Problem Solving and Structured Programming in Pascal

Second Edition

Elliot B. Koffman

Problem Solving and Structured Programming in Pascal

SECOND EDITION

ELLIOT B. KOFFMAN
Temple University

ADDISON-WESLEY PUBLISHING COMPANY, INC.
Reading, Massachusetts • Menlo Park, California •
Don Mills, Ontario • Wokingham, England • Amsterdam •
Sydney • Singapore • Tokyo • Mexico City • Bogotá •
Santiago • San Juan

This book is in the Addison-Wesley Series in Computer Science

James T. DeWolf, Sponsoring Editor

Hugh Crawford, Manufacturing Supervisor
Robert Forget, Art Editor
Fran Palmer Fulton, Production Editor
Karen M. Guardino, Production Manager
Richard Hannus, Hannus Design Associates, Cover Design
Maureen Langer, Text Designer

Library of Congress Cataloging in Publication Data

Koffman, Elliot B.
Problem solving and structured programming in Pascal.

Includes index.
1. Pascal (Computer program language) 2. Structured programming. I. Title.
QA76.73.P2K63 1985 001.64'24 84-16811
ISBN 0-201-11736-3

Turbo Pascal™ is a trademark of Borland International, Inc.

Reprinted with corrections, September 1985

CDEFGHIJ-DO-898765

Reference Guide to Pascal Statements

Statement	*Example of Use*
program heading	`program GUIDE (INPUT, OUTPUT, INFILE, OUTFILE);`
comment	`{This section shows examples of Pascal statements.}` `(* Comments are ignored by Pascal. *)`

constant declaration
integer
character
string
real
	`const`
	`  STRINGSIZE = 20;`
	`  BLANK = ' ';`
	`  SCHOOL = 'TEMPLE UNIVERSITY';`
	`  DEANSLIST = 3.5;  PROBATION = 1.0;`

type declaration
enumerated
subrange
string
pointer
record
	`type`
	`  COLLEGE = (BUSINESS, ARTS, EDUCATION, GENERAL);`
	`  STUDENTRANGE = 1..100;`
	`  STRING = packed array [1..STRINGSIZE] of CHAR;`
	`  CLASSPOINTER = ^STUDENT;`
	`  STUDENT = record`
	`  NAME : STRING;`
	`  GPA : REAL;`
	`  INCOLLEGE : COLLEGE;`
	`  NEXTSTU : CLASSPOINTER`
	`  end;  {STUDENT}`

array
file
set
	`  MAJORARRAY = array [STUDENTRANGE] of COLLEGE;`
	`  STUFILE = file of STUDENT;`
	`  GRADESET = set of 'A'..'Z';`

variable declaration
record
set
text file
file
pointer
array
character
integer
	`var`
	`  CURSTU : STUDENT;`
	`  GRADES : GRADESET;`
	`  INFILE : TEXT;`
	`  OUTFILE : STUFILE;`
	`  CLASSLIST : CLASSPOINTER;`
	`  MAJOR : MAJORARRAY;`
	`  NEXTCH : CHAR;`
	`  I, COUNTPROBATION : INTEGER;`

declaring function
with BOOLEAN
result
	`function MEMBER (NEXTCH : CHAR;`
	`  TESTSET : GRADESET) : BOOLEAN;`
	`{Returns TRUE if NEXTCH is a member of TESTSET.}`

assignment (BOOLEAN)
set membership
	`begin {MEMBER}`
	`  MEMBER := NEXTCH in TESTSET  {Is NEXTCH in set?}`
	`end;  {MEMBER}`

(*continued on last page*)

Problem Solving and Structured Programming in Pascal

SECOND EDITION

To my family—Caryn, Richard, Deborah, and Robin Koffman, for their constant love and understanding.

To my parents—Edward and Leah Koffman, for all that they have given me.

Preface

There have been many changes in the way the first course in Computer Science is taught since the first edition of this book was published in 1981. During the past two years I have been the chairman of the ACM Task Force that has been studying these changes with an eye towards updating the description of the recommended first course for Computer Science majors (CS1).[1] Parallel with this effort, the Educational Testing Service (ETS) published a set of guidelines for an Advanced Placement course in Computer Science.[2] The text has been completely revised and reorganized to conform to both of these guidelines.

This text can be used in any introductory programming course that emphasizes a careful, disciplined approach to programming in Pascal. Since the Advanced Placement course is a full year course, this text covers more material than would normally be completed in one semester. The additional material on searching and sorting algorithms (Chapter 9) and dynamic data structures (Chapter 10) are optional advanced topics in CS1 and would normally be deferred until CS2.

