The New Apple II User's Guide
David Finnigan

ISBN-10: 0615639879

ISBN-13: 978-0-615-63987-1

This book was prepared using Adobe software products
on Macintosh computers. The display face is Futura.
Text is in Bookman Old Style.

Published by Mac GUI

Chapter 3 : Beginning BASIC

This chapter will teach you the basics of programming, starting off with a few fundamental ideas. You will then learn about the two different types of BASIC on the Apple and how to distinguish them. The rest of the chapter will then be devoted to learning enough commands to make a simple, yet functional BASIC program.

If you wish to learn BASIC, it is imperative that you follow along with the examples in this book, and then experiment on your own. You cannot learn a programming language by just reading and memorizing; you must practice it too.

About BASIC

If you're interested in programming your Apple, then BASIC is likely the first language that you'll use. It comes built into every Apple computer. BASIC stands for Beginner's All-purpose Sym-

bolic Instruction Code and was developed at Dartmouth University in the 1960s. Using English words, BASIC is easy to learn, even by those with no previous programming experience.

If you're used to other high-level languages such as C, Java, or Pascal, then you'll find that BASIC is quite a bit different. It is not a structured language. Instead, BASIC uses line numbers to separate program statements.

Programs, Statements, and Commands

In case you are new to programming computers, there are a few basic terms to learn first. What you will be doing in this chapter is writing what is known as a computer program. A *program* is a set of instructions which are followed exactly by the computer to carry out a task. The shortest program consists of just one instruction, but as you might guess, it wouldn't be very useful. The longest programs for the Apple contain many thousands of instructions.

Taken by themselves, these instructions are formally known as *statements*. An example of a statement is one which multiplies two numbers and stores the result. A further statement could then print the stored product on the screen. A statement makes up just a small part of a program, but if even one critical statement is wrong or missing, the program will not perform as desired.

A statement usually includes a *command*, an operation for the computer to perform. In BASIC, a command is always a single English word or abbreviation. Take, for example, a command to display a number that was stored earlier. By itself, this command is not a statement, since it is missing a critical component: the number to be displayed. It has no meaning otherwise. Once these pieces are brought together, the command becomes part of a meaningful statement.

Syntax

Just like human languages such as English and French have rules of grammar which should be followed, so too do computer

languages have such a grammar, known as *syntax*. The main difference is that if you make a small mistake while speaking a language, odds are that the other person will be able to compensate and still understand what you mean. The Apple can do no such thing. To a computer, there are only two results: right and wrong. Either you have entered a command correctly, or you have not.

Fortunately for you, BASIC is based on commands which are common English words or abbreviations. Therefore, with the exception of typographical errors, you should have little or no trouble with syntax errors after you have had some practice.

The Two BASICs

In the course of the Apple II's life, two major versions of BASIC were released. The first, available on the original model of Apple II, is known as *Integer BASIC*. If you recall from Chapter 1, its prompt is a > character. Integer BASIC was written by Steve Wozniak mostly for gaming purposes, but is not limited to just games. As a result of it only handling integer (whole) numbers, it is quite speedy. Starting with the Apple II Plus, Integer BASIC no longer came built-in as a standard option. Therefore, Integer BASIC is less commonly used.

The other version of BASIC is known as *Applesoft*. It is somewhat slower than Integer BASIC, but has more features, such as support for floating point (decimal) numbers, and high-resolution graphics. These features make it suited to many scientific, business, and mathematical applications. Applesoft is built in to the Apple II Plus and all later models and is far more common than Integer BASIC. It is available as an option on the original model of Apple II. The Applesoft prompt character is a] symbol.

Differences between the two versions of BASIC will be pointed out where they exist. In the absence of such documentation, one may assume that the two BASICs behave identically.

Getting into BASIC

Before you can start telling your Apple what to do in the BASIC language, you need to get into the BASIC mode. The process to do so varies by type of Apple. First, if any disk drives are connected to your Apple, make sure that there aren't any disks in them, and turn on your monitor (or television) and Apple II. While first learning BASIC, you won't be using the Apple disk system.

If you have an original model Apple II which starts in the Monitor, press CONTROL-B and then RETURN to enter Integer BASIC. As soon as you press RETURN, a > prompt will appear.

