# CAPITULO 3.- ESTATUTOS DE CONTROL

Hasta ahora se han cubierto los diferentes tipos de datos que Pascal ofrece. Los datos son representaciones de conceptos que deseamos que un programa manipule. Este punto se refiere a cómo Pascal manipula los datos.

Como ya se explicó, una variable es una localidad de memoria manejada por el compilador. Su tamaño y su manejo dependen del tipo de ella.

El operador fundamental en Pascal es el de asignación, cuyo símbolo es ":=", con el cual probablemente usted ya esté familiarizado. El operador de asignación es (generalmente) la forma como los valores son puestos en las variables. Considere este estatuto de simple asignación:

```
I := 17;
```

Un valor del lado derecho del operador de asignación se asigna a la variable que está del lado izquierdo. En un estatuto de asignación, siempre hay una variable del lado izquierdo del operador. Del lado derecho puede haber una constante, una variable o una expresión.

Una expresión en Pascal es una combinación de datos y operadores que dan como resultado un valor. Los datos pueden ser constantes o variables. Los operadores son símbolos especiales que desarrollan alguna acción involucrando el valor o valores que se le han dado. Estos valores se llaman "operandos" de los operadores.

El más simple y familiar ejemplo de expresión es el aritmético:

```
17+3
```

El operador de adición desarrolla una operación de suma con los dos operandos: 17 y 3. El valor de la expresión es 20.

## OPERADORES RELACIONALES

Los operadores relacionales se usan para construir expresiones booleanas, que son las más ampliamente usadas en Pascal. Todos los estatutos de lazos y brincos en Pascal dependen de las expresiones booleanas.

La siguiente tabla resume los operadores relacionales implementados en TURBO Pascal. Todos los resultados son booleanos.



## OPEARADORES RELACIONALES EN Turbo Pascal

| Operador | Símbolo | Tipo de Operando |
|---|---|---|
| Igual | = | escalar, string, apuntador, registro |
| No igual | < > | escalar, string, apuntador, registro |
| Menor que | < | escalar, string |
| Mayor que | > | escalar, string |
| Menor o igual | < = | escalar, string |
| Mayor o igual | > = | escalar, string |
| Es miembro | IN | conjunto, miembros de conjuntos |
| El izq. está en el der. | < = | conjuntos |
| El der. está en el izq. | > = | conjuntos |
| Negación | NOT | booleanos |
| Conjunción | AND | booleanos |
| Disjunción | OR | booleanos |
| EXOR | XOR | booleanos |

**IGUALDAD**

Si dos valores se comparan y son iguales, la expresión que se evalúa es verdadera (True). En general, TURBO Pascal considera dos valores iguales comparando bit por bit. Esto es cierto para comparaciones de tipos iguales.

La excepción son las comparaciones hechas entre valores numéricos de diferentes tipos. TURBO Pascal permite comparaciones entre enteros, reales y bytes con libertad, pero deben hacerse con mucho ciudado.

Los conjuntos son considerados iguales si tienen los mismos elementos. Dos apuntadores se consideran iguales si ambos apuntan a NIL. Dos registros son iguales si ambos son del mismo tipo y todos sus campos son bit por bit idénticos. Dos strings se consideran iguales si tienen la misma longitud lógica y contienen los mismos caracteres.

**NO IGUALDAD**

Las reglas para las no-igualdades son las mismas que las de la igualdad. La única diferencia es que el valor booleano resultante es el contrario.

En general, se puede usar el operador de no igualdad en las mismas situaciones que el de igualdad.

```
17   =   17   { true }
17   <>  17   { false }
42   =   17   { false }
42   >   16   { true }
```

### MAYOR QUE / MENOR QUE

Los operadores MAYOR QUE, MENOR QUE, MAYOR O IGUAL A y MENOR O IGUAL A, añaden una nueva dimensión a las notaciones de comparación. Se asume que los operandos existen en algún orden definido para que la comparación pueda ser hecha.

Esto inmediatamente descalifica a los apuntadores, conjuntos y registros. Decir que un apuntador es más grande que otro no tiene sentido. Lo mismo se aplica a conjuntos y registros. Ordenarlos no tiene sentido lógico, así que los operadores involucrados en el orden no pueden usarse.

