture
260 GOTO 110 ' Do next picture
```

Figure 5.10: FANS.BAS

Each fan is generated from a random line defined in program lines 150-180. The fan is then opened up from this starting line by subtracting a small random amount from the coordinates of one of the ends (line 210). The number of fans is kept small to understate the resulting splashes of forms.

Try changing the color patterns to see the effects produced. For example, you might try these changes:

```
125 PAL=(PAL+1) MOD 2
185 CLR=1+(INT(RND*3) MOD 3)
220 LINE (X1,Y1)-(X2,Y2),CLR
```

Textures

Building on FANS.BAS, Figure 5.11 frees the fan's pivot to let the lines flow through space in a controlled but random manner. Type it into memory and RUN it.

```
100 ' TEXTURES.BAS
110 KEY OFF:CLS
120 SCREEN 1,0
130 COLOR 1,1
140 ' Start drawing
150 CLS ' Clear screen
160 FOR I=1 TO 6 ' Count off splashes
170 ' Define beginning of first line
180 X1=INT(RND*320):Y1=INT(RND*200)
190 ' Define end of first line
200 X2=INT(RND*320):Y2=INT(RND*200)
210 ' Define direction of motion
220 IF RND<.5 THEN SN=-1 ELSE SN=1
230 FOR J=1 TO 50 ' Count off lines
240 ' Move ends of line in random pattern
250 Y2=Y2+SN*INT(1+RND*5):X2=X2+SN*INT
    (1+RND*5)
260 Y1=Y1+SN*INT(1+RND*5):X1=X1+SN*INT
    (1+RND*5)
270 LINE (X1,Y1)-(X2,Y2),3 ' Draw line
```

```
280 NEXT ' Do next line
290 NEXT ' Do next splash
300 FOR K=1 TO 2000:NEXT ' Pause to admire picture
310 ' SPACE extends admiration period
320 A$=INKEY$:IF A$=" " THEN 350
330 GOTO 150 ' Make another picture
340 ' Another SPACE renews picture-making
350 A$=INKEY$:IF A$="" THEN 350
360 GOTO 150 ' Make another picture
```

Figure 5.11: TEXTURES.BAS

You quit TEXTURES.BAS with <Fn-Break>. If you want to pause longer to study the generated picture, hit <Space>. To continue generating pictures, just hit <Space> once more.

The visual effect of TEXTURES.BAS is similar to the hachuring or shading an artist does. On a color TV the textured patterns are quite attractive, giving the impression of twills and other coarse weaves. The effect is not so interesting on RGB and black and white monitors. Being of better quality, they filter out the video noise effects of the TV that can actually enhance a picture of this character. This filtering condition is especially noticeable with the moire patterns we generated earlier.

TEXTURES.BAS is designed as a conversation piece that you might play at parties. Use <Space> to pause and ask what the pattern represents—birds, ships in the night, crystals and so on. It's sort of an inkblot game.

Compare TEXTURES.BAS closely with FANS.BAS to appreciate how the differences generate the effects seen.

9. CIRCLES AND OTHER ROUNDISH STUFF

Circles

There's a neat BASIC command for PCjr that you've got to learn about—CIRCLE. Let's try it. Clear memory and type these lines in:

```
100 ' Circle fun
110 KEY OFF:SCREEN 1,0:COLOR 1,1
120 CLS
```

To get started, just

Type: <Fn-2>

We now have a quick way to reset the screen after our little experiments. We'll use these instructions further on for a program.

Type: CIRCLE (150,100),30,3 <ENTER>

We've drawn a white circle of radius thirty units at the center of the screen, right? Using the cursor, move up to the CIRCLE command and change the radius to fifty and type <Enter>.

Using the cursor again, change the center of (150,100) to (50,150) and hit <Enter>.

Now modify the command so it reads this way:

CIRCLE (50,100),50,3,,,4 <ENTER>

Watch the number of commas between the 3 and the 4. Other stuff goes there that we don't need just now.

Ooh! An ellipse. The 4 is called the aspect ratio. We ran into it earlier with our ellipse-of-lines subroutine (Figure 5.9). To make the ellipse go the other way, try these changes:

Type: CIRCLE (200,100),40,3,,,.5 <ENTER>

What figure do you get if the aspect ratio is 1? How about 5/6 (i.e., 0.833)?

Arcs

Okay, you've got the idea of how to put circles and ellipses of various sizes and aspects all over the screen. But there's something else we can do with CIRCLE. First we need the value of pi, so put this line in memory:

130 PI=3.141593

Pi, you remember, is the ratio of the circumference of a circle to its diameter. It comes up all over the place.

Now try this:

Type: <Fn-2>
Type: CIRCLE (150,100),50,3,0*PI,.5*PI,1 <ENTER>

Did you get the arc of an ellipse? That's what the extra two numbers defined, the start and the end of the arc. Change the CIRCLE command to read this way:

Type: CIRCLE (150,100),50,3,1*PI,1.5*PI,1 <ENTER>

Now you've got the arc halfway around the ellipse. You can start or end the arc any place between -2*PI and +2*PI.

Try some examples of your own.

Rounding the corners of shapes built from lines is an important use for elliptical arcs. But this kind of application can get you into some messy math. Remember that you have to figure the center of the ellipse and the aspect ratio so that everything fits flush.

Spiral of Arcs

Let's try to build a spiral of arcs from a series of ellipses of larger and larger radii. We'll place the spiral at center screen and vary the radius and arc start and end points using our constant companion, the FOR...NEXT loop.

```
140 FOR R=0 TO 100 STEP .5 ' Change radii
150 START=START+.03*PI ' Increase arc start point
160 IF START>2*PI THEN START=START-2*PI ' Correct if too
    big
170 ND=START+1 ' Arc end 1 greater than start
180 IF ND<=2*PI THEN 190 ' Test for end of cycle correction
185 NDD=ND-2*PI:ND=2*PI
187 CIRCLE (150,100),R,3,0,NDD,.8
190 CIRCLE (150,100),R,3,START,ND,.8 ' Draw arc
200 NEXT ' Go draw next arc
```

For convenience, change the program name too.

```
100 ' SPIRAL.BAS
```

Two comments about this program. First, guard against making the arc start and end points greater than 2*PI. Such values will render an error. Lines 160 and lines 180-187 do the needed corrections. Always be careful when dealing with periodic functions. From interpolated lines 185 and 187, you can detect that I wasn't careful the first time round.

Second, we can't use END as a name for the end of arc—it's the reserved name of a BASIC command. Thus, I used the similarly pronounced name ND instead.

Try a bunch of experiments making spirals. How can you control the thickness of the spiral? The number of cycles? The distance between successive loops?

Can you put some sound to the spiral generation to make it more interesting to watch?

Here's a simple line on which to build sound effects.

```
105 SOUND OFF:BEEP OFF ' Use internal speaker
145 PLAY "mb":SOUND 4*R+37,3.6
```

Can you make moire patterns using overlapping spirals?

Serendipity from Errors

I think all great discoveries must be made by accident. Columbus sure wasn't looking for America when he bumped into it. And penicillin derived from an experiment spoiled by a mold. What turns a spoiled experiment into a discovery is the attitude of the experimenter toward his frustration. Understanding why an experiment was spoiled can open the door to greater things.

While my son Tim was running through this chapter, he made a mistake typing in SPIRAL.BAS. He typed line 180 this way, omitting the pi:

```
180 IF ND<=2 THEN 190 ' Test for end of cycle correction
```

Make the change to see what happens.

That's a much more interesting spiral. Sort of like a nebula or galaxy—one of those great spirals of stars. And with the interruptions

in the sound as the pitch gets higher, you can feel a strain, as if something were about to explode. Let's see if we can actually make the nebula "explode" as it reaches climax.

Figure 5.12 gives a first approximation to an explosion with sound effects. Add it to SPIRAL.BAS and SAVE the result as NEBULA.BAS. Then experiment to see if you can improve the program.

```
1000 ' Explosion—sort of
1010 SOUND ON:BEEP OFF ' Use external speaker
1020 CLS:X0=150:Y0=100 ' Define center of explosion
1030 PSET(X0,Y0),3
1040 FOR I=1 TO 7 ' Repeat rumble
1050 COLOR I,1 ' Add color—a throb
1060 V=16-2*I:D=I/400 ' Decrease volume & increase duration
1070 NOISE 7,V,8 ' Make noise at pitch of SOUND command
1080 FOR F=1400 TO 600 STEP-350 ' Range of rumble
1085 ' Explosion of lines
1090 LINE (X0,Y0)-(X0+F*(RND-.5)/5,Y0+F*(RND-.5)/5),3
1100 SOUND F,D,V,2 ' Create rumble with various frequencies
1110 SOUND 37+F/3,D,V,2 ' around F as center
1120 SOUND 5*F,D,V,2
1130 SOUND 37+F/2,D,V,2
1140 SOUND 7*F,D,V,2
1150 SOUND 37+F/4,D,V,2
1160 NEXT
1170 NEXT
1175 ' Continue throb
1180 FOR I=0 TO 7:COLOR I,1:FOR K=1 TO
     500:NEXT:NEXT:GOTO 1180
```

Figure 5.12: Explosion for NEBULA.BAS

Just a couple of notes. The idea I had for the explosion sound is to mix high and low frequencies that sweep down through a range controlled by the FOR...NEXT loop at line 1080. Notice how the frequencies in the SOUND commands (lines 1100-1150) jump back and forth around F, the central frequency.

To add some "fog" to this rumbling, I used the NOISE generator (line 1070). This command adds a buzz to the frequency being SOUNDed if you play it with the voice 2. See the 2's in the SOUND commands? (Voices are numbered 0, 1 and 2, and we'll see more about them later.)

Since an explosion starts loud and rapidly fades away, I made the volume and duration variables V and D, respectively. Their values are controlled in line 1060.

The visuals here leave much to be desired because of the random lines radiating from the center of the screen (line 1090). With the facilities we've covered thus far, the graphics operate too slowly to give an effective result. We'd have to program individual pieces of the exploding nebula, constructed in advance, and then flashed on the screen in successive positions outward from the center. This is animation, a subject we take up toward the end of the book.

You've seen the flashing colors before in the throbbing eye. Try running NEBULA.BAS minus the line generator in line 1090 to appreciate its impact.

10. USING A "HAKE" BRUSH

So far we've been using a narrow brush to form most of our figures, one the width of a single line. Even where we've filled up an area with color, it's been by drawing lots of narrow lines. The exception is where we used the "fill" option of the LINE command to paint the interior of a box. And we did use the PAINT command once.

This fill facility, available with the PAINT command, is good for any closed shape. Try it. (Set up the screen in medium resolution with white on a blue field first.)

Type: <Ctrl-L>
Type: CIRCLE (150,100),50,3 <ENTER>

If you don't specify the "aspect," you get a circle.

Don't erase the screen. You're going to paint the circle's interior "green." Right below the "Ok,"

Type: PAINT (150,100),1,3 <ENTER>

NOTE: According to the BASIC manual (under COLOR) color 1 of palette 1 is "cyan," a blue of some sort. My set produces a green color, and I will refer to it as such. Differences in TVs and monitors can produce quite different colors for the same input signals, so don't be surprised if you generate colors nobody else gets.

We needn't select the center of the circle, (150,100), as the point from which to paint. Any point inside the boundary of the figure will do. The number 1 is the default green of palette 1. The 3 is the white of the boundary. If you use another color, the PAINT command won't know where to stop and in this case will paint the entire screen green.

Let's build on this example. Move the cursor up to the CIRCLE command and change the aspect to 0.4.

Type: CIRCLE (150,100),50,3,,,.4 <ENTER>

Now there's an ellipse inside the circle. It looks something like the CBS "eye." Try painting the ellipse magenta. If the cursor is not at the PAINT command, move it there and make these changes:

Type: PAINT (150,100),2,3 <ENTER>

Now put a circle inside the ellipse and erase the magenta fill color by using the CIRCLE and PAINT commands on the screen and making these changes in the order shown.

Type: CIRCLE (150,100),24,3 <ENTER>
Type: PAINT (150,100),0,3 <ENTER>

The blue inside the pupil of the eye is the field color. We can change it this way. You should be at the fifth line down.

Type: COLOR 6,1 <ENTER>

The orange-brown on my set contrasts reasonably well with the palette 1 colors. Let's see what palette 0 gives. Modify the COLOR command on the screen so it reads this way:

Type: COLOR 6,0 <ENTER>

All the commands disappeared because the color 3 of palette 0 is brown, the same as the field color. Not to worry. Carefully now—don't make an error because you can't see to correct it—

Type: COLOR 1,1 <ENTER>

The screen should be intact again. If you got fouled up and it didn't work, there's a recovery procedure you can try. Carefully now, press the UP arrow key twice, quickly. This moves you up above the picture.

Press <Esc(ape)> to clear the line.
Retype: COLOR 1,1 <ENTER>

If you still screw up, try again. And if you can't get this procedure to work for you, you'll just have to begin over—say, with a warm start.

11. CHANGING THE PALETTE

Some of the default color combinations of palette 0 and 1 are nauseous. Fortunately, you can change them. Move the cursor up to the top line and erase the CIRCLE command with <Esc(ape)>. Let's change the cyan (attribute 1) that is green on my TV to a light color such as yellow—number 14 (see the COLOR command in the BASIC manual, the table of colors for the text mode screen).

Type: PALETTE 1,14 <ENTER>

The 1 is the palette attribute. The 14 is the yellow we want.

Let's also change the magenta (attribute 2) to red (color number 4). Change the PALETTE command to this:

Type: PALETTE 2,4 <ENTER>

Play around with PALETTE, changing the assignment of colors to attributes. You can also change the attributes of palette 0. Let's see what they look like at the moment. Clear a line with <Esc(ape)> and

Type: COLOR 1,0 <ENTER>

Move back up to the PALETTE command and change it to this:

Type: PALETTE 1,8 <ENTER>

Try combinations of color assignments to the palette attributes so you'll get a feel for the possibilities. When you finish playing, restore the default assignments this way. Clear a line and

Type: PALETTE <ENTER>

12. KEEP GOING BACK OVER EARLIER GROUND

By now you should be getting a pretty good idea of what it takes to build pictures on the medium-resolution graphics screen. The emphasis here has been to use formulas to form shapes from points, lines and ellipses. Using formulas is not the only way to do computer graphics, as you'll see shortly, but they do make up an important part of the bag of tricks you need to be a successful practitioner of the art.

If there's one thing you should have learned from this chapter, it is the importance of experimenting to combine ideas for new graphic effects.

Go back through the chapter to see whether some of the techniques you learned toward the end can be used to enhance ideas introduced earlier. Save your results and make notes both of ideas that led to great results and of actions that got you into trouble.

Also be sure you understand the BASIC commands used to develop the examples. Figure 5.13 lists those used in the chapter. Run down the list and ask yourself whether you really understand what each does. If you have some hesitation, look up the problem command in the BASIC manual, and review how it was used in this chapter. Try the examples in the BASIC manual too. They'll add to your perspective.

BASIC COMMAND	FUNCTION
CIRCLE	Draw an ellipse or arc at the specified point of designated color and aspect
CLS	Clear screen
COLOR	Set field and palette colors
COS	Calculate trigonometric cosine
FOR...NEXT	Repeat a sequence of commands
GOSUB	Take next command from line number, and RETURN to this point when finished
GOTO	Take next command from line specified
IF...THEN	Make decision on program flow
KEY OFF	Toggle screen prompts off
LINE	Draw a line between specified points; build a box
LIST	List program memory to the screen
LOCATE	Position cursor on screen for printing text
NOISE	Generate various noise patterns through the external speaker
PAINT	Fill closed figure with specified color
PALETTE	Reassign the color attributes of a palette
PRINT	Print text on the screen
PSET	Print dot at specified location and color on the graphics screen
RENUM	Renumber the lines of a BASIC program
RND	Generate random number between 0 & 1
SCREEN 1,0	Set medium-resolution graphics screen
SIN	Calculate trigonometric sine
SOUND	Play a pitch of specific duration

Figure 5.13: Summary of Commands in Chapter 5

CHAPTER 6
MORE ADVANCED PICTURE-MAKING

Objectives:

This chapter will build on the techniques presented thus far and will turn toward composition and other graphics goals.

1. IT'S IMPORTANT TO COLLECT SUBROUTINES

Curves Generated by Straight Lines

To do computer graphics programming successfully, you must build a repertoire of subroutines that you'll use over and over again to produce your brand of special effects. Sometimes they are very simple programs, like the one in Figure 6.1. Type it into memory so you can do something with it.

```
1000 ' Corner net generator
1010 U1=(X2-X1)/N ' Define step lengths
1020 V1=(Y2-Y1)/N
1030 U2=(X3-X2)/N
1040 V2=(Y3-Y2)/N
1050 FOR K=0 TO N
1060 LINE (X1+U1*K,Y1+V1*K)-(X2+U2*K,Y2+V2*K),3
1070 NEXT
1080 RETURN
```

Figure 6.1: Corner Net Generator

The subroutine of Figure 6.1 is just a variation on what we've already done to draw crisscrossed lines. In this case the end points lie on two lines that intersect on the screen. Three points are involved: (X1,Y1), (X2,Y2) and (X3,Y3), where (X2,Y2) is the point of intersection of the lines that also run through (X1,Y1) and (X2,Y2), respectively. N is the number of crisscrossed lines to draw.

Type: X1=50:Y1=100 <ENTER>
Type: X2=250:Y2=50 <ENTER>
Type: X3=150:Y3=175 <ENTER>
Type: N=10 <ENTER>
Type: <Ctrl-L>
Type: GOSUB 1000 <ENTER>

If we had made the angle between the base lines larger, the net would look better, but the pretty curve the straight lines carve out wouldn't be so nice. As you can easily visualize, you might develop many variations of this curve by varying the angle at which the two base lines intersect, by varying the length of these base lines, and by making the step length for one base line different from that of the other.

Try a few examples of your own.

If you make N larger, you can make the network practically solid.

You've Got to Have Heart

There are all sorts of interesting curves you can dig out of analytic geometry books and program as subroutines. Figure 6.2 is one I particularly like. Without erasing the corner net generator, type Figure 6.2 into memory. You'll use the net generator shortly.

