micro books

Peter Brown

# Pascal FROM BASIC

# Pascal from BASIC

PETER BROWN

**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*

Composition in Times by Filmtype Services Limited,
Scarborough, North Yorkshire.

Printed in Finland by Werner Söderström Osakeyhtiö, Member of Finnprint 1982.

ISBN 0-201-10158-0

ABCDEFGHIJ-898765432

To my wife who nagged me to finish;
my girlfriend who kept me cheerful;
my colleague who criticized;
my boss who understands me all too well.

To the one person who is all of these.

The publishers wish to thank Dave Farris for the cartoon illustrations.

# Preface

VISITOR: *Can you tell me the way to the Post Office?*

LOCAL: *If I was going to the Post Office I wouldn't start from here.*

This book assumes that you are a reasonably competent BASIC programmer and that you want to learn Pascal. Most Pascal books are written as if the reader does not know how to program. Your problem is quite different, however, from that of the novice. You do not want explanations of basic concepts such as variables, loops and so on, but you do want help in adjusting your manner of thinking from BASIC to Pascal. In fact your task is probably *harder* than that of the novice, because BASIC is actually a maverick among programming languages – though it has become the world's most popular language in spite of (because of ?) this. It is harder to learn concepts and then to relearn them in a new way, than to start from scratch. Make no mistake, switching from BASIC to Pascal does mean a radical change. If you write your Pascal programs in the same way as your BASIC programs you will be like an English-speaking tourist in France translating your sentences word for word from a dictionary.

This book is aimed specifically to help you conquer the problems that BASIC programmers have with Pascal, and to adapt your programming style accordingly. It is not assumed that you are forsaking your old friend BASIC entirely. Instead the book is quite critical of some aspects of Pascal, and tries to make clear to you where Pascal gains and where it loses.

The book also aims to be light, and fun to read. The goal is to be serious without being solemn and stodgy. Concepts are introduced almost entirely in terms of examples, and, where relevant, these are related to BASIC. If you prefer a more formal approach, look elsewhere.

When you read this book you should gain two capabilities. You should be able to write good Pascal programs, and to read Pascal reference manuals and appreciate what they are all about.

The book is strenuously neutral towards competing versions of Pascal: with absolute fairness, it does not mention any of them. Fortunately, different Pascal systems are much more similar than are BASIC systems, so it is possible to give a detailed discussion without commitment to a particular implementation. Thus it is immaterial whether you are planning to use Pascal on a personal computer or on a large time-shared mainframe computer.

## Acknowledgements

I would like to thank Laurence Atkinson who, by his enthusiasm, unknowingly inspired me to write this book. Thanks are also due to Tony Hoare for valuable guidance and to several of my colleagues, including Stephen Binns, David Turner and, most especially, Peter Welch. A kind and friendly Pascal system provided the layout of the programs shown in this book.

Finally let me record my gratitude to two of the world's greatest. Marianne Kong typed the manuscript with amazing care and conscientiousness, and put up nobly with all my changes. Heather Brown spent as many hours in checking and criticizing as I did in writing. Her caustic comments and her savage condemnations annihilated the worst parts of the book before they reached print.

Peter Brown

Canterbury
October 1981

# Contents

# CHAPTER 1

# An example to show the fundamentals

**begin** *at the beginning.*

DYLAN THOMAS

## An example

To get a feel for the task in hand, we shall start with a very simple BASIC program and show its Pascal equivalent. The program takes as data a number N, which is followed by N further numbers. All the program does is to print the average of the numbers. In BASIC the program can be written

```
10  REM---FINDS THE AVERAGE OF N NUMBERS---
100 INPUT N
110 LET S = 0
120 FOR K = 1 TO N
130   INPUT X
140   LET S = S + X
150 NEXT K
200 PRINT"AVERAGE =";S/N
999 END
```

A direct translation of this into Pascal – which, as we shall see, is not a good Pascal program – is

```
program average(input, output);

{ ---finds the average of N numbers--- }

var
    n: integer;
    k: integer;
    s: real;
    x: real;

begin
    read(n);
    s:= 0;
    for k:= 1 to n do
    begin
        read (x);
        s:= s + x;
    end;
    writeln ('Average =', s / n);
end.
```

If you have never seen a Pascal program before, many differences from BASIC will strike you straight away.

