The Pascal Programming Language
Bill Catambay, Pascal Developer

Chapter 2

The Pascal Programming Language
by Bill Catambay

Return to Table of Contents

II. The Pascal Architecture

Pascal is a strongly typed, block structured programming language. The "type" of a Pascal variable consists of its semantic nature and its range of values, and can be expressed by a type name, an explicit value range, or a combination thereof. The range of values for a type is defined by the language itself for built-in types, or by the programmer for programmer defined types. Programmer-defined types are unique data types defined within the Pascal TYPE declaration section, and can consist of enumerated types, arrays, records, pointers, sets, and more, as well as combinations thereof. When variables are declared as one type, the compiler can assume that the variable will be used as that type throughout the life of the variable (whether it is global to the program, or local to a function or procedure). This consistent usage of variables makes the code easier to maintain. The compiler detects type inconsistency errors at compile time, catching many errors and reducing the need to run the code through a debugger. Additionally, it allows an optimizer to make assumptions during compilation, thereby providing more efficient executables. As John Reagan, the architect of Compaq Pascal, writes, "it was easy to write Pascal programs that would generate better code than their C equivalents" because the compiler was able to optimize based on the strict typing.

Declaring variables in Pascal is straightforward. The Pascal VAR declaration section gives the programmer the ability to declare strings, integers, real numbers and booleans (to name a few built-in types), as well as to declare variables as records or other programmer defined types. A variable defined as a RECORD allows a single variable to track several data components (or fields).

```
Type
   Employee_type   =   (Hourly, Salary, SalaryExempt);
   InputRec        =   RECORD
      emp_name:      packed array[1..30 ] of char;
      social:        packed array[1..9] of char;
      salary:        real;
      emp_type:      Employee_type;
      end;

Var
```

```
        index:      integer;
        ratio:      real;
        found:      boolean;
        inpf:       file of InputRec;
```

Figure 1: Sample Code - Types and Records

Pascal also supports recursion, a powerful computing tool that allows a function or procedure within a program to make calls to itself. This allows for elegant and efficient coding solutions, eliminating the need for tedious loops. A good example of recursion is the following Pascal solution to the classic Towers of Hanoi puzzle (see Figure 2). The puzzle is to take a stack of disks in increasing sizes from top to bottom, and move them from the first peg to the second peg, with the rule that a disk must always be placed on a disk larger than itself, and only one disk can be moved at a time.

```
Program TowersOfHanoi(input,output);


Var
   disks:   integer;

Procedure Hanoi(source, temp, destination: char;  n: integer);


   begin
   if n > 0 then
      begin
      Hanoi(source, destination, temp, n - 1);
      writeln('Move disk ',n:1,' from peg ',source,' to peg ',destination);
      Hanoi(temp, source, destination, n - 1);
      end;
   end;

begin
write('Enter the number of disks: ');
readln(disks);
writeln('Solution:');
Hanoi('A','B','C',disks);
end.
```

Figure 2: Sample Recursive Code ó Towers of Hanoi Solution

The solution to the Towers of Hanoi puzzle involves moving all but one disk from peg to peg, repeatedly, until the entire stack has moved from one peg to the other. The elegance of recursion is that this solution is illustrated clearly, without mundane loops and logic checks. The three steps of the solution are depicted by three lines of code. The movement of a stack, regardless of size, is always done by a call to Hanoi, thus ensuring that the rules are adhered to.

Pascal eliminates the need for clumsy "goto" statements by supporting REPEAT/UNTIL, WHILE/DO, and FOR loops; by providing an intelligent CASE statement; and by providing a means to consolidate common lines of code into PROCEDUREs and FUNCTIONs. Using the English words of BEGIN and END to delimit blocks of code within a clause, enforcing strong typing, providing ordinal-based arrays, and other useful linguistic features, Pascal facilitates the production of correct, reliable, and maintainable code. Any language can be commented and indented for better readability, but Pascal, by the nature of its syntax and architecture, encourages structured programming practices and allows the programmer to focus on developing solutions. It's important to emphasize this element of Pascal. Even programmers with the most sophisticated and disciplined of programming styles will find themselves in a time crunch. With deadlines quickly approaching, it's likely that a programmer will focus more on achieving a result and less on making the code understandable for future maintenance. The key to Pascal is that a programmer tasked with maintaining Pascal code will be able to make the same assumptions that the compiler makes about program flow and data usage. This gives the maintenance programmer a fighting chance of figuring out the behavior, purpose, and operating conditions of the code, even if itís poorly-written.

Block Structure

In *An Introduction to Programming and Problem Solving with Pascal*, the author writes,
"A block is a sequence of declarations, a begin, a sequence of statements that describes actions to be performed on the data structures described in the declarations, and an end." A Pascal program consists of a PROGRAM heading, which names the program, followed by a block. Within that main program block, there exist subprograms, each of which have their own heading followed by a block. Within each block, there can be inner blocks, and within each inner block there can exist further inner blocks. In essence, a Pascal program is a hierarchical construction of blocks ; hence, Pascal is a block-structured programming language.

All data values declared at the beginning of a block are accessible to the code within the block, including inner blocks, but not to any others. The usefulness of a block-structured language, therefore, is not only the modularization of the program, but also the protection of data that is exclusive to one set of modules within a program from compromise by other modules.

