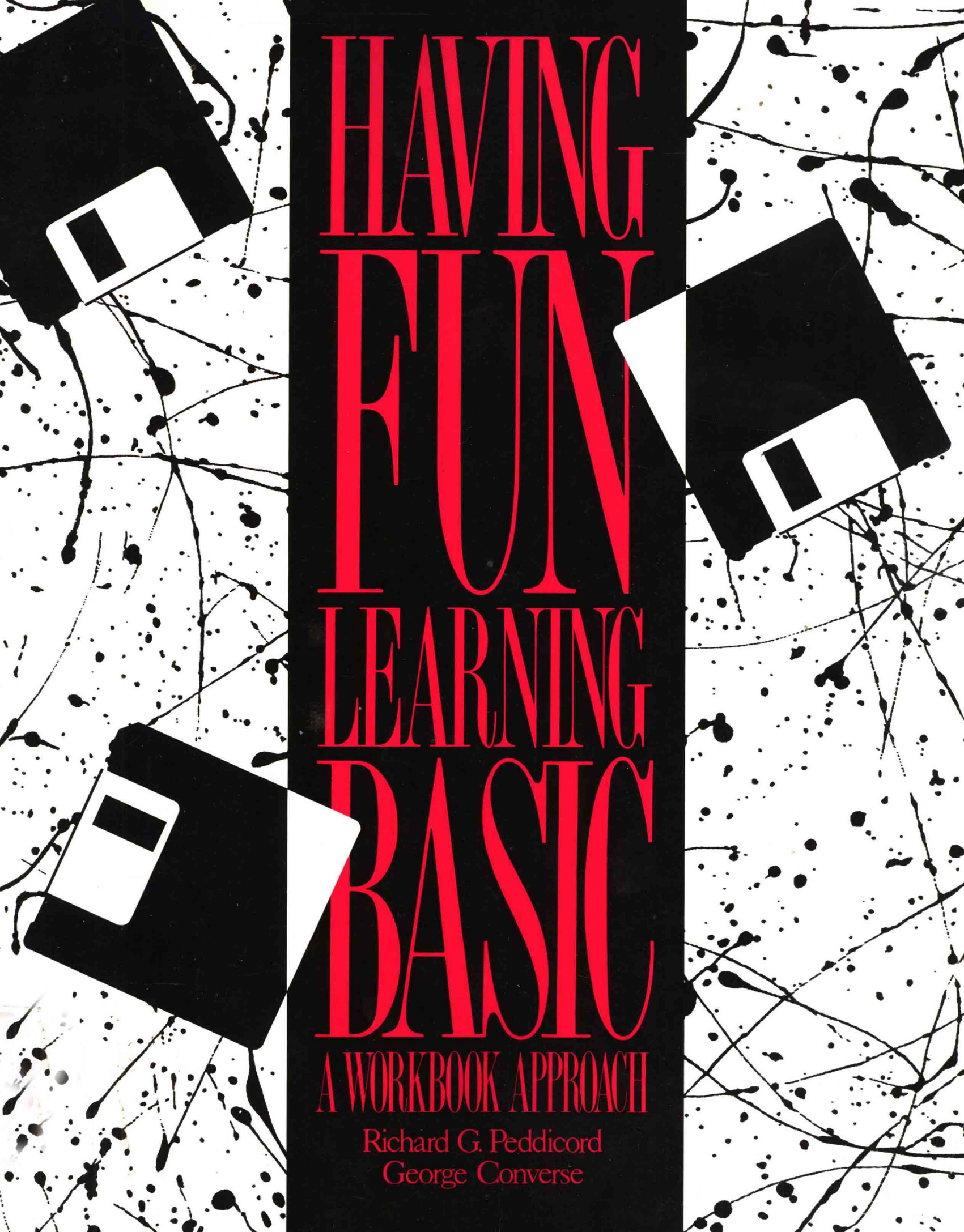

HAVING FUN LEARNING BASIC

A WORKBOOK APPROACH

Richard G. Peddicord
George Converse

HAVING FUN LEARNING BASIC

A WORKBOOK APPROACH

Richard G. Peddicord · Southern Oregon State College
George Converse · Southern Oregon State College

WCB Wm. C. Brown Publishers

Book Team

Editor *Kathy Shields*
Software Coordinator *Lisa Schonhoff*
Production Coordinator *Peggy Selle*

