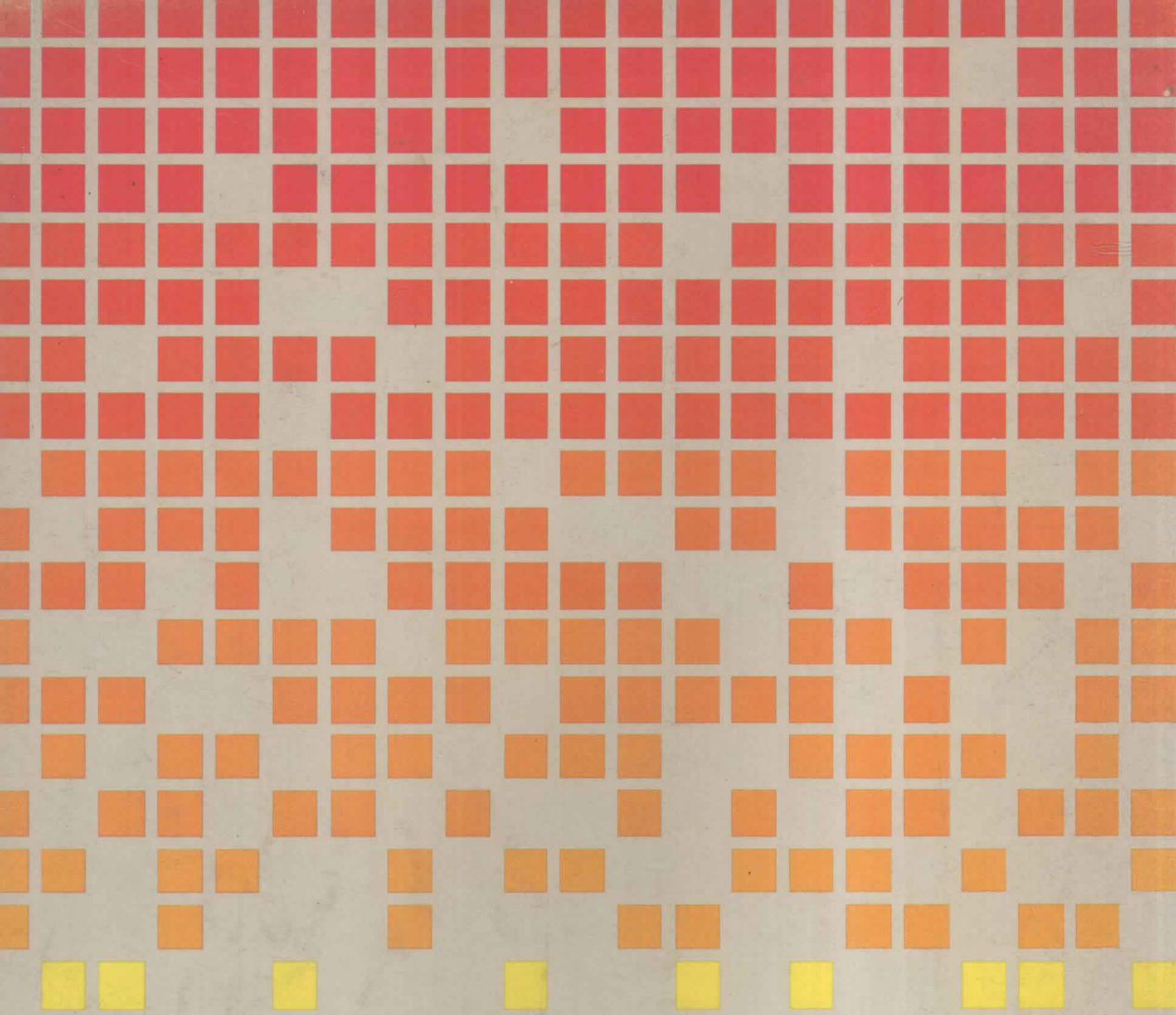


# SCIENTIFIC PASCAL

Harley Flanders

# SCIENTIFIC PASCAL

**HARLEY FLANDERS**
Florida Atlantic University

**Reston Publishing Company, Inc.**
**A Prentice-Hall Company**
**Reston, Virginia**

**Library of Congress Cataloging in Publication Data**

Flanders, Harley.
Scientific Pascal.

