# STRUCTURED PROBLEM SOLVING WITH PASCAL

Lawrence Mazlack

# STRUCTURED PROBLEM SOLVING WITH PASCAL

Lawrence J. Mazlack
Information Systems
University of Cincinnati

Holt, Rinehart and Winston
New York Chicago San Francisco Philadelphia Montreal
Toronto London Sydney Tokyo Mexico City
Rio de Janeiro Madrid

The photograph for the cover is courtesy of Chromatics, Inc.


Address correspondence to: 383 Madison Avenue, New York, NY 10017

**Library of Congress Cataloging in Publication Data**

Mazlack, Lawrence J.
Structured problem solving with Pascal.

Includes index.
1. PASCAL (Computer program language) 2. Structured programming. I. Title.
QA76.73.P2M36 1983 001.64'24 82-21245
ISBN-0-03-060153-3

Printed in the United States of America

Published simultaneously in Canada

3 4 5 6 016 9 8 7 6 5 4 3 2 1

CBS COLLEGE PUBLISHING
Holt, Rinehart and Winston
The Dryden Press
Saunders College Publishing

# PREFACE

A computer language is a tool that can be used for solving many different types of problems. Computer languages are used to communicate with the computer. The language that is used influences the structure of the design of the problem's computer solution. There are a variety of different computer languages. Languages are different because of what they are designed to do, how they go about doing it, and how difficult they are to use.

The structure and form of a computer language have a great deal to do with how a problem is solved on a computer. Some languages encourage programming styles that create programs that are easier to design, implement, and modify. Pascal has been designed to help people learn how to program in the best possible style.

When people first started to use computers, the best method of using them to solve problems was unknown. More and more has been learned about solving problems with the aid of a computer. Recently, a powerful way of using computers to solve problems has been developed. This better way addresses how a problem is to be analyzed before it is written in a computer language and how the problem should be described to the machine. This powerful way of problem analysis is known as *top-down analysis*. The better way of programming that has been developed is known as *structured programming*. This book addresses the technique developed of top-down problem solution design and the structured use of a computer programming language.

Computer languages may be divided into "low" and "high" level langauges. A low level language requires the programmer to be very concerned with precisely how a given computer goes about solving or executing a program. A high level language is intended to resemble human communication forms. A high level language approaches either a "natural" language such as English ("story like") or a mathematical language that has the form of a set of mathematical statements. Consequently, the use of a low level language results in a program which has been developed in terms of the machine being used, whereas a high level language results in a program which has been developed in terms of the problem being solved.

The advantage of using a high level language is that the problem solver can spend more time on how to go about solving a problem and less time on the mechanical details of how the computer executes a given data manipulation. A greater freedom from concern with mechanical details usually allows the problem solver (the programmer) to resolve many more information processing problems in a given time.

There are many high level languages available. This book describes the

essential elements of Pascal, a high level language. Pascal was originally introduced by Niklaus Wirth in the early seventies as a first language for students with no programming experience. It is now widely available on both large and small computers. It has the capacity to easily manipulate arithmetic, character, and boolean values. Pascal has significant data description facilities that allow the development of useful data structures. However, more important than its considerable manipulative capability is that the problem solution method can be reflected in the program's structure. Pascal has a variety of powerful "structured" control facilities and useful data structures.

The design of Pascal makes it an ideal tool for writing structured programs. Structured programs generally are simpler in form and are more easily reasoned about. The structured programming technique has been found to have both theoretical and practical advantages. The theoretical advantage in using structured programming is that it produces programs that are closer to being provably correct than any other programming technique.

The practical advantage in structured programming is that the structure of the programs produced is usually simpler and can reflect a top-down/problem solution analysis. Top-down analysis is a progressive problem subdivision analysis technique. This technique produces solutions that are more understandable and allow greater control over errors.

Pascal was designed by one man, Niklaus Wirth. For a period of time, there was no agreed upon standard description of Pascal. Wirth's book (K. Jensen and N. Wirth, *Pascal User Manual and Report*, 2nd ed., Springer-Verlag, New York, 1974) served both as the best description of the goals of Pascal and as a description of what should be in the Pascal language. Several professional groups worked together to develop a standard description of Pascal that all compiler implementations should conform to. The last result of this was ANSI X3J9 "Second Draft Proposal ISO/DP 7185.1." This is available in *Pascal News*, No. 20, December 1980. In 1982, this version became the international standard and is what is referred to as "standard" Pascal in this book.

## OF PARTICULAR INTEREST TO THE INSTRUCTOR

This text is oriented towards solving problems that are concerned with handling general collections of data as opposed to problems of a highly mathematical nature. The programming examples have been kept as simple as possible. This focuses attention on what the Pascal program statements are to be doing instead of on the design of the problems that are being solved. The formation of data into groups known as records will be addressed.

The examples and problems in the text require only simple computational skills. Specifically, problems requiring any scientific knowledge or background have been avoided. A few of the problems at the end of the chapters come from scientific or engineering origins; however, the computations required do not include complex

equations or background knowledge not provided in the problem statement.