Wm. C. Brown Publishers

President *G. Franklin Lewis*
Vice President, Publisher *George Wm. Bergquist*
Vice President, Operations and Production *Beverly Kolz*
National Sales Manager *Virginia S. Moffat*
Group Sales Manager *Vincent R. Di Blasi*
Vice President, Editor in Chief *Edward G. Jaffe*
Marketing Manager *Elizabeth Robbins*
Advertising Manager *Amy Schmitz*
Managing Editor, Production *Colleen A. Yonda*
Manager of Visuals and Design *Faye M. Schilling*
Production Editorial Manager *Julie A. Kennedy*
Production Editorial Manager *Ann Fuerste*
Publishing Services Manager *Karen J. Slaght*

WCB Group

President and Chief Executive Officer *Mark C Falb*
Chairman of the Board *Wm. C. Brown*

Cover design by Dale Rosenbach

Copyedited by Jane Dowd

Library of Congress Catalog Card Number: 91-55472

ISBN 0-697-14956-0

Printed in the United States of America by Wm C Brown Publishers
2460 Kerper Boulevard, Dubuque, IA 52001

10 9 8 7 6 5 4 3 2 1

PREFACE

This workbook attempts to make the task of learning BASIC a successful and enjoyable one. For this to happen, the student must have access to a computer that runs some version of BASIC, and someone to help them--an instructor, lab assistant, classmate, or friend. This 1991 Edition relies heavily on the student trying all the exercises on a computer. Only with extensive practice is it possible to get the keyboarding and logic skills necessary to work successfully with computers.

The main features of the workbook are its nonthreatening, hands on, keep it light and simple, do-it-right approach. Although designed for college students, the workbook can be used by practically anyone wanting to learn BASIC and willing to put in the time and effort.

The workbook contains enough material for a one-year course in generic BASIC for college students. It has evolved from the authors' Introduction to BASIC (CS151) and Advanced BASIC (CS152) at Southern Oregon State College. The first-term course is designed to give liberal arts students some valuable experience with a programming language. Part One, chapters 1 through 6, supports this first term. Part Two, chapters 7 through 10, supports a second term, for computer-and math-oriented students who wish to continue their study of BASIC, either on their own or for credit. The material in Part Two becomes quite challenging, and we use it to prepare our students for local programming contests.

This workbook has evolved over several years, and we hope that it gets better with each edition. This edition relies on students having access to a computer.

In keeping with this emphasis on lab work, we have incorporated specific data, where possible, and have provided the correct answers to many exercises. This allows students to check their work as they go along, which gives them a strong sense of accomplishment.

Solutions are presented in generic BASIC, so they will run on almost all versions of BASIC. When a command has different forms for different computers, we give the correct syntax and usage instructions for the popular versions of BASIC now running. We thank our CS151 and CS152 students for their many valuable suggestions and corrections; Jon Montgomery for proofreading the original manuscript; and Lee Hill and Bob McCoy for suggesting many exercises. Barbara Cantrell made the revisions suggested by the proofreader.

We also thank Kathy Shields, of Wm. C. Brown Publishers, for soliciting several reviews and returning their comments, all of which were most helpful. We have made the recommended changes, and we feel the workbook is much improved as a result. In this edition, we stress structured programming, restrict the use of GOTO, and encourage WHILE/WEND.

The following Trademarks appear in this workbook:

IBM is a registered trademark of International Business Machines

WordPerfect is a registered tradermark of WordPerfect, Inc

Richard Peddicord and George Converse
Department of Computer Science
Southern Oregon State College
Ashland, Oregon
September 4, 1991

TABLE OF CONTENTS

CHAPTER ONE: GETTING STARTED

BASIC is one of the most popular programming languages in use today, and this popularity stems from several features of the language. It is easy to learn, it has a powerful, complete set of instructions, and many personal computers use BASIC as their "native" language, the language that comes up when the power is turned on. For these reasons, it has become the current standard for a beginner's programming language. In fact, the name is short for Beginners All-Purpose Symbolic Instruction Code. It was developed at Dartmouth College to give students an easy-to-learn, but complete, programming language.

In this first chapter, we will describe step by step how to enter and run your first program, and we will introduce you to four BASIC statements, REM, INPUT, LET, and PRINT, and the system commands LIST, RUN, and NEW. With these few statements and system commands you can document your program, input numbers or words from the keyboard, process the input, and show the results on the screen.

