Vardell Lines

# MATH AS A SECOND LANGUAGE

# PASCAL AS A SECOND LANGUAGE

**VARDELL LINES**
*National Applied Computer Technologies*
*Orem, Utah*

*PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632*

*Library of Congress Cataloging in Publication Data*

Lines, M. Vardell.
Pascal as a second language.

Includes index.
1. PASCAL (Computer program language) I. Title.
QA76.73.P2L55 1984 001.64'24 83-9493
ISBN 0-13-652925-9

Editorial/production supervision
and interior design: *Lynn S. Frankel*
Cover design: *Photo Plus Art* (Celine A. Brandes)
Manufacturing buyer: *Gordon Osbourne*



Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-652925-9

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

# PASCAL AS A SECOND LANGUAGE

# PREFACE

In 1965, Niklaus Wirth proposed a language that would encourage and facilitate systematic software development. This language, which Wirth named Pascal in honor of Blaise Pascal, a seventeenth-century mathematician and philosopher, was officially announced in 1971. Some revisions were made to the language in 1972 and a report and user's manual was published n 1973. Since then its use and popularity have spread, and it is now available on most computers from micros to largescale.

A standard definition of the Pascal language was adopted in early 1981 by the International Standards Organization (ISO). The ISO standard is very close to the Wirth definition described in his report. This book is a comprehensive coverage of ISO standard Pascal with the exception of conformant arrays. It is intended for readers who have some programming experience.

Examples are a valuable learning tool, so I have included many example programs in the book. Some of them, especially in the early chapters, are trivial and are of no benefit beyond an aid to learning. Some of the programs in the later chapters are intended to serve the additional purpose of presenting useful algorithms. In these example programs, user-defined identifiers are lowercase and the Pascal reserved words are uppercase.

Most of my Pascal experience has been on microcomputers in which the primary input device is a keyboard and the primary output device is a CRT screen. Many of the example programs reflect the interactive mode of operation implied by such a system. Some of the programs take advantage of the screen output by formatting and highlighting the text.

Basic concepts are covered in Chapter 1. These concepts include the Pascal vocabulary, structure of Pascal programs, Pascal operators, and data types. Statements in Pascal may be simple or structured. All of the statements are described in Chapter 2. Pascal in-

cludes two kinds of subprograms—procedures and functions, as discussed in Chapter 3. Chapters 4 through 7 cover the four types of structured data. Arrays are described in Chapter 4, sets in Chapter 5, records in Chapter 6, and files in Chapter 7. The third category of Pascal data types, pointers, is described in Chapter 8.

The appendixes include syntax diagrams for ISO standard Pascal and a brief description of several methods used to associate files in a Pascal program to external files.

VARDELL LINES

# PASCAL AS A SECOND LANGUAGE

# CONTENTS

# 1

# BASIC CONCEPTS

Programming languages have several attributes that fall into two broad classes: ease of use and efficiency. The ease of use attributes include language fluency, program legibility, and program maintainability. Language fluency refers to the level of difficulty of both learning the language and writing programs in it. Program legibility refers to the level of knowledge required in order read and understand programs written in it. Legibility is a function of the particular program as well as the language. The ease with which someone can correct and enhance a program is program maintainability. This attribute depends on legibility, in particular on the control structures available in the language.

Efficiency attributes include processing time, program storage, and control of hardware resources. High-level languages such as Pascal are generally less efficient than assembly languages. The efficiency of a high-level language depends on the language features and on how the language is implemented.

A language that has too many features may become difficult to understand for the compiler as well as human readers. In such cases, the compiler will generate less efficient code unless it is an optimizing compiler. Optimizing compilers require more memory, thus limiting their use on small computers, and take longer to compile. The net result is less efficient programs. On the other hand, a language that has too few features may result in less efficient programs because extra program statements will be required to make up for the lack of features.

The efficiency of a high-level language depends on its implementation. If it is implemented with a compiler that generates machine or native code, it will be more efficient than if it is implemented with an interpreter. Since any high-level language can be implemented either way, the effect on efficiency is not language dependent.

Most high-level languages cannot directly control hardware resources. However, some high-level languages include intrinsic routines written in assembly language that can be called to control hardware resources and some allow embedded assembly language statements for the same purpose. These features are extensions to the standard language and may be added to any high-level language. For example, there are versions of BASIC, FORTRAN, and Pascal that provide these extended features.

