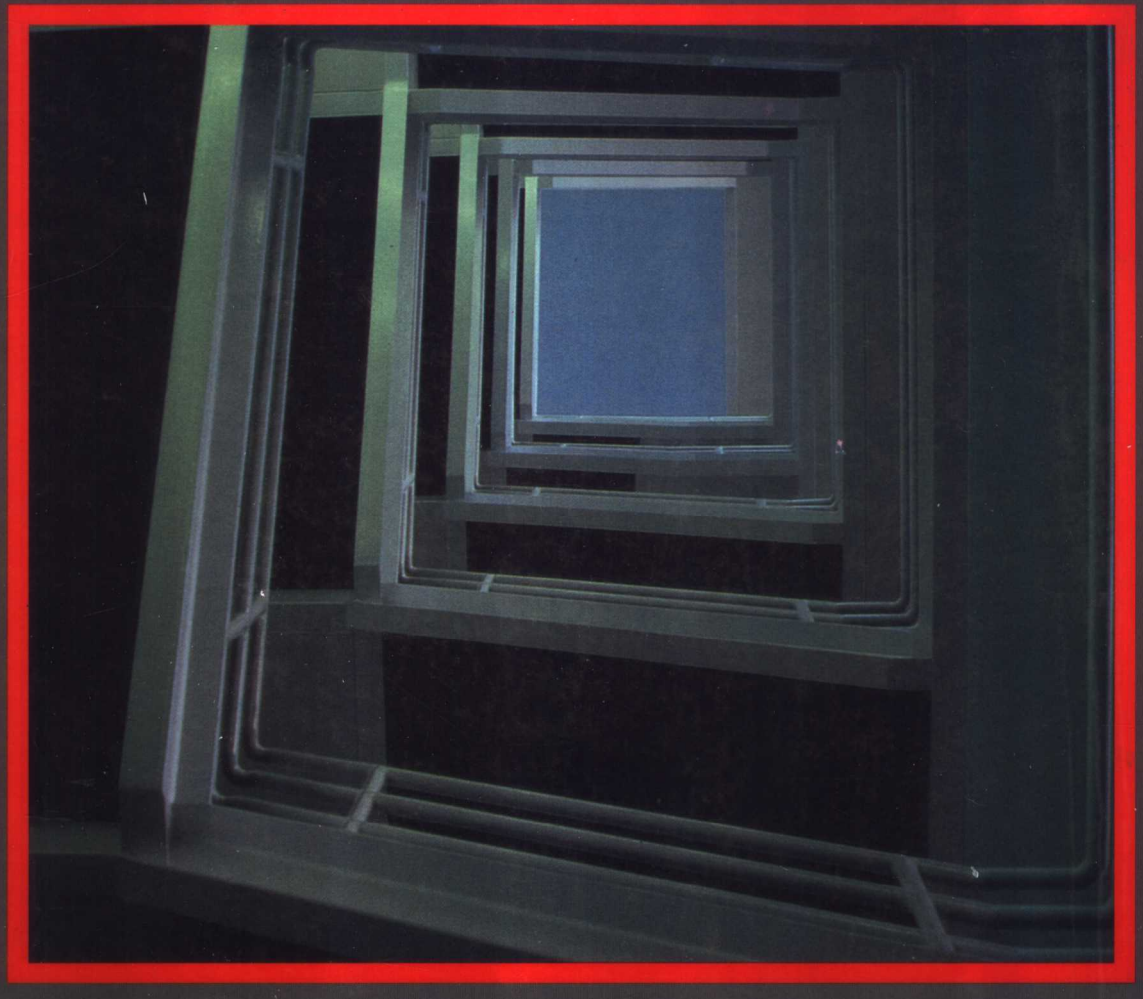
Problem Solving and Structured Programming with Pascal

Ali Behforooz

Martin O. Holoien



Problem Solving and Structured Programming with Pascal

Ali Behforooz
Moorhead State University

Martin O. Holoien
University of California, Santa Barbara

Brooks/Cole Publishing Company
Monterey, California

To my parents —*A.B.*

To my parents and my grandchildren,
Annie, Kari, and Isaac —*M.H.*

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

Library of Congress Cataloging-in-Publication Data
Behforooz, Ali, [date]
Problem solving and structured programming with Pascal.