The most manifest is, paradoxically, one of the least significant: the fact that the BASIC program uses upper case letters (i.e. capital letters) whereas the Pascal program uses lower case. The first point to make is that this is largely a matter of habit rather than requirement. We could have written the BASIC program in lower case and the Pascal in upper case, and the programs would still be acceptable to most compilers.

The habit – and it is not a universal habit – of using upper case letters in BASIC results largely from history. BASIC grew up in the early sixties when computers had very small character sets; input to computers was often from punched cards or primitive typewriters, and lower case letters were usually not available. More recently, computers and devices for entering data into computers have supported richer character sets. You no longer have to SHOUT AT THE COMPUTER IN UPPER CASE, but can communicate in a much more refined way in lower case. The pendulum has now swung so far that many programmers avoid upper case altogether, and programming manuals for more recent languages tend to do likewise. Doubtless it will swing back one day to a more moderate position.

The current convention is, however, quite useful for this book. We shall present BASIC programs in upper case and Pascal programs in lower case. We can thus talk about and compare programs without ambiguity.

It can also be seen that the Pascal program contains bold face words, such as **begin**. Again, this is a habit of presentation rather than a fundamental property of the language. When you type the program into the computer you type the bold face letters as ordinary letters.

## Declarations

A second striking facet of the programs is that in BASIC the action starts in line one (or, strictly speaking, line two as the first line is a comment), whereas in the Pascal program there are half a dozen lines of introduction before anything actually happens. The first line is

```
program average(input, output);
```

A line of this form needs to come at the start of every Pascal program – the name you choose to give the program, in this case *average*, being inserted into what is almost a fixed template. We shall explain this program heading line later. The line after the program heading is a Pascal comment, akin to a REM in BASIC.

Much more important are the lines

```
var
    n: integer;
    k: integer;
    s: real;
    x: real;
```

These are examples of *declarations*. In BASIC, if you want to use a variable X you can do so with no fuss or preparation; in Pascal, on the other hand, you must declare the variable X before you can use it. BASIC does, of course, contain the concept of declarations. If you want to use an array (table) Q in BASIC you need to declare it by a line such as

```
DIM Q (5, 8)
```

Pascal has the same idea, but you have to declare *everything*.

When you declare a variable you need to specify its *data type*. Data types are one of the most powerful and exciting features of Pascal, though this is not brought out by our present sample program. Pascal has some built-in data types, like the *integer* and *real* types used above, and also allows you to define your own types, as we shall see. BASIC has an embryo idea of data type. Variables are normally of *numeric* data type, as N, K, S and X in our example. However, if a dollar sign is appended to its name a variable is of *string* data type, e.g. A$, P9$. Some BASICs also support an *integer* data type, often represented by names such as K%, T%.

A data type specifies the set of values that an object may take. Associated with each data type is a set of operators; you can for example apply a multiplication operator to numeric data. You cannot multiply two strings together, but there are other operations that apply uniquely to strings, such as extracting a substring. Some operators are *polymorphic* in that they apply to more than one data type. In BASIC this applies, for example, to the assignment operator, and to some operators on IF statements, e.g.

```
LET X = 3
LET X$ = "STRING"

IF X = Y THEN 30
IF X$ = Y$ THEN 50
```

Similar rules apply to Pascal. BASIC data types are manifest from the name of a variable: X is numeric and X$ is a string. In Pascal any name can be used for any variable. When you declare the variable you give its name and its associated data type. We shall see later that there is freedom of choice with Pascal names. We could, and indeed should, have used the name *sum* or even the name *sumofvariables* rather than the name *s*. As a general principle it is better to choose names that make the purpose of the variables clear; our use of single character names was to maintain a direct correspondence with BASIC.

The data types used in the *average* program are *real*, which is the same as the numeric type of BASIC, and *integer*. The variables *s* and *x* are declared to be real variables, and *n* and *k* to be integer variables. (We could have written these four declarations in any order, just as in BASIC you can declare arrays in any order.) It is important in Pascal to distinguish variables that can only take on integer values, like *n* and *k* in the *average* program, from those that can take any numeric value, like *s* and *x*. The reasons for this stem mainly from the design of current computers. If a variable is known to take only integer values, it can be stored in a different and more compact way from a variable that can take any numeric value. Moreover, operations on integers run much faster than on real values; the factor may range from two to over one hundred. Finally, the integer operations are exact. Real operations are inexact, and can give small round-off errors. You may well have experienced the effects of this round-off when running a BASIC program; perhaps you expected the answer 6 from a program and instead the answer came back as 6.000000001.