All programming languages have a set of rules that describe how programs may be written using the language. These programming rules include a syntax part and a semantics part. Syntax rules define how the vocabulary of the language may be combined to form statements. Semantic rules assign meaning to the statements.

The syntax and semantics of Pascal mean what you would expect them to mean, with the result that programs written in Pascal are quite easy to read and understand. This means that compilers can generate efficient code as far as memory required and execution speed is concerned. It also means that the programs are legible and maintainable and that the language itself is quite easy to learn and use.

Computer programs consist of two essential parts: data and procedures that process the data. Most programming languages stress the processing part, but Pascal is as powerful in its ability to describe data as it is in its ability to process data.

## PASCAL VOCABULARY

The Pascal vocabulary consists of letters, digits, and special symbols. Letters in the standard definition of Pascal include the 26 letters in the Roman alphabet. Pascal implementations may include both upper- and lowercase letters in their vocabulary. In such implementations, there is no distinction between upper- and lowercase except in a character string. For example, the names alpha and ALPHA would be equivalent.

Digits are the ten Arabic numerals or digits 0 through 9.

Special symbols in Pascal are tokens with special meanings. They are used to delimit the syntactic units of the language. Special symbols are represented by single special characters; two special characters, called compound symbols; and words, called wordsymbols or reserved words. In BNF form, the special symbols are:

```
special symbol = "+"|"-"|"*"|"/"|"="|"<"|">"|"["|"]"|
                 "."|","|":"|";"|"^"|"("|")"|
                 "<>"|"<="|">="|":="|".."|
                 "and"|"array"|"begin"|"case"|"const"|"div"|
                 "do"|"downto"|"else"|"end"|"file"|"for"|
                 "function"|"goto"|"if"|"in"|"label"|"mod"|
                 "nil"|"not"|"of"|"or"|"packed"|"procedure"|
                 "program"|"record"|"repeat"|"set"|"then"|
                 "to"|"type"|"until"|"var"|"while"|"with"
```

Various Pascal implementations may have additional special symbols in their vocabularies. For example, several implementations of Pascal, including UCSD and IBM, include the reserved word "string."

Special symbols are also referred to as lexical tokens. Pascal includes other lexical tokens that are formed from the basic vocabulary. These additional lexical tokens are identifiers, directives, unsigned numbers, labels, and character strings.

### Identifiers

Identifiers are names used to represent constants, types, variables, procedures, and functions. Identifiers are composed of a letter followed by zero or more letters or digits. Identifiers may be any length, although most implementations of Pascal limit the number of characters that are significant. For example, several implementations recognize only the first eight characters as significant. In implementations in which the length of identifiers are limited, two apparently different identifiers may be considered identical. For example, the two identifiers, employeename and employeenumber, would be considered identical if only eight characters are significant. An identifier cannot be spelled the same as a reserved word.

*Examples of valid identifiers:*

```
alpha
R3780
N
studentname
```

*Examples of invalid identifiers:*

| | |
|---|---|
| 3780RJE | identifiers must begin with a letter |
| tax@5% | special characters are not allowed |
| student-name | special characters are not allowed |

Some Pascal implementations allow the underline (_) special character to be used in identifiers to enhance program legibility. Thus, the invalid identifier "student-name" could be written as "student_name."

### Directives

Pascal directives are used only in place of procedure or function blocks. The directive forward is the only standard directive, but various Pascal implementations may define other directives. Directives are described with procedures and functions in Chapter 4.

### Unsigned Numbers

Decimal notation is used to represent numeric constants of integer type or real type. Scientific notation, in which the letter "e" precedes a scale factor, can be used to represent real constants. The range of values for both integer and real numbers is implementation dependent.

*Examples of numbers:*

```
94
1.3e12
3.141593
-37
```

### Labels

Labels are a sequence of from one to four digits and have an apparent integer range of 0 to 9999.

### Character Strings

A character string consists of a sequence of characters enclosed in apostrophes. Characters include letters, digits, and special characters of the character set being used. An apostrophe can be included in the string of characters by including a second apostrophe, as shown in the second example below.

*Examples of character strings:*

```
'Error'
'I can''t find the file.'
```

### Token Separators

Pascal separators are spaces (except in character strings), ends of lines, and comments. A comment is any sequence of characters and ends of lines enclosed in braces, { }. Since braces are not found on all keyboards, the Pascal standard provides an alternative set of delimiters for comments. The alternative set of comment delimeters are (* in place of { and *) in place of }. Zero or more separators may be used between any two consecutive tokens or before the first token of a statement. At least one separator must be used between consecutive tokens made up of reserved words, labels, identifiers, or unsigned numbers.