You will also learn about variables, how to put values in them, how to print them, and how to use them in arithmetic expressions.

1.1 REM, LIST, RUN, and NEW

Now it is time to find a computer that runs BASIC and someone to help you when you get stuck. Beginners get stuck all the time, and it is very important to have help available, that is, someone familiar with the particular system you are working on.

In what follows, we assume that you are at a computer, in BASIC, and that you have a formatted diskette ready to store your programs.

When BASIC comes up in WAITING mode, waiting for you to enter a program statement or a system command, enter the following program, by typing one line at a time, exactly as printed below. Press the Enter (Return) key when your typed line looks correct. Turn on the CAPS LOCK function if you want all capital letters. Many older BASIC systems require all capital letters.

```
10 REM  THIS PROGRAM WILL NOT DO ANYTHING
20 REM  BECAUSE IT CONSISTS ENTIRELY OF
30 REM  REMARK STATEMENTS
```

Type each line starting with the statement number, then a space, then REM, then three spaces, and then the message. You can change the line, using the editing keys, up until the time you press Enter.

When you are satisfied that the line the cursor is on is what you want the system to see, press the Enter (Return) key. The system will read your line, interpret it, and then switch back to WAITING mode, where it is waiting for you to type in another line of your program or a system command.

If at any time you wish to see your program on the screen, type LIST (without a line number) and press Enter. Your program will be listed on the screen, in increasing order of the line numbers that start each line.

So as soon as you are done with all three lines above, type LIST, press Enter, and your program should appear.

The purpose of the REM statement is to make a remark about the program for the benefit of someone reading it, either a comment that clarifies how the program works or some identifying information.

When working on exercises to be turned in, begin each separate program with your name, the exercise number, and the date. This identifies the work.

Notice in the above program that the individual lines are numbered, in this case, by 10, 20, and 30. These are called line numbers. You make them up as you enter your program, line by line. Most people start at 10 and go up by 10s, so that they can insert statements later if necessary.

To run your program, that is, have the system execute the commands you have programmed, type RUN and press Enter. The system will start at the lowest numbered statement and begin executing your program statements, one after the other, until it either runs out of instructions, comes to a STOP or END, or encounters an error condition.

If you haven't already done so, RUN your program. It should do nothing, and the system should return to WAITING mode. If you get an error message when you RUN your program, LIST the program and check it carefully. Call someone to help you if you are stuck at this point.

Before beginning a new program, erase the current program by typing NEW (no line number), and then press Enter.

So type NEW, press Enter, then type LIST, press Enter. No program will appear. You are ready to start the first exercise.

The solution to each exercise appears immediately below the exercise. Try each exercise on your own first, then compare your work with the solution given. There are many ways to solve a particular exercise, but we show only one.

****

Exercise 1.1.1 Write a program containing three remark statements that give, respectively, your name, the phrase LAB ASSIGNMENT 1, and the date. LIST your program, and then RUN it. It works if it does nothing and produces no error messages. If you get an error message, you will have to debug your program.

Solution and Discussion:

```
10 REM  DIRE STRAITS
20 REM  LAB ASSIGNMENT 1
30 REM  01/01/91
```

1.2 INPUT, PRINT, SAVE, and LOAD

In what follows, you will write a simple program that asks you for a response and then prints your response twice. Go ahead and enter it, LIST it, and then RUN it. Remember to type NEW and press Enter before you begin work on a new program.

```
10 REM      SIMPLE PRINT
20 INPUT "Do you want to learn BASIC";A$
30 PRINT A$
40 PRINT A$
```

Switch the CAPS LOCK off and on as needed or, if you prefer, you can type all lowercase or all uppercase. Some systems require all uppercase (capital) characters.

As soon as you press Enter you see the message

Do you want to learn BASIC?

appearing below the RUN command and starting at the far left of the screen, with the cursor flashing after the question mark. The system is waiting for you to type in something and then press Enter. As soon as Enter is pressed the system takes your response and stores it in the memory location named A$, and goes on to the next instruction, in this case 30.

Because the name A$ ends in a dollar sign ($) the system expects you to type in words rather than a number. So it will accept just about anything you can type on the keyboard, and it will store all the characters you entered, starting a certain location in main memory. The system takes your variable name, A$, and puts it in a table, and follows it by the actual location in memory. Thereafter, whenever the variable name A$ appears in your program, the system uses the actual memory location. This is a standard feature of all programming languages: you just name your variables, and the system takes care of where to put them.