```
2000 ' HEART Generator
2010 CLR=0:A=.5:B=.025:C=.8
2020 RF=3.1459/180
2030 GOSUB 2160
2040 FOR FX=-A TO A STEP B
2050 FY=C*(EXP(LOG(FX^2)/3)-SQR
     (A^2-FX^2)):GOSUB 2170
2060 X=300-(SX*HX+U0):Y=200-(SY*HY+V0)
2070 LINE -(X,Y),CLR
2080 CLR=3
2090 NEXT FX
2100 FOR FX=A TO -A STEP -B
2110 FY=C*(EXP(LOG(FX^2)/3)+SQR
     (A^2-FX^2)):GOSUB 2170
2120 X=300-(SX*HX+U0):Y=200-(SY*HY+V0)
2130 LINE -(X,Y),CLR
2140 NEXT FX:GX=-A:GY=C*(EXP(LOG(A^2)/3)):
     GOSUB 2180
2150 X=300-(HX*SX+U0):Y=200-(HY*SY+V0):LINE
     -(X,Y),CLR:RETURN
```

```
2160 STH=SIN(THETA*RF):CTH=COS(THETA*RF)
2170 GX=FX:GY=FY
2180 HX=GX*CTH+GY*STH:HY=-
     GX*STH+GY*CTH:RETURN
2190 ' Heart plots at center u0,v0
2200 ' Scale with sx and sy
```

Figure 6.2: Heart Generator

This subroutine is a bit complicated, even if you know analytic geometry, but that doesn't mean you can't use it. A lot of people drive cars without knowing what's under the hood. All you need to know is that the heart needs the following numbers to specify it:

THETA	The rotation angle
SX, SY	Scale factors to stretch or shrink the heart in the horizontal, vertical directions
U0, V0	Where to center the heart on the screen

With this information let's try the subroutine.

Type: THETA=180 <ENTER>
Type: SX=100:SY=100 <ENTER>
Type: U0=150:V0=100 <ENTER>
Type: <Ctrl-L>
Type: GOSUB 2000 <ENTER>

To turn it rightside up, try again with THETA=0 degrees. Everything else is okay.

Type: THETA=0 <ENTER>

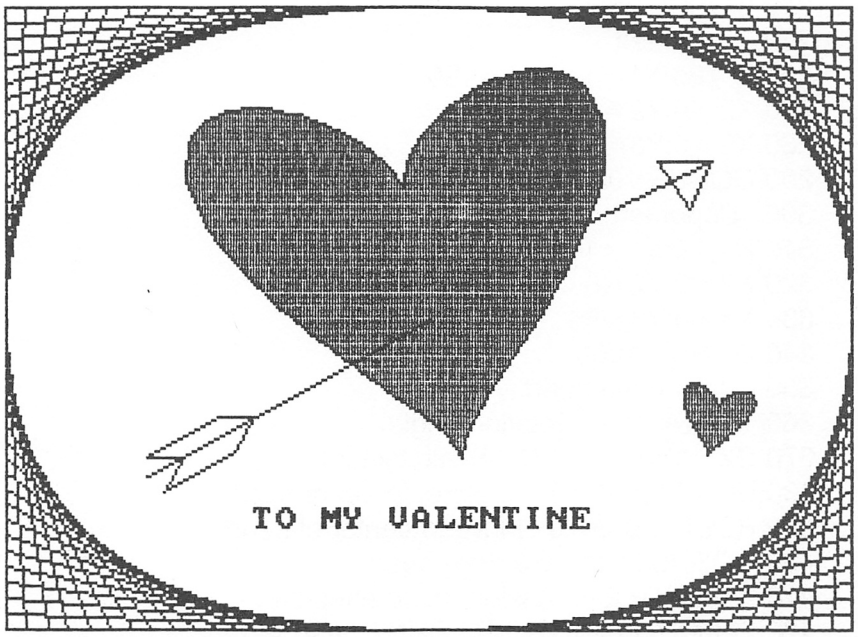
You can distort the heart to get other figures. Does this set of parameters come close to a bird of prey in your mind's eye?

Type: SX=80:SY=17:THETA=5 <ENTER>
Type: <Ctrl-L>
Type: GOSUB 2000 <ENTER>

2. PUTTING IT ALL TOGETHER

To My Valentine

I'm sure this hint is enough for you to conceive of combining the heart and the corner net subroutines to make a valentine on the screen. There are lots of ways you might do so. Figure 6.3 is what I came up with.



The program to generate the valentine is given in Figure 6.4. Type in the program on top of the subroutines you've already typed to learn how the picture is generated.

```
100 ' VALNTINE.BAS
110 KEY OFF
120 SCREEN 1,0:CLS ' Medium res graphics
130 COLOR 1,1 ' Blue field, palette 1
140 ' Draw lace in corners
150 ' Upper right
```

```
160 N=20:X1=149:Y1=0
170 X2=299:Y2=0
180 X3=299:Y3=99
190 GOSUB 1000
200 ' Lower right
210 N=20:X1=149:Y1=199
220 X2=299:Y2=199
230 X3=299:Y3=99
240 GOSUB 1000
250 ' Lower left
260 N=20:X1=149:Y1=199
270 X2=0:Y2=199
280 X3=0:Y3=99
290 GOSUB 1000
300 ' Upper left
310 N=20:X1=149:Y1=0
320 X2=0:Y2=0
330 X3=0:Y3=99
340 GOSUB 1000
350 ' Draw main heart
360 THETA=10 ' Rotation angle
370 SX=150:SY=110 ' Scale factors
380 U0=149:V0=99 ' Define center of heart
390 PSET (U0,V0),0 ' Move to center of heart
400 GOSUB 2010 ' Go draw heart
410 PALETTE 2,4 ' Assign red to attribute 2
420 PAINT (U0,V0),2,CLR ' PAINT heart red
430 ' Draw second small heart
440 THETA=-10 ' Rotation angle
450 SX=25:SY=20 ' Scale factors
460 U0=50:V0=65 ' Define center of heart
470 PSET (300-U0,200-V0),0 ' Move to center of heart
480 GOSUB 2010 ' Go draw heart
490 PAINT (300-U0,200-V0),2,CLR ' PAINT heart red
500 ' Draw feathers on arrow
510 LINE (62.5,143.75)-(50,143.75),CLR
```

```
520 LINE -(75,131.25),CLR
530 LINE -(87.5,131.25),CLR
540 LINE -(81,142),CLR
550 LINE -(56,155),CLR
560 LINE -(62.5,143.75),CLR
570 ' Draw shaft of arrow
580 LINE (50,150)-(150,100),CLR
590 LINE (205,70)-(250,50),CLR
600 ' Draw arrowhead
610 LINE -(231.25,50),CLR
620 LINE -(241.5,65),CLR
630 LINE -(250,50),CLR
640 ' Print message
650 LOCATE 21,12:PRINT"TO MY VALENTINE";
660 GOTO 660
```

Figure 6.4: VALNTINE.BAS

The valentine program is easy to follow. First a lacy net is placed in each corner (lines 160-340), using the edges of the screen as base lines. Next the main heart figure is drawn (lines 350-490). I've rotated it slightly so it won't look so mathematical. By not correcting for aspect, the heart is a little asymmetric, more like a heart an artist would draw. Similarly for the second, little heart (lines 430-490). It is included for balance.

Notice that I changed attribute 2 of palette 1 from magenta to the more appropriate color red (lines 410-420).

Drawing the arrow, a corny feature, was a little work—an exercise with graph paper. I don't know where to buy graph paper that's in the same aspect ratio as my TV screen. The plots on square grids are always distorted when transferred to the screen. Well, it took some eyeball adjustments to make the feathers reasonably symmetric. With the arrowhead I was less successful. I lucked out in guessing where the head end of the arrow would appear from behind the heart. This kind of thing is always a lot of work.

Adding Sounds

More often than not, sound can improve a graphics demonstration. Of course, if the approach used is inappropriate to the audience or the subject, sound can be a big negative. So what can we do with sound to enhance a presentation such as the valentine? Well, it rather depends on your point of view. Do you want to be serious, romantic, comic, mischievous, or what?

To illustrate, let's take the approach seen in many cartoons. We'll let PCjr mumble and hum to itself as it draws the figures. To start, add this line to the valentine program:

```
105 SOUND OFF:BEEP ON (Use internal speaker)
```

Now let's have PCjr put the lace in each corner with a little flourish, and a slight pause in between for thinking. The sound produced will depend on where you draw the lace on the screen. The sound should be put inside the corner net generator subroutine to get maximum control. Copy this:

```
1055 PLAY"mb":SOUND 200+(X2+U2*K)*(Y2+V2*K)/2,.5
```

When the lace has been laid in one corner, we can pause a bit with this line.

```
1075 FOR Z=1 TO 500:NEXT
```

To add sound to the drawing of the heart, we also have to splice the sound generator in the subroutine, in two places in fact. The subroutine draws the heart in two halves. These lines will render a kind of tapping sound.

```
2085 PLAY"mb":SOUND 37+5*(199-Y),.1
2135 PLAY"mb":SOUND 37+5*(199-Y),.1
```

The next action is to paint the heart. How about a sound that denotes a "filling" process, starting with a high, loud sound, gradually becoming softer and lower as the heart fills.

```
415 PLAY"mb":FOR Z=500 TO 200 STEP -10
416 SOUND Z,1,Z/20-10:NEXT
```

This sound pattern is initiated before the PAINT command in line 420, so it must run long enough in the background to span the time it takes to fill the heart.

Coordinating sound to the action on the screen, as in this case, can take a lot of fiddling with STEP sizes and other parameters. And if you decide to add even more frills, you can upset the timing and have to start fiddling all over again.

While filling the second heart, let's use a tinkle sound.

```
485 PLAY"MB":SOUND 2000,.5
```

Finally, we'll try to make a sound that indicates the arrow plopping into the heart. To add a little surprise, we'll pause a moment to give the viewer the mistaken idea that the action is over.

```
504 FOR Z=1 TO 1500:NEXT
506 PLAY"MB":SOUND ON:NOISE 2,15,3
```

What's this NOISE command? All it does is generate a crackling or hissing sound reminiscent of old-fashioned radios. You have the choice of eight pitches and qualities of hiss. We're using type 2 in line 506, the lowest pitch, and playing it as loud as possible (volume=15) for a duration of three clock ticks. (See the NOISE command in Chapter 4 of the BASIC manual.)

3. SAVING PICTURES ON DISK

Is VALNTINE.BAS still in memory? I want to show you how to save a copy of it on disk in picture form. With the SAVE command, you save only the program to generate the picture; every time you want to look at the picture, you have to generate it from scratch. This is not a problem with the valentine, because it doesn't take long. But some pictures could take half an hour or longer to generate. In these cases, you'll want to save the generated picture so you can load it back on the screen quickly next time you're showing off to friends.

Add these lines to VALNTINE.BAS.:

```
654 DEF SEG=&HB800 ' Block out the screen buffer
656 BSAVE "VALNTINE.PIC",0,&H4000 ' Transfer buffer to disk
```

You don't need to understand these lines to use them in your programs for saving pictures. Just change the name VALNTINE.PIC to one of your own choosing. For those of you who are aficionados of command-level language and hexadecimal codes, however, I'll tell you that the DEF(ine) SEG(ment) statement sets a pointer to the beginning of the buffer (memory area) where the picture on the screen is stored.

Enough explanation. RUN VALNTINE.BAS again and watch the picture being written on diskette when it executes line 656.

Did it work correctly? Let's verify. (SAVE the program first.)

Type: <Fn-Break>
Type: NEW <ENTER>
Type: <Fn-L>
Type: BLOAD "VALNTINE.PIC" <ENTER>

At this point you should see the picture being transferred from disk to the screen—by alternate lines in two phases.

If you don't want the "Ok" prompt to mar the screen when BLOAD finishes, key in and RUN a little program like this one:

```
100 SCREEN 1,0:COLOR 1,1:PALETTE 2,4
110 BLOAD "VALNTINE.PIC"
120 GOTO 120
```

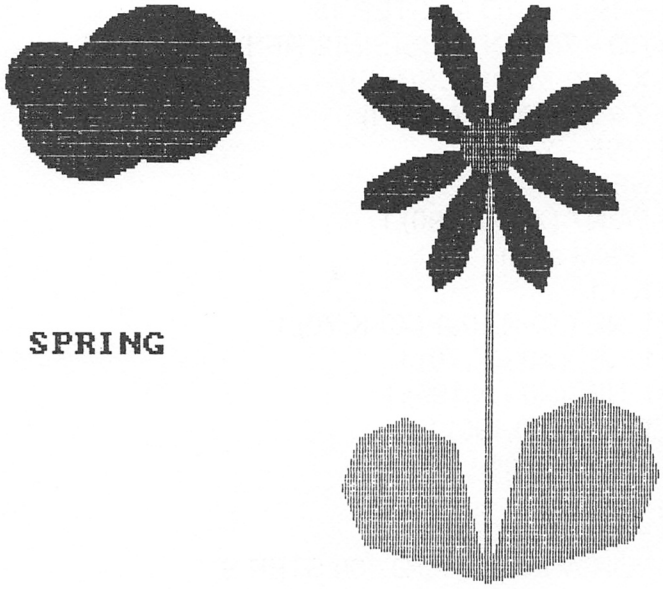
In particular, note that I defined SCREEN, COLOR and PALETTE. If you don't do this, the picture will be loaded as if it were defined for the current screen settings. You can get some surprising results.

Try BLOADing VALNTINE.PIC into the text mode screen (SCREEN 0.)

4. NOW IT'S YOUR TURN

Spring Is Here

Here's one more picture constructed with LINEs, CIRCLEs and a few math formulas, though on the surface it has little to do with math. It's called SPRING.BAS and is illustrated in Figure 6.5.



The program to generate "SPRING" is in Figure 6.6. Type it in and RUN it to see how it operates.

```
100 ' SPRING.BAS
110 KEY OFF:SCREEN 1,0:COLOR 1,1
120 CLS
130 RF = 3.14159/180
140 X0 = 186:Y0 = 60
150 ' Paint leaves
160 PSET (X0,199)
170 FOR I = 0 TO 90 STEP 15
180 RO = 65*SQR(ABS(SIN(2*RF*I)))
190 X = X0-RO*COS(RF*I)
200 Y = 199-RO*SIN(RF*I)
210 LINE -(X,Y),1
220 NEXT
230 PAINT (X0-5,190),1
240 PSET (X0,199)
```

```
250 FOR I=0 TO 90 STEP 15
260 RO=75*SQR(ABS(SIN(2*RF*I)))
270 X=X0+RO*COS(RF*I)
280 Y=199-RO*SIN(RF*I)
290 LINE -(X,Y),1
300 NEXT
310 PAINT (X0+5,190),1
320 ' Paint stem
330 K=1
340 LINE (X0-K,199)-(X0-K,Y0),1
350 LINE -(X0+K,Y0),1
360 LINE -(X0+K,199),1
370 PAINT (X0,190),1
380 ' Draw petals
390 S=8:N=4
400 PSET(X0,Y0),3
410 FOR ANGLE=0 TO 360 STEP S
420 RO=50*SIN(N*RF*ANGLE)
430 X=RO*COS(RF*ANGLE)+X0
440 Y=RO*SIN(RF*ANGLE)+Y0
450 LINE -(X,Y),3
460 NEXT
470 ' Paint petals
480 FOR I=0 TO 360 STEP 45
490 X=20*COS(RF*(I+22))+X0
500 Y=20*SIN(RF*(I+22))+Y0
510 PAINT (X,Y),3
520 NEXT
530 ' Paint center of flower
540 CIRCLE (X0,Y0),10,2
550 PAINT (X0,Y0),2
560 ' Paint cloud
570 CIRCLE (45,50),25,3
580 PAINT (45,50),3
590 CIRCLE (70,40),30,3
600 PAINT (70,40),3
610 CIRCLE (30,40),15,3
620 PAINT (30,30),3
630 LOCATE 16,4:PRINT"SPRING"
640 GOTO 640
```

Figure 6.6: SPRING.BAS

Orchestrating Nature

Don't be put off by the math formulas that draw the leaves and petals. After playing with them a little, you'll be able to adapt them to your own needs by putting them into a subroutine for future use. For now, there's another job for you to do.

Insert appropriate sounds and pauses into the program to slow the action of progressive steps and to accompany these actions with noises.

Here are some hints and background information on the program.

I used the equation of a two-leaved-rose lemniscate (= ribbon) from my dog-eared analytical geometry book to draw the leaves (lines 160-230 for the first, lines 240-310 for the second). This curve turns out to be more rounded than perhaps is desirable. But equations that generate pointy leaves are too complicated for the purposes of this book.

You might vary the sounds used in VALNTINE.BAS while generating the large heart for outlining the leaves and coloring them. Do one leaf, then pause dramatically before doing the second.

Similarly, wait before producing the stem. The sound for the stem, which is drawn very quickly, might be a quick upward sweeping sound, ending with a high-frequency NOISE. Then pause before doing the petals.

This rose will have eight petals. Since this curve is drawn in a periodic up and down motion, you might consider a sound that varies frequency in phase with this motion. You might make the frequency a function of Y in line 450 and insert the SOUND statement at line 455. Make the duration relatively long so you get a sustained note minus all the clicks and beats you get from short durations.

Consider accompanying the petal-painting action with random high-frequency sounds so you get a bell-like or tinkling effect for each petal, and slow down the action from petal to petal with a short pause in line 515.

Wait before painting the center of the flower. Consider using a NOISE like the one used for the little heart in VALNTINE.BAS.

Again, pause before drawing the cloud. Use your imagination to program sound for this action.

It would be a worthwhile exercise to carry out this sound project on your own. Everything has been done for you except actually writing the BASIC statements.

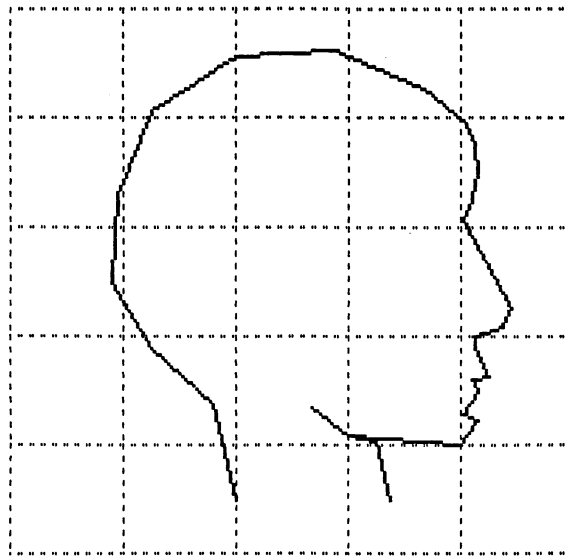
5. RUBBER-SHEET GEOMETRY

Do you have as much fun as I do on visiting the Hall of Mirrors in a fun house? My favorite is to stand in front of the mirror that makes me look skinny—and hope! Then there's the one that gives every girl the classic hourglass figure, and then some. And the one that transforms you into a basketball champion, or that grants you a certain likeness with the Goodyear blimp.

You might imagine imitating these painless physical deformations by drawing your outline on a sheet of rubber and then stretching the sheet in various ways. Or perhaps by modeling in clay.

The earliest formal studies of deformations of the human body are Albrecht Durer's (1471-1528). He and other artists of the time were interested in reproducing correct body proportions in their work.

Durer had a particularly geometric turn of mind, and among his studies are figures of human heads plotted on a square grid, like the diagram of Figure 6.7. Durer shows how a normally shaped head can be deformed to produce jutting chins, large faces and noses, pinched looks, and other head shapes—all by changing the proportions of the grid.



Closer to our time, Sir D'Arcy Thompson applied the same idea to the comparison of related animal and plant forms. Chapter IX of his *On Growth and Form* (first published in 1917) is filled with grids overlaid on bones of various animals, showing how they are related through continuous deformations of the grid. The correspondences seem so close for mammals that you have little trouble concluding they evolved from a common source.

For years I've been fascinated by Thompson's work. Although his figures have been published widely, and his ideas discussed in the scientific literature, little was done with his concepts until the computer became available. Now it's possible to actually illustrate the changes that might take place when molding a form like the head in Figure 6.7 into another shape.

Why don't you see for yourself? Figure 6.8 is a program that lets you explore the world of rubber-sheet geometry. Just type it and SAVE it. Then I'll tell you how to operate it, how it works, and how you might modify it for other experiments.

