PARAMETROS FORMALES Y ACTUALES

Pasar datos a los procedimientos y a las funciones puede hacerse de dos formas:

1) usando variables globales para que cualquier procedimiento y/o función pueda leerse.

2) a través de una lista de parámetros para cada procedimiento y/o función.

La primera es una muy mala idea, por no mencionar las limitantes que el hardware pueda imponer. Es una buena práctica que un procedimiento tenga solamente que manejar una lista de valores.

Los parámetros definidos en la declaración del procedimiento se llaman "parámetros formales". Los valores que están presentes en una lista de parámetros de la invocación en particular se llaman "parámetros actuales". Si los parámetros son pasados por referencia, los parámetros actuales deberán tener variables de tipos iguales para guardar en ellos los parámetros formales.

TRANSFERENCIA DE DATOS POR VALOR Y REFERENCIA

PARAMETROS PASADOS POR VALOR

Un parámetro pasado por valor es justamente eso: Un valor copiado del parámetro actual al parámetro formal. El movimiento del valor hacia el procedimiento es en un sentido. Nada regresa ni puede ser usado por el programa que lo llamó.

Hay ventajas poderosas en el movimiento de valores dentro del procedimiento. El procedimiento puede guardar, modificar y mutilar el parámetro en la medida que las necesidades del procedimiento lo exijan, y no existen efectos colaterales fuera del procedimiento. La copia del parámetro actual es una copia privada, estrictamente local para el mismo procedimiento.

La forma de declarar pase de valores por valor es:
PROCEDURE (variable : tipo);

PARAMETROS PASADOS POR REFERENCIA

En muchas ocasiones el punto central de pasar un parámetro a un procedimiento o función es tener ese mismo, pero modificado para su uso. Para que un procedimiento pueda modificar un parámetro y regresarlo así, éste debe pasarse por referencia.

A diferencia de los parámetros pasados por valor, los valores pasados por referencia NO pueden ser literales, constantes o expresiones. Los valores de constantes y literales, por definición, no pueden ser cambiados.

Para pasar un valor por referencia, un parámetro actual debe ser una variable del mismo tipo que el parámetro formal. No hay tipos compatibles; deben ser exactamente del mismo tipo. La única excepción a esta regla son los tipos string, los cuales pueden ser definidos por una longitud de hasta 255.

La forma para definir valores pasados por referencia es:
PROCEDURE (VAR variable : tipo);

CAPITULO 5 ARREGLOS (Arrays)

Las variables que se han venido manejando hasta ahora son todas tipos simples. Iniciaremos ahora el estudio de algunos tipos estructurados que posee Pascal. Se puede decir que un tipo estructurado difiere de un tipo simple en que el tipo estructurado posee mas de un componente. Y cada componente del tipo estructurado es una variable, la cual a su vez puede ser de tipo estructurado o simple. Lo importante con este tipo de variables es la forma en como se tiene acceso a cada uno de sus componentes.

ARREGLO DE UNA DIMENSION (VECTORES)

Un arreglo no es mas que una colección de variables, en donde todas son del mismo tipo. Se puede decir que una línea de texto es un arreglo de caracteres; un vector se puede representar como un arreglo de números; una matriz se puede pensar como un arreglo, cuyos elementos son a su vez vectores.

Un arreglo se declara en términos de un índice y un tipo base, o sea, el tipo de cada uno de sus componentes.
Visto de otra forma, podemos decir que un arreglo nos va a permitir:

- Denotar un grupo de variables mediante un solo identificador.
- Poder distinguir un elemento en particular del grupo mediante un índice. De esta forma podemos referenciar el iésimo componente del arreglo.

El índice podrá ser cualquier tipo ordinal. Así, por ejemplo, una declaración del tipo array podrá quedar de la siguiente forma:

```
Numeros : array[1..100] of real;
```

Esta declaración permite disponer de un arreglo de 100 elementos. En este caso el índice es del tipo entero y el tipo de cada elemento es real.

Según se mencionó antes, el índice no es necesariamente numérico; puede ser cualquier tipo ordinal:

```
Grupo : array['A'..'G'] of integer;
```

Un elemento individual de un arreglo variable se denotará como una variable indexada, escribiendo el nombre (identificador) del arreglo seguido por el correspondiente valor del índice, enmarcado entre corchetes cuadrados. Ejemplo:

```
Grupo['E']
```

está haciendo referencia a aquél elemento del arreglo Grupo cuyo subíndice es 'E', el cual tiene o puede tener almacenado un valor del tipo integer.