The presentation of the material in the text has been modularized to reduce the need for serial presentation. The shaded box at the start of each section indicates what other sections are suggested as background. This allows the order in which the material is learned to be different than the sequence chosen by the author. Also, some topics can be de-emphasised while others can be eliminated.

The sequence of topics will probably be most satisfactory to a person who is very comfortable with the concepts of top-down analysis and programming. A person who is primarily interested in Pascal as a language, with only a developing interest in the top-down process, may well prefer a different topic order. For example, the first topic in this book that aids in program organization and control is subprograms (Chapter 6, PROCEDURE and FUNCTION Fundamentals). Subprograms are covered before any other control structure (such as iteration or selection). This is somewhat different than the sequence that has been common in the past. Although it is felt that subprograms should be used early, users of this book are not constrained to do so. They can just delay covering the contents of Chapter 6, which introduces subprograms, until just before Chapter 12, which expands the discussion of Chapter 6. Likewise, some people may be more comfortable covering selection (IF, CASE) before iteration (WHILE, REPEAT-UNTIL, FOR). This can be done easily. Also, the topics can be interchanged or delayed since the examples in most of the chapters of the book do not require a complete knowledge of the preceding chapters. In much the same manner, most sections within a chapter do not require that all the previous material in the chapter be covered before a particular section is covered. To aid in course design, the prequisite background required for each section is indicated in the shaded box at the start of each section.

The Pascal standard specifies a set of characters that can be used to construct a Pascal program. The standard also describes a set of alternate characters that can be used for the special purpose symbols. (Appendix C describes the standard and alternate character sets along with character sets available for various computer systems.) As few computer systems are capable of printing the complete Pascal standard character set, virtually all programs written in Pascal that are to run on existing computing systems must use some combination of standard and alternate characters. (Some relatively popular computer systems cannot represent all the necessary characters using a combination of standard and alternate characters.) This book uses the combination of standard and alternate character sets that can be displayed by machines using the EBCDIC representation. Many Pascal implementations for computer systems not using EBCDIC either also use this representation for their character set or will accept all of the alternate characters as valid input. Whatever character set your computer uses, this book can be used with it as long as your Pascal compiler conforms to the Pascal standard.

*Lawrence J. Mazlack*

# CONTENTS

## APPENDIXES

# Chapter 1

# INTRODUCTION

### Objectives
Provide a general introduction to computers and to problem solving on computers.

### Suggested Background
This chapter does not require any specific background. Although a prior knowledge of computers and the role that they play in the world would be helpful, it is not necessary. This chapter presents a general discussion of the topics included in it. The reader is encouraged to seek additional information in any of the numerous introductory books to computers that are available.

Computers are a part of today's world. Almost every day, they affect us by what they do. They can do some things very well and are inadequate for other tasks. The things that they can do well are often things that are simple for people to do. The computer's strength is that it can do simple things faster, cheaper, and sometimes more reliably than people can.

## 1.1 THE PARTS OF A COMPUTER

### Objectives
Identify the five basic functions of a computer and relate these functions to the major parts of a computer.

### Suggested Background
None.

Before a person learns to drive a car, there is a need to have a general idea of the pieces of a car and their functions. For example, the ideas needed include the knowledge that: the engine makes the care move, the brakes stop it, the horn makes a warning noise, the steering wheel is used to change direction, etc. Likewise, before

trying to use a computer, a person should have a general idea of what makes up a computer.

There are big computers, small computers, and those that are in between. They all have some things in common. All computers are machines. The type of computers that this book will help a person to use to solve problems is a digital computer. The term *digital* means that the machine manipulates digits or symbols to perform its assigned tasks.

All digital computers can perform five functions: (a) read, (b) write, (c) store data, (d) manipulate data (including arithmetic), and (e) control their own actions and make decisions based on previously supplied instructions. The instructions that tell the computer what to do are called *a program*. How these functions are related is indicated in the diagram shown in Figure 1.1. In the diagram, the direction of the arrows indicates the direction(s) in which data can be passed.

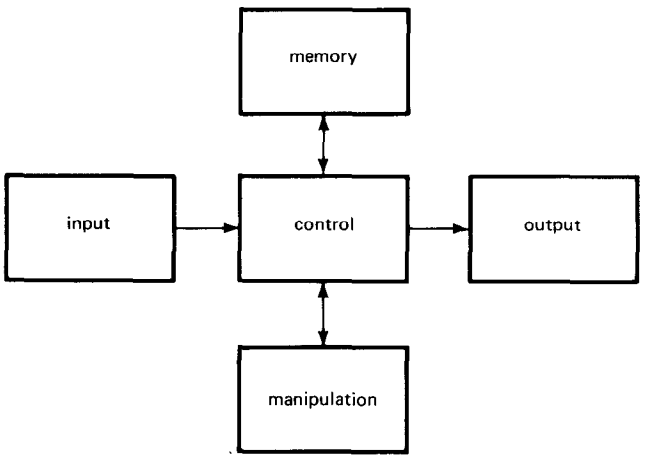


Figure 1.1 Basic parts of a computer.

