VISUAL STUDIO 2008

Matrices (Arreglos)
Subrutinas y Funciones
Tipo *String*
Funciones Intrínsecas
Programación por eventos

COLEGIO DE FOMENTO MONTECASTELO

Matrices (Arreglos)

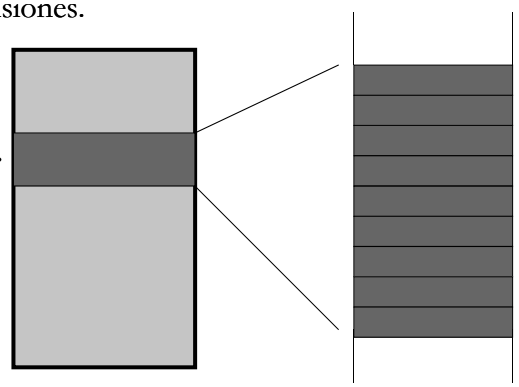
Matriz (*Array*) - I

- Es un conjunto de datos situados de forma contigua en memoria.
- Todos los datos son del mismo tipo y comparten un nombre común.
- Para acceder a una localización determinada se especifica el nombre de la matriz y la posición que ocupa dentro de la misma.
- Las matrices se clasifican en:
  - Matrices de variables
    - Matrices estáticas
    - Matrices dinámicas
  - Matrices de controles

Matriz (*Array*) - II

- Elementos en la declaración de una matriz:
  - Nombre.
  - Tipo de dato.
  - Número de elementos.
  - Cantidad de dimensiones.

Una matriz de **enteros** llamada ***mat***, que posee **10** elementos.



Matriz estática (I)

- Se declara con un número constante de elementos que se define al declarar la variable.
- La forma sintáctica de declaración es:

  Dim *Nombre*([*Dimensión1*, *Dimension2*, …]) As *Tipo* [= {*Valor1*, *Valor2*, …}]

  Nombre: Es un identificador para la variable.

  Dimensión1, *Dimension2*,…: Son las dimensiones que puede tener la matriz, hasta un máximo de 32.

  Tipo: Es uno de los tipos de datos definidos en el lenguaje o por el usuario.

  Valor1, *Valor2*, …: Son los valores de inicialización.

Matriz estática (II)

- Tres formas de declarar una matriz de 10 elementos enteros:

```
Dim MatEnt(10) As Integer
```

```
Dim MatEnt(0 to 9) As Integer
```

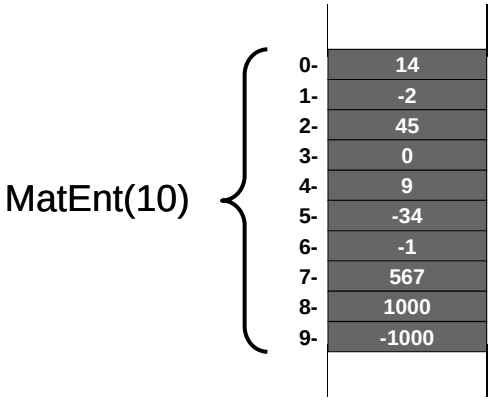
```
Dim MatEnt As Integer() = New Integer(10){}

Dim MatEnt As Integer()
MatEnt = New Integer(10){}
```

Matriz estática (III)

- Al declarar una matriz también se pueden inicializar sus valores:

```
Dim MatEnt(10) As Integer = {14,-2,45,0,9,
-34,-1,567,1000,-1000}
```



Matriz estática (IV)

- A cada elemento se accede mediante el número que identifica su posición, llamado **índice**.
- Cada posición de la matriz se comporta como una variable independiente del mismo tipo.
- Ejemplo:

```
Const N As Byte = 50
Dim mat1(N), mat2(N), i As Byte

For i = 0 To N - 1
     mat1(i) = i
     mat2(N – i - 1) = mat1(i)
Next
```