Includes index.
1. PASCAL (Computer program language) 2. Structured programming. I. Holoien, Martin O., [date]
QA76.73.P2B4445 1986 005.13'3 85-28008

ISBN 0-534-05736-5

Sponsoring Editors: *Neil Oatley, Cynthia C. Stormer*
Editorial Assistants: *Gabriele Bert, Corinne Kibbe*
Production Editor: *Michael G. Oates*
Production Associate: *Dorothy J. Bell*
Manuscript Editor: *Susan Thornton*
Permissions Editor: *Carline Haga*
Interior and Cover Design: *Katherine Minerva*
Cover Photo: *Walter Nelson/Fran Heyl Associates*
Art Coordinators: *Michèle Judge, Lisa Torri*
Interior Illustration: *Vantage Art, Massepequa, New York*
Typesetting: *Syntax International, Singapore*
Cover Printing: *Phoenix Color Corp., Long Island City, New York*
Printing and Binding: *R.R. Donnelley & Sons Co., Crawfordsville, Indiana*

Preface

Problem Solving and Structured Programming with Pascal is designed for a first course in problem solving and computer programming in which Pascal is the programming language. No previous knowledge of programming is presumed. The mathematics background required is that equivalent to three years of high school mathematics or a course in college algebra and trigonometry.

Our aim in writing this book is to help teach students the skills and concepts needed to become good problem solvers capable of using the Pascal language well. Not until these skills are developed is the Pascal language presented. Subsequently, as more powerful aspects of the language are developed, so are good programming techniques—all based on the solid foundation of problem solving and algorithm development presented in the first chapter.

Algorithm and program development are presented from the top-down approach. Students learn how to produce precise statements of problems as well as analyze input and output requirements. They are taught techniques for developing problem solutions in modules and introduced early to the concept of subprograms in Pascal. As students learn the programming phase of problem solving, they are taught how to design and test each module individually first, then put modules together in a single program that produces the required results. As a consequence of our approach, students should develop confidence in solving any problem presented to them and should be able to produce good, understandable solutions to such problems in which the computer is the major tool.

All program examples in this book have been run on a DEC VAX11/750 computer system running under DEC's RMS operating system. Any file processing done in the programs is, therefore, done according to the requirements of the VAX11/750.

To provide a guide to the pedagogical development used, we now review the approach taken in each chapter. In Chapter 1, the fundamental concepts of problem solving are discussed quite apart from any programming language features. Four phases of problem solving—precisely defining the problem, analyzing the input and output requirements, developing the solution algorithm, and developing the Pascal program—are presented, as is the top-down approach to problem solving. Considerable time is devoted to algorithms, their properties, their modular development, and methods for representing them.

Chapter 2 presents some elementary programming language concepts with particular emphasis on how they occur in Pascal. The standard data types, and how to make Pascal declarations that relate to each of them, are discussed. Elementary input and output concepts are explained, as well as other features of Pascal needed to be able to write a simple but complete program. Chapter 3 leads the student into the basics of control structures, such as the *if-then-else, for-loop, while-loop,* and *repeat-until* structures.

Chapter 4 takes a careful look at program errors and often-overlooked concepts related to that topic. Three kinds of program errors (syntax, logic, and run-time) are discussed, and techniques are introduced to prevent them in the first place. Methods for locating and removing the errors, if they do occur, are also presented. Actual programmed examples are analyzed for various errors, and students are shown how to debug them. No new Pascal concepts are introduced in this chapter, so if lack of time prevents its inclusion in the course, students will not miss any aspects of the Pascal language. However, the authors have found it important to formally discuss in class the occurrence and removal of program errors.

Subprograms are introduced in Chapter 5, as well as the syntax and application of functions and procedures in Pascal. From this point on in the book, modular programming is emphasized and examples are given to reinforce that principle.

Array declarations and applications are considered in Chapter 6, as are text files and the processing of text files. After thorough instruction in the topics in this chapter, students will be able to write programs more like those developed by professional applications programmers.