Includes bibliographical references and indexes.
1. Pascal (Computer program language) I. Title.
QA76.73.P2F55 1984 001.64'24 83-16065
ISBN 0-8359-6932-0
ISBN 0-8359-6931-2 (pbk.)


Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia 22090



Set by SCIENCE TYPOGRAPHERS in
Times Roman and
Helvetica Bold

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America.

# Scientific Pascal

# PREFACE

The first purpose of this book is to teach the programming language Pascal to those whose primary use of the computer is for scientific applications. The second purpose is to teach the skill of tackling a computing problem, developing an algorithm for solving it, and writing a program that does the job. The third purpose is to present a number of important programs in useable form.

Why Pascal? First, because it has built-in clarity. A well-written Pascal program can be read easily because of the way the language handles subroutines, links between subroutines, and repetitive processes. Second, the language is widely used. Virtually every large and middle sized (mini) computer has Pascal software, and more and more microcomputers include Pascal. What is more, Pascal is rapidly becoming the first language taught in introductory computer science/systems courses because the structured programming discipline of Pascal engenders better programming technique in general than does, say, BASIC or FORTRAN. A third reason is Pascal's broad flexibility for data types. Finally, the recursive nature of Pascal is especially attractive.

The algorithmic and recursive language ALGOL was developed in the 1950s and 60s by a group of computer scientists. Pascal, an offshoot of ALGOL, was invented by Niklaus Wirth in the late 60s. Compared to ALGOL, Pascal is simpler and clearer. It has much better data handling capability and built-in I/O procedures (input/output), which ALGOL lacked.

I believe one learns most efficiently by studying examples and by working exercises. Students learn calculus by working through hundreds of solved examples and by solving thousands of exercises. I think the same principle applies to learning

a programming language. A student should see solved examples on each topic and work on hundreds of exercises. The examples and exercises, culled from many sources, are probably the bulk of this otherwise modest text. Both the examples and the exercises are graded in an overall way. At first they are brief, single task fragments of programs. It is only in Chapters 3 and 4 that we get to some substantial programming problems. The polynomial program near the end of Chapter 4 is our first large multi-purpose program. Needless to say, organizing a large program presents a rather different type of problem than implementing a single short algorithm. Starred exercises are the most difficult ones. Pascal is particularly suited to the decomposition of a large task into manageable components.

The solutions section contains a solution for each exercise in the text. Each example of a program or procedure, whether in the text or the solution section, has been tested on a computer. To guarantee accuracy, we have reproduced computer-generated printout of all our programs directly from disk files (after running successfully) without the intervention of typists, copyeditors, or typesetters. Yet, there may still be some errors lurking for the following reasons, and the author will be grateful for having errors brought to his attention, large or small. First of all, many of the programs went through several revisions in the course of testing, and in some cases an earlier version (or even the wrong program) may have slipped in. Second, the Apple* versions of Pascal that I have used do not implement function and procedure parameters or the standard procedure Dispose. Therefore, the few programs using these features are not fully tested (although I am pretty sure they are correct). Finally, a few (harmless) deviations from my format may have slipped through, and in the longer programs, there may occasionally be a variable declared that is not actually used (also harmless). I hope there are no errors more serious than these latter two.

The publisher expects to make many of the programs in the book available on diskettes, first for the Apple® computer, later for the IBM PC computer and possibly also for other operating systems.

Let us look briefly at the contents. Chapters 1 and 2 cover most of Pascal. I have tried to get into the subject briskly, so the reader can solve interesting problems in Pascal almost from the start. The last sections of Chapter 1 emphasize how program flow is controlled. By the last sections of Chapter 2, some challenging algorithms are presented in the exercises. The examples and exercises on sets are a feature. Chapter 3 contains a detailed discussion of recursion. One unique feature is a section on translating recursive programs to (faster) iterative ones. Other features are the in-depth sections on program flow and the section on format and errors.

Chapter 4 covers records, pointers, and files, the more sophisticated parts of Pascal, and completes our study of the language. Features are the programs on matrices and on polynomials that exploit pointer types, and the exercises on files.