Con esta variable (Grupo['E']) es posible hacer todas las operaciones en las cuales el operando sea de tipo entero:

```
Acumulado := Grupo['A'] + Grupo['E'];

Prom      := Grupo['B'] + Grupo['C'] / 2;
```

Como lo deja entrever la definición, el índice dentro de una variable indexada no necesita ser una constante, bien puede ser una variable, e incluso una expresión, contal de que el tipo de la expresión y/o de la variable concuerde con el tipo del índice. De forma que si por ejemplo tenemos la siguiente declaración:

```
Numeros : array[1..100] of real;
```

podemos denotar el i-esimo elemento de la variable Numeros con una variable o una expresión de tipo entero, con tal de que su valor esté en el rango permitido para el índice (1-100), de forma que podrá escribirse:

```
Numeros[j];

Numeros[i+j];

Numeros[trunc(x*4.2/12) div 6];
```

Si el valor de j, i+j, trunc(x*4.2/12) div 6 cae fuera del rango 1-100, se estará tratando de hacer referencia hacia un elemento que no existe y el compilador marcará error.

En ocasiones necesitamos guardar durante la ejecución de un programa el contenido de un arreglo en otro arreglo. Turbo Pascal permite hacer lo siguiente:

```
Numeros1 := Numeros2;
```

lo cual sería equivalente a hacer lo siguiente:

```
Numero1[1] := Numeros2[1];

Numero1[2] := Numeros2[2];

Numero1[3] := Numeros2[3];

Numero1[100] := Numeros2[100];
```



```
program Orden1;
uses
  Crt;
var
  Temporal,
  N, i, j  : integer;
  Numero   : array[1..50] of integer;
begin
  clrscr;
  writeln ('** CAPTURA DE NUMEROS ENTEROS **');
  write ('Cuantos numeros? ');
  readln (N);
  for i:=1 to N do
  begin
     write ('Numero ',i:2,': ');
     readln (Numero[i])
  end;

  writeln;
  writeln ('** IMPRESION DE NUMEROS COMO SE CAPTURARON **');
  for i:=1 to N do writeln ('Numero ',i:2,' = ',Numero[i]);


  for i:=1 to (N-1) do
     for j:=(i+1) to N do
          if (Numero[i]Numero[j]) then
          begin
             Temporal  := Numero[i];
             Numero[i] := Numero[j];
             Numero[j] := Temporal;
          end;
  writeln;
  writeln ('** IMPRESION DE NUMEROS EN ORDEN **');
  for i:=1 to N do writeln ('Numero ',i:2,' = ',Numero[i])
end.
```

```
{
   Hay que ser cuidadoso con las mayusculas/minusculas, pues eso puede

   alterar el orden de los nombres. Se recomienda probar primero con

   puras mayusculas.
}


program Orden2;
uses
   Crt;
var
   N, i, j  : integer;
   Temporal : string;
   Nombre   : array[1..50] of string;
begin
   clrscr;
   writeln ('** CAPTURA DE NOMBRES **');
   write ('Cuantos nombres?');
   readln (N);
   for i:=1 to N do
   begin
      write ('Nombre ',i:2,': ');
      readln (Nombre[i])
   end;
   writeln;
   writeln ('** IMPRESION DE NOMBRES COMO SE CAPTURARON **');
   for i:=1 to N do writeln ('Nonbre ',i:2,' = ',Nombre[i]);

   for i:=1 to (N-1) do
      for j:=(i+1) to N do
         if (Nombre[i]Nombre[j]) then
         begin
            Temporal  := Nombre[i];
             Nombre[i] := Nombre[j];
             Nombre[j] := Temporal;
         end;
   writeln;
   writeln ('** IMPRESION DE NOMBRES EN ORDEN **');
   for i:=1 to N do writeln ('Nombre ',i:2,' = ',Nombre[i])
end.
```


ARREGLOS DE DOS DIMENSIONES (MATRICES)

En los ejemplos de los arreglos manejados hasta ahora sólo se han hecho uso de arreglos de una dimensión (sólo usan un subíndice). Sin embargo Pascal permite arreglos de más de una dimensión. Por ejemplo, cada una de las páginas de este escrito se puede considerar como un arreglo de renglones, en donde cada renglón consta de 65 caracteres, o sea, que se puede definir la variable página de la siguiente manera:

```
pagina : array[1..52,1..65] of char;
```

o bien:

```
pagina : array[1..52] of array[1..65] of char;
```

así, para el i-esimo caracter de la j-esima línea de PAGINA se puede referenciar:

```
pagina[i,j]
```

o bien:

```
pagina [i][j]
```

Cualquier arreglo definido con más de una dimensión se considera un arreglo multidimensional, aunque es muy raro que se exceda de tres dimensiones. Los arreglos bidimensionales, llamados matrices, son bastante comunes, especialmente en estadística e ingeniería. Por ejemplo, cuando se está midiendo la conductividad de un metal, una matriz se adapta perfectamente para llevar el registro de la temperatura con respecto a la conductividad medida:

```
Temp_Conductividad : array[1..200,1..2] of real;
```

La mejor manera de visualizar una matriz es pensar en una tabla con filas y columnas: el primer rango significa las filas (1..200) y el segundo las columnas (1..2).

