The Pascal Programming Language

Bob Fraley
HP Labs

January 15, 1980

The Pascal language has gained popularity in recent years. It has a number of features which simplify programming and make programs more readable. We shall examine some major features of Pascal. The principal reference for Pascal is the book by Jensen and Wirth [1].

Pascal Basics

Pascal is like most other Algol-based languages. Identifiers (words) consist of letters and digits. Certain identifiers are reserved keywords (such as if and end). There are integer, real, and character constants:

```
1234   123.4567  1.2e20  'x'  'Pascal string'
```

Pascal has lots of the expected operators, like "+" and "*", and a variety of built-in functions, like "abs" and "sin".

Statements in Pascal are similar to those of Algol. They may appear anywhere on the line, and are separated by ";". A collection of statements may be treated as a block by surrounding them with the keywords "begin" and "end". Pascal programs are divided into "procedures" and "functions"; functions return results and procedures do not.

So much for the similarities. The interesting part of Pascal is the differences.

Pascal Data Types

The most interesting new feature of Pascal is its data type facilities. These distinguish Pascal from its predecessors. Pascal allows the declaration of data structures, much like Cobol or PL/I, but differs in allowing the structure itself to have a name, rather than only naming the data. All other kinds of data in Pascal can also be named. A type name may refer to some range of integers, to an array containing some specific element type, to a file or to a pointer. These named types may then be used in defining variables and other types.

But we're getting ahead of ourselves. Let's look at some Pascal data types. When we declare an integer variable, we would normally write the Pascal declaration:

```
Var    I: Integer;
```

A person reading the program now knows that I has an integer value. But the declaration gives neither the human reader or the machine reader any hint as to the size of the integer which I might represent. An alternate declaration in Pascal would be:

```
                         Var    I: 1..10;
```

This declaration specifies that I may only represent an integer in the specified range 1 to 10. Different variables may have different ranges. If a given range is used often, it may be assigned a name:

```
                     Type    Small = 0..255;
```

This type may be used in later declarations:

```
             Var    J: Small;
                    A: Array [Small] of real;
```

The second declaration is equivalent to writing the subscript range directly in the array declaration.

Pascal records describe structured data. Each component of a structure may have any type. Fortran or Basic programmers get along without structured data by making several arrays and using the corresponding entry of each array as components of a structure. The relationship between the arrays is not documented in the Fortran source code. As an example, the following declarations could define a table of names and their definitions. The "Const" portion of the program defines constants which may be used within the body of the program.

```
      Const  stringlen = 30;
             tablelen  = 579;
      Type
             string = packed array [1..stringlen] of char;
             entry  = record
                         name:   string;
                         value:  real;
                      end;
      Var
             table  : array [1..tablelen] of entry;
```

A terse version of the same declaration could be made as follows:

```
      Var    table : array [1..579] of record
                      name: packed array [1..30]of char;
                      value: real end;
```

The first declaration has several advantages. First of all, the sizes have been placed where they can be found easily. Secondly, by defining the name "entry", the programmer can declare local variables which look like entries, or can place an entry inside of another structure.

Some Pascal data types exist primarily to increase readability. For example, if a programmer wants to define some numeric codes to represent colors, explicit code numbers need not appear in the source code. Enumeration types define the code values and keeps the programmer from confusing different code types.

```
               Type Color = (Red, Orange, Yellow, Green,
                             Blue, Purple);
                    Fruit = (Apple, Pear, Plum, Cherry);
```

A variable of type "Color" can be assigned the value "Red" but not the value "Apple".

Pascal set types may be used when several values might apply simultaneously. In the example below, there are three access rights which might be granted in any combination.

```
                Type
                     Access = (Read, Write, Execute);
                Var
                     Rights: Set of Access;
```

Sets may be specified by listing the elements of a set between square brackets. A set of brackets containing no elements designates the empty set. Set operators + (or), * (and), and - (set difference) are provided, as well as the conditions in (element of), = (equality), and <= (subset). Here are some sample statements which use set operations:

```
                Rights := [read, execute];
                Rights := Rights + [read] - [execute];
                if write in rights then ...
                if rights * [read, write] = [read] then ...
```

Statements

Pascal has a small assortment of statements, which include assignment; procedure call; if and case selection; for, while, and repeat loops; and the with statement. Two statements which hold particular interest are the case statement and the with statement.

The case statement allows the selection of an action based on the value of a variable. The variable can be an integer, character, or an enumeration type. Here are some examples [1]:

```
      case operator of              case i of
        plus:  x := x+y;              1: x := sin(x);
        minus: x := x-y;              2: x := cos(x);
        times: x := x*y;              3: x := exp(x);
      end                             4: x := ln(x);
                                    end
```

The with statement establishes a record whose field names may be referenced like ordinary variables. This is best understood by an example. A symbol table entry (using the type declaration specified earlier) may be created using the statements:

```
             table[i].name := 'John Q. Jones                  ';
             table[i].value := balance;
```

where "balance" is the name of a variable. Notice that the "table[i]" specification had to be written twice. This could be time consuming for both the programmer and the machine. The "with" statement allows the same statements to be written:

```
              with table[i] do begin
                 name := 'John Q. Jones                  ';
                 value := balance;
                 end;
```

This technique is most useful when a record has many fields.