Funciones *LBound* y *UBound*

- Son funciones que se han mantenido desde la versión 6 de Visual Basic.
- Devuelven los valores de los límites de la matriz.
  - LBound(*Nombre_Matriz*): Devuelve el límite inferior de la matriz.
  - UBound(*Nombre_Matriz*): Devuelve la cantidad de elementos de la matriz.
- En las versiones actuales *LBound* siempre devuelve 0.
- Ejemplo:

```
Dim mat1(100) As Byte

'LBound(mat1) devuelve 0
'UBound(mat1) devuelve 100
```


Matriz dinámica (I)

- Se declara sin especificar la cantidad de elementos.
- Se utiliza la instrucción *Redim* para asignarle un espacio en memoria.
- Además de *ReDim* se puede utilizar *Preserve* para no cambiar el contenido de la matriz.

```
Dim F As Integer
Dim Matriz_A () As Single
… …
F = 1024
… …
ReDim Matriz_A(F)
… …
```

Matriz dinámica (II)

- Ejemplo:

```
Dim Dias As Short, i
Dim Temperatura() As Single

Dias = Console.ReadLine()
ReDim Temperatura(Dias)
For i = 0 To UBound(Temperatura) - 1
   Temperatura(i) = Console.ReadLine()
Next
```


Matriz dinámica (III)

- Ejemplo:

```
Dim Dias As Short, i
Dim Temperatura() As Single

Dias = Console.ReadLine()
ReDim Temperatura(Dias)
For i = 0 To UBound(Temperatura) - 1
    Temperatura(i) = Console.ReadLine()
Next
'Se agregan más elementos preservando los anteriores
ReDim Preserve Temperatura(Dias + 10)
For i = 10 To UBound(Temperatura) - 1
    Temperatura(i) = Console.ReadLine()
Next
```

Matriz dinámica (IV)

- Ejemplo:

```
Dim i As Uinteger = 0
Dim str() As Char

‘Se asigna espacio en la matriz y se agregan los
‘elementos en la medida que se leen. Se preservan los
‘anteriores
Do
    c = Convert.ToChar(Console.ReadLine())
    i += 1
    ReDim Preserve str(i)
    str(i) = c
Loop Until c = ‘Z’
```


Procedimientos y Funciones

Organización de las aplicaciones

- Los proyectos de Visual Basic se componen de módulos, entre otros tipos de archivos.
- Los módulos incluyen los bloques de instrucciones que componen un programa.
- Estos bloques de instrucciones se organizan en piezas más pequeñas llamadas procedimientos.
- Los procedimientos representan tareas que un programa puede ejecutar una o más veces.
- En las aplicaciones de consola se genera el procedimiento principal (*Main*) por defecto.

Procedimientos (I)

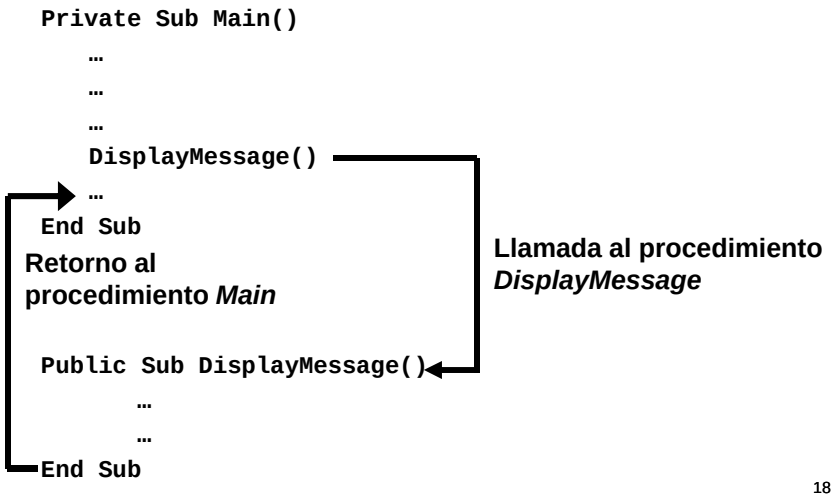
- Un procedimiento es un bloque de instrucciones incluido entre una instrucción de inicio de procedimiento y una instrucción *End*.
- Las instrucciones de inicio de procedimiento pueden ser: *Function*, *Sub*, *Operator*, *Get* y *Set*.
- Todas las instrucciones ejecutables deben incluirse en algún procedimiento dentro de la aplicación.
- Desde el punto de vista algorítmico:
  - Son la base de la programación modular.
  - Deben ser diseñados para que funcionen de forma independiente.
  - Permiten la reutilización del código.