For all other models of Apple II, press CONTROL-RESET to enter Applesoft BASIC. The] prompt will appear.

All examples shown will use the square bracket prompt of Applesoft BASIC, unless the information applies only to Integer BASIC, in which case the angle bracket will be shown.

Switching to Integer BASIC

To switch from Applesoft to Integer BASIC, type INT, followed by RETURN. The Applesoft prompt will be replaced with the Integer BASIC greater-than bracket (>). Integer BASIC must be available on your model of Apple in one of three ways:

- Built-in to the motherboard ROMs
- Located on an Apple Firmware (ROM) card
- Loaded from DOS 3.3 to an Apple Language Card.

The first method is only available on the original model of Apple II. The second method is restricted to the Apple II Plus. Finally, the third method, using a Language Card, can be used with any model of Apple. Integer BASIC cannot be used with ProDOS on any model of Apple. See Chapter 7 for an explanation of DOS 3.3 and ProDOS.

Use the FP command (which is short for floating point) to return to Applesoft.

Upper Versus Lowercase

Unfortunately, there can be quite a bit of confusion for the new Apple II user as whether to use uppercase or lowercase letters when typing commands.

This problem arose in 1983, with the introduction of the Apple IIe. Before then, the two earlier models of Apple, the II and II Plus, had keyboards which could only type capital letters. The SHIFT key existed only to produce the upper symbol on a key cap with two symbols, such as the P key, or the number keys. There was absolutely no way to get lowercase letters without performing a hardware modification to the computer. Therefore, the vast majority of software operated in, and expected commands in, uppercase only.

Then everything changed when the Apple IIe was released. It was the first model of Apple to have a built-in keyboard capable of typing both lower and uppercase letters on screen. However, the generation of software written before it expected only uppercase commands. Even the BASIC included with the original model of Apple IIe expected commands to be in uppercase. Therefore, the user of the Apple IIe typically operated it with the CAPS LOCK key depressed.

Later on, when further models of Apple were introduced, lowercase became standard, and thus more common. The built-in BASIC in the later revisions of Apple IIe, as well as all revisions of the IIc, IIc Plus, and IIgs, can accept commands in either upper or lowercase. To summarize, the only model of Apple whose keyboard can generate lowercase letters, but whose BASIC only accepts uppercase is the standard, unenhanced Apple IIe.

To make things simpler for all models of Apple, this book will show examples in uppercase only. It is recommended that users of an unenhanced Apple IIe make sure that the CAPS LOCK key is depressed.

Your First BASIC Command

Now it's time to learn and type your first BASIC command. If you have an unenhanced Apple IIe (that prints "Apple][" at startup), be sure that the CAPS LOCK key is depressed. Now, type:

```
]NEW
```

and press RETURN. Remember that either the angle bracket (>) or square bracket (]) is provided on screen for you; do not type it yourself. The bracket is merely shown in these examples to illustrate what the final screen line should look like. If everything went correctly, it should appear as if nothing happened. The Apple will print another prompt and flash the cursor at you.

However, something in fact did happen. The command NEW tells the Apple that you intend to start a new BASIC program. If any BASIC program was already in memory, it will be discarded. The Apple can only have one BASIC program in its RAM (memory) at a time. If you forgot to type NEW, and an old program was already in memory, your new program could get mixed in with the old one. Since there wasn't already a program in the Apple this time, typing NEW here was a mere formality, but it's a good habit to get into.

Errors

The Apple is not shy when it comes to telling if you've done something that it cannot understand. If you enter a command and the Apple beeps and shows either:

```
*** SYNTAX ERR
```
(Integer BASIC)

or

```
?SYNTAX ERROR
```
(Applesoft)

on the screen, then it means that you entered something which the Apple couldn't comprehend, known as making a *syntax error*. If you've been experimenting on your own, then you have probably

already received this error message. BASIC on the Apple only understands a limited, predefined set of words. It will not recognize any synonyms or other command words. Once you've learned the BASIC commands and functions, the most likely cause of a syntax error is a typing mistake. Simply retype the line and try again.