Las asignaciones de valores pueden ser:

```
Temp_Conductividad [1,1] := -99.34;

Temp_Conductividad [1,2] := 12.3;
```

	1	2
	Temperatura	Conductividad
1	-99.34	12.3
2	-97.76	12.2
3	-96.01	11.9
.	.	.
.	.	.
200	99.01	2.9

Concentrándose en el manejo de las matrices, y dejando de lado el significado de cada localidad de ella, es posible manejarlas como cualquier sistema de ecuaciones y obtener sus resultados tales como el determinante, los elementos de las diagonales, resolver el sistema, etc. A continuación se presenta un programa que maneja los métodos para la solución de sistemas por medio de los algoritmos de Montante y Gauss-Jordan. Obsérvese el tratamiento tanto para la captura como la impresión de la matriz.

Los diagramas de flujo de los algoritmos se dejan de lado para concentrar la atención el el código.


```
Program Test;
uses
  Crt;
type
  rango  = 1..20;
  tipo   = real;
  Matriz = array[rango,rango] of tipo;


procedure CapturaMatriz (var M : Matriz; var Orden : rango);
var
  i,j : rango;
begin
  writeln; writeln;
  writeln ('** CAPTURA DE MATRIZ **');
  writeln;
  write ('Orden de la matriz: ');
  readln (Orden);
  for i:=1 to Orden do
      for j:=1 to Orden do
          begin
              write ('Elemento (',i,',',j,'): ');
              readln (M[i,j])
          end;
  writeln
end; {-------------------------------------------------}


procedure ImprimeMatriz (M : Matriz; Fil,Col : rango);
var
  i,j : rango;
begin
  writeln; writeln;
  writeln ('** MATRIZ **');
  writeln;
  for i:=1 to Fil do
  begin
      for j:=1 to Col do write (M[i,j]:10:3);
      writeln;
  end;
  writeln
end; {-------------------------------------------------}


procedure ConcatenaMatrizIdentidad (var M : Matriz; Orden : rango);
var
  i,j : rango;
begin
  for i:=Orden+1 to 2*Orden do
      for j:=1 to Orden do
          if (i-Orden=j) then M[j,i]:=1 else M[j,i]:=0;
end; {-------------------------------------------------}
```

```
procedure CambiaFilas (var M : Matriz; Orden,Fila,Pivote : rango);
var
   Factor : tipo;
   i      : rango;
begin
   Factor := -M[Fila,Pivote];
   for i:=1 to Orden do
      M[Fila,i] := Factor * M[Pivote,i] + M[Fila,i];
end; {-------------------------------------------------------}


procedure TransformacionElemental (var M : Matriz; Fil, Col : rango);
var
   Diagonal,
   ColCero  : rango;
   Factor   : tipo;
begin
   for Diagonal:=1 to Fil do
      if (M[Diagonal,Diagonal]0) then
      begin
         Factor := M[Diagonal,Diagonal];
         for ColCero:=1 to Col do
            M[Diagonal,ColCero] := M[Diagonal,ColCero]/Factor;
         ImprimeMatriz (M,Fil,Col);
         readln;
         for ColCero:=1 to Fil do
            if (ColCeroDiagonal) then
               CambiaFilas(M,Col,ColCero,Diagonal);
      end;
end; {-------------------------------------------------------}


var
   UnaMatriz : Matriz;
   N         : rango;
begin { main }
   CapturaMatriz (UnaMatriz,N);
   ConcatenaMatrizIdentidad (UnaMatriz,N);
   ImprimeMatriz (UnaMatriz,N,2*N);
   TransformacionElemental (UnaMatriz,N,N*2);
   ImprimeMatriz (UnaMatriz,N,2*N);
   writeln ( 'Presione <Enter> para continuar ...');
end..
```



```
{
        MANEJO DE OPERACIONES ELEMENTALES CON MATRICES
}
program ManejoDeMatrices;
uses
   Crt;
const
   MAX = 20;
type
   rango  = 1..MAX;
   tipo   = real;
   Matriz = array [rango,rango] of tipo;
var
   MatrizA,
   MatrizB,
   MatrizC  : Matriz;

   Fil_A, Fil_B, Fil_C,
