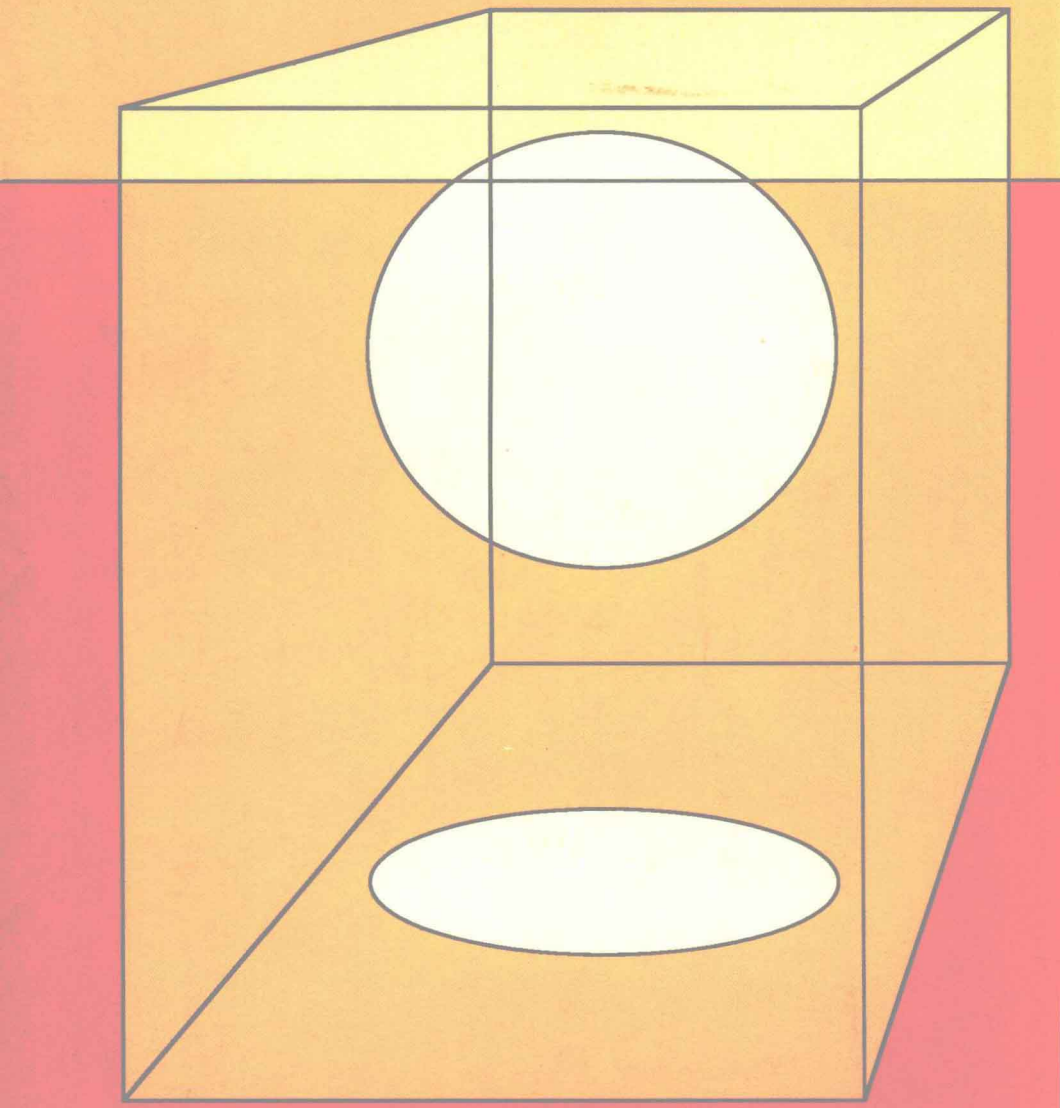
William L. Raiser

Introduction to Structured Pascal Programming



Introduction to Structured Pascal Programming

William L. Raiser
Graceland College

Gorsuch Scarisbrick, Publishers
Scottsdale, Arizona

Gorsuch Scarisbrick, Publishers
8233 Via Paseo Del Norte, Suite E-400
Scottsdale, AZ 85258

10 9 8 7 6 5 4 3 2 1

ISBN 0-89787-417-X

Printed in the United States of America.