```
100 ' DEFORM.BAS
110 GOTO 890 ' Initialize
120 ' Coordinate transformation
130 X1=50+SX*
    (A*XX*XX+S*XX*Y+D*Y*Y+F*XX+G*Y+H)
140 Y1=10+SY*
    (Z*XX*XX+X*XX*Y+C*Y*Y+V*XX+B*Y+N)
150 RETURN
160 ' Draw grid
170 ' First the vertical
180 FOR I=1 TO 6
190 XX=X(1):Y=Y(I)
200 GOSUB 130
210 PSET (X1,Y1),0
220 FOR J=2 TO 6
230 XX=XX(J)
240 GOSUB 130
250 LINE -(X1,Y1),CLR1,,STYL
260 NEXT:NEXT
```

```
270 ' Then the horizontal
280 FOR J=1 TO 6
290 XX=XX(J):Y=Y(1)
300 GOSUB 130
310 PSET (X1,Y1),0
320 FOR I=2 TO 6
330 Y=Y(I)
340 GOSUB 130
350 LINE -(X1,Y1),CLR1,,STYL
360 NEXT:NEXT
370 RETURN
380 ' Draw figure
390 XX=FX(1):Y=FY(1):GOSUB 130 ' Calc new
    position
400 PSET (X1,Y1) ' Move to start position
410 FOR K=2 TO P ' Trace out figure
420 XX=FX(K):Y=FY(K):GOSUB 130 ' Calc next
    position
430 LINE -(X1,Y1),CLR2 ' Connect with line
440 NEXT ' Do next point
450 RETURN
460 ' Mainline
470 A$=INKEY$:IF A$="" THEN 470
480 IF A$="Q" OR A$="q" THEN
    CLS:PRINT"FINISHED":BEEP:END
490 IF A$="P" OR A$="p" THEN 1150 ' Save current
    picture
495 ' Select parameter to modify
500 PARAM=INSTR("ASDFGHJasdfghj",A$)
510 ON PARAM+1 GOTO
    520,540,550,560,570,580,590,
    600,610,620,630,640,650,660,670
520 PARAM=INSTR("ZXCVBNMzxcvbnm",A$)
530 ON PARAM+1 GOTO
    880,680,690,700,710,720,730,740,
    750,760,770,780,790,800,810
```

```
535 ' Modify parameters
540 A=A+.5*DEL:GOTO 830
550 S=S+DEL:GOTO 830
560 D=D+.5*DEL:GOTO 830
570 F=F+DEL:GOTO 830
580 G=G+DEL:GOTO 830
590 H=H+5*DEL:GOTO 830
600 JJ=JJ+DEL:GOTO 830
610 A=A-.5*DEL:GOTO 830
620 S=S-DEL:GOTO 830
630 D=D-.5*DEL:GOTO 830
640 F=F-DEL:GOTO 830
650 G=G-DEL:GOTO 830
660 H=H-5*DEL:GOTO 830
670 JJ=JJ-DEL:GOTO 830
680 Z=Z+.5*DEL:GOTO 830
690 X=X+DEL:GOTO 830
700 C=C+.5*DEL:GOTO 830
710 V=V+DEL:GOTO 830
720 B=B+DEL:GOTO 830
730 N=N+5*DEL:GOTO 830
740 M=M+DEL:GOTO 830
750 Z=Z-.5*DEL:GOTO 830
760 X=X-DEL:GOTO 830
770 C=C-.5*DEL:GOTO 830
780 V=V-DEL:GOTO 830
790 B=B-DEL:GOTO 830
800 N=N-5*DEL:GOTO 830
810 M=M-DEL:GOTO 830
820 ' Draw new grid & figure
830 CLS:GOSUB 180
840 GOSUB 390 ' Draw figure
845 ' Manage visual and active screens
850 APAGE=(APAGE+1) MOD 2
860 VPAGE=(VPAGE+1) MOD 2
870 SCREEN ,,APAGE,VPAGE
880 BEEP:GOTO 470 ' Go accept new changes
890 ' INITIALIZE
```

```
 900 CLEAR,,,32768! ' Double video buffer size
 910 KEY OFF:SCREEN 1,0,0,0:COLOR 0,1
 920 CLS:SOUND ON:BEEP OFF
 930 CLR1 = 1:CLR2 = 3 ' Grid & figure attributes (colors)
 940 PALETTE CLR1,14 ' Try various grid line colors
 950 STYL = &HCCCC ' Set dashed line style
 960 DIM XX(24),Y(24),FX(50),FY(50) ' Grid and figure
     tables
 970 FOR I = 1 TO 6 ' Load grid point table
 980 XX(I) = I-1:Y(I) = I-1
 990 NEXT
1000 P = 29 ' Number of points in figure
1010 FOR I = 1 TO P ' Load figure table
1020 READ FX(I),FY(I)
1030 NEXT
1040 DEL = .06
1050 ' Define initial distortion parameters
1060 A = 0:S = 0:D = 0:F = 0:G = 0:H = 0:JJ = 0
1070 Z = 0:X = 0:C = 0:V = 0:B = 0:N = 0:M = 0
1080 F = 1:B = 1 ' Square grid parameters
1090 SX = 40:SY = 35 ' Scaling factors
1100 APAGE = 1:VPAGE = 0 ' Active & visual page
     selectors
1110 GOSUB 180 ' Draw initial grid
1120 GOSUB 390 ' Draw initial figure
1130 SCREEN ,,1,0 ' Change active screen
1140 GOTO 460 ' Go accept changes to parameters
1150 ' Save picture on current VPAGE
1160 CLS:SCREEN ,,APAGE,APAGE ' Messages on
     active page
1180 PRINT:PRINT"Key name of picture"
1190 PRINT:PRINT" (8 or fewer characters)
1200 PRINT:INPUT NA$ ' Type file name to save
1210 NA$ = LEFT$(NA$,8) + ".PIC" ' Attach .PIC
     extension
```

```
1220 SCREEN,,VPAGE ' Make VPAGE active
1230 DEF SEG=&HB800 ' Start of video art
1240 BSAVE NA$,0,&H4000 ' Save picture
1250 SCREEN,,APAGE,VPAGE ' Reset screen
1260 GOTO 460 ' Go accept more changes
1270 ' Figure points—substitute here for other figures
1280 DATA 2,4.5,1.8,3.62,1.25,3.12,.9,2.5
1290 DATA .95,1.7,1.25,.95,2,.45,2.87,.4
1300 DATA 3.68,.75,4.05,1.05,4.12,1.25,4.15,1.5
1310 DATA 4.1,1.75,4.03,1.95,4.45,2.75
1320 DATA 4.37,2.92,4.12,2.99,4.12,3.12
1330 DATA 4.25,3.37,4.1,3.4,4.15,3.51,4.02,3.7
1340 DATA 4.14,3.77,4,4,3,3.91,2.67,3.67
1350 DATA 3,3.91,3.25,3.97,3.37,4.5
```

Figure 6.8: DEFORM.BAS

6. PREPARING INPUT FOR DEFORM.BAS

When you RUN DEFORM.BAS, a grid of squares is drawn on the screen, followed by the tracing of a figure. For the illustration, you'll see the human head depicted in Figure 6.7. The grid is drawn automatically by the program. You must supply the figure in a set of DATA statements beginning in line 1280.

I prepared the head you see on the screen by first sketching it on a piece of graph paper. I used paper with eight grid lines per inch. Ten is better and doesn't force you to think in eights when you want to read the coordinates of a point. (Trace the figure if your freehand sketching is too primitive.)

Next, I traced over the sketch with bold, straight line segments. You can use a ruler or straightedge if you like. The purpose of this step is to define the endpoints of a series of straight lines to be generated by the LINE command.

Then, starting with a free end of the figure, the lower back of the neck in the illustration, I read off coordinate pairs (x,y—i.e., horizontal and vertical position) of successive end points of line segments. The (0,0) coordinate (origin) is the upper left corner of the grid; (5,5) is the lower right corner.

With this list of coordinates, I prepared the DATA statements starting at line 1280. They move along in successive pairs of coordinates around the figure. The first few are as follows: (2, 4.5), (1.8, 3.62), (1.25, 3.12), (.9, 2.5). Check them against line 1280.

If the figure has forks (e.g., where the neck and jaw break away from the chin), go to the nearest endpoint and double back to complete the figure. For figures with two or more unconnected parts, you'll have to modify the program to draw them properly.

7. DISTORTING THE FIGURE

Now comes the fun part. There are twelve things you can do to transform the character of the picture. They are listed in Figure 6.9. Keep referring to this figure as we try the operations.

Up, Down and Sideways Movements

The simplest operations are moving the figure up, down and sideways. Hit N and watch the screen. In about ten seconds— sorry it's so slow—you'll see the figure jump up a small amount, and you'll hear a beep to signal the completion of the change. You don't see the figure being drawn, because it's done on another page of video memory. When the new picture is ready, the other page is switched to the screen. It's easier to see the effect of the change this way.

Try hitting N a couple more times and see the figure move up above the top edge of the screen.

CAVEAT: Be careful about hitting the keyboard while the program is creating a picture in the alternate page. These strokes are stored in the type-ahead buffer and will be put into action eventually. You may not like what you did accidentally. I decided not to flush the buffer so that I could deliberately load it with several commands (e.g., to shrink the figure to screen size) while I did something else for the minute or two that it takes to execute them.

KEYSTROKE*	EFFECT
Centering	
N	Move figure up/down
H	Move figure left/right
Dilation	
vertically	
B	Expand/shrink (uniformly)
vertically	
C	Expand/shrink (non-uniformly)
horizontally	
F	Expand/shrink (uniformly)
horizontally	
A	Expand/shrink (non-uniformly)
Shear	
V	Shift right edge up/down (uniformly)
Z	Shift right edge up/down (non-uniformly)
G	Shift bottom edge left/right (uniformly)
D	Shift bottom edge left/right (non-uniformly)
Rotation	
X	Compress/expand right edge
S	Compress/expand bottom edge

* Holding <Shift> down as you strike a letter moves you in the opposite direction

Figure 6.9: Distortion Operations

To move the figure downward, hold the Shift key and strike N. It'll take a few of these strokes to reposition the figure.

The Shift key is used with all these operations to indicate movement in one direction. The operation key alone indicates movement in the opposite direction.

Right and left shifts of the figure are done with Shift-H and H, respectively. Try a couple of moves.

Dilation

Suppose you want to make the figure smaller to keep it within the bounds of the screen. Type F and see the figure shrink toward the left. Of course, a <Shift-F> will stretch the figure toward the right.

The F operation shrinks and expands the figure uniformly. That is, the space between grid lines is changed proportionally. With the A operation, the shrinking or expansion can be done non-uniformly. To see what I mean, strike A a couple of times. Just the opposite effect is brought about with <Shift-A>.

The V and Z commands are the vertical movement counterparts of F and A, respectively. Try them to get the feel.

Shear

So far we've done the simplest distortions. To keep matters simple, clean the slate.

Type: Q (Quit)
Type: RUN <ENTER>

There, we're back at the beginning.

Strike <Shift-G> a couple of times. We've given the poor fellow something of a jutting chin, or underbite. Notice that the shear of the bottom edge, relative to the top, is done uniformly. This figure is practically identical to one of Durer's drawings.

By using the <Shift-D> command, we can accelerate the movement. Strike <SHIFT-D> and see. Of course, G and D move you in the opposite direction.

As before, V and Z do the equivalent shearing operations in the vertical direction.

Mess around with the shear operations just to see their effects. Don't go mixing up other operations with them until you've fixed the shear operations in your mind.

Rotation

Rotations are the most difficult to intuit. We'd best get a clean slate to try them. (Type: Q and RUN <Enter>.)

Strike <Shift-S> once. You can see that the bottom edge stretches toward the right without curving or modifying the top and left edges. It's like the F command, except that the amount of stretch depends on the distance from the top edge.

Try the opposite. Type S twice. The poor fellow has a bit of an overbite now.

The X command is D's counterpart for the vertical axis.

Play with the D and X commands to get the idea of the distortions they produce.

8. SOME EXPERIMENTS

Keystroke Notation

Now that you've got the general idea, let's take up more complex movements that involve combinations of the individual operations. To save time, an abbreviated notation will be used to specify the steps one can take to build a certain picture. If you are to strike a given key, I'll just write the name of the key (e.g., D or F). If you are to hold <Shift> down while striking the key, I'll write a small s in front of the letter (e.g., sD or sF).

If I write a series of strokes separated by plus signs (+), e.g., D+sF+G, then you are to strike D, followed by <Shift-F>, followed by G.

If I write a series of strokes in parentheses, preceded by a positive integer, e.g., 3(D+sG), then you are to strike the pair D and sG three times, in that order. If there's only one stroke to be done several times, I'll omit the parentheses, e.g., 3D instead of 3(D).

If you're a little confused, don't worry. These rules will clear up as you go along.

Man of the Future?

Let's start afresh. Quit DEFORM.BAS and RUN it again to get the square grid picture. We'll begin with something easy to get used to the notation. Try this:

2(A+C) (Shrink lower right toward upper right, non-uniformly)

Just do the keystrokes in this order: A, C, A, C.

You might say this series of deformations chronicles the evolution of man as his brain case expands to hold his evolving brain.

Square Sail Forms

Now watch what happens as you do this series of operations:

9sG + 4D (Generate square sail form)

You can get the reverse of this picture by doing 9G + 4sD. You've got to start with the square grid again.

The equivalent changes in the vertical can be obtained with these strokes.

9V + 4sZ (Generate shield form)

Again, by symmetry, the reverse is generated by doing 9sV + 4Z.

Diamond Forms

Let's start again, and pull on the poor fellow's chin.
Type: Q and RUN.

3(sD + sZ) (Generate spearhead shape)

Of course, you know you can shrink the figure with F and B to bring it back onto the screen.

What happens if, starting from scratch, you do 3(D + Z)?

If, instead of the non-uniform shearing operations, we use the uniform ones, we'll get straight-line figures. Do this:

3(sV + sG) (Generate diamond shape)

What would happen if you did 3(V + G) instead?

These diamond forms are useful for studying the effects of stretching and shrinking axial symmetric shapes like fish. But in these cases you should prepare the input so that the axis of symmetry runs along the diagonal—i.e., the head in the upper left and the tail in the lower right.

Foldings

Now for some more complicated motions. Try this one.

```
5(S+X)+5(sH+sN)    (Fold lower right back to and beyond
                    upper left corner)
```

If you had done 3(D+Z) a few more times, you'd have completed the folding and gone beyond as was done here.

With some of these deformations, you'll probably want to rescale the figure with B and F, and then recenter it with N and H.

Here's a process similar to the last one, but headed horizontally and down.

```
4(S+sX)+7N    (Fold lower right out and past lower left)
```

The 7N just gives you a chance to see what's happened below the bottom of the screen. Keeping pictures within the screen is a bit of a bother. You may want to move down even farther.

Try some additional possibilities of your own invention. Make notes of what you discover so you can exploit them later.

The Monkey's Uncle

This subsection might have been titled "Man's Roots," "The Origin of Man" or something equally Darwinian. My inspiration comes from the figures in Thompson's *On Growth and Form*, which show the relationships between the shape of the human skull and those of the chimpanzee and the baboon. I thought it would be fun to see if I could deform the human head on the screen to one approaching a simian prototype. The best result I obtained is shown in Figure 6.10—a figure more like a dog than a chimp. But the smaller brain case and large jaw is the general effect sought.

To give you a feel for the process of reshaping a figure, try this shortened reconstruction of the process I went through to obtain my monkey's uncle.

```
10F+10B+8sA+10F+3sN+Z+2D+4sS
  +10F+G+4sC+10B+3sC+3V+3sN+2sX+8B+2sH
```

CAVEAT: Slow down. Be careful not to overload the type-ahead buffer. It'll scream at you if you do, and you'll probably lose count of the keystrokes. And if you do lose count, you won't end up with the same figure I did.

9. SAVING A FIGURE ON DISK

If you want to save your handiwork to show others, here's how. Is the monkey's uncle still on the screen? Okay,

Type: P (Save Picture)

You're asked to name the file you want to save it in.

Type: BABOON <ENTER>

The disk drive's red light goes on and you hear the file being recorded. That's it.

Let's verify that BABOON.PIC (the extension .PIC is added by DEFORM.BAS) is on the disk.

Type: Q (Quit DEFORM.BAS)
Type: FILES <ENTER>

Do you see it?

10. READING BACK YOUR PICTURES

To show off your hard-won pictures, you can do the following. Clear program memory (NEW) and type in the following little program.

