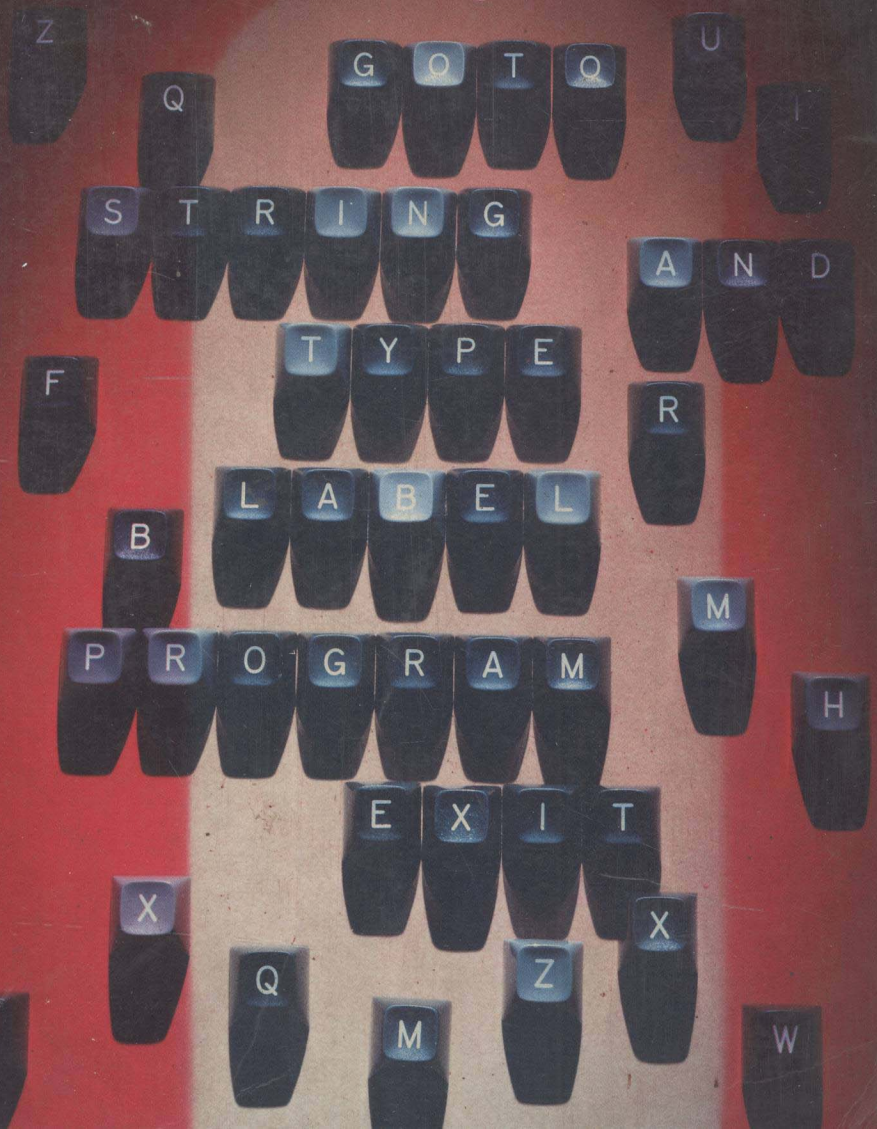
micro COMPUTER books

Including Microsoft™ BASIC and UCSD™ Pascal

Charles Seiter / Robert Weiss

PASCAL

for BASIC Programmers



Pascal for BASIC Programmers

CHARLES SEITER □ ROBERT WEISS

Addison-Wesley Publishing Company
READING, MASSACHUSETTS • MENLO PARK, CALIFORNIA
LONDON • AMSTERDAM • DON MILLS, ONTARIO • SYDNEY

This book is in the
Addison-Wesley Microbooks
Popular Series
THOMAS A. BELL, *Sponsoring Editor*
MARSHALL HENRICHS, *Design*

Library of Congress Cataloging in Publication Data

Seiter, Charles.
Pascal for BASIC programmers.

Bibliography: p.
Includes index.
1. PASCAL (Computer program language) I. Weiss, Robert, 1949– . II. Title.
QA76.73.P2S44 1983 001.64'24 82-13750
ISBN 0-201-06577-0 (pbk.)

ISBN 0-201-06577-0
ABCDEFGHIJ-HA-898765432

Preface

BASIC and Pascal are both computer languages that have been adapted elegantly to microcomputers. Although they are similar in many details, the flavor of programming in them is quite different. Each has its strengths and its weaknesses.

This book is oriented toward a reader who has a home or small business computer, is familiar with BASIC, and wants to investigate the power and convenience of Pascal. It should also be useful to a BASIC programmer encountering Pascal on a larger computer.