On the other hand, if you make a logic error, the Apple will never notice. It can only detect syntax errors. As an example, if you mean to multiply two numbers but type an addition sign instead of multiplication, it is up to you to detect and correct this mistake.

Printing Text on Screen

If you weren't too impressed with your first command, this next one should be better: it at least has visible results! One of the most common ways that computers communicate with humans is to display text, whether it be the results of a calculation, or a prompt to enter in some data, or merely instruction on how to use the program.

In the case of BASIC, the command to display something on the screen is called PRINT. The name of this command is a throwback from the old days when computers would display output on an actual line-printer or teletype, instead of a display screen.

To try out your first PRINT command, type the following:

```
]PRINT "HELLO, WORLD!"
```

and press RETURN. The Apple should respond by displaying the phrase "HELLO, WORLD!" on the screen (without the quotes). If not, check your typing and reenter the line (remember that you don't need to type the] symbol).

Congratulations, you've just written your first line of BASIC! Notice that only the text in between the quotes is displayed. The quotes themselves are not part of the subsequent output.

Now enter this line:

```
]10 PRINT "HELLO, WORLD!"
```

When you press RETURN, the Apple *won't* print anything on the screen. What happened? Here's what this line, or statement, means. To begin, it has two parts: a line number, and a command.

The line number in this example is 10. All BASIC programs must have line numbers, which can range from 0 to 32767 in Integer BASIC, and 0 to 63999 in Applesoft. The Apple will run a program from the lowest line number to the highest, unless told otherwise by a command.

You may be wondering about the first two commands you entered, NEW and PRINT, both of which did not have a line number. This leads us to the next part.

Immediate Versus Deferred Execution

When you typed the NEW command, the Apple II instantly executed it by clearing out any existing program. In other words, the command was executed immediately. Likewise, with the first PRINT statement, the line was executed as soon as you pressed RETURN. This is called *immediate execution*.

However, when you typed the line 10 PRINT "HELLO, WORLD" the Apple did not actually execute that command. Instead, the computer saved it in memory, allowing you to execute it and any other commands you entered at a later time. This is an example of *deferred execution*: storing a set of statements (also known as a program) to be executed later. Every proper BASIC program is made up of one or more numbered lines of deferred execution commands. Sometimes you will see the term "programmed mode" used in place of "deferred execution"; they mean the same thing.

In many cases, BASIC commands can be used in both the immediate and deferred execution modes. The PRINT command is one example. Other commands do not make sense if used in the opposite mode, and the Apple will not allow them.

The Rest of the PRINT Command

Now, let's take a look at the final two parts of the PRINT command which you just entered.

After the line number comes the actual command itself. In this case, it's PRINT, telling the Apple to display some information. How does it know what to display? Well, the third and final part of the statement is the answer. The argument to the command is "HELLO, WORLD!" The Apple takes everything between the quotation marks and displays it on the screen when this command is executed.

What happens when you type a PRINT command with no argument? Try it. The Apple just prints a blank line.

Printing Numbers and Calculations

The PRINT command can do more than just write text onto the screen. It can also display numerals and mathematical expressions, such as in the following examples:

```
]PRINT 10 - 3
7

]PRINT 2*8
16

]PRINT -115 + 200
85

]PRINT 64/2
32

]PRINT "12*6"
12*6
```

Pay close attention to the last line. If you type an expression in quotation marks, the Apple does not evaluate it. Instead, it just

prints it as if it were any other ordinary text. You must remember this distinction, unless you really do want to print an expression and not the results of it.

In every other case, the Apple prints the results of the mathematical expression on the next line. However, there are some limitations to what the Apple can compute in BASIC mode.

Integer BASIC supports a numerical range of -32767 to +32767. Therefore, attempting a calculation such as the following will result in an error:

```
>PRINT 32767+1
*** >32767 ERR

>PRINT 5/0
*** >32767 ERR
```

As its name suggests, Integer BASIC cannot deal with non-integral numbers (decimal numbers having a fractional part, such as 2.5). The following examples illustrate this limitation:

```
>PRINT 5/2
2

>PRINT 1 + 0.5
*** SYNTAX ERR
```

In the first case, the Apple discards the remainder instead of displaying the true value of 2.5. The number is not rounded. In the second case, Integer BASIC cannot accept decimals, so the entire line is rejected as having a syntax error.

