QUICK PASCAL QUIC

QUICK PASCAL QUICK PA

QUICK PASCAL QUICK PASCAL

QUICK PASCAL

UICK PASCAL QUICK PASCAL

PASCAL QUICK PASCAL

CAL QUICK PASCAL

QUICK PASCAL

DAVID L. MATUSZEK

QUICK PASCAL

DAVID L. MATUSZEK
The University of Tennessee

JOHN WILEY & SONS
New York • Chichester • Brisbane • Toronto • Singapore

Copyright © 1982, by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of
this work beyond that permitted by Sections
107 and 108 of the 1976 United States Copyright
Act without the permission of the copyright
owner is unlawful. Requests for permission
or further information should be addressed to
the Permissions Department, John Wiley & Sons.

Library of Congress Cataloging in Publication Data:

Matuszek, David L.
Quick Pascal.

Includes index.
1. PASCAL (Computer program language)—Study and teaching. I. Title.
QA76.73.P2M35 1982 001.64'24 82-8354
ISBN 0-471-86644-X AACR2

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

QUICK PASCAL

To My Father,
Chester Matuszek

试读结束：需要全本请在线购买： www.ertongbook.com

PREFACE

The purpose of this book is to teach Pascal to programmers who need to know the language yesterday. It is not suitable for teaching Pascal to non-programmers. My intention is to make it practical to use Pascal in computer science courses even though students have not previously been taught this language.

The first chapter is an overview of Pascal and provides enough information for the student to be able to read most Pascal programs written by other people. The second chapter covers assorted details that the student will need to know in order to begin writing his or her own programs. Taken together, Chapters 1 and 2 provide a complete short course in Pascal. The remaining chapters are organized by topic, so that the student attempting to use a particular feature of Pascal (for example, variant records) will find all the necessary information in one place. This necessitates some redundancy; for example, all the information presented in Chapter 1 is repeated in greater detail at the appropriate places in later chapters. To further increase the value of the book as a reference, considerable summary information is provided in the appendices, and there is an extensive index.

The emphasis throughout the book is on teaching Pascal as it really is, not as it ought to be. Most Pascal books describe only an idealized "Standard Pascal," or else Pascal for a particular machine. This book describes not only Standard Pascal, but also four real implementations. Whatever dialect of Pascal you use, the chances are that it will be enough like one of these described here to enable you to find what you need. Throughout the book you will find hints for exploring the particular Pascal you are working with.

The language presented here is full Pascal; nothing from Standard Pascal has been left out, and many variations from Standard Pascal have been included, because Standard Pascal is certainly not what you have to work with on your machine.

Although Pascal is a useful language, it is not perfect. The quirks and

problems you are likely to encounter are discussed in detail to help you get over the rough spots. The four representative implementations are

PASLC The upper/lowercase compiler on the DEC-10, developed at the University of Hamburg, Germany.

Pascal 8000 The compiler for the IBM 360/370, developed at the University of Tokyo, Japan, by Teruo Hikita and Kiyoshi Ishihata for the Hitac 8800/8700 computer, and adapted for the IBM system by Gordon Cox and Jeffrey Tobias at the Australian Atomic Energy Commission, Australia.

CDC Pascal The Pascal 6000-3.4 compiler for the CDC 6000 computers, as described by Kathleen Jensen and Niklaus Wirth.

UCSD Pascal The Pascal compiler developed for use on microcomputers at the Institute for Information Science at the University of California at San Diego, under the direction of Kenneth L. Bowles (Apple version).

In this text we will refer to particular Pascal compilers by the above abbreviations. However, it should be understood that the companies that make these computers are not necessarily responsible for the Pascal compilers on those machines.

I thank Charles Hughes, Eugene Getchell III, and Hal Harrison for their careful reading of the manuscript. They caught many errors and made many helpful suggestions. Any errors that survive are, of course, entirely my own responsibility.

DAVID L. MATUSZEK

CONTENTS

PART ONE

INTRODUCTION

试读结束：需要全本请在线购买： www.ertongbook.com

Chapter 1

AN OVERVIEW OF PASCAL

Pascal has become popular because it is a simple, easily understood language. In exchange for simplicity, it is less flexible than big languages like PL/I.

