BENT ▲ SETHARES THIRD EDITION

BUSINESS BASIC

Business BASIC

Third Edition

Robert J. Bent
George C. Sethares
Bridgewater State College

Brooks/Cole Publishing Company
Pacific Grove, California

Brooks/Cole Publishing Company
A Division of Wadsworth, Inc.

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data
Bent, Robert J., [date]
Business BASIC.

Includes index.
1. BASIC (Computer program language) 2. Business—Data processing. I. Sethares, George C., [date] II. Title.
QA76.73.B3B47 1988 005.13'3 87-35802
ISBN 0-534-09024-9

Sponsoring Editor: *Cynthia C. Stormer*
Marketing Representative: *John Moroney*
Editorial Assistant: *Mary Ann Zuzow*
Production Editor: *Sue Ewing*
Production Assistant: *Linda Loba*
Manuscript Editor: *Harriet Serenkin*
Permissions Editor: *Carline Haga*
Interior and Cover Design: *Lisa Thompson*
Cover Photo: *Lee Hocker*
Art Coordinator: *Lisa Torri*
Interior Illustration: *Maggie Stevens*
Photo Editor: *Lisa Thompson*
Photo Researcher: *Monica Suder*
Typesetting: *Progressive Typographers, Emigsville, PA*
Cover Printing: *Lehigh Autoscreen, Pennsauken, NJ*
Printing and Binding: *Diversified Printing and Publishing Services, Inc., Brea, CA*

Photo Credits
We wish to acknowledge the following people and companies for the photos used in this book.

Chapter 1. 1, Figure 1.1, Frank Keillor. **2,** Figure 1.3, Sperry Corporation. **3,** Figure 1.4, Texas Instruments. Figure 1.5, NCR. **4,** Upper left, Honeywell, Inc. Upper right, General Electric. Lower, IBM Corporation. **5,** Upper, Judy Blamer, Lower, IBM Corporation. **6,** Figure 1.6, Apple Computer, Inc. **7,** Upper left, Frank Keillor. Upper right, Digital Corporation. Lower left, Dataproducts Corp. Lower right, Frank Keillor. **8,** Left, Judy Blamer. Right, IBM Corporation.
Chapter 2. 14, Both photos, NASA/Ames.
Chapter 3. 26, Upper, Apple Computer, Inc. Lower, Honeywell, Inc.
Chapter 4. 40, Upper, Texas Instruments. Lower, AT&T.
Chapter 5. 52, Upper, IBM Corporation. Lower, Apple Computer, Inc.
Chapter 6. 60, Upper right, General Electric Company. Upper left, IBM Corporation. Lower, Hewlett-Packard Company.
Chapter 7. 74, Upper, Hewlett-Packard. Middle, IBM. Lower, Electronic Arts.
Chapter 8. 98, Upper left, Hewlett-Packard (Integral PC at Krug Winery). Upper right, Chrysler Corporation. Lower, Hewlett-Packard.
Chapter 9. 130, Upper, General Electric Corporation. Lower, Hewlett-Packard.
Chapter 10. 144, Upper, Hewlett-Packard. Lower, Adrian Bradshaw/Visions.
Chapter 11. 164, Upper left, Apple Computer, Inc. Upper right, Hewlett-Packard. Lower, Apple Computer, Inc.
Chapter 12. 180, Upper, J. Ross Baughman/Visions. Lower, IBM.
Chapter 13. 200, Upper, Evans & Sutherland. Middle, Evans & Sutherland. Lower, Manos/Magnum—Burroughs.
Chapter 14. 226, all courtesy of Hewlett-Packard.
Chapter 15. 254, Upper, Texas Instruments. Lower, Koala Technologies.
Chapter 16. 266, Upper, Control Data. Lower, Commodore Electronics Ltd.
Chapter 17. 294, Upper, Prime Comp., Inc. Lower, Apple Computer, Inc.
Chapter 18. 308, Upper, Kurzweil Music Systems, Inc. Middle, Hewlett-Packard. Lower, AT&T.