Applesoft does not deal with fractions such as 1/4, but it can work with the decimal equivalent, such as 0.25. Furthermore, it is limited to a total of nine significant digits in all fractional numbers which it displays. This includes all significant digits after the decimal point, and not just the whole number part.

```
]PRINT 3.141592653589793
3.14159266
```

Despite this limitation, Applesoft is superior to Integer BASIC in that it can handle numbers of much greater magnitude than 32,767.

```
]PRINT 3.141 * (500 ^ 2)
785250.002

]PRINT 785250.002 ^ 3
4.84198945E+17
```

Applesoft represents large numbers using scientific notation, as shown in the previous example. In case you are unfamiliar with this form of notation, one simply takes the value after the *E* (known as the exponent*)* and moves the decimal point by that many places. A positive number means that the decimal point is to be moved to the right, whereas a negative number dictates that the decimal point be moved to the left.

Thus, 4.84198945E+17 is written in standard form notation as 484,198,945,000,000,000.

The range for exponents is E-38 to E+38.

Combining Calculations and Characters

If you would like a PRINT line to show both characters, such as some text, as well as a calculation, you may separate the two using a semicolon, like so:

```
]PRINT 365-120;" DAYS LEFT IN THE YEAR"
245 DAYS LEFT IN THE YEAR
```

You can append many more characters and calculations to a single line. Remember that letters and numbers which should be displayed literally (“as-is”) must be placed within quotation marks.

```
]PRINT "I AM ";365*28;" DAYS OLD AND ";6
*12;" INCHES TALL."
I AM 10220 DAYS OLD AND 72 INCHES TALL.
```

In Applesoft, using the semicolon to join parts in a PRINT statement is optional.

A Shorter PRINT Command

Applesoft has a convenient shortcut for PRINT, a command that gets used fairly often. If you want to cut down on the amount of typing you do, you'll appreciate this alternative. Anywhere you want to type the PRINT command, type a single question mark (?) instead. You'll save four letters of typing. Here are a few examples of its usage:

```
]?"VENI, VEDI, VICI"
VENI, VEDI, VICI

]?10*24
240
```

The question mark syntax can be used in both deferred and immediate mode execution. When you LIST your program (this command will be explained shortly), the question mark will be converted to a PRINT command.

Finishing the Program with END

Now that you have a one-line program, it is time to finish it off with a final line. The END command is required in Integer BASIC, but is optional in Applesoft. Whenever the Apple encounters END, it stops running your program and returns to the BASIC prompt.

To add an END command to your first program, type

```
]20 END
```

and press RETURN. You now have a simple, yet complete BASIC program.

If you don't put an END command in your program, then Applesoft will stop when it runs out of lines to execute. Integer BASIC

will also stop under the same condition, but it will also return a *** NO END ERR message.

Running Your Program

So far you have a program with two lines of instructions. Can you guess what command to enter to run it and see the results?

Type

```
]RUN
```

and press RETURN. By now, you should realize that every line typed on the Apple must be completed with a tap of the RETURN key. In future, this direction will not be explicitly given. If the Apple does not appear to be responding to your command, perhaps you have forgotten to press RETURN!

If all went well, your efforts will be rewarded with the line "HELLO, WORLD!" and another command prompt. You can type RUN again if you wish, and the Apple will execute the program once more. The Apple II won't ever get tired of running your program over and over.

If nothing happens, it may be that you did not type in line 10 from a few sections ago, or that you typed NEW at some point, thus obliterating it. To recover, type the entire program again:

```
10 PRINT "HELLO, WORLD!"
20 END
```

And then RUN it again:

```
]RUN
HELLO, WORLD!

]
```

Examining the Program

By now, there is likely a whole mess of program lines and output lines on your screen. To make matters worse, some of the program lines could have changed from what is shown on the screen if you retyped them. The solution is to ask the Apple to print the current *program listing*. The command to do so is called LIST. Try it now to see what your program looks like:

```
]LIST
```

The LIST command always prints what the program currently looks like, and always in ascending line order. Even if one were to type line 20 first, then line 10, internally, the Apple rearranges those lines in correct order. The LIST command always reflects the proper order.