So type in something and press Enter. You should see your response appear in duplicate on the two lines below the input prompt message, because in lines 30 and 40 whatever is stored in A$ is printed.

You should see your response printed twice, once per line, and the system will then return to WAITING mode.

If you have trouble at this point, enter LIST (without a line number) and press Enter. Your program will appear on the screen, in increasing order of line number. Locate and correct any errors, and RUN your program again.

Ask your instructor or consult the Appendix about the best way to fix errors on your particular system. For most personal computers, you can fix any line that appears on the screen, and then press Enter and the system will accept the changed line. On mainframe systems, this may not be possible. You can always retype the line and press Enter. The system will replace the old line with the new line. If you just type a line number and press Enter, the line with that number will be removed from your program.

If you need to insert one or more characters in the middle of a line, press the INSert key,

if you have one, and start typing. Everything to the right of the cursor should move to the right as you type. See if the cursor changes when you go into INSert mode.

When you RUN the above program, BASIC starts at the lowest numbered line, in this case 10. But since it is a REM statement, BASIC skips over it and immediately goes to the next numbered line, in this case, the INPUT statement at line 20. In line 20, the system prints the input prompt (the message within double quotes) and waits for the user to enter text. As soon as it is entered, the text is stored in A$. In line 30, the contents of A$ are printed.

Variable names that end in a dollar sign are called string variables because they hold a string of symbols; that is, letters, digits and punctuation marks are strung together from left to right. In short, string variables hold words.

Variable names can be any length, but only the first few characters are used by the system. Earlier versions of BASIC may be limited to 2 characters plus the dollar sign; more recent permit at least 8 characters in a name. The shorter the name the less you have to type. The longer the name the more descriptive it can be.

As you have seen, a BASIC program consists of one or more lines of code. Each line has a line number followed by one or more statements. You will learn what the various statements look like later on.

It is customary to begin numbering your lines at 10, and to go up by 10 each new line. If you forget something, or have to insert a short program between two lines, this will give you enough room. The system always lists your program in increasing order of line number, both on the screen and when it is printed. We want you to try the following exercise on your own. Don't look at the solution until you have LISTed and RUN your own solution and tried to debug it. Remember to type NEW to clear the current program, so that you can start a new one. The system only holds one program at a time.

****

Exercise 1.2.1 Ask for the user's name, and then print

THANK YOU, <name>
THANK YOU, <name>
THANK YOU, <name>

as shown above, that is, three times, once per line. The notation <name> means to put an actual name here, the name entered by the user. If the user's name is ALICE the output should look like this:

THANK YOU, ALICE
THANK YOU, ALICE
THANK YOU, ALICE

Solution and Discussion:
Use an INPUT statement with an input prompt and put the result in a string variable called N$. Then construct the PRINT statement and make two copies of it. The first PRINT statement should start with the message THANK YOU, in double quotes, followed by a

semicolon (;), followed by the variable name N$ (without quotes around it). To duplicate the PRINT statement, add 10 to its line number and press Enter. Do this again to get the third PRINT statement. If you LIST your program, you will see all three PRINT statements.

```
10 REM  SOLUTION 1.2.1
20 INPUT "WHAT IS YOUR NAME";N$
30 PRINT "THANK YOU, ";N$
40 PRINT "THANK YOU, ";N$
50 PRINT "THANK YOU, ";N$
```

The semicolon (;) in line 30 means "close it up", that is, do not insert any extra spaces between the preceding print item and the one after the semicolon. If you put a comma between two print items, the screen will "tab" to the beginning of the next tab zone. There are usually five tab zones across the screen, each about 15 characters wide.

The semicolon in line 20 means something different. It means to display a question mark after the input prompt, just before the flashing cursor. This reminds the users that they are supposed to answer something. If you don't want the question mark to appear, try a comma instead and see what happens on your system.

If you are having trouble getting your program to run, LIST it and see what you have. If you need to change part of a line, use the cursor keys (the ones with arrows pointing in all four directions) to move to the area you want to change. Make the change and then press Enter. The new version of that line will replace the older version. You must always press Enter before the system will replace the old with the new.

ALWAYS PRESS ENTER AFTER YOU HAVE FIXED A LINE OF CODE. OTHERWISE, THE SYSTEM WON'T TAKE THE CHANGE!