In most BASICs you ignore the difference between integers and real numbers by making everything real (numeric), and you put up with the occasional eccentricity such as the 6.000000001. In Pascal you can sometimes get away with this, but cannot use a real variable on a **for** statement or as an array subscript. Given these requirements and their indirect consequences, it is best to distinguish integers from reals. The only possible problem with using the integer data type is that some implementations place quite severe limits on the size of integers – integers may, for example, be forbidden from exceeding 32,767. Consult your local manual for details.

## The program

When you look at the executable instructions in the Pascal program you are on more familiar ground. The statements in the Pascal example are in one-to-one correspondence with the BASIC example except that Pascal has a couple of **begin**s and **end**s. The **begin**s and **end**s are one of the *structuring* concepts of Pascal. Favourite concepts in programming come and go, but structuring has remained on the best sellers' list for several years now. The reason is quite fundamental: the only way we can hope to understand a large program is to build it out of smaller and simpler substructures.

To get an idea of the purpose of **begin** and **end** it is instructive to consider two examples of FOR statements in BASIC and Pascal

```
BASIC                                  Pascal

10 FOR K = 1 TO 10                      for k := 1 to 10 do
20   LET S = S + K                         s := s + k;
30 NEXT K

60 FOR K = 1 TO 10                     for k := 1 to 10 do
70   LET S = S + K                     begin
80   LET T = T + K * K                    s := s + k;
90 NEXT K                                 t := t + k * k;
                                       end;
```

The FOR statement is the only structuring construct in standard minimal BASIC. Each FOR is matched by a NEXT and the statements in between are treated as a unit. In Pascal there are many structuring constructs within programs, of which **for** is one. Several of these constructs use **begin** and **end** to enclose a group of statements that is to be treated as a single unit. This unit is called a *compound statement*. The **for** construction in Pascal is not in itself a statement. It is a *clause* that must be prefixed to a single statement. If you want to make several statements the subject of the **for** you put a **begin** and **end** round them; this turns them into a compound statement, which counts as a single statement. In our first **for** example above we do not need a **begin** and **end** because there is only one statement to be iterated. However it does not matter if you insert extra redundant **begin**s and **end**s, so we could have put them in. Our second example needs the **begin** and **end** because more than one statement is to be repeated.

The body of an entire Pascal program is enclosed by a **begin** and **end**. The final **end** is written

```
end.
```

The dot means it is the very end of the program. Like FOR and NEXT, **begin** and **end** nest in a natural way, e.g.

```
begin {A}
  {...}
begin {B}
  {...}
end; {B}
  {...}
begin {C}
  {...}
end; {C}
  {...}
end; {A}
```

Here, the **begin** we have marked with the comment {*A*} matches the **end** similarly marked, and so on. A good program is laid out so that it is visually obvious which **begins** and **ends** match e.g.

```
begin
    {...}
    begin
         {...}
    end;
    {...}
    begin
         {...}
    end;
    {...}
end;
```

It is, of course, wrong to have **begins** that are not matched by **ends** or vice-versa.

You will find that, because of the preponderance of structuring concepts in Pascal, a little care in program layout will reap big rewards in readability. If you are lucky, you will have a 'prettyprinting' utility on your computer. This is a program that takes a Pascal program and turns it into a decently laid out form. (Prettyprinters also exist for BASIC, but the task is a smaller one.) However, although such programs are useful for a final tidy up, it is much better for you to get in the habit of laying out programs decently from the outset.

## Individual statements

The individual statements in our *average* program are similar in both BASIC and Pascal, e.g.

| *BASIC* | *Pascal* |
|---|---|
| INPUT N | *read*(*n*); |
| LET S = S + X | *s* := *s* + *x*; |
| PRINT "AVERAGE = "; S/N | *writeln*('*Average* =', *s* / *n*); |

Pascal, unfortunately from the point of view of the BASIC programmer, uses *read* to mean INPUT. Pascal has no concept similar to READ in BASIC, so if you