```
100 SCREEN 1,0
110 BLOAD"BABOON.PIC"
120 GOTO 120
```

Of course, you substitute the name you want for BABOON.PIC. Line 120 just keeps you from ruining the picture with the "Ok" prompt.

11. DEFORM.BAS PROGRAM NOTES

Program Layout

When you RUN DEFORM.BAS, the first thing it does is jump from line 110 to 890 to do the one-time setup actions. Because of the way BASIC operates, it's important to reserve the low-order line numbers for subroutines that are used most often. Figuring out the new positions for the various plotted points (coordinate transformation, line 120), and drawing the grid (line 160) and figure (line 380), are all at the front of the program. The mainline (line 460) is sandwiched between these subroutines and the initialization activity.

Initialization

First define the video space. We have to double the size of the graphics buffer so we'll have two pages between which we'll switch back and forth. Normally you're given a 16K buffer. To double it, we use the CLEAR command in line 900. With the space allocated, we can define the graphics screen type (medium resolution) and which pages will be active and visual—page O to start with (line 910).

NOTE: It is essential to follow this discussion with your BASIC manual at hand. Read slowly and look up each of the commands discussed. Some of them have a variety of purposes. To put them in perspective, and even understand what I'm talking about, you must relate our application of them here to these other functions.

I did a little experimenting with colors by assigning various attributes and palette combinations in lines 930 and 940. You might try some youself to see if there's a more satisfying combination of colors for you.

To avoid making the grid lines solid, a dashed-line style is defined in line 950, to be used in lines 250 and 350 when the grid is drawn. You might try some experiments with other styles of lines (e.g., all dots, or dot-dash combinations).

In line 960, space is allocated for the grid points and the points of the figure you supply as input. You can make them larger if you need more space for a figure, or want to use a finer-grained grid.

Lines 970-990 load the grid points into the grid table. Lines 1000-1030 do the same for the figure. You've got to specify the number of points, P, in the figure at line 1000 so that the subsequent FOR...NEXT loop knows when to stop READing DATA statements (line 1020).

The amount by which the figure is permitted to change from step to step is specified by DEL in line 1040. For instance, if you double its size, the amount you can shift the figure's center is doubled each time you use the N or H command. This is a parameter with which you might experiment.

Transforming Coordinates

The mechanism for distorting the grid and the figure is a pair of equations that relate the original square grid to one with curved grid lines. The equations I use in DEFORM.BAS are quadratic forms. They are spelled out in lines 130 and 140. The square-grid coordinates are XX and Y. The transformed coordinates are X1 and Y1.

Each equation involves six coefficients—A, S, D, F, G, H; and Z, X, C, V, B, N, respectively. I picked these letters so they would correspond to successive letters on rows of the keyboard. In this way I could literally feel them in the home position of my left hand, and easily know which term of the quadratic form I was dealing with. Since X is one of the coefficients, I used XX to represent the horizontal coordinate.

It turns out that if you let all the coefficients be 0 (except F and B, which you set to 1) then the transformed grid is identical with the original, square grid. These initial values are set in lines 1060-1080.

In addition, there's a scaling and centering problem to consider—fitting the grid to the screen. The scale factors (SX and SY) are set in line 1090 and are used in the transformation equations in lines 130 and 140. I picked values that seemed to produce a square grid of reasonable size. (Remember the aspect ratio problem?) Centering is done by the constants 50 and 10 in the transformation equations at lines 130-140.

Drawing the Transformed Grid

The next initialization step (line 1110) is to draw the grid using the subroutine at 180. This subroutine is also used in the mainline. The grid is drawn in two steps—first the horizontal lines, then the vertical—by almost identical routines. The coordinate transformation routine at line 130 is used to change the original grid coordinates to the new positions (i.e., lines 200 and 240). The appropriate changed positions are then connected by straight-line segments (i.e., line 250). It's a question of picking a point along the left edge or top edge (lines 210 and 310), and then zipping horizontally or vertically to form the grid lines.

Drawing the Figure

After the grid is drawn, the figure is superimposed. Initially this is done in line 1120 using the subroutine at 390. The process is similar to that of drawing the grid. The starting position is calculated (lines 390-400), then each successive position of the figure is transformed (line 420), and a line is drawn to it from the previous point (line 430).

The Mainline

Just before jumping to the mainline, the active and visual page indicators are set (line 1100), and the screen setting is changed. Page 0 remains visible and page 1 is made active so we can build the next transformation on it while we're still viewing the original figure.

The mainline, which starts at line 470, is similar to keyboard input loops we've used before, though perhaps a little more involved. Line 470 looks for you to strike a character on the keyboard. If you hit Q, the program is over (line 480). If you hit P, you're sent off to line 1150 to save the current picture.

Generally you'll strike one of the parameter keys to change the coefficients of the quadratic equations. There are two cases. Coefficients of the equation for X1 (line 130) are detected in line 500; those of the equation for Y1 (line 140) are detected in line 520. If the variable PARAM turns out to be zero, you struck an undefined key. You're BEEPed at (line 880) and sent back to line 470 to try again.

Lines 510 and 530 take care of selecting how to handle the changed coefficient. Depending on whether you typed a capital or lower-case letter, you'll add or subtract a multiple of DEL from that coefficient. You can see all the possibilities delineated starting at line 540. I add/subtract only half of DEL to/from coefficients of squared terms to compensate a bit for the large change these terms introduce. I accelerate the constant terms by adding/subtracting five times DEL, so changing the center of the figure moves it along more quickly. You might do some experimenting here too.

Once the coefficients have been modified, the grid and the figure are replotted (lines 830-840), but you don't see the new picture until the active and visual pages are flipped (lines 850-870). I felt it was important to BEEP when the change is made, thereby letting you know you can make another change.

Saving a Picture

If you enter a P, you are sent off to line 1160 to save the current screen image. You are switched to the active screen so you can type in the name of the file you want to save (line 1200). A .PIC extension is added to the name (line 1210). If the name contains more than eight characters, the machine recognizes only the first eight.

Now the visual page is made active (line 1220). This is important because the BSAVE command saves only from the active page. The standard formula for saving video memory is the commands at lines 1230 and 1240.

Finally the active page is switched back and you're sent back to line 460 to make additional transformations.

Other Transformations

If you know what you're doing, it's fairly easy to change the transformation equations in lines 130-140 to other mathematical forms. The quadratic is fairly limited in its possibilities. The monkey's uncle would probably have been more successful if the equations contained logarithmic or exponential terms.

12. REVIEW, REVIEW, REVIEW

This chapter contained some rather advanced graphics programming ideas, but you have seen how the simple techniques we used in earlier chapters arose again in these more difficult programs. Still, as programming techniques became more and more involved, you may have had some difficulty with them.

The fact is that if you want to do interesting things with a computer, there's a lot of stuff to know. Although your interest may be graphics, much of what you should be familiar with is equally applicable to inventory control or accounting or other application areas. Manipulating data by computer is done pretty much the same way in all applications.

Before going on, be sure you understand the BASIC commands used to develop the examples in this chapter. We're building on what we've learned thus far, so you want to be sure you're prepared for what lies ahead.

Figure 6.11 lists the BASIC commands used in this chapter. Read over the list to be sure you understand what each command does. Look up problem commands in the BASIC manual. Then go back over the chapter and try introducing your own. You'll be surprised what you can learn and how satisfying it can be when you get the sense that you're controlling the PCjr yourself. You may want to delve into other books on BASIC to get other perspectives on the language and what you can do with it.

BASIC COMMAND	FUNCTION
ABS	Change a number to its positive value
BEEP	Make speaker beep

BLOAD	Reload a section of memory saved with BSAVE
BSAVE	Write a portion of memory to disk or cassette
CIRCLE	Draw an ellipse or arc at the specified point of designated color and aspect
CLEAR,,,32768	Set up two pages for medium-resolution graphics buffer
CLS	Clear screen
COLOR	Set field and palette colors
COS	Calculate trigonometric cosine
DATA	Store numeric or character data for access by READ command
DEF SEG	Define the beginning of a block of memory
DIM	Set up a table or array
FOR...NEXT	Repeat a sequence of commands
GOSUB	Take next command from line number, and RETURN to this point when finished
GOTO	Take next command from line specified
IF...THEN	Make decision on program flow
INKEY$	Read character from type-ahead buffer
INPUT	Transfer keyboard data to a variable
KEY OFF	Toggle screen prompts off
LINE	Draw a line between specified points; build a box; specify style of line.
LIST	List program memory to the screen

LOCATE	Position cursor on screen for printing text
NOISE	Generate various noise patterns through the external speaker
ON...GOTO	Computed GOTO multiple line numbers
PAINT	Fill closed figure with specified color
PALETTE	Reassign the color attributes of a palette
PLAY	Sound the notes and execute the commands in a music command string
PRINT	Print text on the screen
PSET	Print dot at specified location and color on the graphics screen
READ	Transfer values from a DATA statement to specified variables
RENUM	Renumber the lines of a BASIC program
RND	Generate random number between 0 and 1
SCREEN 1,0	Set medium-resolution graphics screen
SCREEN,,apage, vpage	Select active and visual pages of video buffer
SIN	Calculate trigonometric sine
SOUND	Play a pitch of specific duration
SQR	Take the square root of a number

Figure 6.11: Summary of Commands in Chapter 6

CHAPTER 7
TWO AND THREE-PART HARMONY

Objective:

This chapter will further explore PCjr's musical capabilities and demonstrate how to program harmonized tunes.

1. THE KEY IDEA IS SIMPLE

So far we've only considered how to transcribe single-voice tunes using the PLAY command, but PLAY permits chords and harmonies to be included in a tune. Here's how.

Three-Voiced Chords

You need SOUND ON to create simultaneous voicings with PLAY.

Type: SOUND ON:BEEP OFF <ENTER>

Now for some chords. First we'll anchor them at the octave of middle C.

Type: PLAY "O3","O3","O3" <ENTER>

Notice there are three O commands specified, one for each voice. The voices are labeled from left to right, 0, 1 and 2, respectively. Now for some notes.

Type: PLAY "C","E","G" <ENTER> (C Major)
Type: PLAY "C","E-","G" <ENTER> (C Minor)
Type: PLAY "C","E-","G-" <ENTER> (C Diminished)
Type: PLAY "C","E","B-" <ENTER> (C Dominant 7th)
Type: PLAY "C","F","A" <ENTER> (F Major)

Let's try a sequence of them. To avoid confusion, write a table of the chord sequence and the assignment of notes to voices.

Voice 0	Voice 1	Voice 2	Chord
O3C	03E	03B-	C Dominant 7th
C	F	A	F Major
<<B-	D	A-	B-flat Dom 7th
E-	<B-	G	E-flat Maj
A-	>C	G-	A-flat Dom 7th
D-	<A-	F	D-flat Maj
G-	B-	E	G-flat Dom 7th
<B	B	D#	B Maj 3rd

Note how PCjr music notation has been used to indicate changes of octave within a voice, and how the starting octave has been reset to O3 to be sure it's properly selected. As you see, Voice 0 is the bass, Voice 1 the alto, and Voice 2 the soprano. It doesn't make any difference which voice you assign to which part. To hear how this noodling sounds, write each part as a separate tune, but not in Direct Mode. There's too much typing at risk. Clear memory (NEW) and enter this line:

```
100 PLAY "O3CC<<B-E-A-D-G-<B",
         "O3EFD<B->C<A-B-B",
         "O3B-AA-GG-FED#"
```

Notice that each string of letters between quotes corresponds to a column (voice) in the table we set out above.

Now put in the SOUND ON line:

```
90 SOUND ON:BEEP OFF
```

Now run the program. This is the classic cycle of fifths that runs through all of Occidental music. I was going to say "western" music, but you might have thought I was talking about the cowboy variety.

Easier Manipulation of Music Fragments

Though there were only eight notes in each voice in this last example, you can tell immediately that line 100 is not the way to write polyphonic music (several voices supporting a melody). In fact, there is an easier way for some purposes. Let's test your editing skills as we move into it.

Clear the screen (<Ctrl-L>, and LIST 100 to put line 100 at the top of the screen. Move the cursor up to PLAY command and change the line number to 200. *Don't* hit <Enter> yet.

Now type the characters A$= over the PLA in PLAY and delete the Y with the Del key.

Next, move the cursor right to the first comma and type <Ctrl-Fn-End> to erase the tail end of the line.

Now you can hit <Enter> to register the new line 200. What you've done is to assign the string of notes for Voice 0 to the pigeonhole labeled A$. (The dollar sign, of course, tells PCjr that what's in the pigeonhole is a string of ASCII characters, not a number.)

LIST the growing program to see if the music string in line 200 matches Voice 0 in line 100.

Using the editing controls, pull out the music strings for Voice 1 and Voice 2, and put them in pigeonholes labeled B$ and C$ at lines 210 and 220, respectively. You should end up with program lines that look like this:

```
200 A$="O3CC<<B-E-A-D-G-<B"
210 B$="O3EFD<B->C<A-B-B"
220 C$="O3B-AA-GG-FED#"
```

To all this, add the following line:

```
230 PLAY "XA$;","XB$;","XC$;"
```

Don't omit the semicolons. Okay, RUN the program. You should hear the sequence of chords twice.

The X is a special PCjr music language character. It says to treat the characters following it as the label of a music string that is stored elsewhere. The semicolon simply specifies the end of the label so that PCjr can switch back to interpreting the characters as music language operations.

While these variables are still active, try these Direct mode commands:

```
Type: PLAY "XA$;","","XC$;" <ENTER>
Type: PLAY "","XB$;","XC$;" <ENTER>
Type: PLAY "XA$;","XB$;","" <ENTER>
```

Kind of fun to hear the voices by twos.

2. ROUND AND ROUND

Play Rounds on PCjr

Perhaps the earliest personal experience that children have with harmonizing is the singing of rounds, songs with two or more voices overlapping a melody. Obvious examples are tunes such as "Three Blind Mice," "Row, Row, Row Your Boat" and "Frère Jacques."

Figure 7.1 is a program called ROUNDS.BAS, which permits PCjr to play a round in three parts.

ROUNDS.BAS consists of three sections. First there are a few lines that set the initial conditions (lines 10-70). Then there's the round itself—"Frère Jacques" in the figure (lines 100-200). It is represented as a series of strings, loaded into the array B$, measure by measure. Finally, there's the round-playing controller (lines 1000-1160). This is what does all the work of selecting which measures to PLAY together.

```
 10 ' ROUNDS.BAS
 20 ' Play musical rounds
 30 SOUND ON:BEEP OFF ' Select external speaker
 40 BARS=50:DIM B$(BARS) ' Define array to store
    measures
 50 I=0:J=0:K=0:A$="":M=2:T=120 ' Defaults
 60 V1=10:V2=8:V3=6 ' Voice volumes
 70 ' Insert tune from 100-999
100 ' FRERE JACQUES
110 BARS=8:M=2:T=150 ' No. of measures, offset &
    tempo
120 B$(0)="P1" ' Dummy measure of rests
130 B$(1)="O3L4CDEC" ' First measure
140 B$(2)="O3L4CDEC"
150 B$(3)="O3L4EFL2G"
160 B$(4)="O3L4EFL2G"
170 B$(5)="O3L8GAGFL4EC"
180 B$(6)="O3L8GAGFL4EC"
190 B$(7)="O3L4C<G>L2C"
200 B$(8)="O3L4C<G>L2C" ' Last measure
1000 ' Round controller
1010 ON KEY(1) GOSUB 1160 ' Set <Fn-1> stop playing
1020 KEY(1) ON ' Activate function key 1
1030 PLAY "t=T;","t=T;","t=T;" ' Set tempo
1040 I=I+1 ' Increase measure count for voice 1
1050 IF J=0 AND I<1+M THEN 1070 ' Voice 2 starts at
    offset
```

```
1060 J=J+1 ' Increase measure count for voice 2
1070 IF K=0 AND I<1+2*M THEN 1090 ' Voice 3 starts
     at offset
1080 K=K+1 ' Increase measure count for voice 3
1085 ' Tests whether to stop playing or not
1090 IF K>BARS THEN IF V2=0 AND A$=CHR$(27)
     THEN END ELSE K=1
1100 IF J>BARS THEN IF V1=0 AND A$=CHR$(27)
     THEN V2=0:J=1 ELSE J=1
1110 IF I>BARS THEN IF A$=CHR$(27) THEN
     V1=0:I=1 ELSE I=1
1130 ' Here's where the objective is fulfilled
1140 PLAY "v=v1;XB$(I);","v=v2;XB$(J);
          ","v=v3;XB$(K);"
1150 GOTO 1040
1160 A$=CHR$(27):RETURN ' Executed when <Fn-1>
     struck
```

Figure 7.1: ROUNDS.BAS, for Playing Rounds on PCjr

When you have keyed in the program, LLIST it, correct the typos, and SAVE it under the name ROUNDS.BAS. Then RUN it. You can stop its playing with <Fn-1>.

The Playing Defaults

Consider turning off Voices 2 and 3 so you can check aurally to make certain you keyed the music correctly. With multiple voices playing, it's difficult to hear precisely where a sour note is occurring. To turn off these voices, set V2 and V3 to 0 in line 60. Restore their values later.

Note that by using different values for V1, V2 and V3, you can control how the voices dominate.

Although the default number of measures (or bars) is 50, you must redefine it to its exact value in the music, as in line 110. Similarly, you must also define the offset, M—the number of measures to delay before the next voice starts. And don't forget the tempo, T, at which you want the tune played. If you care to, you may also redefine the voice volumes, V1, V2 and V3, in the tune near line 110.

Pulling a Tune from a Program

We can extract the tune "Frère Jacques" from the program so that it can be used for other things. At the same time, we'll doctor up ROUNDS.BAS so that it can work with other tunes. With the *Test Programs* diskette in the disk drive,

Type: LIST 100-200,"BRO-JOHN" <ENTER>

This variation of LIST writes the program lines of "Frère Jacques" to the file BRO-JOHN.BAS instead of to the video screen. You can check whether it got there by typing FILES <ENTER>.

The Generic Form of ROUNDS.BAS

Now let's purge ROUNDS.BAS of the tune "Frère Jacques" so that other tunes can be inserted.

Type: DELETE 100-200 <ENTER>

Use LIST to check that the tune is deleted.

ReSAVE ROUNDS.BAS in its generic form. That is,

Type: SAVE "ROUNDS.BAS",A <ENTER>

Now we have the tools for generating and playing additional rounds. So how about trying another tune.

Row, Row, Row Your Boat

Figure 7.2 is the music for this old chestnut. Clear memory and type it in.

```
100 ' ROW, ROW, ROW YOUR BOAT
110 BARS=8:M=2:T=110 ' No. of measures, offset &
    tempo
120 B$(0)="P2" ' Dummy measure of rests
130 B$(1)="O3 L4D D"
140 B$(2)="O3 L8D. L16E L4F#"
150 B$(3)="O3 L8F#. L16E L8F#. L16G"
160 B$(4)="O3 L2A"
```

```
170 B$(5)="O4 L12DDD <L12AAA"
180 B$(6)="O3 L12F#F#F# L12DDD"
190 B$(7)="O3 L8A. L16G L8F#. L16E"
200 B$(8)="O3 L2D"
```

Figure 7.2: ROWBOAT.BAS—"Row, Row, Row Your Boat"

When you finish typing "Row, Row, Row Your Boat," LLIST it and correct any typos you may have made. Then SAVE it as an ASCII file this way:

Type: SAVE "ROWBOAT",A <ENTER>

When you've finished SAVing, reLOAD ROUNDS.BAS. That is,

Type: LOAD "ROUNDS" <ENTER>

Next, we're going to insert ROWBOAT into ROUNDS with another trick:

Type: MERGE "ROWBOAT.BAS" <ENTER>

Use LIST to verify that ROWBOAT was inserted into ROUNDS correctly. If it's okay, RUN the program.

When you tire of playing this tune, SAVE the combined program under a different name so you can play it again without merging ROWBOAT with ROUNDS each time you want to hear it.

Dona Nobis Pacem

One of the loveliest of rounds is the Austrian traditional "Dona Nobis Pacem"—"Give us peace." It consists of three parts of eight bars each. Each voice sings the three parts in series before returning to the first part to repeat the process. The first voice sings the first part through. As the second part is taken up, the second voice starts the first part. Similarly, as the second voice starts the second part, the third voice starts the first part. And, of course, the first voice is starting the third part.

The whole round is given in Figure 7.3, ready for you to key it into memory.

```
100 ' DONA NOBIS PACEM
110 BARS=24:M=8:T=120
120 B$(0)="P4P4P4"
130 B$(1)="O3MLL8FMNCL2A"
140 B$(2)="O3MLL8GMNCL2B-"
150 B$(3)="O3MLL4AMNGF"
160 B$(4)="O3L4FL2E"
170 B$(5)="O4MLL4DL8CMN<B-MLAMNG"
180 B$(6)="O4MLL4C.MN<L8B-L4A"
190 B$(7)="O3MLL8AGL4FMNE"
200 B$(8)="O3L2F."
210 B$(9)="O4L2C."
220 B$(10)="O4L2C."
230 B$(11)="O4MLL4CMN<B-A"
240 B$(12)="O3L4AL2G"
250 B$(13)="O4L4DL2D"
260 B$(14)="O4L4CL2C"
270 B$(15)="O4MLL8C<B-L4AMNG"
280 B$(16)="O3L2F."
290 B$(17)="O3L2F."
300 B$(18)="O3L2E."
310 B$(19)="O3MLL4F.MNL8GMLAMNB-"
320 B$(20)="O4L4CL2<C"
330 B$(21)="O3L4B-L2B-"
340 B$(22)="O3L4AL2A"
350 B$(23)="O3MLL8EG>L4CMN<C"
360 B$(24)="O3L2F."
```

Figure 7.3: PACEM.BAS—The Round "Dona Nobis Pacem"

Once again, SAVE this tune as an ASCII file under the name PACEM.BAS. Then LOAD the generic program ROUNDS.BAS into memory, and MERGE PACEM.BAS into it. You're ready to RUN it and SAVE it under a new name, say PEACE.

Try your hand at preparing the music for, and playing, other rounds of your acquaintance. You should be able to adapt BLNDMICE.BAS to this new format. Or how about trying "O How Lovely Is the Evening." Then there's "The Donkey," "Scotland's Burning," "Peter Piper" and "Hey, Ho! To the Greenwood". You might even try "The Old Grey Mare," "She'll Be Coming Round the Mountain," and other old-timers, to see if they work out as rounds.

Many books of folk songs and traditional melodies contain rounds you can adapt for the PCjr. A particularly good one is Mary C. Taylor's *Rounds & Rounds* (Hargail Music Press, 1959).

I'll leave you with one more short round in Figure 7.4, the seventeenth-century "Banbury Ale."

```
100 ' BANBURY ALE
110 BARS=4:M=1:T=150 ' No. of measures, offset &
    tempo
120 B$(0)="P1" ' Dummy measure of rests
130 B$(1)="O4 L4C L8CC <L2B"
140 B$(2)="O4 L4C <C L2G"
150 B$(3)="O4 L8C D E C L4D. L8D"
160 B$(4)="O3 L8G G L4G L2G"
```

Figure 7.4: BANBURY.ALE (17th century)

NOTE: Don't forget the filename extension when you MERGE Figure 7.4 into ROUNDS.BAS, i.e.,

Type: MERGE "BANBURY.ALE" <ENTER>

3. MAKE YOUR OWN PLAYER PIANO

Figure 7.5 is a variation on ROUNDS.BAS that plays up to three parts of precoded music. You should have little difficulty reading the program at this point. Just type it in so we can use it.

```
 10 ' THREE-PART PLAYER
 20 SOUND ON:BEEP OFF ' Select external speaker
 30 T=120:V1=10:V2=10:V3=10 ' Set defaults
 40 BARS=100:DIM B$(BARS,3) ' Define array to store
    music
 50 ' SANDWICH VOICE PARTS AFTER HERE AND
    BEFORE LINE 1000
100 ' Test tune
110 T=120:V1=8:V2=8:V3=8 ' Set tempo and volume
120 BARS=1 ' Number of measures
130 B$(1,1)="o2cdefgab>c"
140 B$(2,1)="o3cdefgab>c"
150 B$(3,1)="o4cdefgab>c"
1000 ' The section that plays the tune
1010 FOR I=1 TO BARS ' Count measures
1020 A$=INKEY$ ' Check keyboard for tempo change
1025 ' Test for change in tempo signal
1030 IF A$="s" THEN T=T-10:IF T<32 THEN T=32
1040 IF A$="S" THEN T=T+10:IF T>255 THEN T=255
1050 PLAY"mbt=T;","mbt=T;","mbt=T;" ' Set tempo
1055 ' Play a full measure in three parts
1060 PLAY "V=V1;XB$(I,1);","V=V2;XB$(I,2);
          ","V=V3;XB$(I,3);"
1070 NEXT I
1080 PRINT"HIT <SPACE> TO REPEAT; <Esc> TO
     QUIT"
1090 A$=INKEY$:IF A$="" THEN 1090
1100 IF A$=CHR$(27) THEN PRINT"FINISHED":END
1110 IF A$=" " THEN 1010
1120 BEEP:GOTO 1090
```

Figure 7.5: 3-PART.BAS—Three-Part Player

Lines 100-150 are a test tune to check out the program—three C Major scales in parallel octaves. Delete these lines when you've used them, then SAVE the program under the name 3-PART.BAS.

Lines 110-120 set substitutes for the tempo and volume defaults, and specify the exact number of measures of the tune.

Lines 130-150 typify how each measure of music is coded in the two-way array called B$(measure,voice). They are substituted in the PLAY command of line 1060.

If you wish to override the tempo (T) and volume (V1, V2, V3) settings, include them in the B$ for each measure. But be sure the tempos for all the voices are the same. Otherwise, they'll clash when played together. In fact, PCjr is quite fussy about your making the timing of the same measures for different voices match. When peculiar combinations of things play, that's probably your doing.

Working 3-PART.BAS

Figure 7.6 gives the harmonization for a medieval tune called "Attende Domine." It's set up to operate with 3-PART.BAS. Key it into memory and SAVE it as an ASCII file under the name HEARLORD.BAS. Watch out for lower-case L's—don't misread them as 1's. Then LLIST the file and check it for typos before merging it into 3-PART.BAS.