Precisely how the machine accomplishes its functions is not the concern of this book. This is much the same as a person who knows that flipping a light switch causes the light to come on. How electricity is generated and delivered has little to do with knowing how to turn on a light. The person only need understand the general idea that electricity flows along wires and that a switch can control whether or not the electricity will make the light bulb glow.

The smallest computers perform only one task at a time. Larger computers often do several different things at the same time. For example, one program may be in the process of being read in, another may be manipulating data, and a third may be completed with the results in the output process. When several things are happening at the same time, what is happening needs to be controlled. Control is the job of a powerful program called an *operating system*. An operating system's job is to allocate resources (input devices, output devices, etc.) and to schedule tasks (when to read,

what program to run, etc.). The operating system program decides when a Pascal program can do its work.

The operating system program to control the computer handling your Pascal program has already been developed. The course instructor will provide information on how to use it so that Pascal programs can be processed by the computer.

## 1.2 COMPUTER SYSTEMS

### Objective

Provide a general introduction to the hardware and software components of a computer system.

### Suggested Background

Section 1.1 should be done before this section. This section is intended to provide a general summary of the topics presented in it. The reader is encouraged to seek supplemental information.

As was discussed in Section 1.1, a computer performs several different functions. Usually a single device does all of these functions. However, modern computers usually are made up of several separate pieces. The pieces are connected together by bundles of wires called cables. There may be many different combinations of pieces that can be used together. When several pieces are connected together, the result is usually called a *computer system*. Sometimes people call the collection of equipment that is used the *hardware* and the programs that are used the *software*.

The hardware that makes up a computer system can include a large variety of input and output devices. Most students will submit their data to a computer using either punched cards or a terminal and the results will be displayed either on a terminal or on a printer. Other input devices include those that read characters optically or magnetically. Computers can also accept data input from devices that read paper tape, magnetic tape, or magnetic disk.

Magnetic tape or disk is used as *auxiliary* storage. Programs and data can be stored temporarily on tape or disk by a computer program during the running of a program or for future use. Data and programs are also often put on tape, and sometimes disk, for transportation from one location to another.

As computer systems grow larger, more and more devices are collected together. Also, external communication capabilities are added to allow people who are not close to the computer to use the computer's facilities. Terminals sometimes communicate with a computer using telephone lines. Making all of the pieces work together is the job of the system software. In large computer systems, a considerable portion of what a computer does is to supervise and control its own activities. (The program called the operating system effects this control.)

## 1.3 USING A COMPUTER TO SOLVE PROBLEMS

**Objective**
Discuss how a program is connected to a clear statement of the problem.

**Suggested Background**
None.

A computer is an extremely powerful device. It can do just about any calculation or series of calculations that are necessary to solve a problem or task. But it can perform tasks only if it is told precisely and exactly what to do. The process of telling a computer what to do can be divided into planning what to do and then specifying the steps necessary to accomplish the plan. The machine must be instructed in its tasks in a way that the computer can use. If the computer was a person, we would say that it has to clearly and unambiguously "understand" what it is to do.

### 1.3.1 The Role of Algorithms

**Objective**
Develop the concept of what an algorithm is.

**Suggested Background**
None.

In order to understand what it is that is to be done, the solution to a problem first must be designed and stated in an algorithm. We all use algorithms. An algorithm is simply a plan of how to solve a problem. A more precise way of defining an algorithm is to say that it is a complete, unambiguous procedure for solving a specified procedure in a finite number of steps. An algorithm should be

(1) unambiguous
(2) precisely defined
(3) finite
(4) effective
(5) specified as a series of steps

### 1.3.2 The Role of Programs

**Objective**
Connect the use of a program with an algorithm.

**Suggested Background**
Section 1.3.1.

A program is the way that a problem solver tells a computer the steps that it is to follow to solve a problem. A program is a detailed and explicit set of instructions for accomplishing an algorithmically stated problem. The program has to be stated in a language that the computer can use.

The purpose of a program is to solve a problem. A program that does not work and solve its problem is worthless. Additional goals of a program may include finding the solution in the cheapest and fastest manner possible, but cost and speed are unimportant if the problem has not been solved properly. (It is often the case that the problem is misunderstood by the problem solver and that an algorithm and program are designed to solve the wrong problem. The use of structured walkthroughs, discussed in Chapter 15, helps clarify the understanding of what is to be accomplished.)

## 1.4 PROGRAMMING LANGUAGES

**Objectives**
Discuss the purpose of a programming language, how programming languages are described, and the variations in programming languages.

**Suggested Background**
Sections 1.3, 1.3.1, and 1.3.2

A programming language is used to communicate with a computer. As with any language, there are rules on how a language statement should be formed. The rules for constructing a computer language statement are stricter than those concerning human language statements. For example, people might be instructed to write their ages on a piece of paper in a variety of different valid ways: