Apunte Laboratorio ALPI - El lenguaje de programación Pascal

Índice

1. Estructura de un Programa en Pascal

El lenguaje de programación Pascal es un lenguaje estructurado, lo que significa que cada programa requiere una forma específica de escritura para que sea entendido por el compilador.

En esta sección veremos la estructura que tendrá un programa en Pascal:

```
Program Nombre;                          // Cabecera
Uses                                     // Declaraciones
     Lista de Bibliotecas;
Const
     Lista de Constantes;
Type
     Lista de Tipos de Datos;
Var
     Lista de Variables;

Begin                                    // Programa
     Sentencias;
End.
```

Por ejemplo, el siguiente es un programa básico, que escribirá en pantalla la palabra “Hola”:

```
Program Saludo;
Begin
     Write ('Hola');
     Readln;
End.
```

La línea `Program` es opcional y sirve para ponerle un nombre al programa; de esta manera, se tiene una idea de lo que hace el mismo.

Las palabras `Begin` y `End` marcan el principio y el final del programa.

Cada sentencia de Pascal debe terminar con un punto y coma, salvo el último `End`, que lo hará con un punto. No es necesario un punto y coma después de un `Begin`, ni antes de una palabra `End` o de un `Until`.

La orden `Write` aparece algo más a la derecha que el resto. Esto se llama escritura indentada, y consiste en escribir a la misma altura todos los comandos que se encuentran a un mismo nivel, algo más a la derecha los que están en un nivel inferior, y así sucesivamente, buscando mayor legibilidad.

La sentencia `Readln`, en este caso, nos sirve para que la ejecución del programa tenga una pausa y uno pueda ver lo que apareció en pantalla, hasta que se presione la tecla `<ENTER>`.

2. Sintaxis de Pascal

En esta sección se muestra lo esencial de la sintaxis de Pascal.

2.1. Uso de mayúsculas

La sintaxis de Pascal (a diferencia de la de Haskell) no distingue entre mayúsculas y minúsculas. Es decir, no es *case sensitive*. Por lo tanto, interpreta Minombre, MiNombre, minombre, miNombre y MINOMBRE como equivalentes.

2.2. Comentarios

En Pascal, los comentarios se encierran entre llaves o paréntesis acompañados por un asterisco. Para comentarios cortos también se utiliza //, el cual se extiende hasta el final de la línea. Por ejemplo:

```
{este es un comentario}
(* este es otro comentario *)
// este es un comentario hasta el final de la línea
```

2.3. Tipos de datos

Los tipos de datos más usados en Pascal, son los siguientes:

Integer : Es un número entero con signo, que puede valer desde -32768 hasta 32767. Ocupa 2 bytes de memoria.

Byte : Es un número entero, que puede valer entre 0 y 255. El espacio que ocupa en memoria es el de 1 byte, como su nombre indica.

Char : Representa a un caracter (letra, número o símbolo). Ocupa 1 byte y se escribe entre comillas simples. Ejemplos: é', '8', 'B', '#'.

String : Es una cadena de caracteres. Ocupa 256 bytes. Para cadenas de longitud n se utiliza el tipo String[n].

Real : Es un número real con signo. Puede almacenar números con valores entre 2.9e-39 y 1.7e38. Tendremos 11 o 12 dígitos significativos y ocupan 6 bytes en memoria.

Boolean : Es un valor booleano, puede valer TRUE o FALSE.

También es posible introducir nuevos tipos de datos mediante constructores de tipo. A estos tipos se les puede dar un nombre para usarlos más tarde o se pueden aplicar directamente a una variable. Cuando se le da un nombre a un tipo se lo define en una sección del código específica. Por ejemplo, el siguiente fragmento de código define los tipos Letras, Dias, Arreglo10, Fecha y Matriz.

```
Type
   Letras = 'a'..'z';                          // Tipo de subrango

   Dias = (Lun, Mar, Mie, Jue, Vie, Sab, Dom);  // Enumeración

   Arreglo10 = array [1..10] of Integer;

   Fecha = record
      Dia: Byte;
      Mes: Byte;
      Año: Integer;
   End;

   Matriz = array [0..9] of array [0..9] of Integer;
```

Un *tipo de subrango* define un rango de valores dentro del rango de otro tipo. El tipo `Letras` es un tipo de subrango, ya que comprende un rango de valores del tipo `Char`, los caracteres correspondientes a letras minúsculas.

Los tipos de datos enumeraciones, son definidos en Pascal utilizando el formato de tuplas.

El constructor `array` se utiliza para definir un arreglo. En la definición de un tipo `array` se debe especificar entre corchetes los índices superior e inferior del arreglo, los cuales deben ser constantes, y el tipo de los elementos del mismo.

Los tipos definidos con el constructor `record` definen conjuntos de elementos de distintos tipos. Cada elemento o campo, tiene un nombre, de manera que se pueda acceder a ellos mediante éste. Por ejemplo, el siguiente código declara e inicializa una variable de tipo `Fecha`:

```
Var
  hoy: Fecha;

Begin
  hoy.Año := 2008;
  hoy.Mes := 10;
  hoy.Dia := 15;
```

2.4. Declaraciones de Variables

Las variables se utilizan para guardar datos dentro de un programa. Pascal requiere que todas las variables sean declaradas antes de ser utilizadas. La declaración de una variable tendrá la forma:

```
var
   nombre_variable: Tipo;
```

Por ejemplo, las siguientes son algunas declaraciones:

```
var
   valor: Integer;
```

```
    esCorrecto: Boolean;
    a, b: Char;
```

El primer carácter del nombre de una variable deberá ser siempre una letra. Es conveniente utilizar nombres descriptivos, lo cual hacen al programa más legible.

También pueden declararse variables al principio del código de una función o procedimiento, el alcance de estas variables es el cuerpo de la función o procedimiento donde son declaradas y se denominan *variables locales*.

2.5. Asignaciones

La forma de dar valor a una variable es mediante una asignación, la cual tendrá la siguiente forma:

```
variable := expresión;
```

También se puede dar valor a una variable en el mismo momento en que se declara (inicialización). Por ejemplo:

```
var
    valor: Integer = 10;
    correcto: Boolean = True;
```

Esta técnica de inicialización funciona sólo en variables globales, no en variables locales.

2.6. Constantes

Pascal también permite la definición de constantes, para nombrar valores que no cambian durante la ejecución del programa. Para definir una constante, no en necesario especificar un tipo de dato, sino sólo asignar un valor. Por ejemplo:

```
const
  Longitud = 100;
  Pi = 3.14;
```

2.7. Operaciones básicas

Las siguientes operaciones están predefinidas en Pascal y pueden utilizarse en los programas:

La división (`/`) produce un resultado real independientemente del tipo de sus operandos (por ejemplo 2/4 produce el valor real 2.0), por lo tanto el resultado de una división se debe asignar a una variable real.

2.8. Entrada y salida de datos

Para escribir un texto en pantalla Pascal provee la función `Write`. Todo lo que se desee escribir se indica entre paréntesis. Cuando se trata de un texto que queremos que aparezca tal cual, éste se encierra entre comillas simples.

Operador	Descripción
+, - , /, *	Suma, resta, división fraccionaria y multiplicación
div	División entera
mod	Resto de la división
Sqr	Elevar al cuadrado
Sqrt	Raíz Cuadrada
Ln	Logaritmo natural
Exp	Exponencial
Sin, Cos	Seno y coseno
^	Potenciación

Cuadro 1: Operadores aritméticos

Operador	Descripción
=	Igualdad
<>	Negación
>, <, <=, >=	operadores de orden
And, Or, Not	operadores lógicos $\wedge$, $\vee$ y $\neg$

Cuadro 2: Operadores lógicos y de orden

La función `Writeln`, es similar a `Write` con la única diferencia de que después de visualizar el mensaje, el cursor pasa a la línea siguiente, en vez de quedarse justo después del mensaje escrito.

Para la entrada datos se utiliza la función `Read`, que recoge datos desde el teclado y los almacena en las variables especificadas. La diferencia entre `Read` y `Readln` es que esta última posiciona el cursor en una nueva línea.

En el siguiente ejemplo vemos un programa que suma dos enteros ingresados por el usuario e imprime el resultado en pantalla

```
Program Suma; {Ejemplo de un programa que Suma}
Var
    a,b,resultado:integer;

Begin
    Write ('Ingrese el primer numero:');
    Readln (a);
    Write ('Ingrese el segundo numero:');
    Readln (b);
    resultado:=a+b;
    Writeln ('El Resultado es:',resultado);
End.
```

2.9. La sentencia if-then-else

La sintaxis del operador condicional es la siguiente:

```
If <cond> then
     Begin
          <Bloque de Sentencias>;
     End
Else
     Begin
          <Bloque de Sentencias>;
     End;
```

Donde `<cond>` es una condición que puede ser falsa o verdadera, y `<bloque de sentencias>` es una sola sentencia, o bien varias sentencias encerradas entre un `Begin` y un `End`. El `Begin` y el `End` se usan cuando hay más de una sentencia y `Else` cuando es necesario.

2.10. La sentencia For

La sentencia `For` permite ejecutar un código un número fijo de veces. La sintaxis formal de esta sentencia es:

```
For <vc>:=<vi> To <vf> Do
     <sentencia>;
```

donde `<vc>` es una variable de tipo entero, `<vi>` es un valor inicial para dicha variable y `<vf>` un valor final. El valor inicial se asigna a la variable `<vc>`, y se incrementa en uno cada vez que se ejecuta la sentencia o bloque sentencias, hasta que el valor de la variable exceda el valor final.