Preface

This text is written for beginning Pascal students. No prior knowledge of Pascal, or any other programming language, is expected, and all of the fundamental features of the Pascal language are discussed. The text focuses on structured programming, which includes top-down design, modularization, stepwise refinement, and the use of the appropriate control structures—sequential, conditional execution, and iteration. Numerous programs begin with an elementary solution in one chapter and are developed and refined over the course of several subsequent chapters. This integrates the material in the text and allows the student to see successively more sophisticated program solutions to the same initial problem.

Throughout the book, *structure charts* are used to help communicate the ideas of program design and parameter passing. By graphically representing the major sub-divisions of the problem solution, the charts clarify what needs to be passed from one module, procedure, or function to another, and illustrate how the modules relate to each other and to the main program. Structure charts help the student understand and design the overall organization of the program; thus, the use of these charts also lays the foundation for their later use with more complex programming and system design courses.

As the use of structure charts indicates, a key part of the problem-solving process is conceptualizing the problem in terms of its major components. These components, or modules, become Pascal *procedures* or *functions*. This book uses procedures for every program, which assures that students will learn to think in terms of modularizing problems as they design a solution.

The focus of *Introduction to Structured Pascal Programming* is consistently on the process of problem-solving, rather than on a mere presentation of the technical features of Pascal. A computer is a problem-solving tool, and the programming language unlocks the power of that tool. As such, a language is a *means* and not an end in itself. The structured tools used in this book to develop and write Pascal programs can be applied more generally in many other contexts, enhancing the capabilities of our primary problem-solving tool, the mind.

The relatively informal style of this text puts the liberal arts student, using the material as an introduction to computing, at ease; at the same time, the focus on structured techniques provides the necessary foundation for the beginning computer science student. Hopefully all students will learn something about the learning process itself, and about the ways in which skills used in programming can be applied to other problem-solving situations in life.

Contents

Introduction to Computers and Computing

The history of computers and computing is a fascinating story of people and their ideas embodied in a machine that is in the process of transforming our lives. Like most historical stories, we do not really know when or where it began. We will begin with Blaise Pascal since it is the language named after him which you are about to study.

Pascal, a French mathematician/philosopher of the 17th Century, like most scholars of that age, dabbled in many areas and made significant contributions to several. His interest in mathematics and his desire to aid his father in the many computations necessary for his tax work resulted in Pascal's production of a calculating machine. This was a simple device capable only of addition or subtraction, but it was part of the new study that linked an interest in mathematics with machines to perform the computations.

Two centuries later, in England, Charles **Babbage** became the father of computing. Babbage was a mathematician and also a son of the industrial age who, when confronted with the task of repetitively solving a moderately complex algebraic function, naturally sought a mechanical way to solve the problem. He persuaded the British government to finance his attempt to build a *difference engine.* While he produced a workable design, he never built the engine because his mind was on to bigger and better things, an analytical engine that could solve any mathematical function when properly programmed. This machine was programmable, had stored memory, and performed the various mathematical operations. However, Babbage designed the machine to work with massive amounts of numbers manipulated in decimal form using gears and levers. While the machine worked in principle, it never worked in practice since the metal fabricating facilities of the time were unable to produce parts with enough precision.

Also during the 19th Century in England, George **Boole** conceived another piece in the puzzle. He developed an algebra based on the much simpler binary, rather than decimal, system. Now any quantity could be represented and logically manipulated in a system that required only two states—on, off; high, low; zero, one. Since the digits in the decimal system can represent any

of ten quantities, zero through nine, and the digits in the binary system can represent only two quantities, zero and one, this binary system greatly increased the quantity of parts necessary to perform a set of computations but greatly reduced the complexity of each part.