In Chapter 7, we discuss advanced topics related to control structures. Specifically, the following capabilities of Pascal are taught: case structure, if-then-else-if structure, and nested loops. Several complete programming examples illustrate these concepts.

Advanced topics related to functions and procedures are the focus of Chapter 8. Various kinds of parameters are discussed, in particular, the use of functions and procedures as parameters. The important concept of recursion and recursive subprograms is a significant part of this chapter. Diagrams and examples are carefully presented to make this vital topic lucid for the student.

Chapter 9 is devoted to advanced concepts related to structured data types. In particular, we discuss records, linked lists using pointer-type data, and files of records. If the course being offered is intended primarily to teach the fundamental features of the Pascal language, the instructor may choose to omit the sections on linked lists and pointer-type data.

Finally, Chapter 10 contains some basic concepts of data structures such as stacks, queues, linked lists, and binary trees. Examples are given to illustrate the implementation and processing of these data structures in Pascal. If time does not permit, this chapter may be omitted from a course in problem solving and Pascal programming. Nevertheless, we believe that if students are introduced to these concepts at the end of such a course, they will be better prepared to progress through successive computer science courses. However, if the students are not primarily majors in computer science, this chapter should probably be omitted.

Appendix A is a list of all standard Pascal functions, procedures, and identifiers, as well as a list of all reserved words. Anyone who has taught a course in Pascal will appreciate this ready reference of these facts.

Syntax diagrams are the topic of Appendix B. Every Pascal statement structure is clearly described by means of a syntax diagram. Any questions regarding the punctuation or use of reserved words in any Pascal statement can be answered by referring to the appropriate section of this appendix. Diagrams are arranged alphabetically by the word normally used to identify a given Pascal statement.

Included among the unusual features of this book are the following:

1. The unique approach of teaching problem solving before any mention is made of the Pascal language.
2. A complete chapter devoted to preventing, locating, and removing program errors.
3. Early introduction of subprograms so as to facilitate teaching modular design of programs.
4. A large number of solved problems demonstrating problem-solving techniques and good programming practices.
5. An unusually large number of exercises for each chapter, making it possible to select a specific problem to emphasize a given concept.
6. Chapter summaries that make it possible to review in a short time the concepts associated with a given day's topics.

Many helpful suggestions were made by the reviewers of this manuscript, almost all of which were incorporated in the final version. We extend sincere thanks to Andrew Batts, Murray State University; Dwight Caughfield, Abilene Christian University; James Clark, University of Tennessee–Martin; Henry Etlinger, Rochester Institute of Technology; Joe Grimes, California State University–San Luis Obispo; Norman Lindquist, Saint Andrews College; George Medelinskas, North Shore Community College; Clinton Smullen, University of Tennessee–Chattanooga; and Allan Tharp, North Carolina State University. Their thoughtful comments and creative ideas went far toward making this book as useful as we hope it will be.