Throughout the book we have attempted to conform to the most common BASIC usage. In most cases, our presentation of BASIC conforms to the American National Standards Institute (ANSI) standard for BASIC. Phrases such as "Your system may allow you to . . . " indicate that the BASIC feature being introduced is not included in the BASIC standard. The material contained in this book, including the problem sets, has been carefully organized so that topics that are not a part of standard BASIC may be omitted with no loss in continuity.

A few remarks are appropriate concerning the order in which we have introduced the elements of BASIC. The INPUT statement (Chapter 5) is introduced early, before the READ and DATA statements, to emphasize the interactive nature of BASIC. In Chapter 6, the GOTO statement and a limited form of the IF statement are used to introduce the concept of a program loop. WHILE loops are also described in Chapter 6, since many versions of BASIC allow the WHILE statement as a convenient alternative to the IF and GOTO statements when coding loops. Chapter 7 expands upon the description of the PRINT statement given in Chapter 3 and introduces the PRINT USING statement. These output statements are introduced early so that well-formatted output can be illustrated in the examples and can be produced while carrying out all subsequent programming exercises. The general form of the IF statement and its use as a selection statement are described in Chapter 8. The BASIC statements described in Chapters 3–8 were selected because they allow meaningful structured programs to be written. Section 8.8 describes how structured algorithms can be coded as structured BASIC programs. Chapters 9–11 (FOR loops, READ/DATA statements, and subroutines) complete what is sometimes called Elementary BASIC. Selecting an order in which to present the remaining BASIC statements was not so easy. So that a person using this book will not be tied down to the order we have chosen, the introductory material for the remaining BASIC statements is presented in a way that allows these statements to be taken up in any order after Chapter 11.

▶ Special Features

Several new features have been included in this revision. Following are brief descriptions of the more significant changes.

An Expanded Introductory Chapter on Problem Solving

Fundamental problem solving principles, including the top-down development of algorithms, are discussed in Chapter 2. Greater emphasis is placed on input/output specification, modularization, and stepwise refinement. The chapter also explains the sense in which computer programs and algorithms are equivalent and illustrates the steps leading to the discovery of algorithms.

Improved Examples and Programming Exercises

Worked-out examples and programming exercises that illustrate application areas not covered in the previous edition have been added. Several examples and exercises, some of which were redundant, have been deleted. Also, many new programming exercises that are suitable for one- or two-week programming projects have been added to the problem sets.

A Separate Introductory Chapter on Loops

The IF and GOTO statements are used in Section 6.1 to introduce the concept of a program loop. The use of IF as a selection statement is taken up separately in Chapter 8. Section 6.2 describes the WHILE statement and explains its use as an alternative to the IF and GOTO statements in coding loops. This important structured control statement is allowed in many versions of BASIC.

A New Section on Menu-Driven Programs

The concept of a menu-driven program is described early (Chapter 7) and illustrated in several examples in the subsequent chapters. Writing BASIC code for menu-driven programs by using the ON GOTO statement is described in Section 7.8, and by using the ON GOSUB statement in Section 11.3. Programming exercises that call for menu-driven programs have been added to many of the problem sets beginning with Chapter 7.

An Expanded and Earlier Chapter on Subroutines

The topic of subroutines is taken up much earlier in this edition. Subroutines are described in Chapter 11, just after the READ and DATA statements, and their application is illustrated in many of the subsequent chapters.

A New Section on Table Processing

Section 14.3 illustrates the application of the array data structure to programming tasks that involve tables. The problem of deciding whether or not to use arrays to store tabular data is discussed and illustrated in the worked-out examples. New programming exercises that involve table processing have been included in the problem sets.

Increased Emphasis on the Application of String Variables

Many new string processing examples and programming exercises have been added. String arrays, which in the previous edition were introduced in a separate chapter after the numerical arrays, are now taken up with numerical arrays in a single chapter. The string conversion functions CHR$, ASC, STR$, and VAL are described earlier, with all other string and string-related functions, and their applications have been expanded to several new topics, including the conversion of lowercase letters to uppercase.

An Improved Chapter on Data Files

The chapter on data files has been rewritten with greater emphasis on the use of the end-of-file functions and the ON ERROR GOTO statement. Error trapping and handling are covered in more detail. Other topics have been retained from the previous edition, including file maintenance and a discussion of differences in how BASIC systems handle data files.