As in the first edition, the primary goal of this text is to illustrate and teach problem solving through the stepwise development of algorithms. To facilitate this, procedures are introduced much earlier in this edition. There are also several large case studies throughout the text that integrate

[1] Koffman, E., Miller, P., and Wardle, C. Recommended Curriculum for CS1, 1984. Communications ACM 27, 10 (Oct., 1984), 998-1001.

[2] Advanced Placement Program of the College Board, Advanced Placement Course Description: Computer Science, Educational Testing Service, Princeton, NJ, 1983.

topics and illustrate their application in a substantial programming problem with multiple procedures. Many new examples and programming assignment projects are provided.

Some of the important features of this new edition are:

Early introduction of procedures: Procedures without parameters are introduced in Chapter 2 and are used for self-contained operations that require no global variable access (no side-effects). Procedure parameters are discussed in Chapter 3. Early coverage of procedures will enable students to practice the top-down approach from the beginning and to become more adept at program modularization.

Interactive programming: The emphasis is on modern technology and interactive programming. The majority of examples are written as interactive programs; however, students are shown early how to convert these programs to run in a batch environment. There are some batch-oriented examples as well.

New chapter on recursion: There is a new chapter on recursion that provides many examples of recursive procedures and functions. Additional algorithms for searching and sorting arrays are also provided in this chapter.

Arrays: Single and multidimensional arrays are covered in one chapter instead of in two as in the first edition. Similarly, all material on records is covered in a single chapter.

New expanded case studies: There are a number of new, larger case studies within the text. The solutions to the case studies are all carefully developed. System structure charts are used to represent the flow of control and data between modules in a program system.

Spiral approach: A spiral approach is used to preview topics such as the `if` statement, `for` statement, and input/output. Features are introduced as needed rather than overwhelming a student by providing all the details at once.

Pedagogical aids:

- Self-check Exercises are provided at the end of most sections. Solutions to selected exercises are provided at the end of the text.
- Each chapter ends with a Chapter Review section that includes a summary, a table of new Pascal statements, and review questions.
- *Boxed material*: Syntax display boxes are used to describe the syntax of each new Pascal statement as it is introduced, while Program Style boxes discuss the importance of good programming style.
- *Error warnings*: Each chapter ends with a discussion geared toward helping students prevent and correct common programming errors. Several sections discuss debugging techniques.
- *Program comments*: All programs are carefully commented. Loop invariants and assertions are shown for some loops. For easy identification, the first and last line of each procedure or program is in blue type.

New design: The page layout is larger providing more white space and

the overall tone is more user-friendly. The book has been completely redesigned with an eye towards making it easier for students to find figures, examples, programs, and special display boxes. A second color is used both to improve the appearance of the text and to clarify illustrations.

Pascal dialects: ANSI standard Pascal is covered in the text. Common extensions are described in appendixes on ISO standard Pascal, UCSD Pascal and Turbo Pascal.

Reference appendixes: There are also appendixes covering Pascal language elements, syntax diagrams, character codes, and error messages.

Complete instructor's manual: An Instructor's Manual provides a discussion of how to teach the concepts in each chapter. Sample test questions will be included as well as answers to all exercises, chapter review questions, and the Programming Projects found at the end of each chapter.

Transparency masters: A set of 131 transparency masters illustrating important concepts is available upon request.

Acknowledgments

Many people participated in the development of this text. The principal reviewers were most essential in finding errors and suggesting improvements. They include: William Eccles, University of South Carolina; Frank Friedman, Temple University; David Hannay, Union College; Helen Holzbaur, Temple University; Abraham Kandel, Florida State University; Raymond Kirsch, LaSalle College; David Moffat, North Carolina State University; Tom Nelson, North Carolina State University; Richard Rinewalt, University of Texas at Arlington; Paul Ross, Millersville University of Pennsylvania; Chris Speth, Western Kentucky University; Caroline Wardle, Boston University; Charles C. Weems, Jr., University of Massachusetts at Amherst; and Brad Wilson, Western Kentucky University. I am grateful to all of them for their considerable efforts.

Towards the beginning of this project, several faculty and members of the Addison-Wesley sales and marketing staffs participated in focus groups to discuss the first programming course in Pascal. These discussions were helpful in providing direction to the text and clarifying its organization. The faculty are: Linda Ottenstein, Michigan Tech University; David Neusse, University of Wisconsin at Eau Claire; Richard Rinewalt, University of Texas at Arlington; Ruth Barton, Michigan State University; and Howard Edelman, West Chester State University.