The final two chapters contain a wealth of applications; hopefully readers will find some of these programs really useful. In most cases, I have included careful

*Apple is a registered trademark of Apple Computer, Inc.

analyses and background material on the programs. To my knowledge, much of this material has never before appeared in Pascal.

Chapter 5 contains analytic applications: summation of series, transcendental functions, zeros of functions, derivatives, integrals, differential equations, and the fast Fourier transform. Chapter 6 contains algebraic applications: linear systems, scaling and pivoting, integer matrices, characteristic roots, high-precision integer arithmetic, permutations, and posets. I have tried to present state-of-the-art algorithms where possible. Features are adaptive integration and differential equations programs, a program for invariant factors, and the QR algorithm for characteristic roots of symmetric matrices.

I have tried to find challenging problems for the examples and exercises. Finding an algorithm that will solve a given problem in a reasonable time with available computer memory can be a difficult task. This is part of the solution. The other part is translating the algorithm into a clear Pascal (or other language) program. I hope this text proves useful for developing these problem-solving skills.

As you learn programming in Pascal (or any language) you frequently look up partly forgotten words and symbols. The material inside the covers should be helpful. In front is the vocabulary of Pascal, (reserved) words and symbols. In the back are the standard names (identifiers) the language provides and flowcharts for the control statements of Pascal. The appendices contain the syntax of Pascal, both in verbal and "railroad diagram" form. (But don't turn to the appendices every time you are in doubt about how a particular construction behaves. Instead, experiment; run many short test programs. Doing so is challenging, fun, and highly instructive.) The appendices also contain the ASCII control code and the standard compiler error messages.

I have set several standards for myself in this book. One is to include only programs that have actually been tested on a computer. A second is to print all programs in an uncluttered, clear format. The formatting program is included in Appendix C. It is of some interest in itself as it is the only large program in the book that is non-scientific and full of the kind of fussy detail that requires careful organization. I had hoped to prove the existence of a Pascal text without the seemingly mandatory program on sorting. However, at the last minute I relented and included the heapsort program, in Appendix F, used to alphabetize the index of this book. Finally, I have tried to use every term of the Pascal language in at least one program.

Most of the sources cited in the text refer to the References section at the end. However CACM refers to the collection, *Collected Algorithms of the ACM.*

It is my great pleasure to express my gratitude to the many people and institutions that have made this book possible. Florida Atlantic University supplied my computing needs and a great deal of support over the years. Rita Pelava typed some of the early programs into an Apple®. Dawn Schwartz typed the whole manuscript, using the Applewriter III® word processor, and most of the programs, using the Apple Pascal® text editor. John Sulzycki did the developmental editing, ably assisted by reviewers: Bill Ellis, David V. Moffat, Richard W. Nau, John Goda,

and Keith Doty, who contributed numerous valuable suggestions. The production staff at Reston were most cooperative, and I wish to express my particular thanks to the production editor, Diane Anderson. The compositor, Science Typographers Inc., did a fine job of composition, and the technical artist, Donald Price, produced excellent illustrations; I am grateful to both. Finally, the pager did a remarkable job of page make-up, better than I ever thought possible.

Harley Flanders
Florida Atlantic University
July, 1983

# Pascal Punctuation Marks and Symbols

## Separators

| | |
|---|---|
| ; | statement – statement |
| | end of declaration |
| | end of procedure or function body |
| , | list item – list item |
| : | variable – type |
| | function – type identifier |
| | case label list – statement |
| | label – statement |
| | expression – field length |
| | field length – no. of decimal places |
| .. | first element – last element |
| = | identifier – constant |
| | identifier – type |
| . | record – component |
| | end of program – next input |

## Brackets

| | |
|---|---|
| ( ) | usual arithmetic |
| | parameter brackets |
| | record variant brackets |
| [ ] | array bounds brackets |
| | set brackets |
| ' ' | character or string delimiters |
| { } | comment brackets |
| (* *) | |

## Other

| | |
|---|---|
| . | decimal point |
| E e | power of 10 |
| ↑ ^ | pointer type declarer |
| | pointee |
| | file element |