Organization of this book

The book contains five major parts. First, there's a chapter designed to provide about one evening's entertainment at a terminal that will enable a programmer to be doing something in Pascal right away. This is analogous to learning a few traveller's phrases in a language before beginning a really serious study. Second, there is a simple discussion of the process of translating higher level languages into machine code, and the way this process influences the varieties of Pascal. Third, there is a traditional part in which the features of Pascal are explored systematically. The emphasis in this section is placed on the practical aspects of programming. Fourth, there is a description of the process of planning and writing a Pascal program, along with an examination of two medium-sized programs. Finally, there is a BASIC to Pascal phrasebook.

Most of the examples in the book use a simplified form of BASIC; many microcomputer BASIC implementations have more convenient features, but they differ from one implementation to the next. Unless otherwise stated, the Pascal examples set off in frames will work properly in "UCSD Pascal", the most popular form for microcomputers. When Pascal program examples prompt for user input, they generally signal this with the ">", or right angle bracket, character.

Other Pascal books

The developer of the Pascal language, Professor Niklaus Wirth, has written a book called *Algorithms + Data Structures = Programs*, (Prentice-Hall) itself an interesting source of Pascal programs. In Professor Wirth's view, the real task in programming is to formulate a problem in two parts: first, to arrange the data in a convenient and natural form, and second, to devise the scheme or "algorithm" for transforming the data to the desired solution-form. The reader will note very shortly that the format of a Pascal program emphasizes the distinction between these two programming tasks.

Three other references may be useful to the reader. *The Pascal User Manual and Report* by K. Jensen and N. Wirth (Springer-Verlag) is necessary for the designer's perspective on the language and its formal definition. Most versions of Pascal define themselves in terms of this "standard" Pascal. *Programming in Pascal* by Peter Grogono (Addison-Wesley) is an exceptionally clear work and is also recommended. It may assume, however, more background in mathematics than some readers have. *Software Tools in Pascal* by B. Kernighan and P. Plauger (Addison-Wesley) contains many useful programs, along with detailed descriptions of their planning.

Acknowledgements

Registered trademarks mentioned in this book are: UCSD PASCAL®, Regents of the University of California; CP/M® and Pascal/MT+®, Digital Research, Inc., and Pascal Z®, Ithaca Intersystems, Inc.

We would like to thank Michael R. Murphy, Bill Wensil and Joseph Fukumoto for commenting on the manuscript; and Marilyn Darling of Digital Research and Laurie Moskow of Ithaca Intersystems for providing information on their companies' products.

Contents

CHAPTER 1

A Mini-Book: Pascal in One Evening

Tiny Basic, Tiny Pascal and Tiny Programs

Since Pascal has many features that are not found in BASIC, we first restrict ourselves to comparing the "tiny" versions of the two languages, where there are more similarities. This chapter will focus on the way the simplest programs are organized and particularly on the different approaches in the two languages to control statements. For the tiny programs displayed in this chapter, Pascal may seem more cumbersome than BASIC; the real virtues of Pascal will become apparent later.

An Assignment Program

In order not to burden ourselves with too difficult a first step we write a program to add two plus two and print the result:

```
10 LET A = 2 + 2
20 PRINT A
30 END
```

In Pascal, an analogous program would look this way:

A very simple program

```
program FirstGrade;
var A : integer;

begin
  A := 2 + 2;
  writeln(A)
end.
```

Some of the distinctive features of Pascal are already apparent. First, the program has no line numbers. Pascal makes every effort to keep the flow of the program readable; to this end it has enough fancy gimmicks that the GOTO statement and its associated line numbers are usually not necessary. Second, this program is like all Pascal programs in being divided into two parts: the first describes the variables that are to be used, and the second contains program statements that will be executed:

declaration of variables

```
program FirstGrade;
var A : integer;
```

executable section

```
begin
  A := 2 + 2;
  writeln(A)
end.
```

The two parts will rarely be as simple as these, but the structure will always be the same.

Pascal variables must be announced at the beginning of a program and identified by `type`; here the `type` for variable `A` is `integer`. The variety of data `types` recognized by Pascal will later appear as a feature of great power and convenience, but for simplicity we will work mainly with `integers` in this chapter.

Statements in BASIC are distinguished from each other by appearing on different lines. In Pascal, statements are separated by semicolons. The programmer is free to place statements on lines and indent the lines so as best to express what the program is doing; Pascal only looks for the semicolons. This program contains two statements. `A := 2 + 2` is only slightly different from BASIC's usage; `writeln(A)` tells you the value of A, similar to `PRINT A`.

Although we didn't use it in this case, another nice feature of Pascal allows considerable freedom in making up the names of variables. A name must start with a letter and can contain letters and digits. Pascal has a few special words; these are not available as names:

```
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
```

The program above could use `Aardvark` for the variable name, if we wished. While `Aardvark` is perhaps not especially evocative, it will be apparent that this flexibility in naming can make programs more readable and easier to document.