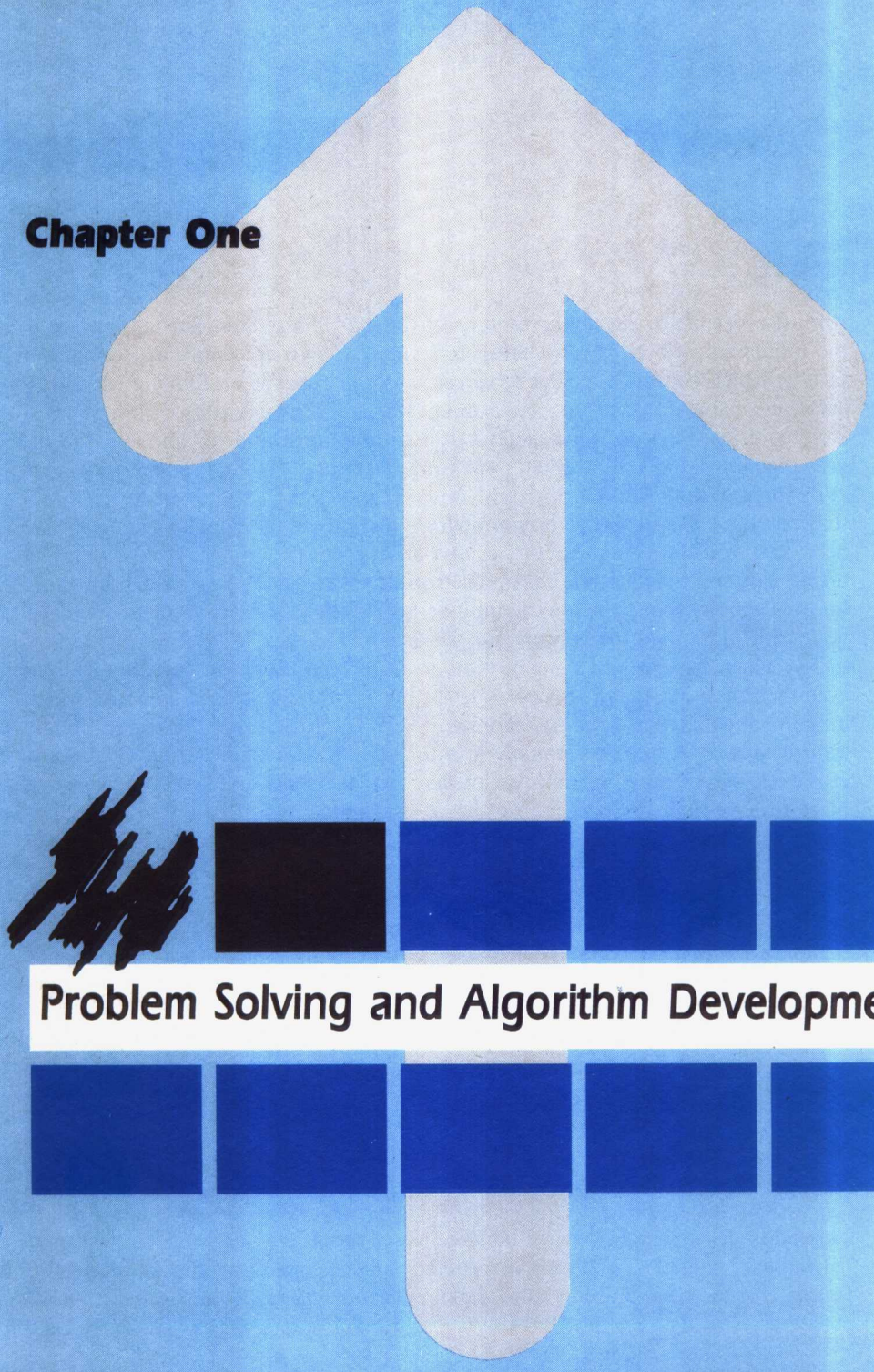
Ali Behforooz
Martin O. Holoien

Contents

Chapter Six
More on Type and Variable Declarations 230

Chapter Seven
More on Control Structures 283

Chapter Eight
More on Functions and Procedures Plus Recursion 328



Chapter One

Problem Solving and Algorithm Development

Introduction

Almost everyone in developed nations of the world is at least partially aware of the influential role played by computers today. There are many aspects of our lives in which computers affect us directly, as, for example, in making travel reservations, bank transactions, retail purchases, college registrations, library transactions, driver's license renewals, payroll transactions, and on and on, the list growing almost daily. How often have you heard from customer service personnel, "I'm sorry, I can't handle this right now because the computer is down"? And that response has likely made you angry. One reason for your anger may have been that you didn't understand what was happening. You may have thought, "How can computers cause so much trouble in my life?" Had you been more knowledgeable about the way computer processing of information takes place, your frustation might have been less.

If you have experienced some hostility toward computers, you probably realized even then that they really are necessary for life to continue in the manner to which millions of us have become accustomed. That's not to say that there aren't many improvements that could be made in a lot of computer applications. There are. However, maybe an important reason that improvements aren't taking place any more rapidly than they are is that the general public is not educated enough to know what they can expect from computers, what changes they can realistically press for. One of the goals of this book is to give you the familiarity with computers that will let you better understand many situations in which they play a role in the world around you. Furthermore, when you finish this book we hope you'll be able to use many computer capabilities to your own significant benefit. The computer is, after all, probably the most important tool available to assist us in problem-solving and decision-making situations.