## Operators

| | |
|---|---|
| := | assignment |
| + | arithmetical unary |
| - | |
| * | arithmetical binary |
| / | |
| + | |
| - | |
| not | boolean |
| and | |
| or | |

| | |
|---|---|
| = | comparison |
| <> (≠) | |
| < | |
| <= (≤) | |
| > | |
| >= (≥) | |
| - (difference \) | operations on sets |
| * (intersection ∩) | |
| + (union ∪) | |

## Not used

" ? & @ % ! _ \ | ~

# Pascal Predefined Identifiers

## Constants

True
False
Maxint

## Types

Integer
Real
Boolean
Char
Text
String {UCSD Pascal}

## Variables

Input
Output

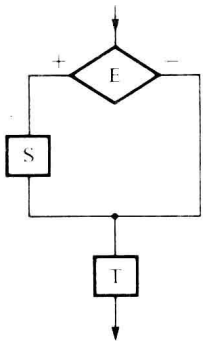
## Procedures

| | |
|---|---|
| Read | Readln |
| Write | Writeln |
| Rewrite | Reset |
| Get | Put |
| Page | |
| New | Dispose |
| Pack | Unpack |

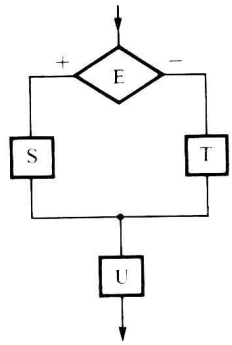
| Function | Domain | Range |
|---|---|---|
| Abs | Integer / Real | same |
| Sqr | Integer / Real | same |
| Sqrt | Real | Real |
| Exp | Real | Real |
| Ln | Real | Real |
| Sin | Real | Real |
| Cos | Real | Real |
| Arctan | Real | Real |

| Function | Domain | Range |
|---|---|---|
| Round | Real | Integer |
| Trunc | Real | Integer |
| Pred | any scalar type | same |
| Succ | any scalar type | same |
| Ord | any scalar type | Integer |
| Odd | Integer | Boolean |
| Chr | Integer | Char |
| Eof | any file type | Boolean |
| Eoln | Text | Boolean |