```
100 ' HEARLORD.BAS
110 BARS=3:T=90
120 B$(1,1)="o3l8mldf#mnaaaml4a."
130 B$(1,2)="p8o3l4mlc#<ba."
140 B$(1,3)="p8o2ml2dl4d."
150 B$(2,1)="p16ml8>dc#<abl4al8aa."
160 B$(2,2)="p16ml8o2b>c#d#ff#edd."
170 B$(3,1)="p8ml8>d<aegamnf#mlf#el4d."
180 B$(3,2)="p8ml8o3c<gf#l4>dl8dl4c#<b."
190 B$(3,3)="p8p8p8p8ml4o2el8el4d<a."
```

Figure 7.6: HEARLORD.BAS—"Attende Domine" (Gregorian)

When you've finished preparing HEARLORD and have SAVEd it on your *Test Programs*, LOAD 3-PART.BAS into memory again. That is,

Type: LOAD "3-PART" <ENTER>

Then MERGE HEARLORD so you have the two program pieces joined together:

Type: MERGE "HEARLORD" <ENTER>

Now RUN the combined program in memory to find out how it sounds. You'll probably get a sour note or two—at any rate, I always do.

A dependable approach to locating music code and typing problems is to play each part separately. Do this by setting the volume controls in line 110 to zero for the voices you're not concerned with at the moment. Also keep at hand a LLISTing of the B$ coding, and the printed music, if available, to aid in the debugging process.

A Little Fun with Bach

I wanted you to get used to 3-PART.BAS on a little tune before I gave you a difficult case to deal with. Figure 7.7 has the coding for Bach's Two-Part Invention No. 1. Keying this tune in will test your mettle. If you have the music you might compare my coding against it.

LOAD 3-PART.BAS first, then type the Bach Invention into the proper slot in memory. Type only a few bars at a time, and check them out bit by bit by RUNning 3-PART.BAS. As you get beyond the middle of the tune, you can save time by modifying line 1010 so that variable "I" starts not at the first measure each time but at a later one of your choosing.

```
100 ' BACH: TWO-PART INVENTION NO. 1
110 T=80:V1=10:V2=10:V3=10 ' Set tempo and
    volume
120 BARS=22 ' Number of measures
130 B$(1,1)="p16l16o3cdefdecmsg8o4c8o
    3b32a32b16o4c8"
140 B$(1,2)="p4p4p16o2l16cdefdec"
150 B$(2,1)="mso4dmno3gabo4co3abgmso
    4d8mng8f32e32f16g8"
```

```
160 B$(2,2)="o2msg8o1g8p4p16mno2gabo3co2abg"
170 B$(3,1)="mso4eagfegfagfedcedf"
180 B$(3,2)="l8o3co2bo3cdeo2gab"
190 B$(4,1)="l16o4edco3bao4co3bo4dco3bagf+agb"
200 B$(4,2)="o3l8co2ef+gmsabmno3mlc4"
210 B$(5,1)="mso3a8d8o4mnc32o3b32o4c
    8d16o3l16bagf+egf+a"
220 B$(5,2)="mno3l16co2def+gef+dmsg
    8mno1b8o2c8d8"
230 B$(6,1)="gbao4co3bo4dcemsdmno3b32o
    4c32dmsgmno3b32o4c32o3b16ag"
240 B$(6,2)="mso2l8ef+gemlo1b.o2c16msdo1d"
250 B$(7,1)="o3g8p8p4p16l16gabo4co3abg"
260 B$(7,2)="mnp16l16o1gabo2co1abgo2msd
    8mng9l8f+g"
270 B$(8,1)="l8mnf+p8p4p16l16ab>cd<b>c<a"
280 B$(8,2)="l16msamndef+gef+dl8msamn>dcd"
290 B$(9,1)="l8bp8p4p16l16>dc<ba>c<b>d"
300 B$(9,2)="l16<g>gfedfegmsl8fefd"
310 B$(10,1)="l8cp8p4p16l16edc<b>dc+e"
320 B$(10,2)="msl16emnagfegfamsl8gfge"
330 B$(11,1)="l8dc+def<ab>c+"
340 B$(11,2)="l16mnfb-agfagb-agfedfeg"
350 B$(12,1)="d<f+g+amsb>cmll4d"
360 B$(12,2)="l16fedc<b>dcedc<bag+ba>c"
370 B$(13,1)="l16dmn<ef+g+af+g+e>edcedc<b>d"
380 B$(13,2)="l8ms<bemnl32>dcl
    8dl16ec<bagf+ag+b"
390 B$(14,1)="cag+baefd<g+>fedl32cdl16cba"
400 B$(14,2)="l16a>c<b>dcedfl8mse<a>e<e"
410 B$(15,1)="o3l16msamn>agfegfal2mlg"
420 B$(15,2)="o2l8msa<ap4p16l16>>edc<b>dc+e"
430 B$(16,1)="mnl16gefgafgemll2f"
440 B$(16,2)="l2mldmnl16d<ab>cd<b>c<a"
450 B$(17,1)="mnl16fgfedfegmll2f"
460 B$(17,2)="l2mlbmnl16b>dc<ba>c<b>d"
470 B$(18,1)="mnl16fdefgefdl2mle"
```

```
480 B$(18,2)="l2mlcl16mnc<gab->c<ab-g"
490 B$(19,1)="mnl16ecdefdecdefgafge"
500 B$(19,2)="l8ab-agf>dc<b-"
510 B$(20,1)="fgab>c<abg>l8c<gl31efl16edc"
520 B$(20,2)="a>fedl16msemn<defgefd"
530 B$(21,1)="c<b-agfagb-ab>c<ed>c<fb"
540 B$(21,2)="l8ecdel16fdefl8msg<g"
550 B$(22,1)="mnl1>c"
560 B$(22,2)="l1mn>c"
```

Figure 7.7: Bach's Two-Part Invention No. 1

NOTE: You'll have noticed I'm pretty inconsistent about transcribing music. Sometimes it's all capitals. Other times it's lower-case letters. Sometimes I put spaces to separate notes. Sometimes the bass is Voice 0, other times it's Voice 2. I've no justification. I'm just inconsistent. Sorry.

Music Minus One—YOU!

Do you play an instrument in a chamber orchestra or a small band—or would you like to? 3-PART.BAS gives you an opportunity to practice your part when the other band members are elsewhere. Here's how.

Let's say you play violin in a quartet or trio. Get copies of the musical parts your fellow musicians play and code them for 3-PART.BAS as shown above. If it's a trio, you might code your own part too—especially if you're learning a new part.

Now you're all set to go. As PCjr plays the other parts, you play along with your instrument. And if your part is also coded, you might let it play too, perhaps at a lower volume so that it won't interfere with your live playing, but loud enough to hear how you're doing by comparison.

One nice characteristic of this approach to practicing your part is that you can slow the tempo (just vary T in line 110) without changing the pitch—just like a real music group. As you know, slowing or speeding a tape or a record changes the key in which you hear the arrangement.

4. EXPERIMENTAL MUSIC

Another feature of the PCjr's music language is the N command. It permits you to treat notes as numbers instead of letters. For instance,

Type: SOUND ON:BEEP OFF <ENTER>
Type: PLAY "N37 P2 O3C" <ENTER>

Middle C should have played twice. Look at the top of Figure 4.1 again and you'll see how each note is numbered.

This note-number correspondence makes it possible to process music arithmetically before you play it. This presents a great advantage when you transpose a piece of music into another key. Try writing a program to transpose by manipulating the letters of the alphabet. You'll soon find out that if the music were represented by numbers instead of letters, you could transpose simply by adding or subtracting a constant from each note.

You can do transposition on your own. Right now, there is a much more interesting application—creating music by computer. This is a tough one, but there are some simple things you can do that might be fun.

Imitate a Musical Pattern

The trick is to avoid purely random sequences of notes and chords, but to use randomness to embellish and extend music patterns you're familiar with. Figure 7.8 is an example that may give you an idea of how to go off on your own.

Here's the musical process: Pick a scale—almost any scale will do. The left hand starts playing the scale upward in one of the lower octaves, say O1 or O2, beginning with almost any note. The right hand plays parallel thirds, fourths, or fifths in almost any combination, in one of the upper octaves, say O3 or O4. But the right hand runs down the scale at half the rate the left hand is coming up. Try to keep the hands two or more octaves apart for best results.

After playing a sequence of five or six chords this way, the left hand jumps down about an octave to change the contrast as the right hand continues down the scale. After playing three or four more notes, the right hand jumps up a fifth, an octave, or an octave and a half, and then starts down the scale again. This time you might change the interval of parallel thirds to parallel fifths, or whatever.

Figure 7.8 is an attempt to simulate this process on the PCjr in a simple way. The program does not take into account how rhythms and other nuances may be changed. You may want to change scales in the middle of a progression, or play different scales at different tempos in each hand. The possibilities are endless.

The program has a certain fuguelike character. Play around with it to see what other musical effects you can generate. Besides experimenting with the tempo and lengths of runs, use some different scales. Here are some possibilities:

Minor Scale. Make these changes:

```
160 S(3)=3
190 S(6)=8
```

Whole Tone (French Romantic). Make these changes:

```
130 ST=6
160 S(3)=4
170 S(4)=6
180 S(5)=8
190 S(6)=10
DELETE 200
```

Pentatonic (Oriental). Make these changes:

```
130 ST=5
160 S(3)=5
170 S(4)=7
180 S(5)=9
DELETE 190 and 200
```

There are a whole bunch of other scales (modes) you can try too.

When you've gotten the idea, try a pattern of your own devising. That's when the fun really begins.

```
100 ' FUGAL.BAS—Experimental music
110 ROOT=11 ' KEY OF B
120 ' SCALE NOTES—DIATONIC
130 ST=7 ' 7 SCALE TONES
```

```
140 S(1)=0
150 S(2)=2
160 S(3)=4
170 S(4)=5
180 S(5)=7
190 S(6)=9
200 S(7)=11
210 V=10
220 T=110' TEMPO
230 R=3.141593/180 ' Radians per degree
240 SOUND ON:BEEP OFF ' External speaker
250 PLAY"L16MBT=t;","L8MBT=t;","L8MBT=t;"
260 BROOT=ROOT+4:IF BROOT>12 THEN
    BROOT=BROOT-12
270 OCTV11=INT(RND*2) ' SELECT OCTAVE
280 OCTV2=OCTV1+3
290 N11=INT(RND*ST)+1 ' SELECT STARTING NOTE
300 N12=N11+1:OCTV12=OCTV11
310 IF N12>ST THEN N12=N12-
    ST:OCTV12=OCTV12+1
320 N2=INT(RND*ST)+1
330 N3=N2-3:OCTV3=OCTV2
340 IF N3<1 THEN N3=N3+ST:OCTV3=OCTV3-1
350 MAX=7+INT(RND*7) ' SELECT NUMBER OF
    NOTES
360 K=0 ' PLAYED NOTE COUNT
370 NOTE11=BROOT+S(N11)+12*OCTV11
380 NOTE12=BROOT+S(N12)+12*OCTV12
390 NOTE2=BROOT+S(N2)+12*OCTV2
400 NOTE3=BROOT+S(N3)+12*OCTV3
410 PLAY "MN","ML","ML"
420 A=(A+15) MOD 360
430 V=4+3*SIN(R*A)
440 PLAY "V9N=NOTE11;N=NOTE12;","V=v;N=
    NOTE2;","V=v;N=NOTE3;"
450 IF N11>=ST THEN N11=N11-
    ST:OCTV11=OCTV11+1:GOTO 470
```

```
460 N11=N11+2
470 IF N12>=ST THEN N12=N12-
    ST:OCTV12=OCTV12+1:GOTO 490
480 N12=N12+2
490 IF N2=1 THEN N2=ST:OCTV2=OCTV2-1:
    GOTO 510
500 N2=N2-1
510 IF N3=1 THEN N3=ST:OCTV3=OCTV3-1:
    GOTO 530
520 N3=N3-1
530 K=K+1:IF K>MAX THEN 270
540 GOTO 370
```

Figure 7.8: FUGAL.BAS—Experimental Music

Program Notes

Basically, all FUGAL.BAS does is play the scale. You specify which scale to play by inserting the number of the ROOT note (line 110) and the numbers of the scale tones (lines 130-200) relative to "do." In the program I picked, the key of B, I set ROOT=11, the lowest B I wanted to play! There are seven tones in the B Major scale (ST=7). I assign the number 0 to "do," 1 to "do#," 2 to "re," etc. In this way the seven tones of the major diatonic scale are 0, 2, 4, 5, 7, 9 and 11. For the minor scale, change 4 and 9 to 3 and 8, respectively. If you want the pentatonic scale, don't forget you have to make ST=5 in line 130 besides changing the scale tones.

The control of the playing process runs from line 370 to line 540. The scale tones to play are N11 and N12 (left hand), and N2 and N3 (right hand). N2 and N3 play together as a chord; N11 and N12 play sequentially at twice the rate of N2 and N3. The octaves in which they play are OCTV11, OCTV12, OCTV2 and OCTV3, respectively.

You can see how the left hand is set to play up the scale in line 460 and 480, while the right hand runs down the scale (lines 500 and 520). The switching of octaves is done in lines 450, 470, 490 and 510.

The actual notes played must be calculated from the scale tone and the octave. This is done in lines 370-400. Essentially all you need do is multiply the octave number by twelve and add the current scale tone to play. But to obtain the correct value, an offset, BROOT, calculated in line 260, must also be added in.

You can experiment with the qualities of notes played by changing line 410.

Lines 420-430 make the volume vary in a sine wave pattern. You might try other ways of introducing dynamics. For instance, make all three voices vary in volume by different sine waves, no two of which are in phase.

The real work is done in line 440. Notice how the volume and note values, which are calculated values, not constants, are incorporated in the PLAY command. Normally we'd write that command something like this: PLAY "V8O3C," assuming L is set in a previous command. But we might also write the command as PLAY "V8N37," or as a sequence of commands:

```
VOL=8:NOTE=37:PLAY "V=VOL;N=NOTE"
```

The equal sign in the music character string tells PCjr to treat what follows as the label of a number. The semicolon defines the end of the label so PCjr can switch back to reading music notation.

A similar construction is used in line 250 to set the tempo.

The number of notes to play in sequence (MAX) is set in line 350. I've made it vary between 7 and 13. MAX is tested for in line 530. When this limit is detected, the starting conditions are reset for a new sequence of chords, starting at line 270.

The left hand and the top note of the right are selected as random scale tones (lines 290-320). The bass octave is also a random number, but the right hand is set to start three octaves above (lines 270-280). The other note in the right hand is set a fourth below the top note (line 330). Its octave is set to that of the top note, but if it's really in the octave below, this fact is detected in line 340 and corrected.

5. MODERN AND PERSONAL

We've played enough with the classical and traditional. It's time to find out how PCjr fares with music in the modern American idiom. I asked my daughter, Genevieve C. Kelley, a graduate of the Temple University School of Music, and a keyboard player and vocalist with the Intergalactic Bus Tour rock group, to put PCjr to the test. She has kindly

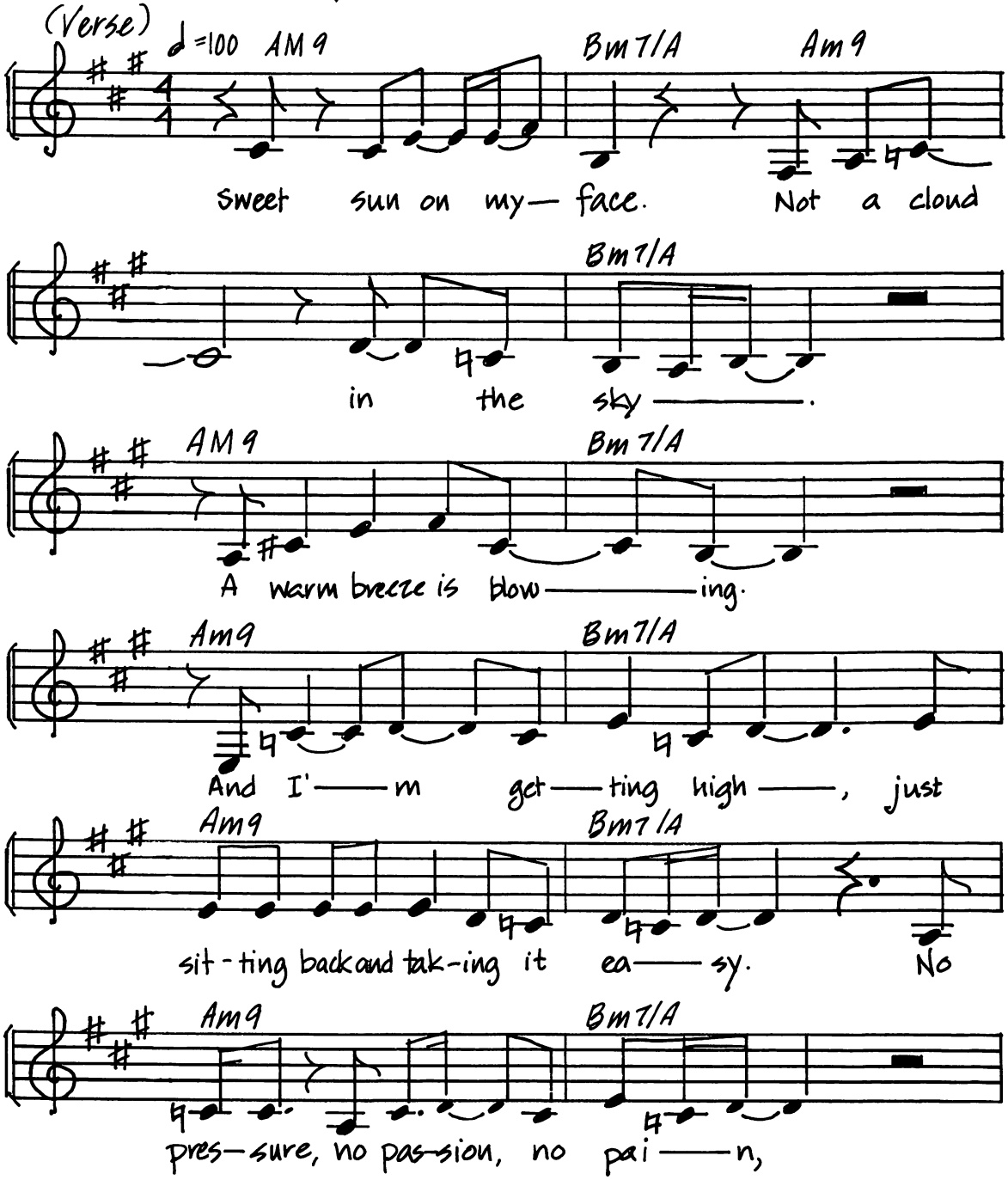
arranged one of her compositions, "Summer Days/Summer Nights," in three voices for the PCjr. The melody and words are given in Figure 7.9.

I've transcribed Jenny's arrangement in Figure 7.10, in a form you might adapt to run under 3-PART.BAS. But don't bother. I have an idea for a home-brew, video-rock short based on Jenny's arrangement that we can develop in the next chapter. To prepare for this experience in animation, we've got to control playing the tune in a somewhat different fashion.

I hope you won't be put off by the amount of typing you've got to do to key in this arrangement. It's really a lovely thing. The melody is handled in Voice 2. The bass is handled in Voice 0. It should be an octave lower than I've coded it, but it sounds better on the PCjr this way. And there's a pretty, flutelike descant in the treble.