*Examples of token separators:*

```
change{this is a separator}DIV{and so is this}10
change DIV 10 {spaces are separators}
change
DIV 10      {the end-of-line between change and DIV is a
             separator}
```

## STRUCTURE OF PASCAL PROGRAMS

Pascal programs are divided into a heading and a body or block. The heading gives the program name and lists the parameters used in the program. The block consists of six

sections, but all except the statement section may be empty. These six sections, in the order in which they must appear in the program, are:

1. *Label declaration:* All labels used in a Pascal program must be declared in this section.
2. *Constants definition:* Values are assigned to identifiers in this section. The identifier cannot be assigned another value anywhere in the program.
3. *Type definition:* Programmers can define their own data types in this section. This is one of the real powers of Pascal.
4. *Variable declaration:* All variables used in a program must be declared in the variable declaration section. The variable is assigned a type, either one of the four predefined or standard types or a new type defined by the programmer.
5. *Procedure and function declarations:* Procedures and functions are program modules. They must be declared before they are called or referenced.
6. *Statements:* This is the processing part of the program.

Syntax diagrams are used throughout this book to describe the syntax of Pascal statements. A syntax diagram is a directed graph with one entry and one exit. Each syntax diagram has a name. For example, the syntax diagram for a complete Pascal program shown in Figure 1.1 is named "program." Names of diagrams can be used in other syntax diagrams. For example, the name "block" in the syntax diagram of Figure 1.1 is the name of another syntax diagram, specifically the one shown in Figure 1.2 that defines a block.

Forks are used in syntax diagrams to represent an alternative sequence. Each path at a fork defines an allowable sequence. Repetitive sequences are represented by loops in syntax diagrams. A loop may be traversed zero or more times, with each traversal defining an allowable sequence. In Figure 1.2 there is a fork near the word LABEL in which one path leads to LABEL and the other continues down. This indicates that the label declaration section is optional. Taking the path to LABEL, we find a loop that contains "unsigned integer" and ",". This loop means that there can be any number of unsigned integers (labels) separated by commas.

The symbols used in the syntax diagrams are the special symbols of the Pascal vocabulary. Names may be reserved words or basic units of the vocabulary (letters, digits, or special symbols) or they may be the names of other syntax diagrams. To distinguish reserved words from other identifiers in the syntax diagrams, names of reserved words will be in uppercase and all other names will be in lowercase. Referring to Figure