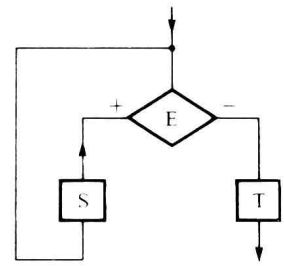
# PASCAL FLOW CONTROL



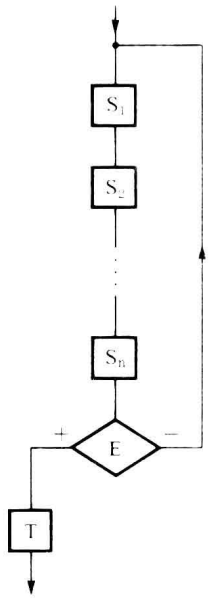

**if** E **then** S;



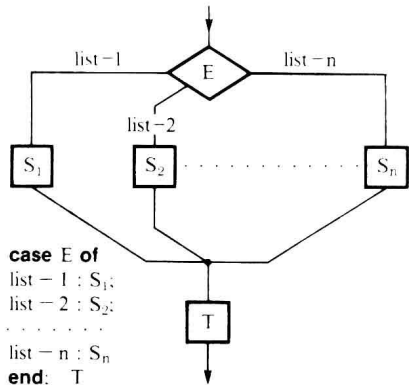

T **if** E **then** S **else** T; U



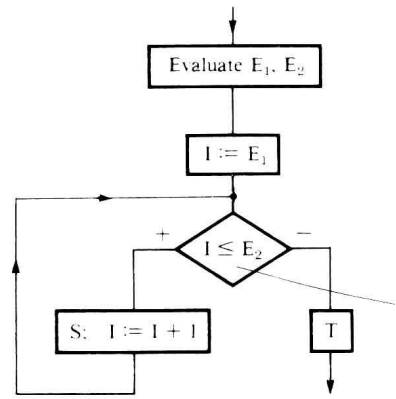

**while** E **do** S; T




**repeat** $S_1$; $S_2$; · · · ; $S_n$ **until** E; T




**case** E **of**
list − 1 : $S_1$;
list − 2 : $S_2$;
. . . . . .
list − n : $S_n$
**end**; T




**for** I := $E_1$ **to** $E_2$ **do** S; T

# Pascal Reserved Words

## Declaration Specifiers

**program**
**label**
**const**
**type**
**var**
**procedure** / **function** } **forward**

## Statement Brackets

**begin**... **end**

## Type Builders

**packed** { **array**[... ] **of**...
**record**... **end**
**file of**...
**set of**...

## Flow Control

**if**... **then**...
**if**... **then**...**else**...
**while**... **do**...
**with**... **do**...
**repeat**... **until**...
**for**... **to**... **do**...
**for**... **downto**... **do**...
**case**... **of**... **end**
**goto**...

## Integer Operators

**div**
**mod**

## Boolean Operators

**not**
**and**
**or**

## Set Membership

**in**

## The Null Pointer

**nil**

# CONTENTS

# CHAPTER 1

# AN OVERVIEW OF PASCAL

## 1.1 INTRODUCTION

Pascal is a block-structured programming language. In contrast, programming languages like BASIC, FORTRAN, and Assembler are line-structured. In Pascal, the line does not exist as a unit, so far as the language is concerned. Of course, we write our programs in lines for clarity, but the lines may all be run together; the program will still run in exactly the same way. Each "block" does a task, and the program structure governs the flow from task to task. By the end of Chapter 2, this will be quite clear; for the moment, the flowchart in Figure 1.1 indicates how we might think of a program that links several tasks together.

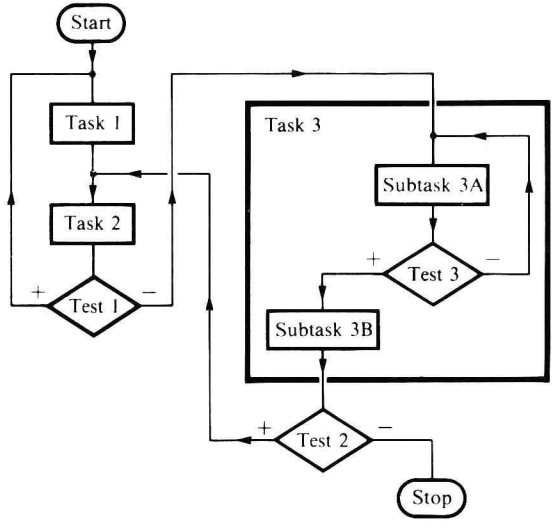



**FIGURE 1.1**
Sample flowchart

## Sample Pascal Program Fragments

The following examples are samples of what pieces of Pascal programs look like. Don't worry if they are not completely clear now; everything will be explained in time. Also, some are not complete programs, so you cannot do computer tests without fleshing them out somewhat. Finally, each fragment is probably not the neatest or most elegant solution of the proposed problem.

**EXAMPLE 1.1** The function $P(N) = N^5 - 5N$ will be evaluated for many integers $N$. Set up its definition in Pascal.

*Solution*

```
function  P(N: Integer): Integer;
   begin  P := (Sqr(Sqr(N)) - 5)*N  end;
```

The first line *declares* the function $P$ of an integer variable $N$ and says that the function is integer valued. The second line *assigns* to $P$ the result of the computation $((N^2)^2 - 5) * N$. (In computing, * is the multiplication symbol.)

**EXAMPLE 1.2** Find the least positive integer $N$ such that $N^5 - 5N > 30{,}000$.

*Solution*

```
program  FINDN;

   var  N: Integer;

   function  P(N: Integer): Integer;

      var  J: Integer;

      begin
      J := N*N;  J := J*J - 5;
      P := J*N
      end;     { end  P }

   begin     { begin  FINDN }
   N := 0;
   repeat  N := N + 1  until  P(N) > 30000;
   Write('N = ', N)
   end.
```

The program begins by declaring an integer variable $N$ and an integer-valued function $P(N)$, where $N$ is an integer variable. The function has its own (local) integer variable, which is used as an aid in assigning its value. The steps are