In order to utilize computers to the greatest advantage, there are at least three areas of knowledge in which one should acquire some facility:

1. problem-solving skills,
2. familiarity with at least one programming language, and
3. acquaintance with a computing environment.

This book deals primarily with the first two of these.

Chapter 1 is devoted entirely to problem solving and algorithm development. In this chapter we explore in some detail concepts and skills that facilitate those processes. You will see that learning a programming language is relatively easy

once a proper foundation has been laid in these areas. Although much more of this book is devoted to teaching the Pascal programming language than it is to problem solving and algorithm development, do not underestimate the importance of Chapter 1. It provides a basis for all that follows.

You will notice also that as we discuss aspects of the programming language Pascal, we will stress certain techniques that are commonly called *programming style*. If you plan to become an effective user of computers, pay special attention to the matter of programming style. It can mean the difference between producing programs that are understandable and useful and ones that even the developer is unable to comprehend once they have become "cold" to the memory.

Some Problem-Solving Examples

Before we discuss certain aspects of the problem-solving process more formally, let's consider some examples to try to gain some insight into how one is successful at problem solving.

EXAMPLE 1

Problem: Bob and Susan Jones and their children, Jim and Annie, had a summer home at which they spent almost every weekend for several years. It was their habit to start for the city at about the same time each Sunday afternoon, and as a result, they invariably met a freight train coming toward them on the railroad tracks as they approached a certain section of the highway. Just as certain as meeting the train was Jim's question "How long is the train, Mom?"

Solution: Susan suggested to Jim, "Count the number of railroad cars as we go by, then I'll find out from the railroad company how long each car is, then we can multiply the number of cars by the length of each and find out how long the train is."

So Jim would begin to count the cars. He was never able to complete the task, however, because he always lost track when some of the cars would disappear behind a clump of trees, or when Annie would ask him a distracting question. So as good as the suggested solution for Jim's problem seemed, it was simply impossible to solve the problem that way.

Some years later when Jim was in the eighth grade, he borrowed a stopwatch from one of his friends. On the family's return to the city at the customary time that Sunday afternoon, sure enough, there was the train coming toward the car, and Jim was ready to solve his problem of several years' standing. They saw the train a way off in front of them as Jim asked his father, "Dad, would you please keep the car's speed at 55 miles per hour as we go by the train today? I'll tell you why when we have gone past the train." Of course, Dad agreed, and soon the train was a short way from the automobile. Just as the front of the locomotive was

alongside the automobile, Jim started the stopwatch, keeping it running until the end of the caboose went by the automobile.

"I'm finally going to find out how long that train is," he told his family. "I checked with the railroad company about the speed of this train at this section of its trip and was told that it is 50 miles per hour. Dad kept the speed of the automobile at 55 miles per hour, so that means that the car and the train were going past each other at the rate of 105 miles per hour. I just now measured how long it took for our car and the train to pass each other and found it to be 36 seconds, which is equal to 0.01 hour because 1 hour equals 60 × 60 seconds, or 3600 seconds. Therefore, the train's length can be computed by multiplying the speed at which the car and the train passed each other by the time it took to do so. This gives the result

$$105 \text{ mph} \times 0.01 \text{ hr} = 1.05 \text{ mi}$$

Thus the train is 1.05 miles long."

Jim's family was pleasantly surprised at Jim's being able to solve the problem but wondered why this second method worked although the earlier suggested solution hadn't. If you think about it for a bit you will soon realize why. Although the first method was simple enough, it was impossible to carry out because no one had thought of the difficulties that would arise in counting the railroad cars. The second method was successful because it involved the completion of tasks possible to perform with the resources readily available, and possible to complete in a reasonable amount of time.

■

EXAMPLE 2

Problem: Suppose you are asked to develop a step-by-step procedure for maintaining a correct record of the balance in a checking account. To do this you need to know the initial amount placed in the account and information about every transaction affecting the account. These transactions would include the following:

1. withdrawals,
2. deposits,
3. service charges, and
4. interest earned.