1.1 FIRST EXAMPLE: ADDING TWO NUMBERS

The fastest way to get familiar with the language is by looking at examples.

```
1  PROGRAM ADD (INPUT, OUTPUT);
2  (* EXAMPLE 1: Program to add two numbers. *)
3  VAR
4        I, J, SUM: INTEGER;
5  BEGIN
6  READ(I, J);
7  SUM := I + J;
8  WRITELN(SUM)
9  END.
```

The line numbers are not part of Pascal. They are there to help us discuss the example.

The first line names the program "ADD" and specifies the files (INPUT and OUTPUT) that it will use. INPUT is Pascal's name for the standard input file (regardless of what your operating system calls it) and OUTPUT is the name for the standard output file. INPUT and OUTPUT are examples of

textfiles, that is, files which contain information in human-readable format (text). Pascal also supports the use of binary files.

Line 2 is a comment. Comments appear between the symbols (* and *). In some implementations and many textbooks you will also see comments enclosed between curly braces { and }. Comments may be put anywhere, except in the middle of a word or a number, or between the : and the = of the := symbol.

In Pascal all declarations are put first, before any executable statements. In this program the declaration part is lines 3 and 4, and the executable part is lines 5 through 9.

All variables used in a program must be declared in a VAR part. Line 3 starts the VAR part; in line 4 the identifiers I, J, and SUM are declared to be integer variables. There are two types of numeric quantities in Pascal: INTEGERs (whole numbers) and REALs (which contain a decimal point). There is no default type—any variable you use, you must declare.

Some words in Pascal are reserved, that is, they cannot be used as variable names. The words VAR, BEGIN, and END in the above example are reserved words.

The executable body always starts with the keyword BEGIN (line 5) and ends with the word END followed by a period (line 9).

READ (line 6) is a call to a built-in procedure to read values from an input file and assign them to variables (in this case, I and J). WRITELN (line 8) is a built-in procedure to print out values.

The assignment statement (line 7) uses := rather than =; it says to set SUM to the sum of I and J. The symbol = means equality, not assignment, and is used in other places.

Semicolons (;) are used to separate one statement from the next, and one declaration from the next. In addition, there is a semicolon between the last declaration and the first BEGIN. This rule is conceptually simple, but difficult to apply; Section 2.2 gives some guidelines for placing semicolons.

PL/I programmers should note that semicolons are used differently in Pascal than in PL/I.

The example program is shown in all capital letters. On computer systems having lowercase letters as well, you can usually use either case, or mix them however you choose, and Pascal ignores the difference (except in quoted strings). Lowercase is easier for humans to read, and is preferable. This text uses all uppercase because so few machines provide lowercase.

Pascal is a free-format language. You do not need to start certain things in certain columns, or put exactly one statement per line. In fact, the Pascal compiler "sees" your program as one lone line, with the line boundaries and comments replaced by spaces. You could rewrite the above example as

```
PROGRAM ADD(INPUT,OUTPUT);
 (* EXAMPLE 1: Program to add two numbers. *)
```

```
VAR I,J,SUM:INTEGER;BEGIN READ(I,J);
SUM :=I+J;WRITELN(SUM)END.
```

and the compiler would be just as happy (but your boss wouldn't).

Pascal is free-format in order to allow you to space and indent in such a way as to make your programs more readable. Increased readability will make your programs easier to debug and to maintain.

There are several different indentation schemes in vogue, and one can argue endlessly about the best way to indent. The scheme the author prefers is used throughout this text without further comment; but whatever indentation scheme is used should be followed consistently. Perhaps the least helpful indentation scheme is to start every line in the same column, as is often done in Fortran.

1.2 CONVENTIONAL DECLARATIONS IN PASCAL

This section describes those declarations in Pascal which should be familiar because they are similar to declarations in most other languages. All variables used in the program must be declared in the VAR section. Declarations have the following syntax,

list_of_variables : *type*

where the *type* may be any of the following: INTEGER, REAL, CHAR (single character), BOOLEAN, or one of the other types in Pascal such as arrays. Variables of type CHAR may have as value any single character. Single character constants are written enclosed in single quote marks, for example, 'A'. BOOLEAN variables (called LOGICAL variables in Fortran) can take on either of the values TRUE and FALSE.

```
VAR
    I, J: INTEGER;                          (* two integer variables *)
    X, Y, Z: REAL;                          (* three real variables *)
    K: INTEGER;                             (* another integer variable *)
    CH: CHAR;                               (* a character variable *)
    P, Q, R: BOOLEAN;                       (* three boolean variables *)
    VEC: ARRAY [1..10] OF INTEGER;          (* see below *)
    BOX: ARRAY [1..10, -5..5] OF REAL;
```

VEC is declared to be a one-dimensional array of ten integers; those integers can be referenced by VEC[1], VEC[2], ..., VEC[10]. The notation 1..10 means that the allowable subscripts may range from one to ten, inclusive.