Style

The flow of Pascal code often reads like plain English, with code indentation playing a crucial role in visualizing conditional clauses. The BEGIN and END statements are key elements of Pascal's architecture. These clause delimiters are often misunderstood and misused, leading even the most avid Pascal programmers to question their usefulness. Used properly, however, they are a vital to visualizing code clauses within a program. Take, for example the Towers of Hanoi solution (see Figure 3 below).

```
Program TowersOfHanoi(input,output);

Var
```

```
     disks:   integer;

  Procedure Hanoi(source, temp, destination: char;  n: integer);

     begin
     if n > 0 then
        begin
        Hanoi(source, destination, temp, n - 1);
        writeln('Move disk ',n:1,' from peg ',source,' to peg ',destination);
        Hanoi(temp, source, destination, n - 1);
        end;
     end;

  begin
  write('Enter the number of disks: ');
  readln(disks);
  writeln('Solution:');
  Hanoi('A','B','C',disks);
  end.
```

Figure 3: Indentation Style & Block Structured Code

The BEGIN and END statements are colored in Figure 3 to illustrate a point. While these words are not typically colored as such in an editor, when a programmer is trained on these words, they are just as visually apparent. Further, when setting the indentation of the clause to match the BEGIN and END statement, a virtual line of sight is established. When clauses are imbedded within other clauses, sometimes several layers deep, these lines of sight become very helpful in deciphering the hierarchy of clauses and flow of execution. Less work is required to understand and rediscover the flow of a program and the programmer can therefore focus on the logic and algorithms.

In a small program with very few lines of code, this advantage is not quite as apparent. The usefulness, however, increases exponentially as complexity and size increase. For example, take an extensive program of several pages, with long clauses that may pass through several page breaks (see Figure 4 below).

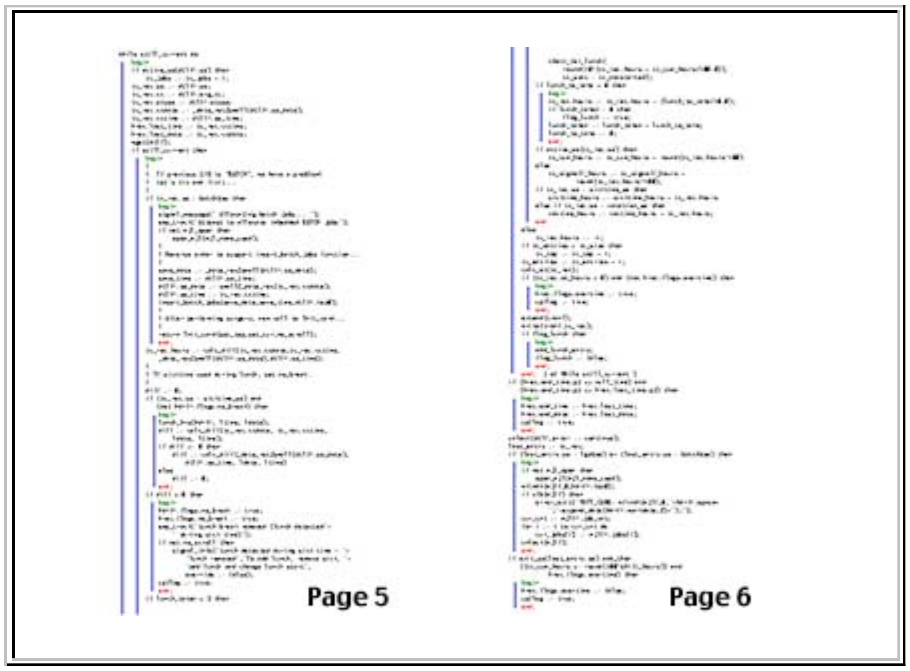

Figure 4: Indentation of Block Structured Code ó At A Glance

The virtual lines of sight are illustrated in Figure 4 with blue lines, and the code itself is purposely blurred to focus attention on the indentation. In a large program, determining the program flow is required each time the programmer attacks the code. The quicker that flow is apparent, the sooner the programmer can get to work at updating the code.

Manageability

Programmers at the beginning of a project face creating a solution to a real world problem using computer code. Over time, however, programmers face the ongoing problem of maintaining and enhancing the computer code as the users needs change and grow. Increasing the manageability of code, both for initial implementation and for long term maintenance, decreases the amount of effort required to work the problem.

Pascal increases manageability of code by enforcing strong typing, supporting block structured programming, and providing a syntax which is easy to read. These aspects of Pascal provide both immediate and long-term benefits to the programmer. Strong typing removes much of the guess work from interpreting data structures. Block structured programming breaks down a program into a hierarchy of tasks. The decomposition of a programming problem into a hierarchy of tasks enhances the manageability of the problem. Finally, the more visually apparent the blocks are within a program, and the more the code reads like English, the easier it is to interpret the program flow, thereby further increasing manageability.

These aspects of Pascal provide the programmer tools for long term manageability in supporting legacy code as well. Each time a programmer attacks legacy code, the logic flow and data structures must be understood before any work can be done efficiently. Guessing incorrectly at what the code or data structures were designed to do can lead to costly bugs in the software. The sooner the programmer can understand the code clearly and with confidence, the quicker the programmer can get "into the groove" of debugging or enhancing the code. Each block, variable, and line of code may represent only seconds of time saved using Pascal, but these seconds add up. A penny saved may not seem like much, but a penny saved every second, 60 hours a week, 50 weeks a year is over $100,000. Moreover, when it takes a very long time to find the groove, the effort to understand the code becomes so frustrating that it impedes creativity and productivity.

Return to Table of Contents Next Chapter