Often times, a program will contain more lines than may be shown on the screen at one time. The LIST command can accept some arguments to limit its output and solve this problem.

To list just one line, type LIST and the desired line number, as demonstrated:

```
]LIST 10
```

To list a sequential range of lines, separate the starting and ending line numbers with a comma as in this example:

```
]LIST 10,20
```

In Applesoft, two additional listing modes are available.

To list all lines starting from the beginning up to a certain line number, type a comma, then the final line number. The Apple will list all program lines up to and including the line number that you entered.

```
]LIST ,20
```

This list mode may be simulated in Integer BASIC by typing a zero before the comma.

Finally, to list all lines after a certain line number, type the line number, then a comma. The Apple will start the listing with the line number that you entered, and stop with the last program line.

```
]LIST 10,
```

In Applesoft, a hyphen (-) may be used in place of a comma in the LIST command. It has the same effect.

Sometimes you may want to abort the listing of a rather long program. To do so in Applesoft, press CONTROL-C. The one drawback to this method is that there is no easy way to immediately continue listing the program from where it was stopped. Integer BASIC does not have a way to abort the program listing.

You may find that the earlier lines in a long program pass by too quickly for you to read. Fortunately, the Apple II Plus and all later models support a listing pause feature. To pause the output of a program listing, press CONTROL-S. The Apple will wait for as long as you need to read the lines on the screen. When you're ready for the rest of the program listing, press any key, such as the SPACE BAR (or CONTROL-S again) to resume.

Modifying the Program

Let's say that you want the program to print something different. How about your name? To revise the program, you can either change existing lines, add more lines, or delete lines.

Changing Lines

To change a line, merely retype the entire line. Alternatively, if the line is especially long, you could make good use of the Escape key sequences covered near the end of Chapter 2 to correct it.

In this case, let's have the Apple print your name. Type line number 10, a quotation mark, your name, and a final quotation mark. The line in my program looks like:

```
]10 PRINT "DAVID FINNIGAN"
```

Type RUN to see the new results. If you successfully changed line 10 by retyping it, the Apple should now display only your name.

Adding Lines

You may add new lines to a program at any time, and in any order. You do not have to enter lines sequentially. The Apple will always arrange the lines in the proper order when it displays the program listing or when it executes the program.

In this example, five program lines are entered out of sequence:

```
]20 PRINT "AS HE FISHED HIS"

]10 PRINT "BOTHER, SAID POOH,"

]40 PRINT "HONEY JAR."

]50 END

]30 PRINT "DISKETTES FROM THE"
```

However, when the program is listed, the lines are in the right order:

```
]LIST

 10  PRINT "BOTHER, SAID POOH,"
 20  PRINT "AS HE FISHED HIS"
 30  PRINT "DISKETTES FROM THE"
 40  PRINT "HONEY JAR."
 50  END
```

Notice that the line numbers are incremented by 10. Allowing this space in numbering makes inserting additional lines convenient.

Deleting Lines

To delete a line, simply type its number followed by the RETURN key. This will replace the existing line with a blank which effec-

tively obliterates it. To verify, use the LIST command, and notice that the line no longer appears, as in this example:

```
]NEW
]50 PRINT "BLUETS AND GRANOLA BARS"
]60 PRINT "MAKE A"
]70 PRINT "CHEWY"
]80 PRINT "SNACK"
]90 END
```

First, five new lines are added. Then line 70 is deleted:

```
]70
```

Now listing the program will show that line 70 is indeed gone:

```
]LIST

 50  PRINT "BLUETS AND GRANOLA BA
     RS"
 60  PRINT "MAKE A"
 80  PRINT "SNACK"
 90  END
```

To efficiently remove a block of lines, use the DEL command, such as in this continuation of the previous example:

```
]DEL 60,80

]LIST

 50  PRINT "BLUETS AND GRANOLA BA
     RS"
 90  END
```

In this example, all line numbers between, and including, 60 to 80 are removed. Even if the range specifies line numbers which do not exist, BASIC takes no exception.