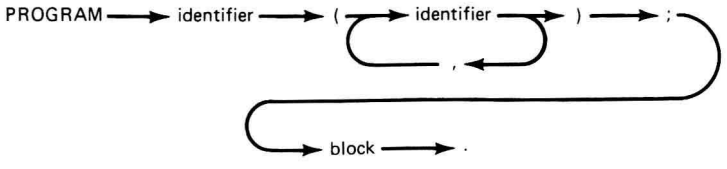


**FIGURE 1.1** Syntax diagram for a complete Pascal program.

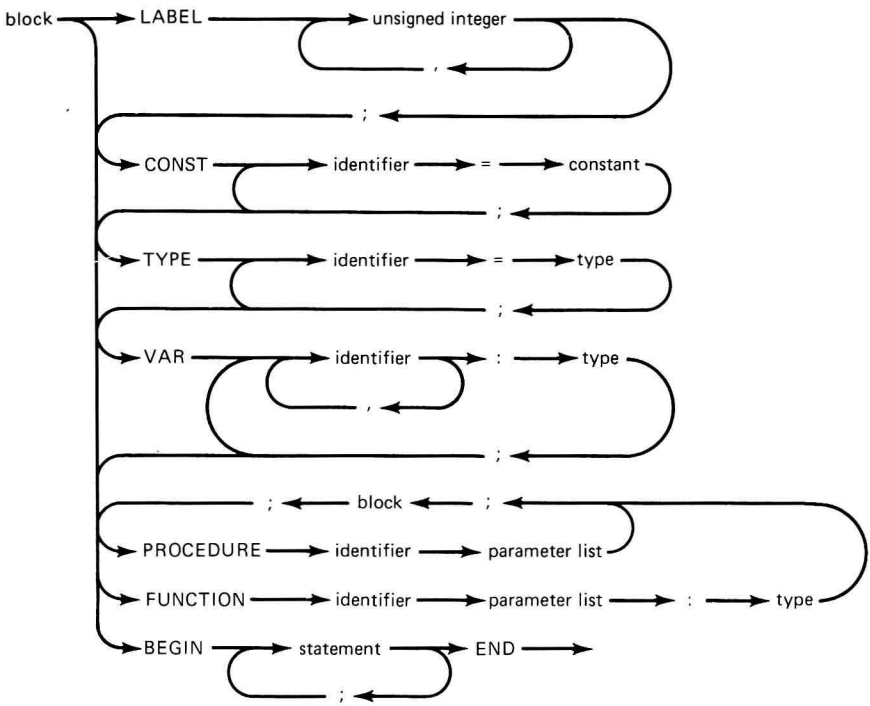


**FIGURE 1.2** Syntax diagram of a block in Pascal programs.

1.1, "PROGRAM" is a reserved word, "identifier" and "block" are names of other syntax diagrams, and the special characters are special symbols of the Pascal vocabulary.

The syntax diagram of Figure 1.1 defines a complete Pascal program. It shows the details of the header part in which the wordsymbol PROGRAM, the name of the program, and a list of parameters, enclosed in parentheses are required. The header is separated from the block by a semicolon and the program is terminated by a period. Semicolons are used to separate the different sections within a block and is also used to separate statements within the statement section. This can be seen by an examination of the syntax diagram for a block shown in Figure 1.2. The syntax diagram for a block also defines the order in which the sections must occur within the block and that all sections are optional except the statement section.

Pascal programs can be divided into subprograms called procedures and functions. These subprograms have the same structure as programs, as can be seen in Figure 1.2. Each subprogram has a heading that includes a reserved word identifying the type of subprogram, the subprogram name, and a list of parameters. Subprograms also include a block consisting of the same six sections as the program block. This implies that labels, constants, types, variables, and other subprograms can be defined within a subprogram. The definition of a label or an identifier representing a constant, type, variable, or another subprogram is valid only within the defining subprogram. This range of validity is called the scope of the label or identifier.

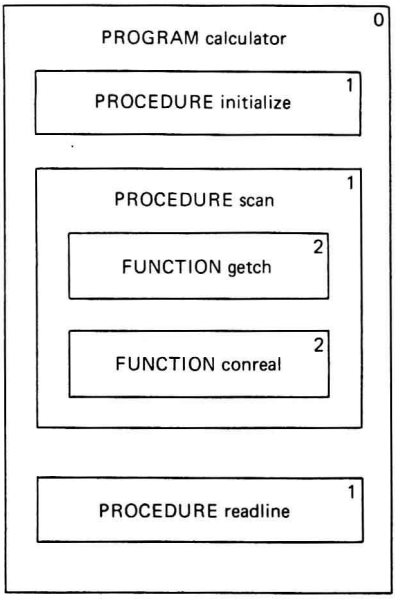


**FIGURE 1.3** Block structure of the Pascal program calculator. The number in the upper-right corner of each block is the lexical level.

Scope can best be explained by the use of examples. Figure 1.3 shows the block structure of a Pascal program and Figure 1.4 illustrates the hierarchical structure of the same program.

The scope of an identifer declared in program calculator is the entire program represented by the block drawn around the program. These identifiers are referred to as globals and they can be referenced from anywhere in the program.

The scope of identifiers declared in procedure initialize is limited to procedure initialize, as indicated by the block representing this procedure. These identifiers are local to procedure initialize and cannot be referenced from anywhere else in the program. If the same identifier is defined in both the program and procedure, there will be two separate and distinct identifiers with different scopes. The identifier defined in procedure initialize will be valid only within the procedure and the global will not be valid in ini-

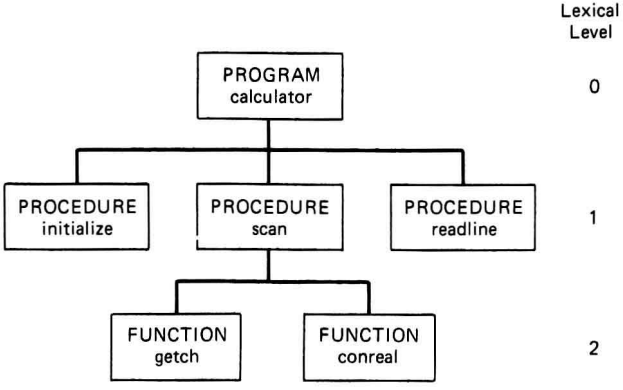


**FIGURE 1.4** Hierarchical structure of the Pascal program calculator.