It remained for those in the 20th Century to translate the mechanical ideas of Babbage and the binary logic of Boole into an electronic machine. John **Atanasoff** was the first to do this, in the basement of one of the science buildings at Iowa State University in Ames during World War II. His accomplishment was so little appreciated at the time that the university felt only three or four such machines could be used in the entire United States. No patent was applied for, and no scholarly advancement or scientific acclaim accrued to Atanasoff. The computer era had begun, however.

During the 1950s several vacuum tube based computers were developed for university research and military and business applications. The next breakthrough came with the invention of the transistor, which replaced the large, hot vacuum tubes, thereby increasing the computing capacity of the machine while reducing its size and cost. This trend has continued to the present with the development of integrated circuits in the 1960s and with very large scale integration in the 1970s. We now have the mass-produced computer-on-a-chip which sells for a few dollars and is smaller than your little finger nail. The movement that began forty years ago is now a revolution.

From a machine that mechanically performed a few mathematical tax computations or filled in the values of a logarithmic table, we have progressed to a general purpose machine which novelist Arthur Clarke suggests, semi-seriously, may be the next step in the evolutionary development of life on earth. Humans are the necessary intermediate step, and bring together the integrated chips and assorted wires necessary to create the computers that are in a sense this "next step on the evolutionary ladder." Computers now design, program, and build other computers and can pass the stored data of one generation to another in a matter of minutes, as opposed to the years required for comparable human transmission. From any perspective, the computer is a very powerful tool.

The computer, as outlined in a classic paper by John **Von Neumann** in the mid 1940s, consists of a central processing unit **(CPU), memory,** and input/output **(I/O)** devices. The CPU consists of an operational control unit which translates program instructions into appropriate machine activities; the ALU, arithmetic logic unit, which performs the various mathematical and logical operations; and local memory where various values are held momentarily during program execution. Memory consists of RAM (random access memory), which the computer can access to store, retrieve, or modify data, and ROM (read only memory) containing various constants and commonly-used subroutines which the computer can retrieve but cannot store or modify. I/O devices are the point of contact between the computer and the external

world and come in a variety of forms. Input most typically comes through a keyboard or some external storage device such as a disk or tape drive. Output goes to a CRT, cathode ray tube or monitor screen, a printer, or some external storage device again. As the use of the computer expands, the number and type of I/O devices expands greatly. For example, input may come from some sensory device in the main engines of a rocket and output may go to the mechanical arm of a robot. All of these pieces constitute computer **hardware.**

The activities of the computer hardware are controlled by **software.** Software consists of programs that are fetched and executed by the CPU. The fact that a computer can run many different programs and that programs can be changed from time to time gives the flexibility necessary to make computers general-purpose devices. By modifying the software, computers perform a multiplicity of tasks without requiring a special piece of hardware for each task.

Software can be broken into two broad categories. The **operating system** is software that controls the resources of the computer, without which the computer cannot function. The operating system of a computer can be compared to the human brain, which controls the resources of the body and without which the body cannot function. The **application program** is software that uses the resources of the computer to perform useful work. We will develop various application programs throughout this text.

Software has undergone several key transformations thusfar. The first computers did not have operating systems. The human operators performed these functions. The initial programs were written as a stream of zeros and ones called **machine language.** Next came **assembly language,** which used cryptic acronyms, such as LDA for *load the accumulator,* and was a distinct improvement over simple zeros and ones. The goal has consistently been to make programming more and more like a standard English conversation with the computer. Operating systems developed, and languages moved to a third generation of **high-level languages** such as FORTRAN, COBOL, BASIC, Pascal, and many others. We are coming closer to the original goal of simplified human/computer interaction with fourth generation query languages used with modern database management systems, and much activity in this country and Japan is focusing on fifth generation computers that will make extensive use of artificial intelligence techniques.

Programming was initially an intuitive, make-do process. Programs were very peculiar to the person who wrote them, and they were usually written by one person. Such programs were difficult to modify to incorporate changes and alterations, and were extremely difficult for a person other than the original programmer to maintain. This made for very low programmer productivity and very slow application development. In the early 1960s, Edsger **Dijkstra** revolutionized the software industry by proposing principles of

structured programming, which we will study in this text. While many academicians and data processing managers recognized the power of these software development tools, programmers were slow to adopt them because of old habits and because the existing languages did not encourage their use. Newer languages such as Pascal, however, do incorporate language constructs that facilitate structured programming, which is one of the reasons that Pascal is so widely used as a language of instruction in colleges and universities across the country. It is a very powerful language, and it encourages the development of good programming habits that transfer to any subsequent language the programmer may use.