Los tipos CHAR y BYTE están limitados a 256 posibles valores, y ambos tienen un orden implícito. El tipo CHAR está ordenado por el juego de caracteres ASCII, lo que convierte estas expresiones en verdaderas:

```
'A'   <   'B'
'a'   >   'A'
'@'   < >  '['
```

Los tipos enumerados están limitados a no más de 255 valores diferentes. El orden en que se declararon son los valores de prioridad que se les asigna.

```
TYPE
Espectro = (Rojo, Naranja, Amarillo, Verde, Azul,Indigo, Violeta);
```

Este orden hace que las siguientes expresiones se evalúen como verdaderas:

```
Rojo   <   Verde
Azul   >   Amarillo
Indigo < >  Violeta
```

### NOT, AND, OR, XOR

Estos cuatro operadores trabajan solamente con operadores Booleanos. No comprueban que exista una relación entre dos operandos, sino que combinan los operandos según las reglas del álgebra de Boole, para producir un nuevo valor y evaluar la expresión.

El más sencillo de ellos es el operador NOT. El operando debe seguir del operador. NOT niega el valor del operando.

```
NOT False          { esta expresión es veradera }
NOT True           { esta expresión es falsa }
NOT (6 < >I)       { verdadera si I = 6 }
NOT (J = K)        { verdadera si J < > K }
```



## OPERADORES ARITMETICOS

Los números se manipulan con los operadores aritméticos, los cuales, junto con las variables numéricas, forman las expresiones. El único operador que en Turbo Pascal no existe, es el de exponenciación. La siguiente tabla muestra los operadores aritméticos válidos en Pascal:

| | Operador | Tipo operando | Resultado |
|---|---|---|---|
| Suma | + | entero, real, byte | igual |
| Inversión de signo | - | entero, real | igual |
| Resta | - | entero, real, byte | igual |
| Multiplicación | * | entero, real, byte | igual |
| División de enteros | DIV | entero, byte | igual |
| División de reales | / | | real |
| Módulo | MOD | | igual |

La lista de operandos son los tipos que el operador puede manejar. El compilador da la libertad de mezclar los tipos de las variables numéricas que están en una expresión dada. En otras palabras, se pueden multiplicar bytes por enteros, sumar reales y bytes, etc. Las siguientes son expresiones válidas en Pascal:

```
VAR
  I,J,K   : integer;
  R,S,T   : real;
  A,B,C   : byte;

  I * B              { entero multiplicado por byte }
  R + J              { entero sumado con un real }
  C + (R * I)        { etc }
  J * (A / S)
```

La columna del tipo resultante indica el tipo de dato que se obtiene al ser evaluada la expresión. Los tipos numéricos deben observar estas reglas:

1) Cualquier expresión que incluya un número real, solamente puede ser asignada a una variable real.

2) Las expresiones que contengan la división de reales (/), pueden ser asignadas a variables reales.

Si no se siguen estas reglas, el compilador genera el error #44: Inconcordancia de tipo.

### LA INVERSION DE SIGNO

El signo de inversión es un operador unario, esto es, toma solamente un operando. Cambia el signo del operando, haciendo positiva una cantidad negativa, y viceversa. Nótese que la inversión de signo no se puede usar con el tipo byte, pues éste es absoluto (sin signo). Un byte nunca es considerado negativo.

## DIVISION

Hay tres tipos de división en Pascal: Una para los números reales y las otras dos para los enteros y bytes. La división real (/) puede llevar operandos de cualquier tipo, pero siempre produce valores reales. Intentar asignar una división para reales a un tipo no-real genera el error #44, aun si todos los operandos son del mismo tipo:

```
VAR
   I,J,K : Integer;

I := J / K;              { este estatuto no compila! }
```

Cuando se realiza una división, resultan dos números: Uno representa la parte entera de la operación y el otro la parte fraccionaria. En Pascal, la división entera se separa en dos funciones independientes: DIV, que produce la parte entera de la división; y MOD, que produce la parte fraccionaria:

```
J := 17;
K := 3;
I := J DIV K;            { se asigna 5 a I }
I := J MOD K;            { se asigna 2 a I }
```

```
program ParesyNones;
var
  N, i, Numero,
  NumPares, NumNones : integer;
begin
  NumNones := 0;
  NumPares := 0;
  write ('Cuantos numeros se van a analizar? ');
  readln (N);
  writeln;
  for i:=1 to N do
  begin
     write ('Numero: ');
     readln (Numero);
     write ('El ',Numero,' es un numero ');
     if (Numero mod 2) = 0 then
     begin
        inc(NumPares);
        writeln ('PAR.')
     end
     else
     begin
         inc(NumNones);
         writeln ('NON.')
     end
  end;
  writeln;
  writeln ('Pares=',NumPares,' Nones=',NumNones);
end.
```



## LOS OPERADORES DE CONJUNTOS

Los operadores de conjuntos se usan para manipular los valores de tipo SET. La tabla resume los operadores de conjuntos implementados en TURBO Pascal. Todos ellos toman operandos y producen valores de tipo SET.