```
5000 ' SUMMER DAYS/SUMMER NIGHTS
5010 ' COPYRIGHT 1981 BY GENEVIEVE C. KELLEY
5020 ' USED WITH PERMISSION
5030 A$(1, 1)="V6O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5040 A$(2, 1)="V6P8L16O4BMLEMNEBL8EP8L16O4
     BMLEMNEBL8E"
5050 A$(3, 1)="V10P4O3L8C#P8C#MLEMNL16
     EMLEMNL8F#"
5060 A$(1, 2)="O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8 MNE<E"
5070 A$(2, 2)="P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
5080 A$(3, 2)="O2L4BP4P8L8F#A>MLC"
5090 A$(1, 3)="O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5100 A$(2, 3)="P8L16O4GMLCMNCGL8CP8L16O4
     GMLCMNCGL8C"
5110 A$(3, 3)="O3MNL2CP8L8MLDMNDC"
5120 A$(1, 4)="O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5130 A$(2, 4)="P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
```

Summer Days / Summer Nights
by Genevieve C. Kelley



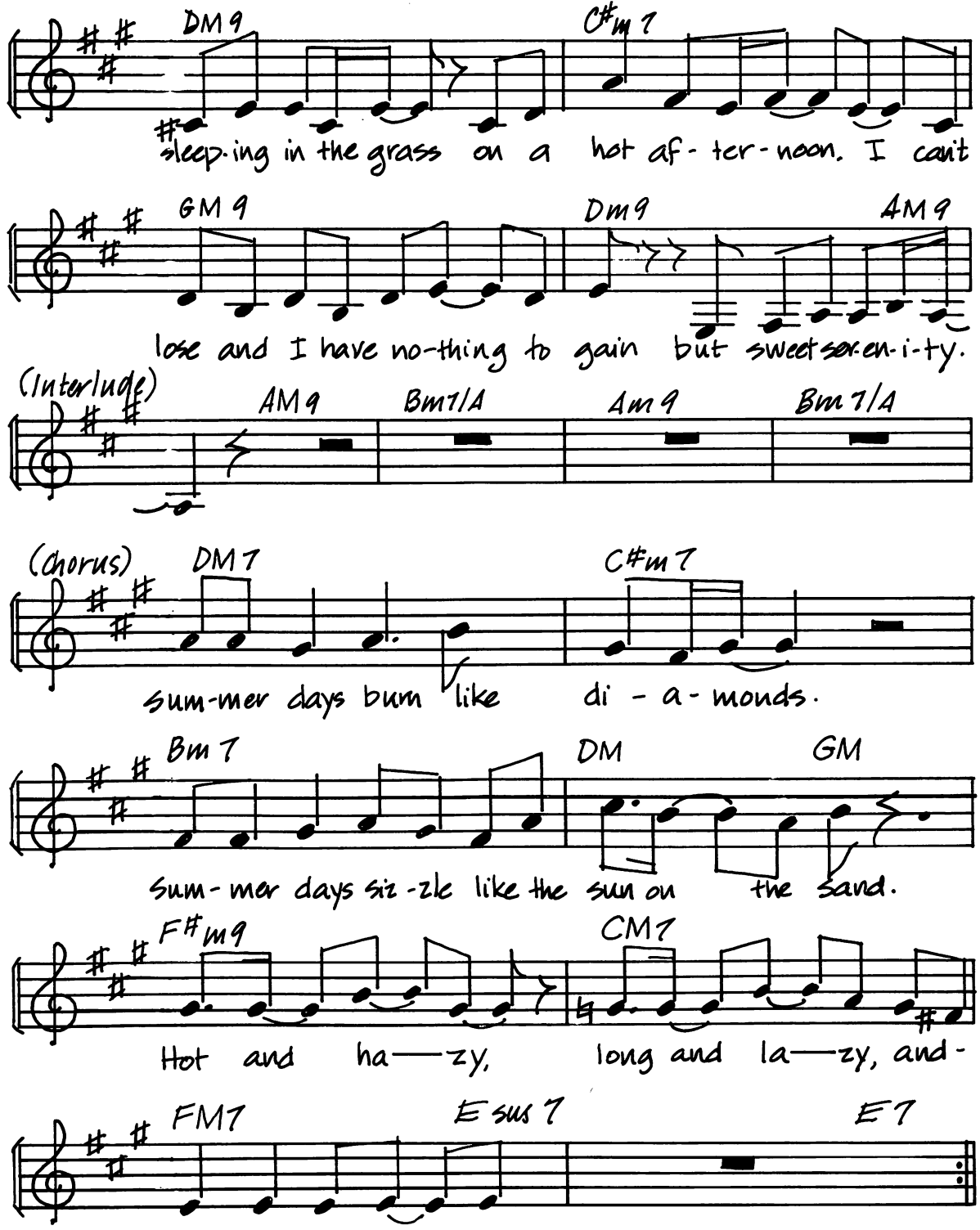


Figure 7.9: "Summer Days/Summer Nights"

```
5140 A$(3, 4) = "O2L8BL16AMLBL4BP2"
5150 A$(1, 5) = "O1L8A.>L16MLEL8MNE<
     EO1L8A.>L16MLEL8MNE<E"
5160 A$(2, 5) = "P8L16O4BMLEMNEBL8EP8L16O4
     BMLEMNEBL8E"
5170 A$(3, 5) = "P8O2L8A>L4C#EL8F#MLC#"
5180 A$(1, 6) = "O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5190 A$(2, 6) = "P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
5200 A$(3, 6) = "O3MNL8C#<MLBMNL4BP2"
5210 A$(1, 7) = "O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5220 A$(2, 7) = "P8L16O4GMLCMNCGL8CP8L16O4
     GMLCMNCGL8C"
5230 A$(3, 7) = "P8O2L8EL4ML>CMNL8CMLDMNDC"
5240 A$(1, 8) = "O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5250 A$(2, 8) = "P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
5260 A$(3, 8) = "O3L4EL8CMLDMNL4D.L8E"
5270 A$(1, 9) = "O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5280 A$(2, 9) = "P8L16O4BMLEMNEBL8EP8L16O4
     BMLEMNEBL8E"
5290 A$(3, 9) = "O3L8EEEEL4EL8DC"
5300 A$(1, 10) = "O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5310 A$(2, 10) = "P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
5320 A$(3, 10) = "O3L8DL16CMLDMNL4DP4.L8<A"
5330 A$(1, 11) = "O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5340 A$(2, 11) = "P8L16O4GMLCMNCGL8CP8L16O4
     GMLCMNCGL8C"
5350 A$(3, 11) = "O3L16CL8C.P8<A>C.
     MLL16DMNL8DC"
5360 A$(1, 12) = "O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
```

```
5370 A$(2, 12)="P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
5380 A$(3, 12)="O3L8EL16CMLDMNL4DP2"
5390 A$(1, 13)="O1L8D.L16MLF#L8F#AD.
     L16MLF#L8F#A"
5400 A$(2, 13)="O5P8L8C#P16<L8B.P8AP16B."
5410 A$(3, 13)="O3L8C#EEL16C#MLEMNL8EP8C#E"
5420 A$(1, 14)="O1L8C#.L16MLEL8EG#L8C#.
     L16MLEL8EG#"
5430 A$(2, 14)="O5P8L8C#P16<B.P8G#P16A."
5440 A$(3, 14)="O3L4AL8F#L16EMLF#MNL8F
     #MLEMNEC#"
5450 A$(1, 15)="V8O1L8G.MLL16BL8B>D<L8G.
     MLL16BL8B>D"
5460 A$(2, 15)="V8O4P8L8GP16A.P8BP16>D."
5470 A$(3, 15)="V8O3L8D<B>D<B>DMLEMNED"
5480 A$(1, 16)="V10O1L8DP16MLL16DMNL8DV8P8
     P16EL16DL8C#<B"
5490 A$(2, 16)="V10O5L8FP16MLL16FL8FMNP8P2"
5500 A$(3, 16)="V11O3L8EP4V9<EF#AAL16BMLA"
5510 A$(1, 17)="V6O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5520 A$(2, 17)="V6P8L16O4BMLEMNEBL8EP8L16O4
     BMLEMNEBL8E"
5530 A$(3, 17)="V6O2L4MNAP4P2"
5540 A$(1, 18)="V4O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5550 A$(2, 18)="V5P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
5560 A$(3, 18)="P1"
5570 A$(1, 19)="V2O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5580 A$(2, 19)="V4P8L16O4GMLCMNCGL8CP8L16O4
     GMLCMNCGL8C"
5590 A$(3, 19)="P1"
5600 A$(1, 20)="V4O1L8A.>L16MLEL8MNE<EO1L8A.
     >L16MLEL8MNE<E"
5610 A$(2, 20)="V5P8L16O4AMLDMNDAL8DP8L16O4
     AMLDMNDAL8D"
```

```
5620 A$(3, 20)="P1"
5630 A$(1, 21)="O1L8D.L16MLF#L8F
     #AD.L16MLF#L8F#A"
5640 A$(2, 21)="O3L8F#F#L4EF#.L8F#"
5650 A$(3, 21)="O3L8AAL4G#A.L8B"
5660 A$(1, 22)="O1L8C#.L16MLEL8EG#L8C
     #.L16MLEL8EG#"
5670 A$(2, 22)="O3L8G#L16F#MLG#L4MNG#P2"
5680 A$(3, 22)="O3L8EL16DMLEL4MNEP2"
5690 A$(1, 23)="O0L8B.MLL16>DMNL8DF#<B.MLL16
     >DMNL8DF#"
5700 A$(2, 23)="O3L8DDL4EL8F#EDF#"
5710 A$(3, 23)="O3L8F#F#L4G#L8AG#F#A"
5720 A$(1, 24)="O1L8D.L16MLF#L8F#AG<GB>D"
5730 A$(2, 24)="O3L8A.L16MLG#MNL8G
     #F#F#P8P4"
5740 A$(3, 24)="O4L8C#.<L16MLBMNL8BABP8P4"
5750 A$(1, 25)="O0L8F#.ML>L16C#MNL8C
     #F#<F#.ML>L16F#L8MNF#C#"
5760 A$(2, 25)="O3L8E.MLL16EL8MNEMLG
     #G#EEP8MN"
5770 A$(3, 25)="O3L8G#.MLL16G#L8MNG
     #MLBBG#G#P8MN"
5780 A$(1, 26)="O1L8C.MLL16EL8MNEGC.
     MLL16EMNL8EG"
5790 A$(2, 26)="O3L8E.MLL16EMNL8EMLGMNGFED"
5800 A$(3, 26)="O3L8G.MLL16GMNL8
     GMLBMNBAGF#"
5810 A$(1, 27)="O1L8F.MLL16AMNL8A>C<F.
     MLL16AMNL8A>C"
5820 A$(2, 27)="O3L4CCL8CMLCMNCD"
5830 A$(3, 27)="O3L4EEL8EMLEMNEE"
5840 A$(1, 28)="V9O1L8EEP8EP2"
5850 A$(2, 28)="V9O4L8EEP8EP2"
5860 A$(3, 28)="P1"
```

Figure 7.10: "Summer Days/Summer Nights"—PCjr Arrangement
Used with Permission

Loading the Music Buffer Only When Needed

All our music programs so far have worked in the same way: They fill the music buffer and then hang around until there's enough space to put more music into it. This strategy won't always do. You'll often want to be able to go off and do something else instead. You can do this now, but what if you don't get back to check on the music buffer before it runs out? Of course, you'll get undesirable breaks in the music.

To deal with these common situations, PCjr permits you to set an "interrupt" that works like this: When the number of notes in the music buffer reaches a limit you preset, you're jumped to a subroutine with which you can put more music in the buffer.

With this idea in mind, add the following statements to the music for "Summer Days/Summer Nights" (Figure 7.10).

```
   10 ' SUMMER.BAS
   20 GOTO 5025 ' Initialize program
   30 ' Play tune subroutine
   40 PLAY "xa$(1,i);","xa$(2,i);","xa$(3,i);"
   50 I=I+1:IF I>28 THEN I=1 ' Select next measure
   60 RETURN
 5025 DIM A$(3,28) ' Define an array for the music
 6000 ' Start music
 6010 SOUND ON:BEEP OFF ' External speaker
 6020 PLAY"T90MBV6","T90MBV6","T90MBV6 ' Set tempo,
      background
 6030 I=1:GOSUB 40 ' Play 1st measure
 6040 ON PLAY(4) GOSUB 40 ' Define interrupt
 6050 PLAY ON ' Turn on interrupt
 6060 PRINT I;:GOTO 6060 ' Dummy program
```

When you've typed in these lines, SAVE the combined program as SUMMER.BAS. Then RUN it.

Let's run through this little program quickly to be sure you understand what's happening when it plays. Lines 10-20 bypass the play subroutine and jump you to the initialization instructions. Since the array A$ is not defined in Figure 7.10, we've got to include the DIMension statement at 5025 so that there will be a place for all the music defined in the subsequent lines.

When all the measures of music have been loaded in array A$, we can start the music. The external speaker is selected (line 6010), and the tempo and background are set (line 6020).

Then the first measure is defined (I=1) and we're sent to the subroutine at line 40. There the first measure is loaded into the music buffer. The next measure is selected by adding 1 to I, or by setting it back to 1 to begin again. We're now returned to line 6040.

Here's the new idea: The ON PLAY(4) statement says, "Whenever there are four or fewer notes left in the music buffer, GOSUB 40 to load another measure." To make this interrupt effective, we must execute a PLAY ON. PLAY OFF stops the interrupt from functioning.

Line 6060 simply keeps the program running. I included a PRINT statement so you could see when each measure is loaded and how many times line 6060 is executed between reloads of the music buffer. This will give you a feel for the amount of work that can be done while the music is playing.

You'll notice that when certain notes are sustained, everything seems to come to a halt. As of this writing, I'm not sure what the cause of this effect is or how one should deal with it. Frankly, it's disconcerting and it breaks the smooth flow of graphics, as we'll see in the next chapter.

CHAPTER 8
VIDEO ROCK A LA PCjr

Objective:

This chapter demonstrates how to design and build an animated graphics scene with an integrated soundtrack.

1. THE SCENARIO

The first thing you'll need for your animation is a story. After all, if you're going to move objects about on the screen in a reasonably realistic way, you've got to have some purpose or objective in mind. It's fine to have a beautiful galloping horse on the video screen, but if it's not in a race or running to the rescue or fleeing from something, what's the point?

In casting about for something simple but useful that we might try, I recalled my daughter Jenny's mentioning that her group was considering making a video-rock tape of her "Summer Days/Summer Nights." We'll try the same thing, but you'll create a more sedate version. We won't have bouncing musicians, foxy birds, hunks, and a light show, but we'll learn how to project our imaginations onto the video screen to the rhythms of music.

"Summer Days/Summer Nights" reminds me of the many happy summer vacations we had at the shore in Milford, Connecticut. Our cottage was right on the beach. On many a hot afternoon I sat in the porch shade, with iced tea in hand, occupied with the inactivity before me. Only a gull or two, or a boy with a new sailboat, ventured into that scorching sun.

The movement of those gulls and the fluttering sails caught the eye with mesmerizing effect, so they will be the principal subjects of the story. But it's the mood this scene projects that I'd like to see if we can recapture using PCjr's graphics.

2. BUILDING A WORLD

Drawing with Lots of Colors

First we'll need a beach, water for the boat to sail on, and a sky for the gulls to fly about in. For this purpose, let's do some experimenting with SCREEN 5. It has sixteen colors we can draw with, but it also requires 32K of memory to hold the video buffer—not the 16K default.

Remember, we solved this memory allocation problem earlier in DEFORM.BAS by using the CLEAR command. (As we go along now, just start entering program statements into a cleared memory.)

```
1000 ' Initialize
1010 CLEAR,,,32768! ' 32K video buffer
1020 SCREEN 5 ' Med Res, 16 colors
1030 COLOR 15,0 ' White foreground, black background
1060 CLS:KEY OFF
```

The COLOR statement works differently for SCREEN 5 from the way it does for SCREEN 1. Now the first number is the foreground color, the second the background. Just don't use black (color 0) for the foreground, or you'll get an "Illegal function call" message.

The Beach

To lay out a nice, curving beach we'll use a trick we learned earlier with the CIRCLE command for drawing the arc of an ellipse—you know, a small aspect ratio.

```
1350 ' Build world
1360 CIRCLE (100,0),380,1,,,.4 ' Light yellow beach
1370 PAINT (130,190),14,1
1380 GOTO 1380
```

Let's see what this looks like so far. Hit RUN. Did you get the beach?

The GOTO in line 1380 is just a trick we'll use throughout to test how the program is developing. We'll wipe it out now as we add in the

Water

Do a <Fn-Break> and type in

```
1380 LINE (0,100)-(319,100),1 ' Dark blue water
1390 PAINT (150,120),1,1
1400 GOTO 1400
```

Go ahead, RUN again. Did the water PAINT in okay? All we need now is

The Sky

```
1400 PAINT (150,50),9,1 ' Sky
1410 GOTO 1410
```

RUN once more to be sure the picture is in good shape.

The Sun

While we're at it, let's add the scorching sun—up in the right-hand corner, with a black border for emphasis.