****

Exercise 1.2.2 Ask for the user's first name. Put it in F$. Then ask for the last name and put it in L$. Then print the full name in "registrar" format, that is, last name, followed by a comma and a space, and then the first name.

Solution and Discussion:
This exercise is kind of tricky to get completely right. The hard part is the PRINT statement. The main things to remember are:

(1) Print items inside double quotes are called literals, and they are printed exactly as they are written.

(2) Print items not inside double quotes are called expressions, and they are computed and then printed. If the expression is a single variable, its current value is printed.

```
10 REM   CONVERT NAME TO REGISTRAR FORMAT
20 INPUT "WHAT IS YOUR FIRST NAME";F$
30 INPUT "WHAT IS YOUR LAST NAME";L$
40 PRINT L$;", ";F$
```

In generic BASIC, a semicolon (;) between print items tells the print routine to close it up, that is, add no extra blanks (spaces) between the items it separates. So in line 40 after the content of L$ is printed, the first semicolon causes the literal ", " to be printed immediately following the last letter of the last name. The second semicolon makes the first letter of the first name start right after the space. The prompt message is optional in the input statement, but it is usually a bad idea to omit it. Try this program without the prompts to get an idea of how confusing it is to just see a '?' on the screen.

You will want to save your programs to your diskette when you stop a work session, so the command is

SAVE "<filename>"

where <filename> is a filename that conforms to your particular system's disk operating system. Ask your instructor which filenames to use.

Whenever you wish to transfer a stored program from diskette to main memory, so that you can work on it, place the proper diskette in the drive and enter

LOAD "<filename"

where <filename> is the same name you used to save the program.

1.3 The LET Statement

In addition to string variables (ending in a dollar sign), the system recognizes two other kinds of variables: integers (the variable name ends in a percent sign) and so-called real numbers (no special symbol after the name). In what follows, we will be working with real variables. Their names have no special symbols at the end, and they store a wide range of values with about nine decimal-digit precision.

Enter and RUN this program:

```
10 REM   PRINT A CONSTANT
20 LET X = 1.23456789
30 PRINT X
```

See how many digits appear.

The LET statement in line 20 always has a variable name after the word LET, and this name is always followed by an equal sign. After the equal sign is always an expression, in

this case, a simple constant.

LET statements also work with string variables. Try this program:

```
10 REM  PRINT A STRING CONSTANT
20 LET X$ = "GOTCHA"
30 PRINT X$
```

You should see the word GOTCHA appear at the left of the screen.

Remember that a variable name, or variable for short, is the name for a place in memory that holds the current value of the item in question. There are two main kinds of data items: string variables (end in dollar sign) and numeric variables (percent sign or no sign).

The system finds a place in main memory to hold items of the specified type, and it links your variable name to the place in memory it has picked out. Every time you refer to the variable name in your program, the system is working with the main memory location that is linked to it.

The real variable, with no type symbol, is the most popular variable. It holds numbers with decimal digits. Some examples of variable names include AA, D, X, XH, Y3, BALDUE, RECEIPTS. The first character must be a letter, and the remaining characters can be letters or digits. Any version of BASIC can handle at least 2 characters, and many systems can handle 8 and more.

If you have a variable like X, you put a constant into it, using the LET statement

LET <variable> = <expression>

where <expression> in this case is a real number constant and <variable> is the single letter X. The resulting statement might look like this:

```
20 LET X=4.23984
```

The word LET serves to remind us that the = sign in BASIC is not the same as the = sign in algebra. LET says to transfer the value on the right side (of the equal sign) into the variable on the left side. The variables on the right, if any, are not changed. Remember, LET goes from right to left. Although we show solutions in capital letters, many systems will accept both upper- and lowercase characters. Clear your program memory by entering NEW, and try the next exercise.

****

Exercise 1.3.1 Put the value 12.345 into a real variable called TH, and then print the label 'TH=' followed by the value of TH.

Solution and Discussion:

Copy the format shown in line 20 above. Then in the print statement use the literal "TH= " for the first print item, follow it by a semicolon, and then put the print item TH without quotes around it. The system will then print the stored value. If you put double quotes around the

variable name, the system will print just the name.

```
10 REM   ASSIGN A VALUE THEN PRINT IT
20 LET TH=12.345
30 PRINT "TH= ";TH
```

Several things should be mentioned. In line 20, you can see that there are no spaces before or after the = sign. It would have been fine to have spaces before and after the equal sign, but none are needed because the equal sign terminates the reading of the variable name TH. The system then knows that whatever lies to the right of the equal sign is an expression that will compute to a real number, and that real number will be placed in the memory location whose symbolic name is TH. In this case the expression is simply the constant 12.345. In line 30, however, we leave one space after the reserved word PRINT so that the system will recognize the word PRINT.