▶ Acknowledgments

While preparing this revision, we were fortunate to have the comments of many users of the second edition. Their thoughtful criticisms and suggestions were carefully considered and, in many instances, incorporated as changes in the book. We are grateful for this assistance. So that we can continue to make improvements for future readers, we would welcome hearing of your experiences with this edition. A reader response form is provided at the end of the book for this purpose.

We wish to take this opportunity to acknowledge the helpful comments of our reviewers: Boyd Ghering, Delaware Valley College; Richard A. Hatch, San Diego State University; John Myer, DeVry Institute of Technology; and Herbert F. Rebhun, University of Houston. We feel that their many thoughtful suggestions have led to a greatly improved book.

A very special thanks goes to Patricia Shea, our typist, proofreader, debugger, and general assistant. Her eleven years of cheerful cooperation are greatly appreciated. Finally, we are happy to acknowledge the fine cooperation of the staff at Brooks/Cole Publishing Company.

Robert J. Bent
George C. Sethares

4.8 On Writing Your First Program 44
4.9 Problems 48
4.10 Review True-or-False Quiz 50

Chapter 5 **Interacting with the Computer 51**

5.1 The INPUT Statement 51
5.2 Problems 56
5.3 Review True-or-False Quiz 57

Chapter 6 **A First Look at Loops 58**

6.1 Coding Loops with the IF and GOTO Statements 58
6.2 WHILE Loops 66
6.3 Problems 68
6.4 Review True-or-False Quiz 70

Chapter 7 **More on the PRINT Statement 72**

7.1 Displaying More Values on a Line 72
7.2 Suppressing the Carriage Return 75
7.3 Problems 76
7.4 The TAB Function 78
7.5 Problems 79
7.6 PRINT USING Statement 80
7.7 Problems 86
7.8 Menu-Driven Programs 88
7.9 Screen Displays 91
7.10 Problems 93
7.11 Review True-or-False Quiz 94

Chapter 8 **The Computer as a Decision Maker 96**

8.1 IF as a Selection Statement 96
8.2 Logical Expressions 102
8.3 Problems 105
8.4 Flowcharts and Flowcharting 107
8.5 Problems 113
8.6 Summing with the IF and WHILE Statements 114
8.7 Problems 118
8.8 Structured Programming 119
8.9 Problems 124
8.10 Review True-or-False Quiz 128

Chapter 9 **Loops Made Easier 129**

9.1 FOR Loops 129
9.2 Problems 135
9.3 Nested Loops 137
9.4 Problems 140
9.5 Review True-or-False Quiz 142

Chapter 10 Data as Part of a Program 143

10.1 The READ and DATA Statements 143
10.2 Problems 150
10.3 The RESTORE Statement 153
10.4 Problems 159
10.5 Review True-or-False Quiz 161

Chapter 11 Subroutines 162

11.1 The GOSUB and RETURN Statements 162
11.2 Problems 169
11.3 The ON GOSUB Statement 171
11.4 Problems 176
11.5 Review True-or-False Quiz 178

Chapter 12 Numerical Functions 179

12.1 Built-In Numerical Functions 179
12.2 Problems 186
12.3 User-Defined Functions: The DEF Statement 189
12.4 Multiple-Statement Functions 192
12.5 Problems 195
12.6 Review True-or-False Quiz 198

Chapter 13 More on Processing String Data 199

13.1 String and String-Related Functions 199
13.2 Combining Strings (Concatenation) 204
13.3 The Search Function INSTR 205
13.4 Problems 207
13.5 The BASIC Character Set 210
13.6 Problems 212
13.7 BASIC Conversion Functions 213
13.8 User-Defined Functions Involving String Data 219
13.9 Problems 222
13.10 Review True-or-False Quiz 223

Chapter 14 Arrays 224

14.1 One-Dimensional Arrays 224
14.2 The DIM Statement (Declaring Arrays) 231
14.3 Table Processing 233
14.4 Problems 238
14.5 Two-Dimensional Arrays 242
14.6 Problems 247
14.7 Review True-or-False Quiz 250