Finally, a number of faculty reviewed and commented on preliminary sections of the text. These faculty include: Gideon Frieder, University of Michigan; Gary Ford, University of Colorado; Abraham Kandel, Florida State University; Paul Hanna, Florida State University; M. Main, University of Colorado; Kent Wooldridge, California State University at Chico; Richard St. Andre, Central Michigan University; C. E. Wolf, Iowa State University; Angela Wu, American University; Yure Gurevich, University of

Michigan; Amir Foroudi, State University of New York at Fredonia; Morris Rang, II, Western Illinois University; Peggy R. Ayres, Linn-Benton Community College; Muhammed H. Chaudhary, Millersville University of Pennsylvania; Stanley Thomas, Wake Forest University; R. J. Del Zoppo, Jamestown Community College; David Rosenlof, Sacramento City College; George Beekman, Oregon State University; George Witter, Western Washington State University; J. M. Adams, New Mexico State University; John Lushbough, University of South Dakota; Dave Valentine, State University of New York at Potsdam; Dennis Martin, State University of New York at Brockport; Chris J. Dovolis, University of Minnesota; Barbara Smith-Thomas, University of North Carolina at Greensboro; Barent Johnson, University of Wisconsin at Oshkosh; Carl Wehner, University of Northern Iowa; Aboalfazl Salimi, Embry-Riddle University; Larry Wilson, Old Dominion University; Cary Laxer, Rose-Hulman Institute of Technology; J. Mailen Kootsey, Duke University; Jerry Waxman, City University of New York at Queens; Bruce J. Klein, Grand Valley State College; Eris Pas, Duke University; Gove Effinger, Bates College; Krishna Moorthy, Framingham State College; Brian Johnson, University of New Hampshire; and John Goda, Georgia Institute of Technology.

There were also many people involved with the actual production of the text. From Addison-Wesley, James DeWolf was the sponsoring editor and recruited reviewers, provided input and suggestions during the writing stage, and coordinated with the production staff. Bill Gruener also was the publisher with overall responsibility for the text. Karen Guardino was the production manager and saw to it that the book was able to meet a very tight production schedule. Maureen Langer refined the design of the text. In Philadelphia, Fran Palmer Fulton served as the Production Editor and coordinated and supervised the typesetting of the manuscript. I am grateful to all of them for their involvement and extra efforts to get this book published on schedule.

Philadelphia, PA
December 1984

E.B.K.

Appendixes

Appendix A
Reserved Words, Standard Identifiers, Operators, Functions, and Procedures

Reserved words

```
and         end         nil         set
array       file        not         then
begin       for         of          to
case        function    or          type
const       goto        packed      until
div         if          procedure   var
do          in          program     while
downto      label       record      with
else        mod         repeat
```

Standard identifiers

Constants:

```
FALSE, TRUE, MAXINT
```

Types:

```
INTEGER, BOOLEAN, REAL, CHAR, TEXT
```

Program parameters:

```
INPUT, OUTPUT
```

Functions:

```
ABS, ARCTAN, CHR, COS, EOF, EOLN, EXP, LN, ODD, ORD,
PRED, ROUND, SIN, SQR, SQRT, SUCC, TRUNC
```

Procedures:

```
GET, NEW, PACK, PAGE, PUT, READ, READLN, RESET,
REWRITE, UNPACK, WRITE, WRITELN
```

Table A.1 Table of Operators

Operator	*Operation*	*Type of Operand(s)*	*Result type*
:=	assignment	any type except file types	—
arithmetic:			
+ (unary)	identity	integer or real	same as operand
- (unary)	sign inversion		
+	addition	integer or real	integer or real
-	subtraction		
*	multiplication		
div	integer division	integer	integer
/	real division	integer or real	real
mod	modulus	integer	integer
relational:			
=	equality	scalar, string, set, or pointer	
<>	inequality		
<	less than	scalar or string	BOOLEAN
>	greater than		
<=	less than or equal	scalar or string	
	-or-		
	subset	set	
>=	greater than or equal	scalar or string	
	-or-		
	superset	set	
in	set membership	first operand is any scalar, the second is its set type	
logical:			
not	negation		
or	disjunction	BOOLEAN	BOOLEAN
and	conjunction		
set:			
+	union		
-	set difference	any set type T	T
*	intersection		

Table A.2 Standard Functions

Name	*Description of Computation*	*Argument*	*Result*
ABS	The absolute value of the argument	real/integer	same as argument
EXP	The value of e (2.71828) raised to the power of the argument	real/integer	real
LN	The logarithm (to the base e) of the argument	real/integer	real
SQR	The square of the argument	real/integer	same as argument
SQRT	The positive square root of the argument	real/integer (positive)	real
ROUND	The closest integer value to the argument	real	integer
TRUNC	The integral part of the argument	real	integer
ARCTAN	The arc tangent of the argument	real/integer (radians)	real
COS	The cosine of the argument	real/integer (radians)	real
SIN	The sine of the argument	real/integer (radians)	real
CHR	Returns the character whose ordinal number is its argument	integer	CHAR
ODD	Returns TRUE if its argument is an odd number; otherwise returns FALSE	integer	BOOLEAN
ORD	Returns the ordinal number of its argument	ordinal	integer
PRED	Returns the predecessor of its argument	ordinal	ordinal
SUCC	Returns the successor of its argument	ordinal	ordinal