   Col_A, Col_B, Col_C  : rango;

   DeterminanteA,
   DeterminanteB,
   DeterminanteC : real;


procedure Captura (var A : Matriz; var Fil,Col : rango);
var
   i,j : rango;
begin
   clrscr;
   write ('Numero de filas    : '); readln (Fil);
   write ('Numero de Columnas : '); readln (Col);
   for i:=1 to Fil do
      for j:=1 to Col do
      begin
         write ('Elemento (',i:2,',',j:2,'): ');
         readln (A[i,j])
      end
end; {-------------------------------------------------------}


procedure ImprimeMatriz (A : Matriz; Fil,Col : rango);
var
   i,j : rango;
begin
   writeln;
   writeln ('** MATRIZ **');
   for i:=1 to Fil do
   begin
      for j:=1 to Col do write (A[i,j]:10:3);
      writeln;
   end;
end; {-------------------------------------------------------}
```

```
function MultiplicaMatriz (    A : Matriz;    Fil_A,    Col_A : rango;
                               B : Matriz;    Fil_B,    Col_B : rango;
                          var R : Matriz; var Fil_R : rango; var Col_R : rango) : boolean;
var
   I,j,k,n : rango;
 begin
   MultiplicaMatriz := false;
   if (Col_A = Fil_B) then
   begin
      N := Col_A;
      MultiplicaMatriz := true;
      Fil_R := Fil_A;
      Col_R := Col_B;
      for i:=1 to Fil_R do
        for j:=1 to Col_R do
        begin
            R[i,j] := 0;
            for k:=1 to N do R[i,j] := A[i,k] * B[k,j] + R[i,j]
        end { for j }
   end { if }
end; {-------------------------------------------------}


procedure ImprimeMatrizAdyacente (A : Matriz; Fil,Col : rango);
var
   i,j : rango;
begin
   writeln;
   writeln ('** MATRIZ **');
   for i:=1 to Fil do
   begin
      for j:=1 to (Col*2) do write (A[i,j]:10:3);
      writeln;
   end;
end; {-------------------------------------------------}


procedure AgregaMatrizIdentidad (var A : Matriz; N : rango);
var
   i,j : rango;
begin
   for i:=1 to N do
      for j:=N+1 to (N*2) do
        if ((i+N)=j) then A[i,j] := 1 else A[i,j] := 0;
end; {-------------------------------------------------}
```



```
procedure Montante (var A : Matriz; var Deter : real; N : rango);
var
   I,j,k,
   M      : rango;
   Factor,
   PivAnt,
   Pivote : tipo;
begin
   AgregaMatrizIdentidad (A,N);
   PivAnt := 1;
   for I:=1 to N do
   begin
      Pivote := A[i,i];
      for k:=1 to N do
        if (ik) then
        begin
            Factor := A[k,i];
            for j:=1 to (N*2) do
              A[k,j]:= (A[k,j]*Pivote - A[i,j]*Factor) / PivAnt;
        end; { if }
      PivAnt := Pivote;
   end; { for }
   Deter := Pivote;
end; {-------------------------------------------------}


function CalculaDeterminante (A : Matriz; N : rango) : real;
var
   DeterAux  : real;
   MatrizAux : Matriz;
begin
   MatrizAux := A;
   Montante (MatrizAux,DeterAux,N);
   CalculaDeterminante := DeterAux;
end; {-------------------------------------------------}



begin
   Captura (MatrizA, Fil_A, Col_A);
   Captura (MatrizB, Fil_B, Col_B);
   ImprimeMatriz (MatrizA, Fil_A, Col_A);
   ImprimeMatriz (MatrizB, Fil_B, Col_B);
   if MultiplicaMatriz(MatrizA, Fil_A, Col_A,
                       MatrizB, Fil_B, Col_B,
                       MatrizC, Fil_C, Col_C) then ImprimeMatriz (MatrizC, Fil_C, Col_C);
   writeln ('Determinante calculado por funcion  : ',CalculaDeterminante(MatrizC,Fil_C):10:3);
   Montante (MatrizC,DeterminanteC,Fil_C);
   writeln ('Determinante calculado por Montante : ',DeterminanteC:10:3);
   ImprimeMatrizAdyacente (MatrizC, Fil_C, Col_C);
end.
```