1
Model Program

This text is designed to teach you Pascal programming. Programming involves first *conceptualizing* a problem in such a way that it can be solved by computer and then analyzing the problem—that is, dividing the problem into appropriate pieces or modules. You thereby structure a solution to the original problem, which you then translate into appropriate code for the computer. The emphasis in this process is on problem solving and not on coding.

Structured programming involves top-down design, modularization, and use of appropriate control structures. Top-down design means that the programmer/analyst starts with a broad overview of the problem the program is to solve. For example, this approach would first look at a map or aerial photograph of your campus or community to gain an understanding of their physical layout. This broad perspective enables us to see major subdivisions of the problem, campus or community, that require focused attention, and these become major modules in the problem solution. For example, programs often require us to get some initial data from computer storage or from the user, to process that data, and to output, as a written report, the new data generated. These become three modules in the program solution, any one of which, or all of which, may require additional decomposition into submodules before reaching manageable size. As each module is coded for computer execution, appropriate control structures are used. The computer can execute one statement after another sequentially, can execute one of two or more possible sets of program code, and can execute sets of code repetitively. The program code determines which of these forms of control—sequential, conditional execution, or iteration—the computer will use at any point in the program, and we will study each of these in subsequent chapters in the text.

Such a problem-solving approach is central to structured programming. As you learn Pascal with this text, you will develop structured programming skills that can be transferred to any language which you may be called upon to learn and use. Niklaus Wirth designed Pascal to encourage the learning and application of these skills, so it is an ideal first language.

To start the process of learning to program and learning to program in Pascal, this chapter will examine the following:

- Key characteristics of both computers and people
- A model program
- Debugging and types of errors
- The general format of all Pascal programs

Computers, People, and Pascal

An understanding of the essential characteristics of both computers and people is critical for anyone attempting to interrelate the activities of the two. Computers are general purpose machines that are capable of many marvelous feats. They cannot, however, solve all the world's problems. One must have a general understanding of computer capabilities to make appropriate use of them. The following is a partial listing of those capabilities. Remember, though, that the computer is new and our understanding of its potentials is limited. We are exploring new frontiers, and there is much room for creativity, which is part of what makes computer science exciting.

Computer Characteristics

Computers are *machines*. They do not think, they are not alive, and they do not have personalities. They do not like or dislike you. And yet each computer is unique regarding interaction with its user and the ways it will respond to instructions. Computers are valuable tools, and they act as extensions of your mental capacities.

Computers are *dumb*. They do not have minds of their own, but are programmed to respond to a predetermined set of instructions in a predetermined manner. Because of this, you must follow set patterns when working with computers: A misplaced semicolon, comma or quote, or a misspelled key word will cause the machine to cease functioning properly because the predetermined patterns have been violated. This does not make working with computers difficult, but it may make the process aggravating at times. Correct application of the rules unlocks the power of this machine.

Computers are *fast* and *accurate*. The individual activities they perform are quite simple. They can, however, perform several thousand of these activities per second. Some computer actions are performed in a few nanoseconds, and there are as many nanoseconds in a second as there are seconds in thirty years. In the process of performing all of these activities, computers do not get bored and make errors as humans in all likelihood would. To err is human; to perform consistently with speed and accuracy is the function of

the computer. These two characteristics, speed and accuracy, are the real strengths of the computer.

Computers have *good memory.* Data is retained for long periods of time on disk and tape drives. These devices are quite reliable and, while their capacity increases everyday, their cost per unit of stored information is rapidly decreasing. Massive amounts of data are currently stored in this way and may be retrieved quickly and accurately.

Human Characteristics

To maximize the potential of the human/computer team, you must understand the human user as well as the computer. People are *smart.* People take a little bit of data and see creative implications. They generalize and see meaningful connections between seemingly unrelated data elements.