BOX is declared to be a two-dimensional array of real numbers. The first subscript can take on any value from one to ten inclusive, while the second subscript can take on any value from -5 to 5 inclusive. The elements

are referred to in the program by BOX[1, −5], ..., BOX[10, 5]. Arrays may have any number of dimensions.

In Standard Pascal, there is no way to initialize variables. In Pascal 8000, however, variables may be initialized in the VALUE section, which must come after the VAR section. Every variable occurring in the VALUE section must previously have been declared in the VAR section. The syntax of an initialization is

variable := *value*

(Note the use of := rather than :.) For example,

```
VALUE
    X := 2.7929;
    CH := '*';
```

1.3 CONVENTIONAL STATEMENTS IN PASCAL

We have already seen examples of the assignment and the procedure call statements. (READ and WRITELN are procedure calls.) Here are some other Pascal statements.

IF *condition* THEN *statement*

The *condition* is a Boolean expression, that is, an expression which results in a value of TRUE or FALSE. If the value of the *condition* is TRUE, then the *statement* is executed; otherwise it is not.

IF *condition* THEN *statement_1* ELSE *statement_2*

If the value of the Boolean *condition* is TRUE, *statement_1* is executed; if the *condition* is FALSE, *statement_2* is executed.

WHILE *condition* DO *statement*

This is a loop with the test at the top. The *condition* is evaluated; if TRUE, the *statement* is executed, and the program loops back to test the *condition* again. If FALSE, the loop exits, which means that control passes to the next statement. Note that if the *condition* is false initially, the loop exits immediately without ever having executed the *statement*.

REPEAT *sequence_of_statements* UNTIL *condition*

This is a loop with the test at the bottom. The *sequence_of_statements* is executed first, then the *condition* is tested. If FALSE, the program loops back to execute the *sequence_of_statements* again; if TRUE, the loop exits.

Note three differences between the WHILE and REPEAT loops.

1 The body of the WHILE loop consists of a single statement, while the

body of the REPEAT loop consists of a sequence of statements. You will soon see that this difference is not important.

2 The body of a WHILE loop may be executed zero times (that is, not at all), but the body of a REPEAT loop is always executed at least once.

3 The condition of a WHILE loop is false after the loop exists; the condition of a REPEAT loop is true after the loop exits.

There is reason to believe that REPEAT loops are harder to use and that people make more errors with them than with WHILE loops. Hence, all things being equal, you should prefer WHILE loops to REPEAT loops.

FOR *variable* := *initial_value* TO *final_value* DO *statement*

This is a loop under count control. The *variable* is a simple unsubscripted variable; the *initial_value* and *final_value* can be any expressions resulting in a value of the same type as the *variable*. The loop executes the *statement* once for each of the values *initial_value* through *final_value*, with a step size of one. If the *initial_value* is greater than the *final_value*, the *statement* is never executed.

FOR *variable* := *initial_value* DOWNTO *final_value* DO *statement*

This statement is like the preceding, except that the step size is minus one, rather than one. If the *initial_value* is less than the *final_value*, the *statement* is never executed.

Note that in all of the preceding statement types, the word *statement* refers to one single statement. Often it is desirable to use a sequence of statements, rather than just a single statement. The "fat parentheses" BEGIN and END make this possible.

BEGIN *sequence_of_statements* END

BEGIN and END enclosing a *sequence_of_statements* form a single, compound statement. (Note that neither keyword by itself is a complete statement.)

Input/output in Pascal is trivially simple. Procedure READ takes any number of variable names as parameters, and reads that many values from the input; it can read integers, real numbers, or characters. Numeric values need not go in any particular columns, but must be separated by at least one space. READ treats a line boundary as if it were a space.

Procedure WRITELN takes any number of expressions as parameters, and writes out the value of those expressions on a single line. (Procedures WRITE and READLN also exist, but are less frequently used; see Section 8.3.)

试读结束：需要全本请在线购买： www.ertongbook.com