Como ejemplo veremos el siguiente fragmento de programa que suma los primeros diez números.

```
var
   k, i: Integer;
begin
   k := 0;
   for i := 1 to 10 do
      k := k + i;
```

2.11. La sentencia While

Para ejecutar el mismo código varias veces, también se puede utilizar la sentencia `While`. Dada una expresión boolana (condición), un bloque se ejecuta hasta que la condición sea falsa.

La sintaxis formal es:

```
While <cond> Do
     <bloque>;
```

Si hay que ejecutar más de una sentencia, se deben de colocar el bloque entre un `Begin` y un `End;`.

2.12. Procedimientos y funciones

Para dividir un programa en bloques pequeños, que puedan ser activados en distintos puntos del programa, Pascal provee una sintaxis que permite encapsular bloques de código de dos tipos: procedimientos (procedure) y funciones (function).

La diferencia entre ellos es que un procedimiento ejecuta una serie de acciones que están relacionadas entre sí, y no devuelve ningún valor, mientras que una función si devuelve valores.

La sintaxis de un procedimiento es la siguiente:

```
Procedure <identificador> (<parámetros>);
Var
    <Variables locales>;
Begin
    <Sentencias>;
End;
```

El siguiente es un ejemplo de un programa que contiene un procedimiento para chequear si un password introducido por el usuario es correcto:

```
Program  Acceso;

Const
    ClaveCorrecta = 'seferino';     // Esta constante es global

Procedure Password;
Var
    clave:String;     // Esta variable es local
Begin
    Writeln ('Introduzca su clave de acceso');
    Readln (clave);
    If clave<>ClaveCorrecta then
        Writeln ('La clave no es correcta!');
End;

Begin
    Password;
    Readln;
End.
```

Dentro de un procedimiento pueden definirse variables y constantes. Éstas se llaman *variables locales* y *constantes locales*, respectivamente, y pueden utilizarse solo dentro del procedimiento. Las variables y constantes definidas al comienzo del programa si pueden utilizarse en cualquier parte del mismo, éstas se denominan *variables globales* y *constantes globales*.

La sintaxis de una función es la siguiente:

```
Procedure <identificador> (<parámetros>): <tipo>;
Var
    <Variables locales>;
Begin
    <Sentencias>;
End;
```

donde **<tipo>** es el tipo del valor devuelto por la función.

Veamos un ejemplo:

```
Program Calculo_productos;
Var
    z: integer;

Function Producto (x1, x2 : integer) : integer;
Begin
    Result:= x1 * x2;
End;

Begin
    z:=Producto(9,3);
    Writeln ('9*3=',z);
    z:=Producto(2,2);
    Writeln ('2*2=',z);
    Readln;
End.
```

El valor devuelto por una función debe guardarse en la variable **Result**. Otra opción es usar el nombre se la función en lugar de **Result**. Por ejemplo, la función producto también podría haberse definido como:

```
Function Producto (x1, x2 : integer) : integer;
Begin
    Producto:= x1 * x2;
End;
```

2.12.1. Parámetros

Los parámetros pasados a una función o un procedimiento pueden ser de dos clases: parámetro por valor o parámetro por referencia. A continuación veremos la diferencia entre estos de tipos de parámetros.

Cuando un parámetro es pasado por valor, se obtiene una copia temporal de la variable usada y dentro de la función o procedimiento se trabaja con esta copia. Por lo tanto la variable no cambia de valor, ya que las operaciones se realizan sobre su copia.

La sintaxis de una declaración de procedimiento que recibe un parámetro por valor es la siguiente:

```
Procedure <identificador> (<variable>: <Tipo>);
```

Veamos un ejemplo:

```
Program PruebaDeParametros;
Var
    dato:integer;

Procedure Modifica(variable:integer);
Begin
    variable:=3;
    Writeln ('Dentro: ', variable);
End;

Begin
    dato:=2;
    Writeln ('Antes: ', dato);
    Modifica (dato);
    Writeln ('Después: ', dato);
End.
```

Si ejecutamos este programa obtemos como resultado:

```
Antes: 2
Dentro: 3
Después: 2
```

Cuando se pasa un parámetros por referencia, los cambios que se realicen a la variable introducida como parámetro, se mantendrán vigentes al terminar el proceso. Su sintaxis es la siguiente:

```
Procedure <identificador> (Var <variable>: <Tipo>);
```

Si en el ejemplo anterior, cambiamos el parámetro del procedimiento para que sea por referencia, es decir cambiamos la línea 5 por:

```
Procedure Modifica(Var variable:integer);
```

el resultado de la ejecución del programa es el siguiente:

```
Antes: 2
Dentro: 3
Después: 3
```

Referencias

[1] M. Delphi: http://www.marcocantu.com/epascal/Spanish/default.htm

[2] http://ar.geocities.com/zonadelprogramador/turbo-pascal.htm