```
1480 'Place sun
1490 CIRCLE (320,0),30,0,5/6
1500 PAINT (315,5),14,0
1510 GOTO 1510 ' Test GOTO
```

Before you test this addition, don't forget to DELETE 1410. Otherwise you won't see the sun.

3. PCjr'S DRAWING LANGUAGE

Now that we've got a backdrop for the action, we ought to address the characters of our drama: the sea gulls, and the boy and his sailboat. Here's where we can exploit PCjr's DRAW command. Let's see how it works.

SAVing Intermediate Results

Before experimenting with DRAW, you should SAVE what you've already typed in. Do a <Fn-Break> and

Type: SAVE "BEACH" <ENTER>

It's always a good idea to SAVE partially developed programs after you've worked out wrinkles. You never know when you may lose power or do something screwy that wipes out memory.

An Aside on DRAW

DRAW is much like the PLAY command in that it executes a character string of instructions that you provide for defining a series of lines to draw on the screen. One way to specify the figure you want to draw is by a series of left-right, up-down and diagonal moves. Here's how you might draw a box. (I'll change screen modes for this, but it's the same in all modes.)

Type: SCREEN 1,0:COLOR 1,1 <ENTER>
Type: PSET (100,100),3 <ENTER>
Type: DRAW "U20 R40 D20 L40" <ENTER>

As you may gather, U means up, R means right, D means down, and L means left. And the numbers that follow them are the numbers of units to move in that direction. These and the diagonal moves that are possible are defined in the compass rose of Figure 8.1.

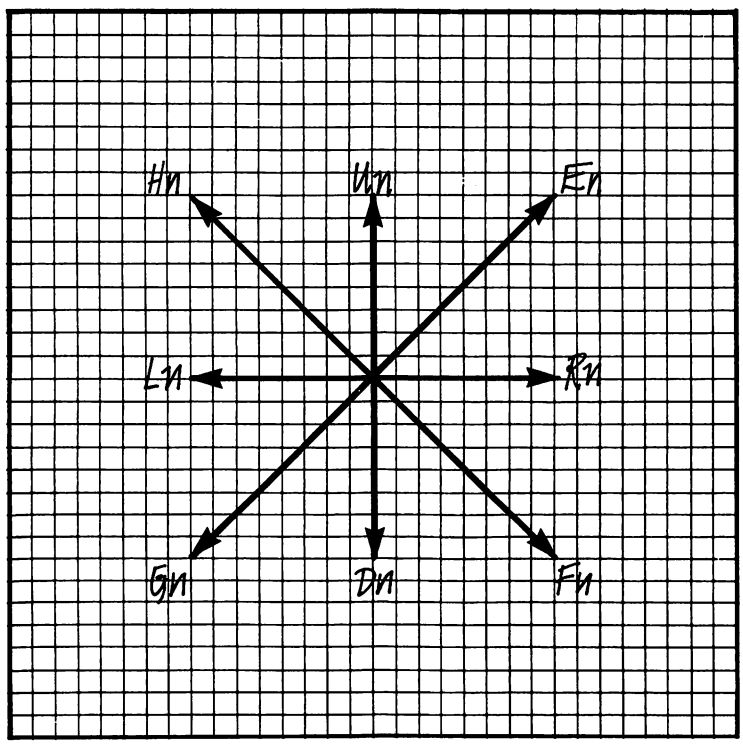


Figure 8.1: Graphic Cursor Moves in the Draw Command

In the illustration still on the screen, the color used to draw the box is defined by the 3 in the PSET command, which moved the graphics cursor to the starting point of the DRAW. However, the color can be set in the DRAW command itself. Move the cursor under the U in the DRAW command, hit the Ins(ert) key, and

Type: C0 <ENTER>

That's a zero, not the letter O.

The box erases because color 0 is "no color." Change the 0 to a 1 and hit <Enter>. Now the box is drawn in green, which is not easily visible on that blue background.

Now move the cursor under the rightmost quotation mark of the DRAW command and

Type: BE5 P3,1" <ENTER>

As you can see in Figure 8.1, the E command says to move diagonally up and right. The B before the E says, "move, but don't draw anything."

When the BE5 instruction combination is through, the graphics cursor is relocated within the box. This permits the P, or PAINT, command to take effect. It paints the box with the attribute 3 (white), provided it has a boundary with attribute 1 (green), which it has.

Change the 3 after the P to a 0 (zero) and hit <Enter>. Do you understand what happened? Put the cursor on the DRAW command once again and hit <Enter>.

Because of the way the DRAW command is set up with the BE5 instruction, we could keep on drawing box over box up the diagonal.

Here, add another wrinkle. Put the cursor under the C at the beginning of the character string of DRAW, hit the Ins(ert) key, and

Type: TA45 S5 <ENTER>

The TA (turn angle) rotates the box (from -360 to +360 degrees), here 45 degrees. S (size or scale) makes the figure larger (max = 255) or smaller (min = 1).

Much of what we've just discussed might also be done with the LINE command. But it's much easier with DRAW.

Retry some of the earlier programs we did with the LINE command to see if you can use the DRAW command instead.

DRAWing Sea Gulls

Being limited to drawing figures with angles of 45 and 90 degrees is rather restrictive. But we can actually draw a line from any point to any other using the DRAW command, and we'll use it to draw the sea gulls for our scene.

You have to start with a picture, typically drawn on a piece of graph paper as shown in Figure 8.2. I start out drawing a freehand sketch of what I want inside a box scaled to the size I'd like to see on the screen. Do this sketch before confusing the issue with numerical coordinates, then try to select the closest coordinate points on the grid for the end points of the line segments to be drawn. The coordinates you see in Figure 8.2 are then incorporated into a PSET-DRAW combination this way:

```
1070 ' Construct sea gulls
1080 PSET(1,4),3:DRAW"m6,4 m8,6 m10,5 m14,8" ' 1st gull
1090 PSET(13,1),3:DRAW"m19,3 m20,5 m22,4 m28,5" '
     2nd gull
1100 GOTO 1100 ' Test GOTO
```

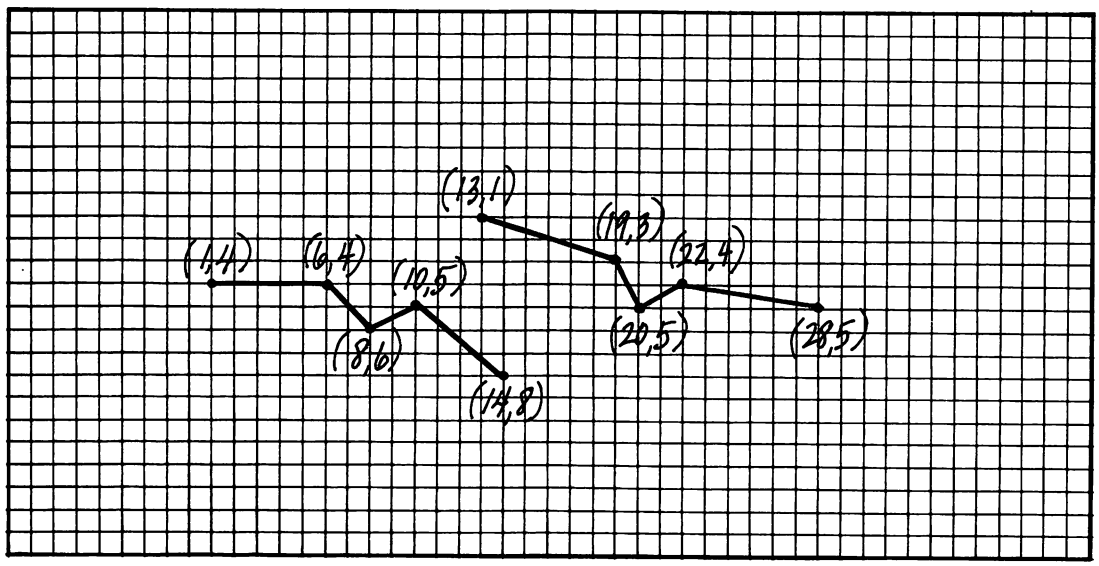


Figure 8.2: Flying Sea Gulls

The first point in the sequence is put into the PSET command, and the drawing color is set. Each successive point is then incorporated in the DRAW command, preceded by an M (move to).

RUN and see what's been drawn.

NOTE: There's another way to initiate DRAW, without the PSET command. Write instead:

```
1080 DRAW "BM1,4 C3 M6,4 ... etc.
```

Saving a Picture in an Array

Now that we have the sea gulls, we must save them in an array so they can be quickly drawn onto and erased from the screen. It's by successively drawing and erasing figures in changing positions that we give the impression of movement and carry out the animation process.

The BASIC command for saving a picture is GET (from the screen).

```
1100 GET (0,0)-(28,8),GULL
1110 CLS ' Clear screen for next figure
```

A whole rectangle of points around the picture is saved. You specify the upper left and lower right corners, as illustrated in line 1100. The array into which the points are transferred is called GULL. It must be defined in a DIM(ension) statement. We'll also need arrays for other figures, so I'll include them now.

```
1040 DIM GULL(50),BT(500),TENT(800)
```

(See the GET command in the BASIC manual for information on calculating the space you need in an array to accommodate a figure. The formula is too complicated for me, so I just pick a number. If it works, good. If not, I make it larger.)

Transferring a Picture Array to the Screen

To verify that the sea gulls are properly stored in the array GULL, we use the PUT (onto the screen) command.

Type: RUN
Type: <Fn-Break>

Don't clear the screen, but just

Type: PUT (150,75),GULL,XOR <ENTER>

Did the sea gulls pop out on the screen? This verifies that GULL has what we want. We'll deal more with PUT shortly.

The Sailboat

I handled drawing the sailboat in the same way as the gulls. It's a slightly more complicated figure, with four parts, and it took longer to construct. Here's what I came up with:

```
1120 ' Construct sailboat
1130 PSET(8,43),3:DRAW"m7,41m40,36m34,40m8,43" ' Hull
1140 PAINT(24,40),3,3
1145 ' Mainsail
1150 PSET(29,38),13:DRAW"m38,5m30,16m16,
     29m5,34m28,35m28,38"
1160 PAINT(24,24),13,13
1170 PSET(37,8),14:DRAW"m41,34m34,32m37,8" ' Jib
1180 PAINT(36,25),14,14
1190 LINE(1,45)-(5,44),15 ' Wake
1200 GOTO 1200 ' Test GOTO
```

RUN to verify that the sailboat is constructed correctly. Then save it in the array BT this way:

```
1200 GET (0,5)-(45,45),BT ' Save in BT
1210 CLS
```

Devise your own test (using PUT) to show that the sailboat is indeed stored in BT.

A Lone Cloud

I figured that something else ought to be in the sky to balance the scene, so I drew a cloud. I used the graphics cursor movements of Figure 8.1 directly on the screen instead of drawing a cloud on graph

paper first. I did this by picking a starting point (PSET) and then entering successive moves in a DRAW command using the BASIC editor.

As the cloud took shape on the screen, I modified the growing character string to shape it to my liking. I used that GOTO trick to prevent the program from proceeding too far. This process took a little patience, but it was faster than drawing a cloud and plotting points.

Here are the clouds. Typing these DRAW statements will really test your patience and accuracy. Don't confuse l's and 1's.

```
1410 ' Cloud
1420 PSET(50,40),0 ' Start drawing in black
1430 DRAW"h3u4e3r6f2ng3e3r7f2bg4u1e2r3
     e1r6f1r4f2r2f2d2g1d1g1l2g1l2"
1440 DRAW"g2bh2f2d1g2l1g2l2g1l5ne2g2l6h2
     l1ne3g1l3g1l3h3l1h1u2e2u1e2r2"
```

Better test what you have so far with RUN. If the outline doesn't look like a cloud, or doesn't connect back on itself, check for errors. When all is okay, paint the cloud.

```
1450 PAINT (60,35),15,0 ' Color white
```

Okay, RUN and see what happens.

4. THE ELEMENTS OF ANIMATION

Now we've got a world and some characters to populate it. The next step is to give them life. We'll start simply and make certain that the ideas are properly developed.

With the setup or initializing work behind us, we want to perform the animation work with low order line numbers. That way, PCjr won't have to search far for a line number when it does a GOTO or IF statement. This saved time could be crucial. So let's begin these animation instructions at about line 100. This will leave us some addi-

tional space for other subroutines that may be even more sensitive to search time. But be sure to do the setup work first. We need

```
100 ' BEACH.BAS
110 GOTO 1010 ' Set up screen
```

We also have to get back to the beginning here, so insert

```
1700 GOTO 120 ' Start animation process
 120 ' Place figures at initial positions
```

Start Positions

Let's try the sea gulls first. To retrieve them from the array named GULL—recall we put them there with a GET (from screen) command—we use a PUT (on screen) command. We'll start them at the left margin of the screen, about a quarter of the way down, at (2,40), this way:

```
170 XS=2:YS=40:X0=XS:Y0=YS:JMP=1 ' Initial position on
    left
180 PUT(X0,Y0),GULL,XOR ' Place gulls on screen
190 GOTO 190 ' Test GOTO
```

XS and YS are the start positions; X0 and Y0 are positions from which we'll erase the gulls. As we'll see, X1 and Y1 are the (changing) new positions of the gulls. JMP is used later, but it must be initialized up front.

DELETE the test GOTO in line 1510 and RUN the program. Notice that the PUT command has almost the same form as the GET. However, there are several actions possible when a figure is placed on the screen with PUT. The XOR action is the only one we'll be concerned with here. It is specialized for animation so that when you PUT a figure twice onto a complex background, that background is replaced unaltered.

Marching Across the Screen

To test this XOR action, we'll need a clock to tick off time units T.

```
190 FOR T=1 TO 150 ' Start clock
270 NEXT T:END
```

Let's make the gulls move on a straight line across the screen, from left to right. To do this we'll just move them right three positions for each tick of the clock, and maintain the vertical position. The new positions are calculated from the start position this way:

```
210 X1=XS+3*T:Y1=YS
```

According to theory, we should restore the background if we PUT the gulls at (X0,Y0) with action XOR. Then we should PUT the gulls at the new position, (X1,Y1).

```
240 PUT(X0,Y0),GULL,XOR:PUT(X1,Y1),GULL,XOR
```

We must not forget to update X0,Y0 for the next time around.

```
250 X0=X1:Y0=Y1 ' Update last position of gulls
```

We should be able to see the gulls move now. Try RUNning. If all goes well, you'll run the birds off the right edge of the screen. You'll get an "Illegal function call" in line 240 because we're trying to PUT GULL beyond the edge of the screen.

More Realistic Flight

The birds are moving but to seem realistic, their flight should be more looping. They might enter the screen high, glide down toward the water, then fly up to gain altitude. They might come back and do the same thing in the opposite direction.

To simulate this action, you must substitute a different calculation for the one at line 210. Also, the action should be slowed down a bit. The calculation for X1 is okay. It's the value of Y1 that needs attention. There are many ways that Y1 could be modified to produce the action you're seeking. Here's the method I used: a parabolic curve. Type it in and RUN.

```
210 X1=XS+3*T:Y1=YS+2.3*T-.02*T*T
260 FOR K=1 TO 50:NEXT ' Slow the action
```

That's better, but there's still an "Illegal function call" to deal with, as well as the problem of bringing the gulls back in the opposite direction.

Back and Forth Motion

Suppose we let the sea gulls go off the screen, where they loop around and come back to trace a similar flight path over the water. We have to test for the edge of the screen to avoid the "Illegal function call." And there'll be a delay while they're looping back, off screen. Let's also make them do the same looping turn while they're off the left end of the screen.

```
150 ' Control gull movement
160 FOR K=1 TO 1000:NEXT ' Off screen delay on left
200 ON JMP GOTO 210,220 ' Select move right or left
210 X1=XS+3*T:Y1=YS+2.3*T-.02*T*T:GOTO 230 '
    Move right
220 X1=XS-3*T:Y1=YS+2.3*T-.02*T*T ' Move left
230 IF X1>290 OR X1<2 THEN PUT
    (X0,Y0),GULL,XOR:GOTO 280
280 ON JMP GOTO 290,160 ' Select move right or left
290 XS=290:YS=40:X0=XS:Y0=YS:JMP=2 ' Initial position
    on right
300 FOR K=1 TO 1000:NEXT:GOTO 180 ' Off screen delay on
    right
```

RUN to see if this addition works okay for you.

Line 230 is the pivotal test. It tells when the sea gulls reach the margin of the screen. When the margin is reached, the gulls are erased.

Now we have to decide how to reset the initial position of the birds. This is done in line 280. The variable JMP keeps track of the direction in which the birds are moving, 1 meaning to the right, 2 meaning to the left. If the birds have been moving to the right (JMP=1), the start position is reset in line 290; otherwise, as initially, in line 170 via line 160. At the same time, a 1,000-tick delay is performed (line 300 or 160) while the birds loop back off screen.

The calculation of X1 also depends on the direction of motion (lines 210 and 211), controlled by JMP in line 200.

Coordinating the Sailboat's Motion

Now to integrate the sailboat into the scene. Suppose we also start the boat from the left margin. We need to specify its initial position and draw it on the screen.

```
130 U0=2:V0=90 ' Boat's initial position
140 PUT(U0,V0),BT,XOR:GOTO 170 ' Place boat on screen
```

The sailboat travels much slower than the birds. Say we move the boat every nine ticks of our master clock, like this.

```
260 IF (T MOD 9)=0 THEN GOSUB 310 ELSE FOR K=1 TO
    50:NEXT
```

Every time T is a multiple of nine, we go to the subroutine at line 310 to move the boat. Otherwise we delay a bit before moving the birds.

The subroutine to move the sailboat follows the same principles we used to move the birds.

```
310 U1=U0+1.7:V1=V0-.04 ' Move boat
330 PUT(U0,V0),BT,XOR:PUT(U1,V1),BT,XOR
350 U0=U1:V0=V1 ' Update last position of boat
360 IF U0<=270 THEN RETURN ' Test for edge of screen
370 PUT(U0,V0),BT,XOR ' Reinitialize
380 PUT(X1,Y1),GULL,XOR
390 GOTO 130
```

Try RUNning to see how this integrated action progresses. And don't forget to SAVE BEACH.BAS.

The subroutine operates a little crudely. When the sailboat reaches the right margin, all it does is put the boat and the birds into their initial positions and continue as if we were starting from scratch. We'd need another boat image to tack on a right-to-left course. In this case you'd want the boat to get smaller as it sails toward the horizon.

5. PAINTING WITH PATTERNS

The scene still looks a little barren. Perhaps a little bathhouse tent on the beach would fill it in a bit. This will give me an opportunity to show you a couple more tricks you can do with PCjr. Just enter these additions to BEACH.BAS and try them out. The explanation will follow.

```
1510 ' Tent
1520 DRAW"bm236,195 c0 m-12,-23 m+12,-28 m+8,+8
     m-8,+43 m+48,-11"
1530 DRAW"m-16,-36 m-24,+4 m+8,-20 m-16,+12
     m+16,-12 m+0,-12"
1540 DRAW"m+20,+4 m-20,+8 m+16,+16 m-6,-8 m-10,-8"
1550 RED$=CHR$(68):GREEN$=CHR$(26) ' Define colors
1555 LBLUE$=CHR$(115):IVORY$=CHR$(127)
1560 TILE1$=STRING$(4,RED$)+STRING$(4,GREEN$) '
     Tile strings
1565 TILE2$=IVORY$+IVORY$+LBLUE$+LBLUE$
1570 PAINT(240,190),TILE1$,0 ' Tile tent panels
1580 PAINT(240,144),TILE1$,0
1590 PAINT(240,155),TILE2$,0
1595 PAINT(254,145),TILE2$,0
1600 PAINT(254,122),13,0 ' Pennant
```

Relative DRAWing Moves

The first trick you learned was that you may use DRAW for drawing lines to destination points that are a specified number of units away from a given starting point, not just to an absolute coordinate position on the screen. In line 1520 you are moved to absolute coordinate position (236,195) to begin with. Subsequent moves are made relative to the last position because each X and Y value has a plus sign (+) or minus sign (-) attached. Thus the first line to be drawn extends from (236,195) to (236-12,195-23) (i.e., absolute position [224,172]).

Defining a figure using relative moves is useful when you want to redraw a figure, perhaps of a different size and orientation, at another place on the screen. You can't really do this with GET and PUT.

Tiling

The second trick you saw was the striped PAINTing of the tent panels. We used a technique called tiling. The PAINT command permits you to specify a sequence of colors with which you want to fill an area. When the PAINTing process takes place, it looks at the sequence of colors you've indicated and fills the successive lines according to that sequence.

The colors themselves are defined by ASCII characters, so there are one hundred twenty-seven possibilities. Unfortunately, many of these colors are repeated, so the true number available for use is much smaller. There is, however, a variety of shades useful in producing shadowing effects that give the impression of three dimensions. I'll show you the available spectrum of shades in a moment.

Notice that the ASCII codes for a few colors are translated in lines 1550-1555. With them I define TILE1$ in 1560 as a string of four red and four green lines. These alternate stripes color the large facing panel and its diagonal (lines 1570-1580). TILE2$ is then defined as a sequence of two ivory and two light blue lines. They paint the other two tent panels.

The pennant is painted with color 13, but you might use any ASCII character or tiling mix you choose.

SAVE BEACH.BAS with the tent now.

The Tiling Rainbow

There's a little utility program in Figure 8.3 that you can use to select ASCII characters corresponding to colors with which you'd like to paint or tile. Type it in and try it out.