On the other hand, people are *slow* and *inaccurate.* The computer is performing activities in fractions of a second, and no matter how fast a person is, the computer is faster. People tire quickly, and as a result, make mistakes.

Contrary to what your school experience may have led you to believe, people have *poor memories.* We may be able to claim *intelligence,* which has to do with reasoning ability, the ability to analyze a problem and see its basic parts, and the ability to see patterns in apparently random pieces of data, but we tend to *remember* incompletely, inaccurately, and for short periods of time.

This survey of characteristics suggests that people formulate the relevant problems and provide the relevant data for the computer. Computers then calculate the relationships that exist between the data elements and logically project the implications of those relationships. People discover the general principles that bring meaning to apparent chaos and the computer helps determine the accuracy of those principles. The computer/user team is very powerful, and we are going to tap the power of the computer with the language Pascal.

Pascal

Pascal was developed by Niklaus Wirth in the late 1960s and early 1970s and was designed to embody structured programming principles in a computer langauge. Because of its strengths, Pascal was soon used widely in colleges and universities, in business, and in industry. Pascal's chief strength, for our purposes, lies in the good habits that it encourages. You will be a better programmer in whatever language or languages you ultimately use for having programmed in Pascal. This is particularly true if Pascal is your first language, since the first language you learn affects your approach to and use of all others.

Business learned long ago that the majority of their programming budget goes for program maintenance—fixing and upgrading. Structured programming techniques make both of these aspects of maintenance much easier, thereby increasing programmer productivity. In addition, structured programming techniques strengthen general problem-solving skills so necessary to higher-level analysis, system design, and management. Pascal also embodies many of the features found in programming languages having more specialized applications, which will give you a strong foundation in contemporary programming languages.

Exercises

1.1 What are the key characteristics of computers? Of people?

1.2 Briefly describe a computer application of which you are aware—for example, word processing—and tell how it does or does not capitalize on the strengths of both the computer and people.

1.3 Why learn Pascal if it is not the most popular business and scientific language?

Programming and Debugging

Pascal Programming

To learn Pascal programming, you must write Pascal programs. There is a big difference between understanding someone else's program and being able to write your own. Your success depends upon your active involvement. You learn from studying the work of others, but the primary focus must be on you and the programs you write. As you write, you will quickly acquire new skills that will act as powerful tools in programming and in problem solving.

Programming is an **iterative process;** that is, it requires you to make perhaps several drafts of a program before you arrive at a satisfactory solution. There is one fundamental principle of programming. Real programmers use pencils, with big erasers. There is nothing so intimidating as a blank sheet of paper, so get your initial ideas down quickly and then refine them. Work from what you know. Much of a program is relatively standardized, so get those parts down. Often one step will lead you to the next, which leads to the next. You do not have to know where you are going to end before you can start. Begin the process, and deal with obstacles as you encounter them.

A Model Program

Since there are considerable differences between computers, you will have to learn the specifics of your computer from your instructor or the manuals that come with that particular machine. The process of entering a program and running it will begin to familiarize you with the workings of both the computer and Pascal. The following program will illustrate many points about Pascal which we shall explore together.

Programs are ways of solving problems. Our problem is how to get the computer to write your name and course title on the screen. The following program will do that *for me.* If you enter this program into the computer exactly as written and then run it, my name and course will appear on the screen. Try it.

Program: Heading

```
program heading (input, output);
(*    print name/course     *)
(*written by: programmer    *)
(*            location      *)
(*            date          *)

procedure nameout;
(*print name/course*)
  begin(*nameout*)
    writeln('William L. Raiser');
    writeln('Computer Programming 1')
  end; (*nameout*)

begin(*main*)
  nameout
end.
```

Debugging

If you do not follow the rules for writing Pascal programs, the computer will quickly get lost and be unable to respond as it executes your program. In such a case, the computer will give you an error message.

This may happen, for instance, if you make a typographical error when entering the model program into the computer. If so, find the problem, correct it, and run the program again. Programs seldom run properly the first