Chapter 15 Sorting 252

15.1 The Bubble Sort 252
15.2 The Shellsort and Binary Search Algorithms 258

15.3 Problems 260
15.4 Review True-or-False Quiz 263

Chapter 16 **Data Files 264**

16.1 File Statements 265
16.2 Detecting the End of a Sequential File 271
16.3 Problems 276
16.4 File Maintenance 279
16.5 Problems 285
16.6 External Sorting—The Merge Sort 287
16.7 Problems 290
16.8 Review True-or-False Quiz 291

Chapter 17 **Random Numbers and Their Application 292**

17.1 The RND Function 292
17.2 Problems 296
17.3 Random Integers 298
17.4 Simulation 299
17.5 A Statistical Application 301
17.6 Problems 302
17.7 Review True-or-False Quiz 304

Chapter 18 **Matrices 306**

18.1 Assigning Values to a Matrix: The MAT READ and MAT INPUT Statements 306
18.2 The MAT PRINT Statement 309
18.3 One-Dimensional Matrices 310
18.4 Matrix Operations 310
18.5 Matrix Functions 313
18.6 Problems 316
18.7 Review True-or-False Quiz 318

Appendix A **A Typical Session at the Computer Terminal 319**
Appendix B **BASIC Language Cross-Reference Guides 323**
Appendix C **Answers to Selected Problems 333**

Index 347

In addition to a BASIC interpreter or compiler, your system will include other systems programs. These will be programs that produce listings of your programs, "save" your programs on secondary storage devices for later use, assist you in finding errors in the programs you write, and, most important of all, a program that exercises general control over the entire system. This last program is called the **operating system.** It allows you to issue commands to the computer to "call up" and execute any of the other systems programs provided.

The emergence of computer science as a new discipline has been accompanied by a proliferation of new words and expressions. Although they are useful for talking about computers, they are for the most part absolutely unnecessary if your objective is to learn a computer language such as BASIC to help you solve problems. In our discussion of computer hardware and software, we have introduced only fundamental concepts and basic terminology. Even so, if this is your first exposure to computers, you may feel lost in this terminology. Don't be disheartened: Much of the new vocabulary has already been introduced. You will become more familiar with it and recognize its usefulness as you study the material in the subsequent chapters. You will also find it helpful to reread this chapter after you have written a few computer programs.

▶ 1.3 Topics for Independent Study

Prepare a short written or oral report for each suggested title. Reports 1–9 concern the history and evolution of computer technology; the remaining reports concern applications of computers.

1. The UNIVAC I: Its development and applications
2. ENIAC: The first large electronic computing machine
3. John von Neumann and the first stored-program computers
4. IBM's entry into the computer field
5. The Jacquard loom
6. Four generations of computers
7. The "chip": Microminiatured circuits
8. Computer languages: Machine versus high level
9. FORTRAN: The first high-level computer language
10. Personal computers and how they are used
11. Video display devices and their applications
12. Computers in small businesses
13. Database applications
14. Computers in the classroom
15. Computers in banking
16. Computers in the laboratory
17. Computers in entertainment
18. Computers in space
19. Computer networks
20. Distributed data processing

▶ 1.4 Review True-or-False Quiz

1. Any electronic device that can process data is called a computer. T F
2. Input/output devices, external storage devices, and the central processing unit are called computer peripherals. T F
3. An automobile that uses a microprocessor to control the gas and air mixture is correctly referred to as a computer system. T F
4. A computer system must contain at least one printer. T F

5. Disk-storage units are called *random access devices* because information stored on a disk is accessed by randomly searching portions of the disk until the desired data are found. T F
6. Tape units are called *sequential access devices* because the computer reads information from a tape by reading through the tape until the desired data are found. T F
7. The term RAM refers to disk-storage units. T F
8. There is a significant difference between memory units called RAM and memory units called ROM. T F
9. The function of a BASIC compiler is to translate BASIC programs into machine language. T F
10. The terms *compiler* and *interpreter* are used synonymously. T F
11. To solve problems using the BASIC language, you must know and understand what a compiler is or what an interpreter is. T F
12. A computer program written to automate a payroll system is an example of a systems program. T F
13. An operating system is a computer system. T F

b3.1. Multiply the salary by 0.02 to obtain a tentative bonus.
b3.2. If the tentative bonus is less than \$400, set the bonus to \$400; otherwise make the bonus equal to the tentative bonus.