We will use some variable names in the Pascal version of this slightly longer program:

```
10 LET J = 5
20 LET B = 2
30 LET A = 12 * B + J
40 PRINT "THEY BOTH HAD", A, "APPLES"
50 END
```

output:

```
 THEY BOTH HAD  29            APPLES
```

An analogous program in Pascal might be clearer:

Using variable names

```
program ApplesToPlus;
var JohnsApples, BillsDozens, BothApples : integer;

begin
  JohnsApples := 5;
  BillsDozens := 2;
  BothApples := 12 * BillsDozens + JohnsApples;
  writeln('They both had ',BothApples,' apples.')
end.
```

output:

```
They both had 29 apples.
```

The choice of variable names helps show that this program answers the elementary question, "If John had five apples, and Bill had two dozen apples, how many apples did they both have?" Now we have four statements in the executable section; we require three semicolons to separate them.

We have introduced a little bit of calculation with the ordinary arithmetic operators. The reader has every reason to hope that the +, −, * and / operations are the same in Pascal and BASIC, and in fact they usually are. This extends to similarity in the rules for order of calculation in complex statements; i.e., `5*4-6` means `(5*4)-6` and not `5*(4-6)`.

Another analogy between the two tiny languages is in their input of numbers from the keyboard. The BASIC statement `INPUT` corresponds to the Pascal statement `read`, as in these two programs:

```
10 INPUT A
20 REM A=APES, B=BANANAS
30 REM WE'RE FIGURING A DOZEN BANANAS PER APE
40 LET B=12*A
50 PRINT "BANANA ORDER =",B
60 END
```

Reading numbers

```
program ApeLunch;
var Apes, Bananas : integer;

begin
   read(Apes);
   Bananas := 12 * Apes {WE'RE FIGURING A DOZEN PER APE} ;
   writeln('Banana order = ',Bananas)
end.
```

Pascal's `read`, however, does not print a prompt like BASIC's `?`. Pascal will sit patiently and quietly until you type a number. Since this example was so straightforward, it seemed like an appropriate place to put in a note about documentation or comment statements. The `REM` statement in BASIC allows the programmer to leave notes throughout the program; the program ignores them. In Pascal this is accomplished by enclosing the notes in curly brackets { } to form a "comment". (In most versions of Pascal, the pairs `(*` and `*)` may be used as well as { and }.) Comments may be inserted anywhere in the program (except in the middle of names) and similarly do not affect the program's operation. All books on programming contain some authors' prejudices about the way things should be done; here are two rules:

1. Comment on every program feature that is not completely obvious at first glance.

2. Since what is obvious now won't be obvious next year, also comment on every program feature that is obvious at first glance.

In most versions of BASIC the presence of comments slows down the program. This is not true in Pascal, where the computer discards your comments while preparing its own version of your program ("compiling" the program).

Simple Decision Statements

The BASIC statement `IF . . THEN` has a close counterpart in the Pascal construct `if . . then . . else`. These are the simplest deci-

sion elements in the two languages, and can be compared in a few examples.

As an elementary program, we will consider using the computer as a suspicious electronic bartender. The BASIC program for this uses an IF . . THEN:

```
 10 PRINT "HOW OLD ARE YOU"
 20 INPUT A
 30 IF A < 21 THEN 100
 40 PRINT "WHAT WILL YOU HAVE?"
 50 GOTO 110
100 PRINT "GET LOST, JUNIOR!"
110 END
```

Pascal provides a "high-level construct" to replace the common but clumsy BASIC sequence.

The if . . then statement

```
program Bartender;
var Age : integer;

begin
  writeln('How old are you?');
  read(Age);
  if Age > 20 then begin
    writeln('What will you have?')
  end
  else begin
    writeln('Get lost, junior!')
  end
end.
```

For the present, we will distinguish the parts of the program executed depending on the if . . then . . else by enclosing them between begin and end. This is not always necessary, as described in Chapter 5. As a note about punctuation, the example illustrates that

such Pascal words as begin and end, since they are not statements, do not have to be separated from statements by the semicolon. It also demonstrates one of the features of Pascal that makes it so easy to read, namely that the choices of action to be resolved by an if . . then . . else appear on an equal footing following then and else. Long BASIC programs suffer from the problem that IF . . THEN branches to statement numbers are hard to follow and can become a source of confusion. The Pascal construct can be extended repeatedly with no problems of readability, as is demonstrated by the sinister program CorruptBar:

Nested if . . then statements

```
program CorruptBar;
var Age, Bribe : integer;

begin
  writeln('How old are you?');
  read(Age);
  if Age > 20 then begin
    writeln('What will you have?')
  end
  else begin
    writeln('How bad do you need a drink?');
    read(Bribe);
    if Bribe > 10 then begin
      writeln('What will you have, sir?')
    end
    else begin
       writeln('Get lost!')
    end;
  end;
end.
```

Enclosing a series of statements between begin and end indicates that they will be performed one after another; in BASIC it would be possible to branch into the middle of such a sequence and