Procedimientos (II)

- Desde el punto de vista funcional:
  - Se invocan desde otras partes del código, lo cual se conoce como "llamada a un procedimiento".
  - La "llamada" es una instrucción o una expresión contenida en una instrucción, que hace referencia al procedimiento **por su nombre** y le transfiere el control de la ejecución.
  - Cuando el procedimiento finaliza la ejecución, devuelve el control a la instrucción siguiente a la de llamada.

Procedimientos (III)

- Desde el punto de vista funcional (gráficamente):



Tipos de procedimientos

- En Visual Basic hay dos tipos de procedimientos:
  - Procedimientos ***Sub***
  - Procedimientos ***Function***

Procedimientos *Sub*

- Un procedimiento *Sub* es una serie de instrucciones delimitadas por las instrucciones *Sub*, al inicio, y *End Sub*, al final.
- El procedimiento *Sub* ejecuta la tarea y devuelve el control al código de llamada, pero no le devuelve un valor.
- Las instrucciones se ejecutan desde la primera instrucción que sigue al encabezamiento hasta la primera instrucción *End Sub*, *Exit Sub* o *Return* que se encuentre.

Sintaxis de un procedimiento *Sub*

[*nivel de acceso*] Sub *nombre*[(*parámetro1, parámetro2,…*)]
instrucción(*es*)
[Exit Sub]
[*instrucción*(*es*)]
[Return]
[*instrucción*(*es*)]
End Sub

- ***nivel de acceso***: Puede ser *Public*, *Protected*, *Friend*, *ProtectedFriend* o *Private*. Por omisión es *Public*.
- ***nombre***: Un identificador.
- ***parámetros***: Es una o varias variables, separadas por coma, que se corresponden con los argumentos que son pasados cuando es invocado el procedimiento. Constituyen los datos de entrada al procedimiento.

Ejemplo de un procedimiento *Sub*

```
Public Sub Factorial(ByVal n As Integer)
   Dim f As Long

   If n = 0 Then
        f = 1
   Else
        f = 1
        While(n > 0)
           f *= n
           n -= 1
        End While
   End If
   Console.Write("Factorial = {0}",f)
End Sub
```

- La llamada puede ser:

…
`Factorial(10)`
…

Ejemplo de un procedimiento *Sub*

- Ejemplos:
  - Procedimiento Factorial.
  - Cálculo del factorial de los *n* primeros números enteros (No modular).
  - Cálculo del factorial de los *n* primeros números enteros (Modular).

Procedimientos *Function*

- Un procedimiento *Function* es idéntico en definición y funcionamiento que un procedimiento *Sub*.
- El procedimiento *Function* devuelve un valor en su nombre.

Sintaxis de un procedimiento *Function*

[*nivel de acceso*] Function *nombre*[(*parámetro1, parámetro2,…*)] As *tipo*
    instrucción(*es*)
    [Exit Sub]
    [*instrucción*(*es*)]
    [Return]
    [*instrucción*(*es*)]
End Sub

- ***tipo***: Uno de los tipos de datos definidos en el lenguaje o por el usuario.

Ejemplo de un procedimiento *Function*

```
Public Function Factorial(ByVal n As Integer) As Long
   Dim f AS Long

   If n = 0 Then
        f = 1
   Else
        f = 1
        While(n > 0)
           f *= n
           n -= 1
        End While
   End If
   Return f
End Sub
```

- La llamada puede ser:

```
Dim k As Long
…
k = Factorial(10)
…
```

Ejemplo

Parámetros

- Los parámetros constituyen la definición de los datos que se le pasan a los procedimientos.
- Permiten a los procedimientos funcionar con datos diferentes cada vez que se invocan.
- Constituyen variables locales al procedimiento.
- Hay dos formas de pasar parámetros:
  - ***ByVal***: El procedimiento no puede modificar el valor de la variable original. Es la forma predeterminada.
  - ***ByRef***: El procedimiento puede modificar el valor de la variable original.