About Program Writer

The limitations of the Apple's small screen and line-editing commands make writing and modifying a large program cumbersome. Program Writer, written by Alan Bird, and formerly published by Beagle Bros, is a utility program designed to make the task of BASIC programming easier. Program Writer features an interface that works like a conventional word processor, allowing automatic line numbering, full screen scrolling and copy-paste functionality.

Putting Multiple Statements on One Line

Sometimes you may find it convenient to place more than one statement on a line, such as if you wish to conserve line numbers, or if the statements are closely related. The one pitfall to this practice is that editing such a line becomes more burdensome. In Applesoft and Integer BASIC, the syntax to add additional statements is the same. Use the colon (:) to separate statements, such as in this example:

```
]10 PRINT "NOTHING" : PRINT "IS REAL"

]RUN
NOTHING
IS REAL

]
```

Remember that you only need to type the line number once. Afterward, type the desired statement, then a colon, and the next statement. You may, of course, have more than two statements on a line, but there are some limits.

In Applesoft, the colon syntax can be used in both immediate and deferred execution modes. The maximum line length is 255 characters, which will place a limit on how many statements can be in a single line.

Integer BASIC restricts such syntax to deferred mode only, meaning that you can only combine BASIC statements in your program,

and not when issuing immediate commands. Integer BASIC also has a stricter limit on the maximum number of statements which may be placed on a single line. The line limit is around 150 characters, but the exact limit depends on the sort of commands involved. Some experimentation on your part will reveal what these limits are.

Adding Comments to a Program

Your first few BASIC programs will be short, and it is unlikely that you will not understand what they are doing. However, long BASIC programs will almost certainly introduce more complexity. To help the programmer, *comments* or *remarks* may be added to a BASIC program. Comments should be used to explain what certain lines of a program are expected to do, or what input should be expected from the user.

To add a comment, use the REM command:

```
]30 REM THIS IS A PROGRAM COMMENT
```

Program comments can appear anywhere and on any line in a program, but typically they are placed just before the line that they describe. If you include a REM command as part of a multiple-statement line, you should add it as the last statement. If you decide to share your program with other people, they will likely appreciate it if you include comments, especially if they try to modify your program. Adding comments will also help you remember what different parts of your program do if you come back to work on the program months or years later.

The Apple does not pay any attention to your comments; when executing your program, it will skip over them as if they were never there. Keep this in mind when using the colon to place multiple statements on one line: be sure that any REM is the last statement on the line.

Clearing the Screen

Running your program and using PRINT statements likely made a lot of text on your screen, not all of which was related. Fortunately, there is an easy way to clean the screen and begin with a "blank slate," so to speak. In Applesoft, the HOME command will clear the screen and return the cursor to the upper-left corner, known as the *home position*. Integer BASIC does not have a HOME command, instead you must use CALL -936, which has the same effect.

Both HOME and CALL -936 can be used in either immediate mode, or as part of your program. It's good to get into the habit of beginning your program by clearing the screen, since you don't know what the user was doing beforehand that may have cluttered it.

A final method of clearing the screen is by an Escape sequence. Press ESCAPE, then type an @ (on the Apple II and II Plus, an @ is generated by typing SHIFT-P). This too will clear the screen, leaving just a flashing cursor at top left. The one difference with this method is that it cannot be used as part of a program.

Automatic Line Numbering

Integer BASIC has a handy feature that when enabled, can automatically number each successive line for you. To enable this feature, use the AUTO command, such as in this example:

```
>AUTO 10
```

When you press RETURN, the Apple will have started the next line with 10 for you. When you type your statement and press RETURN, the next line will be numbered 20.

By default, the line numbering interval is 10.If you would like use a different interval, then you must provide it as a second argument, like so:

```
>AUTO 100, 5
```

In this example, the Apple II will start numbering your program at line number 100, and will advance by 5 every time you enter a valid line.

The Apple II will not increment the line number if you enter an invalid line, such as one with a syntax error, or a blank line. Instead, the Apple will print the same line number again, allowing you to retype the line.

To stop the Apple from automatically numbering each line, first press CONTROL-X to cancel the current line. Finally, use the MAN command to resume manual line numbering.