Heap Storage

The Pascal compiler provides a memory management facility which can simplify many kinds of programs. Pascal's "heap" is a storage management system which is useful for list processing. The "new" procedure allocates space from the heap for storing a variable of a specific type. New returns a "pointer" which may be used to reference the allocated space. The pointer name, followed by the up-arrow symbol "^", is used to refer to the data. The value "nil" is a pointer which points to nothing; it is generally used to indicate the end of a list. Here is an example for building a stack:

```
          Type
              Stack = ^ Node;        { Stack points to type node}
              Node  = Record
                       Data : Integer;
                       Next : Stack;
                      End;
          Var
              P, Top: Stack;
                  .   .   .
               Top := nil;
                  .   .   .
               New (p);
               With p^ do begin
                  Data := 17;  Next := top;
                  End;
               Top := p;
```

When allocated space is no longer needed, it may be reclaimed by calling dispose(p).

Procedures and Functions

Procedures in Pascal are termed "Functions" if they return values and "Procedures" if they don't. The procedure header lists the parameters for the procedure, together with their types, and in function headers the result type follows the parameter list.

The body of the procedure is divided into a number of sections which are separated by keywords. The sections with their keywords are shown below:

```
        Procedure
                 procedure name and parameters;
        Label
                 Labels used within the procedure code block;
        Const
                 Declarations of local constants;
        Type
                 Declarations of local types;
        Var
                 Declarations of local variables;

                 Local procedure and function declarations;
        Begin
                 statement block
        End
```

Procedures, like all other Pascal constructs, must be defined before they can be used. When two procedures refer to each other, one of the procedures must be defined with a "forward" declaration. This consists of the procedure heading (its name, parameters, and result type) followed by the word "Forward". The body of the procedure must appear later in the program.

Input/Output

The I/O facilities of Pascal are basic, omitting some features of other languages. They deal only with sequential files, and are adequate for most sequential file applications. Files may have records of any type (except types containing files); the formats of these files are implementation dependent.

The kind of files which are used most often are "text" files. Such files contain characters which have been organized into lines. Two text files are provided by the language as standard input and output files; they are appropriately called "input" and "output".

The Pascal procedures "Read" and "Write" provide formatted input and output. The functions eoln(f) and eof(f) are useful for determining when the end of a line or end of the file are reached. Finally, Reset(f) and Rewrite(f) open file f for input or for output.
Of all features in Pascal, more people have trouble with input files. Let's look at input to text files and see where the troubles lie. The procedure call read (f, x) reads a character, integer, or real from the file f, and assigns its value to x. Seems simple enough. But lurking behind the scenes is the file's window variable. The window f^ contains the next character to be read from the file. While this notion works well for disk files, it throws off the expected coordination with reading from a terminal. If the procedure read(f, c) reads a character from file f and file f contains the string 'abc', then when c is assigned the value 'a', the b has already been read from the terminal. The consequences of this lookahead are the source of troubles:

1. At the end of each line a extra blank is inserted in the file. When this blank is in the file window, the function eoln(f) returns true. Many people think this means that when read(f, c) returns a blank in variable c, they whould check the eoln flag. On the contrary, eoln would have been true when the previous character was read (and the blank was in the file window), and will have been turned off again when c returns a blank.

2. The procedure readln(f) skips the rest of the current line. The next read(f,c) returns to c the first character of the next line. That means that the first character of the new line was placed in f^ during the readln(f) procedure. Writing a prompt line immediately after the readln call will be too late to prompt for the next input, since it will have been read already.

3. When the program starts, the first input character is in the window variable. Therefore a read has already occurred and the program cannot prompt for input.

Fortunately, there is a movement within the Pascal community to change the point at which the read operation occurs. Rather than indiscriminately reading the next input character into the window variable, the read only occurs when it is needed. A reference to f^ or a call to eof or eoln will read the window character; otherwise, the read is deferred until the next read(f). This solves the second and third problems listed above, allowing prompting to be placed as expected.

Summary

Pascal has a number of features which simplify programming. Data types make program specifications more accurate and improve the error checking properties of the compiler. Pascal's statements add structure to the program to reduce errors and increase readability. I/O has been simplified, but is still confusing.

Pascal has a number of deficiencies, which will become apparent after some programming experience. There is currently a standardization effort which will add a few changes to improve the language; future work will be needed to overcome additional problems. Some of the known solutions have not been added to Pascal in the hopes that simpler, cleaner approaches will be discovered in the future.

Bibliography

[1] Jensen and Wirth "Pascal User Manual and Report" Springer-Verlag, New York 1976.