Making this change, or *refinement,* we obtain the following more detailed algorithm for Step (b):

b1. Open the employee ledger.
b2. Read the next employee's name and salary.
b3.1. Multiply the salary by 0.02 to obtain a tentative bonus.
b3.2. If the tentative bonus is less than \$400, set the bonus to \$400; otherwise make the bonus equal to the tentative bonus.
b4. Write the employee's name, salary, and bonus on the bonus sheet.
b5. If all bonuses have not been determined, return to Step (b2).
b6. Close the ledger.

Our final detailed algorithm consists of eight steps—Step (a) followed by this seven-step algorithm for carrying out Step (b).

▶ REMARK

In this example, we started with a problem statement describing the task to be carried out (produce a year-end bonus report) and ended with an algorithm describing how to accomplish this task. The steps you take while designing an algorithm are called a **problem analysis.** For simple problems, a description of the input and output may lead directly to a final algorithm. For more complicated problems, a thorough analysis of alternative approaches to a solution may be required. In any case, the term *problem analysis* refers to the process of designing a suitable algorithm. ▶

The method used to design an algorithm for Example 1 illustrates three important principles of problem solving:

1. Begin by describing the input (information needed to carry out the specified task) and the output (the results that must be obtained). In the example, we described the input as salary information to be read from the employee ledger and the output as a table showing the names, salaries, and bonus amounts for the employees. An essential first step in the problem-solving process is to read and understand the problem statement. It is unlikely that a correct algorithm will be found if the task to be performed is not understood exactly. *Giving a clear and precise description of the input and output is an effective way to acquire an understanding of a problem statement.*

2. Identify individual subtasks that must be performed while carrying out the specified task. In Example 1 we identified the following two subtasks:

a. Write the title and column heading for the bonus sheet.
b. Read the ledger to determine and fill in the name, salary, and bonus for each employee.

If a complicated task can be broken down into simpler more manageable subtasks, the job of writing an algorithm can often be simplified significantly: You simply describe the order in which the subtasks are to be carried out. This was especially easy to do in Example 1—simply carry out subtask (a) followed by subtask (b). The process of breaking down a task into simpler subtasks is called **problem segmentation** or **modularization**—the subtasks are sometimes called **modules.** As in Example 1, the details of how to carry out these modules can be worked out after an algorithm has been found.

3. If more details are needed in an algorithm, include the additional details separately for each step. In Example 1 we started with the two-step algorithm:

a. Write the title and column headings for the bonus sheet.
b. Read the ledger to determine and fill in the name, salary, and bonus for each employee.

Next, we included more detail in Step (b) by breaking it down into these six steps:

b1. Open the employee ledger.
b2. Read the next employee's name and salary.
b3. Determine the employee's bonus.
b4. Write the employee's name, salary, and bonus on the bonus sheet.
b5. If all bonuses have not been determined, return to Step (b2).
b6. Close the ledger.

Finally, we included more detail in the algorithm by rewriting Step (b3) as follows:

b3.1. Multiply the salary by 0.02 to obtain a tentative bonus.
b3.2. If the tentative bonus is less than \$400, set the bonus to \$400; otherwise make the bonus equal to the tentative bonus.

The important thing to notice is that we introduced details into the algorithm by refining the steps separately—that is, by breaking down the individual steps one at a time—and not by combining steps or otherwise changing the algorithm. This method of designing a detailed algorithm is called the **method of stepwise refinement.** You begin with a simple algorithm that contains few details but that you know is correct. If necessary, you refine one or more of the steps to obtain a more detailed algorithm. If even more detail is needed, you refine one or more of the steps in the derived algorithm. By repeating this process of stepwise refinement, you can obtain an algorithm with whatever detail is needed. Moreover, however complicated the final algorithm, you can be sure that it is correct simply by knowing that you started with a correct algorithm and that each step was refined correctly.

The approach to problem solving used in Example 1 is called the **top-down approach** to problem solving or the **top-down design** of algorithms. The expression *top-down* comes from using the methods of modularization and stepwise refinement. You start at the top (the problem statement), break that task down into simpler tasks, then break those tasks into even simpler ones, and continue the process, all the while knowing how the tasks at each level of refinement combine, until the tasks at the lowest (final) level contain whatever detail is desired. The advantages to be gained by adhering to the three problem-solving principles of the top-down approach will become more evident as you work through the examples and problems in this book. The following example should help you better understand these three principles and their application.