Parámetros por valor y por referencia

- Un parámetro por valor (*ByVal*) contiene una copia del argumento correspondiente.
- Si la subrutina cambia el contenido de un parámetro por valor, el contenido del argumento correspondiente no se afecta.
- Un parámetro por referencia (*ByRef*) contiene la dirección del argumento correspondiente, de forma tal que el parámetro y el argumento se refieren a la misma dirección de memoria
- Los parámetros por referencia son usados para alterar el contenido de los argumentos correspondientes

Ejemplo de procedimientos con parámetros (I)

```
Private Sub CalcAverage(ByVal n1 As Single, _
                        ByVal n2 As Single, _
                        ByVal n3 As Single, _
                        ByRef avg As Single)

Private Function Factorial(ByVal n As Integer)

Private Sub Factorial(ByVal n As Integer, _
                      ByRef f As Long)

Private Sub Swap(ByRef x As Integer, _
                 ByRef y As Integer)
```


Ejemplo de procedimientos con parámetros (II)

```
‘Subrutina para intercambiar el valor de dos variables
Private Sub Swap(ByRef x As Integer, _
                 ByRef y As Integer)
  Dim temp As Integer

  temp = x
  x = y
  y = temp
End Sub
```

Argumentos (I)

- Los argumentos se especifican en la llamada al procedimiento y corresponde a los valores con los que este trabaja.
- Ejemplo:

```
CalcAverage(first, second, third, average)
```

first, *second*, *third* y *average* son los argumentos

- La correspondencia entre argumentos y parámetros se establece por el orden en que aparecen.

Argumentos (II)

- El argumento correspondiente a un parámetro por valor puede ser una variable, una constante literal, una constante simbólica o una expresión.
- El argumento correspondiente a un parámetro por referencia tiene que ser una variable.

Matrices como argumentos de procedimientos

- Las matrices siempre se pasan como argumentos por referencia, independientemente del modificador que se utilice.
- Ejemplo:

```
Public Sub prueba(ByRef mt() As Integer)
  For i = 0 To 4
       mt(i) = i
  Next
End Sub

Sub Main()
  Dim mat() As Integer = {10, 10, 10,
10, 10}
  prueba(mat)
End Sub
```


Ámbito o Visibilidad de elementos declarados

- El ámbito de un elemento declarado es el conjunto de todo el código que puede hacer referencia a él directamente.
- Niveles de ámbito:
  - Ámbito de bloque.
  - Ámbito de procedimiento.
  - Ámbito de módulo.
- Hay que distinguir ámbito de período de duración. Este último se refiere al tiempo que un elemento declarado tiene disponible una zona de memoria.

Ámbito de bloque

- Un bloque es un conjunto de instrucciones incluido dentro de las instrucciones de declaración de inicio y fin. Por ejemplo: *Do…Loop*, *If…End If*, *Select…End Select*, etc.
- Las variables declaradas dentro de un bloque sólo se pueden utilizar dentro de él.
- Ejemplo:

```
While (n > 0)
   Dim i As Integer

   i = n
   …
   n *= 5  'Se modifica el valor de la variable de control
   …
   n = i
End While
```

- El período de duración de una variable con ámbito de bloque sigue siendo el del procedimiento. Las variables con este ámbito conservan su valor anterior al utilizar el bloque varias veces durante el procedimiento.

Ámbito de procedimiento

- Los elementos declarados dentro de un procedimiento solo se pueden utilizar dentro de él.
- El ámbito y el período de duración está restringido a la ejecución del procedimiento.

Ámbito de módulo

- Se pueden declarar elementos fuera de los límites de un bloque o procedimiento, pero dentro de los límites de un módulo.
- Estos elementos podrán ser utilizados en cualquiera de los procedimientos definidos dentro del módulo.
- No se producen conflictos de identificadores iguales cuando tienen diferente ámbito.

Ejemplo