| Operador | Símbolo |
|---|---|
| Set union | + |
| Set diference | - |
| Set intersection | * |

### SET UNION

El resultado de la unión de dos conjuntos es el conjunto en donde están reunidos todos los miembros de los dos conjuntos. El símbolo que lo representa es el de suma (+). Ejemplo:

```
VAR
   SetX, SetY, SetZ : set of char;
   SetX := ['Y', 'y', 'M', 'm'];
   SetY := ['N', 'n', 'M', 'm'];
   SetZ := SetX + SetY;
```

La variable SetZ contiene los caracteres 'Y', 'y', 'N', 'n', 'M', 'm'. Aunque 'M' y 'm' se repiten en los dos conjuntos, en el resultante están solo una vez.

### SET DIFERENCE

La diferencia entre dos conjuntos es aquella que contiene solamente los miembros que NO son comunes. Este operador es lo contrario de SET INTERSECTION, que se verá más adelante. Su símbolo es el de resta (-). Ejemplo:

```
VAR
   SetX, SetY, SetZ : set of char;
   SetX := ['Y', 'y', 'M', 'm'];
   SetY := ['N', 'n', 'M', 'm'];
   SetZ := SetX - SetY;
```

La variable SetZ contiene 'Y', 'y', 'N', 'n', que son los elementos no comunes.

### SET INTERSECTION

La intersección de dos conjuntos es la que contiene como miembros aquellos que están en ambos conjuntos. Es lo contrario que SET DIFERENCE y su símbolo es el asterisco (*). Ejemplo:

```
VAR
   SetX, SetY, SetZ : set of char;
   SetX := ['Y', 'y', 'M', 'm'];
   SetY := ['N', 'n', 'M', 'm'];
   SetZ := SetX * SetY;
```

La variable SetZ contiene 'M' y 'm', que son los únicos miembros comunes en ambos conjuntos.

Controlar el flujo del programa es la más importante faceta de un programa de computación. Estatutos condicionales que cambien el sentido de flujo de control, lazos que repitan líneas un número determinado de veces y estatutos que tomen un curso dado de acuerdo a un valor de control, incrementan la utilidad de los programas.

Pascal permite que el programador direccione el flujo de control de una manera estructurada y racional, para que los programas sean fáciles de escribir, leer y cambiar cuando sea necesario. Por esta razón, incursionar en un programa de Pascal puede ser en un principio difícil.

La sintaxis sugiere por sí misma que los programas empiecen en el inicio de la página y progresen hacia abajo, terminando abajo cuando el trabajo está concluido. Múltiples puntos de entradas, salidas y brincos incondicionales vía el GOTO provocan más problemas que los beneficios que se obtienen.

## BLOQUES BEGIN/END

En Pascal, frecuentemente es necesario que un grupo o número de estatutos sean tratados como un solo bloque o como un solo estatuto. Esto se logra con las palabras reservadas BEGIN y END. Un grupo de estatutos entre un BEGIN y un END se convierten en un estatuto compuesto. Los cuerpos de los programas, procedimientos y funciones, son estatutos compuestos:

```
PROGRAM Raiz;
      VAR
      R, S : real;
  BEGIN
      writeln ('Calculo de una raiz cuadrada');
      writeln;
      writeln ('Dame un numero:');
      readln (R);
      S := sqrt (R);
      writeln ('La raiz de',R:7:7,' es ',S:7:7,'.');
  END.
```

Los estatutos compuestos pueden ser usados en cualquier lugar como un estatuto simple. Todo lo que esté entre un BEGIN y un END es tratado por el compilador como un estatuto sencillo.



```
program Seno;
uses
   Crt;
const
   Eje = 15;
   Escala = 10;
var
   Angulo : real;
   radian  : integer;
   i       : byte;
begin
   clrscr;
   write ('< GRAFICA DE UNA FUNCION SENOIDAL >':60);
   Angulo := 0;

   for i:=1 to 24 do { EJE Y }
   begin
      gotoxy (1,i);
      write ('|')
   end;

   for i:=1 to 80 do { EJE X }
   begin
      gotoxy (i,Eje);
      write ('-')
   end;

   for i:=1 to 80 do
   begin
      radian := round(Eje-Escala*sin(Angulo));
      gotoxy (i,radian);
      write ('*');
      Angulo := Angulo + 0.1
   end
end.
```

```
program Coseno;
uses
   Crt;
const
   Eje = 15;
   Escala = 10;
var
   Angulo : real;
   radian  : integer;
   i       : byte;
begin
   clrscr;
   write ('< GRAFICA DE UNA FUNCION COSENOIDAL >':60);
   Angulo := 0;
```