EXAMPLE 2

A wholesale firm keeps a list of the items it sells in a card file. For each item, there is a single card containing a descriptive item name, the number of units in stock (this can be zero), the number of the warehouse in which the item is stored, and certain other information that will not concern us. Our task is to prepare a list of out-of-stock items for each warehouse.

PROBLEM ANALYSIS

The problem statement says that a separate list of out-of-stock items is needed for each warehouse. Thus, we will need to know the warehouse numbers. Let's assume we are told there are three warehouses numbered 127, 227, and 327. With this information, we can include these numbers as input rather than reading through the entire card file to determine them. We can now specify the input and output for our algorithm:

Input: Warehouse numbers 127, 227, and 327.
Card file: one card for each item.

Output: Three reports formatted as follows:

WAREHOUSE 127
(Out-of-stock items)

Hammers—Model 2960
Hammers—Model 3375
Saws—Model 1233
.
.
.

WAREHOUSE 227
(Out-of-stock items)
.
.
.

WAREHOUSE 327
(Out-of-stock items)
.
.
.

Having given a precise description of the input and output, we should determine what subtask or subtasks must be performed. The problem statement specifies that three reports are to be produced, one for each warehouse. If we arrange things so that reports are produced one at a time (this is a common practice when using computers), we can use the same procedure for each one. Specifically, for each warehouse number W, we will carry out the following subtask (named R for report):

Subtask R. Prepare the report for warehouse W.

Of course, we will need to include details describing how to carry out this subtask. But even without these details, we can write a simple algorithm that obviously is correct:

THE ALGORITHM

a. Assign 127 to W.
b. Carry out Subtask R.
c. Assign 227 to W.
d. Carry out Subtask R.
e. Assign 327 to W.
f. Carry out Subtask R.

Notice that Subtask R specifies that a report is to be prepared. As in Example 1, we can break this subtask down into two subordinate subtasks R1 and R2 as follows:

R1. Write the report header for warehouse W.
R2. Read through the card file to complete the report.

At this point, we should recognize that Step R1 requires no additional details — the output specification shows how the report header should be formatted. We should, however, include more details in Step R2. In the following algorithm for Subtask R, Steps R2.1 – R2.4 show one way to carry out Step R2:

R1. Write the report header for warehouse W.
R2.1. Turn to the first card.
R2.2. Read the warehouse number (call it N) and the units-on-hand figure (call it U).
R2.3. If N = W and U = 0, read the item name and write it on the report.
R2.4. If there is another card, turn to it and continue with Step R2.2.

Our final algorithm for the given problem statement consists of two parts: the original six-step algorithm [Steps (a) – (f)] that tells us when (but not how) to carry out Subtask R and the five-step algorithm (Steps R1, R2.1 – R2.4) that shows us how Subtask R can be accomplished.

▶ REMARK

Let's review the problem analysis carried out in this example:

Input/output specification. Our attempt to give a precise description of the input and output led us to include the warehouse numbers 127, 227, and 327 as input. As a consequence, the final algorithm is simpler than what would have been obtained had we begun by reading the entire card file just to determine the warehouse numbers.

Modularization. Our attempt to identify subtasks led us to conclude that there was but one major subtask, namely,

Subtask R. Prepare the report for warehouse W.

By using this subtask, it was easy to write an algorithm [Steps (a)–(f)] that lacked only in the details needed to carry out Subtask R.

Stepwise refinement. Our next job was to show how to carry out Subtask R. We began by breaking it down into these two subordinate subtasks:

R1. Write the report header for warehouse W.
R2. Read through the card file to complete the report.

Notice that this two-step refinement of Subtask R represents an application of the principle of modularization applied to Subtask R. At this point we recognized that only Step R2 was lacking in details. We supplied these details by writing a short four-step algorithm (Steps R2.1–R2.4) for Step R2.

▶ 2.2 Variables

Algorithms can often be stated clearly if symbols are used to denote certain values. Symbols are especially helpful when used to denote values that may change during the process of performing the steps in an algorithm. The symbols W, N, and U used in Example 2 illustrate this practice.