Now it's time to do some arithmetic.

****

Exercise 1.3.2 Assign the values 3.4, 78.97, and 102.78 to the variables A, B, and C, respectively. Then set D equal to the sum of A, B, and C. Then print the sum D.

Solution and Discussion:
Use three LET statements to assign the values, another LET statement to compute their sum, and a print statement to print the result.

```
10 REM   COMPUTE SUM OF 3 NUMBERS
20 LET A=3.4
30 LET B=78.97
40 LET C=102.78
50 D=A+B+C
60 PRINT D
70 END
```

Enter and run the above program. You should get 185.15.

1.4 Arithmetic Expressions

It is easy to get BASIC to do arithmetic for you: use + for addition, - for subtraction, * for multiplication, / for division, and ^ for exponentiation (the power operator). Some systems use a different symbol for exponentiation, but the two most common today are the carat, a pointed-roof looking character, and the upper arrow, which has a line coming down from the pointed top of the roof. An expression like 2^3 means two to the third power, that is, 2 times 2 times 2. Locate the upper arrow symbol and verify that it works, using the program

```
10 PRINT 2^5
```

you should get 32. Some older versions of BASIC use two asterisks in a row (**) to indicate the power operation (exponentiation).

There is a hierarchy, shown below, that determines which operations are done first. At the top of this hierarchy is the paired parentheses, (...). Whatever expression is inside will be executed before that value is used within a larger expression. Next after the parentheses comes exponentiation (the power operation), below that are the other arithmetic operations of multiplication and division (next level down), and below these are addition and subtraction, on the same level, one below multiplication and division.

For example,

```
20 PRINT 2^(1+3)
```

will print

16

because 1 is added to 3 before becoming the power of 2, whereas

```
20 PRINT 2^1+3
```

will print

5

because 2 is raised to the 1 power, and then added to 3.

Hierarchy of Arithmetic Operators

Done First		
Level 1	( )	parentheses
Level 2	^	exponentiation
Level 3	* /	multiplication and division
Level 4	+ -	addition and subtraction
Level 5		left to right
Done last		

For example, if you write

```
10 LET X=3*(4+5)
```

the machine will add the 4 and the 5 together, and then multiply the result by 3. In this case,

27 will be stored in location X. On the other hand, if you write

```
10 LET X=3*4+5
```

the machine will first multiply the 3 and 4 together (because multiplication is higher than addition in the hierarchy) and then add the 5. In this case, 17 gets stored in X. If a number variable is named somewhere in your program, it gets initialized to 0 before it is used. String variables are initialized to the null string. The null string is the string with no characters, written "" (two double quotes in a row).

It is good programming practice to initialize all variables yourself before they are used. BASIC on some systems does not initialize variables. If you do not initialize the variables yourself, you will get different answers each time the program is run. Furthermore, BASIC programmers who go on to learn other languages, such as Pascal, will find it easier to adjust to the necessity of initializing variables in the new language if they practice it in BASIC.

****

Exercise 1.4.1 Put 4 squared plus 3 cubed into X. Then print "X= " followed by the value of X.

Solution and Discussion:
Use the power operator (^) to get 4 squared and 3 cubed, and use the + sign to add them together. In your PRINT statement, the first print item will be the literal "X= " and the second print item will be the variable X. You can separate the two print items with a space, a semicolon, or a comma. Try each of them and see what they do on your system.

```
10 REM  PRINT 4 SQUARED PLUS 3 CUBED
20 LET X=4^2+3^3
30 PRINT "X=";X
```

****

Exercise 1.4.2 Write an arithmetic assignment statement that puts the sum of the first 7 integers, 1+2+3+4+5+6+7, into a variable called S. Then print S with a label in front that says "SUM=".

Solution and discussion:

```
10 REM  ADD 7 NUMBERS AND PRINT RESULT
20 S=1+2+3+4+5+6+7
30 PRINT "SUM=";S
```

The semicolon between "SUM=" and S tells the system not to add any blank space between the equal sign and the first character of the printout of the contents of S. The printout of a number always starts with one blank character, if positive, or a minus sign if