```
100 ' RAINBOW.BAS—Palette of TILE colors
110 CLEAR ,,,32768!:CLS:SCREEN 5:KEY OFF
120 COLOR 15,0
130 FOR J=1 TO 6
140 FOR I=1 TO 22
150 ROW=I:COL=6*(J-1)+1
160 N=I+22*(J-1)
170 IF N>127 THEN 290
180 TIL$=CHR$(N)
```

```
190 LOCATE ROW,COL:PRINT STR$(N)
200 X0=8*COL+25:Y0=8*(ROW-1)
210 X1=X0+13:Y1=Y0+8
220 VIEW (X0,Y0)-(X1,Y1),0,0
230 GOSUB 270
240 NEXT I
250 NEXT J
260 END
270 PAINT(12,4),TIL$,9
280 RETURN
290 GOTO 290
```

Figure 8.3: RAINBOW.BAS—Tiling Shades

One way you might use RAINBOW.BAS is to RENUMber it to some high line number (e.g., 10000) and SAVE it as an ASCII file—you know, with the ",A" at the end. Then, when you're developing a graphics program, MERGE RAINBOW.BAS into your growing program. When you need a color, just RUN 10000 <Enter> to temporarily display the available colors on the screen. A <Fn-Break> and LIST get you back to where the color character goes in your program. When you've finished your program, just DELETE RAINBOW.BAS and SAVE the rest.

6. INTEGRATING SUMMER.BAS AND BEACH.BAS

The Big Merge

Our video rock production is almost completed. We just have to add the musical soundtrack, which we prepared in the last chapter. If you're sure you've SAVEd BEACH.BAS, we'll combine the programs.

Type: MERGE "SUMMER.BAS" <ENTER>

There are only a couple of connections to make at this point to coordinate the action with the sound. First, redirect the starting point from loading in the music at line 5025 to line 1000, where the graphics are initialized. So make this change:

```
20 GOTO 1000 ' Initialize
```

Then, instead of jumping back to line 120 from line 1700 when the graphics initialization is completed, we should go on and initiate the music.

Type: DELETE 1700 <ENTER>

Finally, when the music has been initialized and the PLAY interrupt set, we can initialize the graphics activity. So change line 6060 to read

```
6060 GOTO 120 ' Start graphics activity
```

As we saw earlier, with the PLAY interrupt set, whenever the music buffer is approaching empty, PCjr will jump from what it's doing with graphics (in this case) to the subroutine at line 40 to insert another measure in the music buffer.

SAVE the integrated program as BEMUSE.BAS. Then RUN it to see what happens.

Where'd the Boat Go?

Did you see the sailboat disappear at times—and the sea gulls too? And how about the way the action comes to a halt on certain music passages? We can do something about the boat disappearing, but I don't know how to fix the frozen action.

The boat disappears because an interrupt to fill the music buffer occurs while the first half of line 330 is erasing the figure. There's a wait then before the redraw action takes place.

To prevent the boat from disappearing, we've got to turn off the PLAY interrupt just before erasing the boat, and then turn it back on just after the redraw. Try these additions to the program:

```
320 PLAY OFF
340 PLAY ON
```

Theoretically we should also be able to prevent the birds from disappearing using this same technique. You might try adding these instructions:

```
235 PLAY OFF
245 PLAY ON
```

The birds don't disappear now, but they're not halfway across the screen before the music cuts out altogether.

This problem puzzled me for a long time. I finally guessed that the music buffer must have fewer than four notes left—the ON PLAY limit—by the time PLAY ON occurs, making it impossible for the PLAY interrupt to function. Inserting the following line seems to fix the problem.

```
243 IF PLAY(0)<=4 AND PLAY(1)<=4 AND PLAY(2)<=4
    THEN GOSUB 40
```

This line cranks up the music again.

7. BREATHING LIFE INTO A DOLL

You may think we cheated a bit in calling BEMUSE.BAS an illustration of animation. After all, we merely used a couple of pictures and moved them about on the screen. But that's half the battle, isn't it?

You've seen enough about how Walt Disney and other animators make cartoon movies using hundreds of overlays called cels (with one l) to laboriously construct their productions one frame at a time. We can do something of the sort right on PCjr. The process doesn't involve any really new ideas—just repetitions of old ones.

To illustrate the animation process, I'll replicate the figure of a marionette that one of my girls gave me. It hangs in the window next to the computer. This cute little man is dressed in an old-fashioned way, with a bushy mustache, red nose and tall hat. His arms and legs are articulated so that if you pull on the string that hangs down, his arms and legs fly up in the air and he does a funny dance.

The animation job in ARKANSAS.JIG (Figure 8.4) is fairly straightforward. However, it involves drawing and storing eighteen cels—three left and three right arms, and six left and six right legs, all articulated differently. This work takes up most of the initialization section of the program (line 480 on).

To get a reasonable sense of proportion, I traced the marionette on a piece of graph paper first. Drawing the body was the most detailed part of the job. I did all the drawing directly on the screen, building each DRAW command character by character.

I really had to draw only one arm and a single leg of three parts to get the eighteen necessary cels. As you read through ARKANSAS.JIG, you'll see that all the arms and legs are generated by rotating and placing the parts appropriately. Placing is the toughest part. It took a lot of experimentation. While developing the figures, I had RAINBOW.BAS resident at line 10000 so the available shades would be conveniently at my disposal.

As you may have guessed, the music that plays to the marionettes jig is "Arkansas Traveler." There's even a mouth-harp to keep the rhythm—defined in line 1020.

The mainline (starts at line 380) is very short. It involves starting the play of one measure of the music, and then making appropriate changes in the positions of the arms and legs—random patterns that exclude repetitions. The actual drawing is done in the many little subroutines, almost all identical, at the head of the program. The old cel is erased before the new one is laid down.

If you're going to do much animation, you'd be well advised to invest in a digitizing pad so you can readily trace things on the screen and save them in arrays (on disk) or in DRAW or DATA statements. You could really have fun, for example, tracing successive frames of Eadweard Muybridge's many photographs of people and animals in motion.

```
100 ' ARKANSAS.JIG
110 GOTO 490 ' Go initialize
120 ON ORA GOSUB 190,200,210 ' Select arms to place
130 ON NRA GOSUB 190,200,210:RETURN
140 ON OLA GOSUB 160,170,180
150 ON NLA GOSUB 160,170,180:RETURN
160 PUT(113,80),LU,XOR:RETURN ' Place arms
170 PUT(102,97),LM,XOR:RETURN
```

```
180 PUT(111,109),LD,XOR:RETURN
190 PUT(160,78),RU,XOR:RETURN
200 PUT(158,99),RM,XOR:RETURN
210 PUT(158,110),RD,XOR:RETURN
220 ON ORL GOSUB 260,270,280,290,300,310 '
    Select legs
230 ON NRL GOSUB 260,270,280,290,300,310:RETURN
240 ON OLL GOSUB 320,330,340,350,360,370
250 ON NLL GOSUB 320,330,340,350,360,370:RETURN
260 PUT(148,135),RLD1,XOR:RETURN ' Place legs
270 PUT(128,135),RLD2,XOR:RETURN
280 PUT(148,133),RLM1,XOR:RETURN
290 PUT(148,133),RLM2,XOR:RETURN
300 PUT(144,131),RLU1,XOR:RETURN
310 PUT(144,131),RLU2,XOR:RETURN
320 PUT(124,135),LLD1,XOR:RETURN
330 PUT(124,135),LLD2,XOR:RETURN
340 PUT(87,132),LLM1,XOR:RETURN
350 PUT(87,133),LLM2,XOR:RETURN
360 PUT(64,131),LLU1,XOR:RETURN
370 PUT(64,131),LLU2,XOR:RETURN
380 ' Mainline
390 IF PLAY(0)>2 THEN 390 ' Hold music to action
400 PLAY"t=T;xa$(I);","t=T;xr$;" ' Load next measure
410 L=L+1:IF L>16 THEN L=1 '
    Keep track of measure
420 OLA=NLA:NLA=1+((NLA+2+SGN(RND-.5))
    MOD 3) ' Arm/leg positions
430 ORL=NRL:NRL=1+INT(6*RND):IF NRL=ORL
    THEN NRL=1+(NRL MOD 6)
440 GOSUB 140:GOSUB 220 ' Go place left
    arm & right leg
450 ORA=NRA:NRA=1+((NRA+2+SGN(RND-.5))
    MOD 3)
460 OLL=NLL:NLL=1+INT(6*RND):IF NLL=OLL
    THEN NLL=1+(NLL MOD 6)
470 GOSUB 120:GOSUB 240:GOTO 390 ' Go place right
    arm & left leg
480 ' Initialize
```

```
490 CLEAR ,,,32768! ' Video buffer space for SCREEN 5
500 RF=3.141593/180 ' Radians/degree
510 SCREEN 5:COLOR 14,0
520 DIM MAN(500),RU(200),RM(200),RD(200)
530 DIM LU(200),LM(200),LD(200)
540 DIM LLU1(500),LLU2(500),LLM1(500),
    LLM2(500),LLD1(500),LLD2(500)
550 DIM RLU1(500),RLU2(500),RLM1(400),
    RLM2(500),RLD1(300),RLD2(400)
560 KEY OFF:CLS
570 ' Construct arms
580 A=45:GOSUB 1470
590 GET(50,20)-(80,50),RU
600 CLS:A=0:GOSUB 1470
610 GET(50,40)-(90,60),RM
620 CLS:A=-45:GOSUB 1470
630 GET(50,50)-(80,75),RD
640 CLS:A=135:GOSUB 1470
650 GET(20,25)-(50,50),LU
660 CLS:A=180:GOSUB 1470
670 GET(10,40)-(50,60),LM
680 CLS:A=225:GOSUB 1470
690 GET(20,50)-(50,80),LD
700 ' Construct right legs
710 CLS:A=0:B=0:GOSUB 1540:GOSUB 1630
720 GET(90,105)-(120,160),RLD1
730 CLS:A=0:B=65:GOSUB 1540:GOSUB 1630
740 GET(70,105)-(120,160),RLD2
750 CLS:A=35:B=0:GOSUB 1540:GOSUB 1630
760 GET(90,105)-(155,150),RLM1
770 CLS:A=35:B=65:GOSUB 1540:GOSUB 1630
780 GET(90,105)-(155,155),RLM2
790 CLS:A=65:B=0:GOSUB 1540:GOSUB 1630
800 GET(90,100)-(180,130),RLU1
810 CLS:A=65:B=65:GOSUB 1540:GOSUB 1630
820 GET(90,100)-(180,140),RLU2
830 ' Construct left legs
```

```
 840 CLS:A=0:B=0:GOSUB 1540:GOSUB 1660
 850 GET(80,105)-(110,160),LLD1
 860 CLS:A=0:B=-65:GOSUB 1540:GOSUB 1660
 870 GET(80,105)-(130,160),LLD2
 880 CLS:A=-35:B=0:GOSUB 1540:GOSUB 1660
 890 GET(45,105)-(110,150),LLM1
 900 CLS:A=-35:B=-65:GOSUB 1540:GOSUB 1660
 910 GET(45,105)-(110,155),LLM2
 920 CLS:A=-65:B=0:GOSUB 1540:GOSUB 1660
 930 GET(20,100)-(110,140),LLU1
 940 CLS:A=-65:B=-65:GOSUB 1540:GOSUB 1660
 950 GET(20,100)-(110,140),LLU2
 960 CLS:GOSUB 1250 ' Build body
 970 NLL=1:GOSUB 320:NRL=1:GOSUB 260
 980 NLA=3:GOSUB 180:NRA=3:GOSUB 210
 990 ' ARKANSAS TRAVELER
1000 DIM A$(16)
1010 SOUND ON:BEEP OFF
1020 R$="o0mnl8fp8fp8" ' jew's-harp
1030 A$(1)="o3ms l16fagf l8dd"
1040 A$(2)="o3ms l8ccff"
1050 A$(3)="o3ms l8ggaa"
1060 A$(4)="o3ms l16gagf mnl8dc"
1070 A$(5)=A$(1)
1080 A$(6)="o3ms l8cc mnl4f"
1090 A$(7)="o4ms l16fef<b>cfed"
1100 A$(8)="o4ms l16c<b-ag mnl8f o4mll16ab-
1110 A$(9)="o5ms l16c<b-a>c<b-agb-"
1120 A$(10)="o4ms l16agfage mnl8c"
1130 A$(11)="o4ms l16fefage mnl8c"
1140 A$(12)="o4mn l16fefa l8g msl16ab-"
1150 A$(13)=A$(9)
1160 A$(14)="o4ms l16agfage l8c"
1170 A$(15)="o4ms l16fefcdfed"
1180 A$(16)="o4ms l16c<b-ag mnl8f p8"
1190 T=122:V=8:L=2
1200 PLAY"t=T;v=V;mb","t=T;v1;mb"
1210 PLAY"t=T;xa$(1);","t=T;xr$;"
```

```
1230 LOCATE 2,14:PRINT"ARKANSAS JIG"
1240 GOTO 390
1250 ' Draw body
1260 PSET(150,100),7
1270 DRAW"l10 g5 d6 f5 r20 e5 u6 h5 l10"
1280 PAINT STEP (+0,+5),CHR$(24),7
1290 DRAW"bm+0,+15"
1300 DRAW"c7 l10 g5 d6 f5 r20 e5 u6 h5 l10"
1310 PAINT STEP (+0,+5),CHR$(24),7
1320 DRAW"c7 bm-10,-5 u4 bm+20,+0 d4 bl10 bu2"
1330 PAINT STEP (+0,+0),14,7
1340 DRAW"c4 bu18 br5 u3 l10 d3 r10 u1 l10" ' Neck
1350 CIRCLE STEP (+5,-11),11,12,,,,.9 ' Head
1360 PAINT STEP (+0,+0),CHR$(92),12
1370 CIRCLE STEP (+0,+2),1,4 ' Nose
1380 PAINT STEP (+0,+0),4,4
1390 CIRCLE STEP (+0,-9),14,6,,,.2 ' Hat
1400 CIRCLE STEP (+0,-16),9,6,,,.2
1410 DRAW"c6 bl9 d13 br18 u13"
1420 DRAW"c2 bl9 d13"
1430 PAINT STEP (+0,+0),CHR$(8),6
1440 DRAW"c0 bd8 bl6 r2h1g1 br12 l2e1f1" ' Eyes
1450 DRAW"bl8 bd6 g3 r1 e3 br3 f3l1h3" ' Mustache
1460 RETURN
1470 PSET(50,50),0 ' Draw arm
1480 DRAW"ta=A;"
1490 DRAW"c7 br8 u2 e2 r4 f2 e1r2 e1 r4 f1 r3 f1 r3 d2 l3
     g1 l3 g1 l4 h1 l2h1 g2 l4 h2 u2"
1500 PAINT STEP (+5*COS(RF*A),-5*SIN(RF*A)),
     CHR$(24),7
1510 DRAW"c12 br19 e2 r2 f1 r2 f1 d2 g1 l2 g1 l2 h2
     u1" ' Hand
1520 PAINT STEP (+3*COS(RF*A),-3*SIN(RF*A)),
     CHR$(92),12
1530 RETURN
1540 PSET(100,100),0 ' Draw leg
```

```
1550 DRAW"ta=A;"
1560 DRAW"c7 bd8 r2 f3 d6 g1 d4 g1 d3 g1d3 l5 u3 h1u3
     h1u4h1u6 e3 r2"
1570 PAINT STEP (+5*SIN(RF*A),+5*COS(RF*A)),
     CHR$(24),7
1580 C=A-B
1590 DRAW"c15 bd17 ta=C; r2 f2 d6 g1 d5 g1 d3 l3 u3
     h1 u5 h1 u6 e2 r2"
1600 PAINT STEP (+5*SIN(RF*C),+5*COS(RF*C)),15,15
1610 RETURN
1620 ' Draw right foot
1630 DRAW"c6 bd14 r6 f1 r4 f1 r2 f1 d3 g1 l18 h2 u2 e1
     u1 e1 r3"
1640 PAINT STEP (+3*SIN(RF*C),+3*COS(RF*C)),
     CHR$(8),6
1650 RETURN
1660 ' Draw left foot
1670 DRAW"c6 bd14 l6 g1 l4 g1 l2 g1 d3 f1 r18 e2 u2 h1
     u1 h1 l3"
1680 PAINT STEP (+3*SIN(RF*C),+3*COS(RF*C)),
     CHR$(8),6
1690 RETURN
```

Figure 8.4: ARKANSAS.JIG—Illustration of Animation

APPENDIX

A GRAPHICS SCREEN-DUMP PROGRAM FOR THE IBM COMPACT PRINTER

I think owners of the IBM Compact Printer should have a way of printing pictures. The screen-dump program in Figure A permits you to print pictures drawn in SCREEN 1 mode. Type it in carefully and SAVE it as MRDUMP.BAS (for medium-resolution screen dump).

To use MRDUMP you must first generate a picture on SCREEN 1 and SAVE it in a picture file. We did this kind of thing with BSAVE in Chapter 6, sections 3 and 9. When you run MRDUMP, you're asked to type in the name (filespec) of the picture file. Then turn on the IBM Compact Printer and hit <Space>.

Try printing VALNTINE.PIC, which we created in Chapter 6.

The printing process is slow, so be patient if it doesn't start immediately. When printing stops, hit <Esc> to start over with another picture. Quit with <Fn-Break>.

NOTE: If you get funny stuff printing out, you might turn off the printer to reset it, and then begin the printing process all over again from scratch.

Working with printers is messy. The printing codes for the IBM Compact differ significantly from those of the standard IBM dot matrix printer (essentially an Epson) that's matched to the IBM PC and for which standard graphics screen-dump programs exist within DOS 2.0 and 2.1. You can see these printing codes toward the end of the pamphlet "IBM PC Compact Printer Installation and Operating Instructions." You should have it in your *Guide to Operations*.

Studying the printer codes in the pamphlet and seeing how they are used in Figure A will give some insights into how you might do other interesting things with them (e.g., double-width and compressed characters).

```
100 ' MRDUMP.BAS—Med Res Screen Dump
110 CLEAR
120 SCREEN 1,0:COLOR 1,1:KEY OFF:CLS
130 DEFINT A-Z ' Integer arithmetic for speed
140 PRINT:PRINT TAB(8);"* * * SCREEN DUMP * * *"
150 PRINT:PRINT:PRINT"Print Medium Res Screen on
    IBM Compact"
160 PRINT:PRINT:PRINT"Key filespec of picture: ";
```

```
170 INPUT"",NA$
180 PRINT:PRINT:PRINT"Hit <SPACE> when ready to
    print"
190 PRINT:PRINT"Hit <Esc> for new picture"
200 A$=INKEY$:IF A$<>" " THEN 200
210 CLS
220 BLOAD NA$
230 OPEN "LPT1:" AS #1 ' Open line printer
240 WIDTH #1,255
250 PRINT#1,CHR$(24); ' Initialize printer
260 PRINT#1,CHR$(27);CHR$(48); ' Set line spacing
270 DEF SEG=&HB800 ' Start of video buffer
280 FOR I=0 TO 79 ' Column count
290 PRINT#1,CHR$(27);"K";CHR$(144);CHR$(1);' Set
    bit graphics
300 P=I+&H1EF0 ' Start position (last row)
310 FOR N=1 TO 100 ' Double row count
320 ROW1=PEEK(P):ROW2=PEEK(P+&H2000)
330 PRINT#1,CHR$(ROW2);CHR$(ROW2);
    CHR$(ROW1);CHR$(ROW1);' Double density printing
340 P=P-80 ' Next start row (count backwards)
350 NEXT
360 PRINT#1,CHR$(13);CHR$(10); ' Carriage return &
    line feed
370 NEXT
380 PRINT#1,CHR$(13);CHR$(24);CHR$(27);
    CHR$(50);:CLOSE
390 A$=INKEY$:IF A$<>CHR$(27) THEN 390
400 GOTO 110
```

Figure A: MRDUMP.BAS—Medium-Resolution Screen Dump

A

B

C

D

E

F

R

S

T

V

W

Creative Computing

Now you can express yourself creatively with the IBM PCjr! With humor and patience, *Sound & Graphics for the IBM PCjr* teaches you the fundamentals of the BASIC programming language and clever programming tricks so that you can make the most of the PCjr's sound and graphics capabilities. You'll learn to use BASIC commands such as PAINT, BEEP, COLOR, NOISE, LINE, PLAY, SOUND, and DRAW, to name just a few of the techniques that generate impressive music and spectacular graphics.

With this book and your PCjr, you can create:

SOUND

SOUND EFFECTS and BIRD CALLS
TWO- AND THREE-PART HARMONIES
FAMILIAR MELODIES
ORIGINAL MUSICAL COMPOSITIONS

GRAPHICS

GEOMETRIC CONSTRUCTIONS
ABSTRACT PATTERNS
NATURALISTIC PICTURES
ANIMATED CARTOONS

The text is chock-full of instructions and program listings that will challenge you to stretch your imagination while you have fun with your PCjr.

SOUND & GRAPHICS FOR THE IBM PCjr

by James E. Kelley, Jr.,
author of *The IBM/PC Guide,*
The IBM/PC & 1-2-3 and *The IBM/PC & Business Software.*

A Banbury Book
ISBN 0-88693-068-5
Distributed to bookstores by The Putnam Publishing Group
$14.95
Cover printed in U.S.A.