A value that can change during a process is called a **variable.** A symbol used to denote such a variable is the *name* of the variable. Thus W, N, and U in Example 2 are names of variables. It is common practice, however, to refer to the *symbol* as being the variable itself, rather than just its name. For instance, Step (a) of the algorithm for Example 2 says to assign 127 to W. Certainly this is less confusing than saying "assign 127 to the variable whose name is W."

The following two examples further illustrate the use of variables in algorithms.

EXAMPLE 3 Let's find an algorithm to determine the largest number in a list of numbers.

PROBLEM ANALYSIS

Input: A list of numbers.

Output: The largest number in the list.

One way to determine the largest number in a list of numbers is to read the numbers one at a time, remembering only the largest of those already read. To help us give a precise description of this process, let's use two symbols:

L to denote the largest of the numbers already read
X to denote the number currently being read

The following algorithm can now be written:

a. Read the first number and denote it by L.
b. Read the next number and denote it by X.
c. If X is larger than L, assign X to L.
d. If all numbers have not been read, go to Step (b).
e. Write the value of L and stop.

To verify this algorithm for the list of numbers

4, 5, 3, 6, 6, 2, 1, 8, 7, 3

we simply proceed step by step through the algorithm, always keeping track of the latest values of L and X. An orderly way to do this is to complete an assignment table, as follows:

5. If the numbers of hours shown on the first four time cards are 20, 32, 40, and 45, respectively, what amounts will be written on these cards?
6. What is the base hourly rate for each employee?
7. What is the overtime rate?
8. Explain Step (c).

In Problems 9–12, what will be printed when each algorithm is carried out?

9. **a.** Let SUM = 0 and N = 1.
   b. Add N to SUM.
   c. Increase N by 1.
   d. If N ≤ 6, return to Step (b).
   e. Print the value SUM and stop.
10. **a.** Let PROD = 1 and N = 1.
    b. Print the values N and PROD on one line.
    c. Increase N by 1.
    d. Multiply PROD by N.
    e. If N ≤ 5, return to Step (b).
    f. Print the values of N and PROD on one line.
    g. Stop.
11. **a.** Let A = 1, B = 1, and F = 2.
    b. If F > 50, print the value F and stop.
    c. Assign the values of B and F to A and B, respectively.
    d. Evaluate A + B and assign this value to F.
    e. Return to Step (b).
12. **a.** Let NUM = 56, SUM = 1, and D = 2.
    b. If D is a factor of NUM, add D to SUM and print D.
    c. Increase D by 1.
    d. If D ≤ NUM/2, return to Step (b).
    e. Print the value SUM and stop.

In Problems 13–19, write an algorithm to carry out each task specified.

13. A retail store's monthly sales report shows, for each item, the cost, the sale price, and the number sold. Prepare a three-column report with the column headings ITEM, GROSS SALES, and INCOME.
14. Each of several 3 × 5 cards contains an employee's name, Social Security number, job classification, and date hired. Prepare a report showing the names, job classifications, and complete years of service for employees who have been with the company for more than 10 years.
15. A summary sheet of an investor's stock portfolio shows, for each stock, the corporation name, the number of shares owned, the current price, and the earnings as reported for the most recent year. Prepare a six-column report with the column headings CORP. NAME, NO. OF SHARES, PRICE, EARNINGS, EQUITY, and PRICE/EARNINGS. Use this formula:

    Equity = number of shares × price

16. Each of several cards contains a single number. Determine the sum and the average of all the numbers. (Use a variable N to count how many cards are read and a variable SUM to keep track of the sum of numbers already read.)
17. Each of several cards contains a single number. On each card, write the letter G if the number is greater than the average of all the numbers; otherwise, write the letter L. (You must read through the cards twice: once to find the average and again to determine whether to write the letter G or the letter L on the cards.)
18. A local supermarket has installed a check validation machine. To use this service, a customer must have previously obtained an identification card containing a magnetic strip and a four-digit code. Instructions showing how to insert the identification card into a special magnetic-strip reader appear on the front panel. To validate a check, a customer must present the identification card to the machine, enter the four-digit code, enter the amount of the check, and place the check, blank side toward the customer, in a clearly labeled punch unit. To begin this process,

the CLEAR key must be pressed, and, after each of the two entries has been made, the ENTER key must be pressed. Prepare an algorithm giving instructions for validating a check.