Saving and Loading Programs

While your first few BASIC programs probably won't be worth saving, you will no doubt conceive of others which you do wish to keep. There are two methods to save a program: on tape, and on disk.

Saving on Tape

Saving a program to tape is rather old-fashioned, but still works. You must have a cassette tape recorder properly connected to your Apple; see Chapter 2 for setup details. When you are ready to save your program, make sure that you have a cassette tape loaded in the recorder. Type the following command, but *do not* press RETURN:

```
]SAVE
```

At the cassette recorder, press the Record and Play buttons. Now press RETURN on the Apple keyboard. After a few seconds, the Apple will beep. Then, after a few more seconds, the Apple will beep again and the BASIC prompt will return. At this point, you may press Stop on the recorder; your Apple has finished.
It is convenient to use the cassette recorder's tape counter to track the positions of multiple programs on a tape.

Saving on Disk

To save a program on disk, which is faster and more convenient than tape, you must have first started a Disk Operating System, DOS or ProDOS. For instructions on this procedure, see Chapter 7. If you load DOS or ProDOS with a BASIC program in memory, that BASIC program will be destroyed!

Using a disk allows you to file your programs by name. The format for the SAVE command is similar to that for saving on tape, except that you must specify a filename, such as in this example:

```
]SAVE MYPROGRAM
```

Entering that command will place a file named MYPROGRAM on your disk containing a copy of your program. The copy of the BASIC program in memory remains unmodified. Depending on what version of operating system you're using, the filename may have some restrictions. Again, take a look at Chapter 7 for the full details on using the disk.

Be aware: if a file with the same name already exists when you SAVE your program, it will be overwritten! The Apple will not warn you that a file already exists, nor will it ask for confirmation to overwrite the previous file.

Loading from Tape

To load an existing program from tape, first position the cassette tape to the beginning of your program. If you are unsure of where your program begins, play back and listen to the tape. When you hear a steady tone, that marks the beginning of your program. Stop the tape. Make sure that the cassette tape recorder is properly attached to your Apple; see Chapter 2 for details.

At the Apple, type in the following, but *do not* press RETURN:

```
]LOAD
```

Press Play on the cassette recorder, then press RETURN on the Apple keyboard. A few seconds will pass as the Apple listens to the steady tone marking the beginning of your saved program. After

that, the Apple will beep, and your program will be read into memory. Finally, if all went well, the Apple will beep again, and the cursor will reappear. Type RUN to start your program, or LIST to see if it was loaded correctly.

If you get an error, rewind the tape, adjust the volume control, and try again. The volume control should be set to around 50-70% of maximum. It can take a few tries to successfully load a tape. Once you have successfully loaded a tape, you should not need to adjust the tape recorder's volume settings anymore.

Loading from Disk

Loading programs from disk is far quicker and more convenient than loading from cassette tape. First, you must have started the disk operating system, either DOS 3.3 or ProDOS. Chances are, the disk on which your program is stored already has an operating system on it. For more details on starting DOS, see Chapter 7.

Once DOS is loaded into your Apple, you can use one of two commands:

```
]RUN MYPROGRAM
```

This will load and then run a program named MYPROGRAM from the disk. Alternatively, if you do not want a program to start running, say if you want to LIST and edit it, use the following command:

```
]LOAD MYPROGRAM
```

This will load the file named MYPROGRAM into memory, overwriting any existing BASIC program. There you may LIST or RUN it, and resume working on it as usual.

If you get a FILE NOT FOUND error, then check your spelling; you may have mistyped the filename. Alternatively, it could be that the file does not exist on the disk; you may have inserted the wrong disk. Finally, if you get a FILE TYPE MISMATCH error, then you have tried to load a file that is not a BASIC program.

Use the CATALOG command to get a listing of each file and its file type present on the disk. Again, see Chapter 7 for full instructions on these procedures.

Listing Programs on a Disk

All of the files saved on a disk are listed in the disk's catalog. After the operating system is loaded, type the following:

```
]CATALOG
```

to list the files saved on the disk. ProDOS will allow you to type CAT instead, which will yield a more compact listing. On a DOS 3.3 disk with more than 18 files the listing will pause to give you time to read it. Press RETURN to see the rest of the files.