Table A.3 Table of Standard Procedures

Procedure Call	*Description*
`DISPOSE (P)`	Returns the record pointed to by pointer variable `P` to free storage.
`GET (F)`	Advances the file position pointer for file `F` to its next component and assigns the value of the component to `F^`.
`NEW (P)`	Creates a record of the type pointed to by pointer `P` and saves its address in `P`.
`PACK (U, I, P)`	Copies the elements in unpacked array `U`, starting with `U[I]`, to packed array `P`, starting with the first element.
`PAGE (F)`	Advances the printer to a new page before printing the next line of file `F`.
`PUT (F)`	Appends the current contents of `F^` to file `F`.
`READ (F,` *variables*`)`	Reads data from file `F` to satisfy the list of *variables*. Only one component of file `F` can be read unless `F` is a text file. If `F` is not specified, file `INPUT` is read.
`READLN (F,` *variables*`)`	Reads data from text file `F` to satisfy the list of *variables*. Skips any characters at the end of the last line read.
`RESET (F)`	Resets the file position pointer for file `F` to the beginning. File `F` may then be read.
`REWRITE (F)`	Resets the file position pointer for file `F` to the beginning; any prior contents are lost. File `F` may then be written.
`UNPACK (P, U, I)`	Copies the elements in packed array `P`, starting with the first element, to unpacked array `U`, starting with `U[I]`.
`WRITE (F,` *outputs*`)`	Writes the data in the order specified by *outputs* to file `F`. Only one output item can be written unless `F` is a text file. If `F` is not specified, the data are written to file `OUTPUT`.
`WRITELN (F,` *outputs*`)`	Writes the data in the order specified by *outputs* to text file `F`. Writes an end-of-line mark after the data.

Appendix B
Additions and Extensions to Pascal

B.1 Additional Features of ANSI/IEEE Pascal

This appendix describes additional features of the ANSI/IEEE Pascal standard not covered in the text.

Forward Declarations

Given the procedure declarations below, procedure B can call A, but procedure A cannot call B. This is because the declaration for procedure B is not processed until after procedure A is translated.

```
procedure A (var X : REAL);
.........
end; {A}

procedure B (var Y : REAL);
.........
end; {B}
```

If a *forward declaration* for procedure B is inserted before procedure A, then A can also call B.

```
procedure B (var Y : REAL); FORWARD;

procedure A (var X : REAL);
.........
end; {A}

procedure B;
.........
end; {B}
```

As shown above, the forward declaration for procedure B consisting of only the procedure heading comes first, followed by the declaration of procedure A, and finally the declaration for procedure B. The parameter list for procedure B appears only in the forward declaration. Now procedure A can call B, and procedure B can call A; so A and B are called *mutually recursive.*

Functions and Procedures as Parameters

A procedure or function may be passed as a parameter to another procedure or function. As an example, we may wish to compute the sum below for the integers 1 through N where f represents a function that is applied to each integer.

f(1) + f(2) + f(3) + ... + f(N)

If f is the function SQR (square), then we wish to compute the sum

a) $1 + 2^2 + 3^2 + \ldots + N^2$

If f is the function SQRT (square root), then we wish to compute the sum

b) $\sqrt{1} + \sqrt{2} + \sqrt{3} + \ldots + \sqrt{N}$

In function SUMINT below, the function F is declared as a *function parameter.* The function designator

```
SUMINT (SQR, 10)
```

computes sum a) above for N = 10; the function designator

```
SUMINT (SQRT, 10)
```

computes sum b) above for N = 10.

```
function SUMINT (function F(X : INTEGER) : REAL;
                 N : INTEGER) : REAL;

{Computes F(1) + F(2) + . . . + F(N).}

var
   SUM : REAL;                    {the partial sum}
   I : INTEGER;                   {loop control variable}

begin {SUMINT}
   SUM := 0.0;                    {initialize SUM}

   for I := 1 to N do
      SUM := SUM + F(I);

   SUMINT := SUM                  {define result}
end;  {SUMINT}
```

The parameter of function F is represented by X in the heading for function SUMINT; any identifier may be used. F can also represent a user-defined function with one type INTEGER parameter.

GOTO Statements and Labels

The GOTO statement is used to transfer control from one program statement to another. The label (a positive integer) is used to indicate the statement to which control is transferred. Labels must be declared in label declaration statements at the beginning of a block. In function SAMEARRAY below, the GOTO statement is used to exit a for loop before the specified number of repetitions (N) are performed.