19. Write an algorithm describing the steps to be taken to cast a ballot in a national election. Assume that a person using this algorithm is a registered voter and has just entered the building in which voting is to take place. While in the voting booth, the voter should simply be instructed to vote. No instructions concerning the actual filling out of a ballot are to be given.

▶ 2.4 Review True-or-False Quiz

1. The terms *algorithm* and *process* are synonymous. T F
2. The following steps describe an algorithm:
   a. Let N = 0.
   b. Increase N by 10.
   c. Divide N by 2.
   d. If N < 10, go to Step (b).
   e. Stop. T F
3. A computer program should describe an algorithm. T F
4. Every algorithm can be translated into a computer program. T F
5. The expression *heuristic process* refers to an algorithm. T F
6. It is always easier to verify the correctness of an algorithm that describes a specific task than the correctness of a more general algorithm. T F
7. The term *variable* refers to a value that can change during a process. T F
8. A *problem analysis* is the process of discovering a correct algorithm. T F
9. The *method of stepwise refinement* is a method of problem solving in which successive steps in an algorithm are combined to produce an algorithm that is easier to read. T F
10. *Top-down design* involves the process of *stepwise refinement.* T F

▶ 3.4 Problems

1. Evaluate the following:

a. `2+3*5`	b. `5*7-2`	c. `-4+2`
d. `-(4+2)`	e. `-3*5`	f. `-3^2`
g. `1+2^3*2`	h. `6/2*3`	i. `1/2/2`
j. `-2*3/2*3`	k. `2^2^3`	l. `(2+(3*4-5))^0.5`

2. For A = 2, B = 3, and X = 2, evaluate each of the following:

a. `A+B/X`	b. `(A+B)/2*X`	c. `B/A/X`
d. `B/(A*X)`	e. `A+X^3`	f. `(A+B)^X`
g. `B^A/X`	h. `B+A/B-A`	i. `A^B+X`

3. Some of the following are not admissible BASIC expressions. Explain why.

a. `(Y+Z)X`	b. `X2*36`	c. `A*(2.1-7B)`
d. `X^-2`	e. `A+-B`	f. `-(A+2B)`
g. `2X+2`	h. `X2+2`	i. `X-2^2`
j. `A12+B3`	k. `A+(+B)`	l. `A2-(-A2)`
m. `5E1.0`	n. `-9^0.5`	o. `-9^1/2`

4. Write BASIC expressions for these arithmetic expressions.

a. $0.06P$	b. $5x + 5y$	c. $p(1 + r)^t$
d. $\dfrac{f}{s - c}$	e. $\dfrac{a}{b} + \dfrac{c}{d}$	f. $\dfrac{a + b}{c + d}$
g. $ax^2 + bx + c$	h. $\sqrt{b^2 - 4ac}$	i. $\dfrac{a + p}{pr}$

▶ 3.5 The LET Statement: Assigning Values to Variables

In Section 3.3 you saw how to write arithmetic expressions in a form acceptable to the computer. You will now learn how to instruct the computer to evaluate such expressions.

A BASIC **program statement,** also called a **programming line,** consists of an instruction to the computer preceded by an unsigned integer called the **line number.** The general form is

line number BASIC instruction

For example,

```
100 LET A=2+5
```

is a BASIC statement with line number 100. This statement, called a LET statement, will cause the computer to evaluate the sum 2 + 5 and then assign this value to A.

A **BASIC program** is a collection of BASIC program statements. The instructions are executed by the computer in the order determined by increasing line numbers, unless some instruction overrides this order. In this book we'll often use the expression *BASIC statement* to refer to a BASIC program statement without its line number. This practice conforms to current programming terminology.

The general form of our first BASIC statement, the LET statement, is

ln LET **v** = **e**

or, more simply,

ln v = **e** (LET is optional.)

where **ln** stands for line number, **v** denotes a variable name, and **e** denotes a BASIC expression that may simply be a constant. This statement directs the computer to evaluate