Solution: Assume the bank complies with these rules for checking accounts:

1. If at any time during the month the balance becomes less than \$200.00, a monthly fee of \$5.00 is charged, plus a service charge of \$0.20 per check. These charges are automatically charged to the account.
2. If the balance remains greater than or equal to \$200.00 and less than \$1000.00, no monthly fee is charged, nor are there any charges for writing checks.
3. If the balance is \$1000.00 or more, not only are there no charges assessed but interest is paid at the rate of 0.5 percent per month on the balance over \$200.00. This interest is automatically added to the account on the last day of the month.

Next we identify some names to be associated with the solution we develop:

InitBalance: The initial balance in the account.
Balance: The current balance in the account.
TrValue: The amount of the transaction.
TrCode: A code for identifying the transaction. Let's assume W for a withdrawal, D for a deposit, I for interest, and S for a service charge.
Date: The date of the transaction.

For each deposit or withdrawal we read the transaction code and the amount of the transaction from a notebook. To determine interest or service charge, we check the current balance at the end of the day's transaction and apply the rules stated previously.

The procedure that is developed is to be applied daily to give the correct checkbook balance at the end of each day. Here is such a procedure:

```
1.0 Assign to Balance the value of InitBalance.
2.0 Perform steps 3.0, 4.0, and 5.0 until there are no more transactions; then proceed to step 6.0.
3.0     Read TrCode and TrValue.
4.0     If TrCode = W then
            assign to Balance the difference Balance – TrValue.
5.0     If TrCode = D then
            assign to Balance the sum Balance + TrValue.
6.0 If Balance < 200 then
        assign to TrCode the value S;
        assign to TrValue the product of the number of checks and 0.20 plus
        5.00 if this charge has not been added this month.
7.0 If Balance > = 1000 then
        assign to TrCode the value I;
        assign to TrValue the product (Balance – 200) * 0.005.
8.0 If TrCode = S then
        assign to Balance the difference, Balance – TrValue.
9.0 If TrCode = I then
        assign to Balance the sum Balance + TrValue.
10.0 Terminate all processing.
```

Upon checking this solution to our problem it becomes apparent that all is not well.

1. We need to know the number of withdrawals (checks written) so that the service charge can be computed if necessary.
2. If the balance drops below $200.00 we must know whether the $5.00 service charge has already been added to the account this month.
3. When the balance reaches $1000.00 or more, we add interest only if we are at the last day of the month.
4. No provision has been made for overdrafts.

To revise our solution we shall introduce a variable name to indicate the number of withdrawal transactions, and another to let us know whether the fixed monthly service charge has been made to the account this month. Suppose we call the number of withdrawals *WCount* and use the name *Added* to specify whether the monthly service charge has already been added this month. At the start of each month we initialize Added to *no* and at the time when we add the monthly service charge we set Added to *yes*.

Here is a revised procedure for maintaining a checkbook balance:

```
1.0 Assign to Balance the value of InitBalance.
2.0 Read the date, and if it is the first of the month, assign to Added the value no.
3.0 Set WCount to zero to indicate that no withdrawals have occurred yet.
4.0 Perform steps 5.0, 6.0, and 7.0 until there are no more transaction data; then
    proceed to step 8.
5.0     Read values for TrCode and TrValue.
6.0     If TrCode = W then
            if Balance > = TrValue then
                assign to Balance the value Balance – TrValue.
                Add 1 to WCount.
            else report an overdraft.
7.0     If TrCode = D then
            assign to Balance the value Balance + TrValue.
8.0 If Balance < 200 then
        Set TrCode = S.
        Assign to TrValue the product WCount * 0.20.
        If Added = no then
            add 5 to TrValue.
            set Added to yes.
9.0 If Balance > 1000 and Date is end of month then
        set TrCode = I.
        Set TrValue = (Balance – 200) * 0.005 (this is the monthly interest)
10.0 If Code = S then
        assign to Balance the difference Balance – TrValue.
11.0 If Code = I then
        assign to Balance + TrValue.
12.0 Terminate all processing.
```

Note that in the process of obtaining this solution to the problem, we started with one procedure, saw some fallacies in it, and modified it until the desired solution was obtained.

■

EXAMPLE 3

Problem: Develop a step-by-step procedure that will read a set of numbers and compute and report the average of those numbers.