Comencemos a programar con

VBA - Access

Entrega 06

Estructuras de datos

Eduardo Olaz

Matrices ó Arrays

Una matriz en VBA es un conjunto de variables del mismo tipo, a las que se puede acceder mediante un índice, que indica su posición en ella.

Imaginemos que queremos almacenar en el código, para su posterior utilización, el número de días de cada mes del año.

Por ejemplo, podemos hacer esto

```
Public Mes01 As Integer, Mes02 As Integer
Public Mes03 As Integer, Mes04 As Integer
Public Mes05 As Integer, Mes06 As Integer
Public Mes07 As Integer, Mes08 As Integer
Public Mes09 As Integer, Mes10 As Integer
Public Mes11 As Integer, Mes12 As Integer

Public Sub CargarMeses()
    Mes01 = 31
    Mes02 = 28
    Mes03 = 31
    Mes04 = 30
    Mes05 = 31
    Mes06 = 30
    Mes07 = 31
    Mes08 = 31
    Mes09 = 30
    Mes10 = 31
    Mes11 = 30
    Mes12 = 31
End Sub
```

Frente a esta declaración de variables, un tanto engorrosa, vamos a considerar estas alternativas:

```
Public Mes(12) As integer

Public Sub CargarMeses()
    Mes(1) = 31
    Mes(2) = 28
    Mes(3) = 31
    Mes(4) = 30
    Mes(5) = 31
    Mes(6) = 30
    Mes(7) = 31
    Mes(8) = 31
    Mes(9) = 30
```

```
    Mes(10) = 31
    Mes(11) = 30
    Mes(12) = 31
End Sub
```

Para asignar los valores a los elementos de la matriz, ejecutaremos el procedimiento `CargarMeses`

La tercera forma aún resulta más directa

```
Dim Mes() As Variant

Public Sub CargarMeses()
    Mes = Array(0, 31, 28, 31, 30, 31, 30, _
                31, 31, 30, 31, 30, 31)
End Sub
```

A veces, por facilidad de lectura del código, interesa distribuir la escritura de una única línea de código, en varias líneas físicas. Para ello se pone al final de la línea un espacio en blanco seguido de la barra baja, como se ve en el ejemplo anterior.

Para acceder a los días de un mes, por ejemplo Julio, en el ejemplo primero tenemos que utilizar directamente la variable

`Mes07`

Para hacer lo mismo en los ejemplos 2º y 3º, tenemos que tomar en cuenta que la variable **`Mes`** contiene los doce valores de los días del mes.

`Mes(7)`

Este método es mucho más práctico y da muchas más posibilidades para usar estructuras de bucle, como veremos más adelante.

Pero, ¿por qué he puesto `Mes = Array(0, 31, . . .?`

Por defecto las matrices, si no se indica el rango de sus índices, empiezan con el índice `0`.

Lo de añadir un valor más al principio como valor de **`Mes`**, en este caso `0` aunque podría haber puesto cualquier otro valor, es para que haya una coherencia entre los casos 2º y 3º, y que por ejemplo **`Mes(7)`** sea el valor de **Julio** en las dos matrices.

La declaración `Dim Mes(12) As integer` genera trece variables, accesibles desde el índice **0** `Mes(0)` al índice **12** `Mes(12)`.

Una segunda puntualización:

La línea de código

```
Mes = Array(0, 31, 28, 31, 30, 31, 30, _
            31, 31, 30, 31, 30, 31)
```

hace que, `Mes(i)`, para cualquiera de los valores de i, sea del tipo integer.

Si quisiéramos que `Mes(i)` devolviera un tipo Long, deberíamos poner el sufijo de declaración de tipo `Long` "`&`" detrás de cada número:

```
Mes = Array(0&, 31&, 28&, 31&, 30&, 31&, 30&, _
            31&, 31&, 30&, 31&, 30&, 31&)
```

Si quisiéramos que las matrices por defecto comenzaran con el índice `1`, deberíamos escribir en uno de los módulos, antes que cualquier procedimiento o declaración de matriz, la instrucción **`Option Base 1`**

Si hubiéramos escrito en la cabecera del módulo

```
Option Base 1
```

Y a continuación `Dim Mes(12) As integer` se puede acceder a la variable `Mes` mediante índices que van del **1** al **12**.

Existe la posibilidad de incluir en la declaración el rango de índices que va a manejar una matriz. La forma de hacerlo es

`Delimitador NombreDeMatriz(`**`IndiceInferior To IndiceSuperior`**`)`

En nuestro caso podríamos haber hecho

```
Public Mes(1 to 12) As integer
```

En los casos que hemos visto hasta ahora, estamos utilizando Matrices unidimensionales, cuyos elementos son accesibles mediante 1 único índice.

Son del tipo `Matriz(Indice)`

Matrices de varias dimensiones

Podemos declarar matrices de varias dimensiones; es decir que utilicen más de un índice.

Supongamos que nos encargan hacer un programa, en uno de cuyos puntos tenemos que controlar el número de personas que viven en un conjunto de bloques de vivienda numerados del 1 al 20. Cada bloque tiene 4 plantas que van de la planta baja (0) a la planta 3. Cada planta tiene 4 viviendas numeradas de la 1 a la 4.

Podríamos hacer una declaración de este tipo:

```
Dim Vivienda(1 To 20, 0 To 3, 1 To 4) As Integer
```

Si en la vivienda nº 2 de la planta baja del bloque 13 vivieran 3 personas, para asignar este valor a la variable sería así.

```
Vivienda(13, 0, 2) = 3
```

Si después en algún punto del código hacemos

```
intPersonas = Vivienda(13, 0, 2) = 3
```

la variable `intPersonas` contendrá el valor 3.

Matrices dinámicas

Supongamos que estamos haciendo un programa de ventas y que uno de sus condiciones es poder tener controladas en memoria, una vez seleccionado un tipo de producto, las referencias que existen del mismo.

Nos podremos encontrar tipos de producto con 1 referencia, otros con 4 ó con cualquier número de ellas.

A priori desconocemos el número de referencias que puede tener un tipo de producto, es más su número puede cambiar con el tiempo.

Para solucionar este tipo de problemas existen las llamadas Matrices Dinámicas.

Una matriz dinámica debe ser declarada, a nivel de módulo o de procedimiento, sin índices.

En nuestro caso se haría

```
Dim Referencias() As String
```

Más adelante, en un procedimiento, podríamos asignarle el número de elementos.

Si vamos a trabajar con un tipo de producto que tuviera 8 referencias, podremos redimensionar la matriz mediante la instrucción **Redim**.

```
ReDim Referencias(1 to 8)
```

Supongamos que posteriormente cambiamos de tipo de producto y pasamos a uno con sólo 2 referencias. En el código haremos

```
ReDim Referencias(1 to 2)
```

Tras redimensionar una matriz con **Redim**, los valores que contenía la matriz se **reinicializan**, tomando el valor por defecto del tipo de dato declarado.

En el caso de las cadenas el valor por defecto es la cadena vacía "", en el de los números el valor es **0** y en el de los Variant el valor **Empty**.

Si por necesidades de programación deseáramos conservar los valores que tenía una matriz dinámica antes de su redimiensionado, hay que utilizar la instrucción **Preserve** entre **Redim** y el nombre de la variable matriz.

Para ver esto vamos a analizar este código:

```
Public Sub PruebaRedim()
    Dim n As Long
    Dim Referencias() As String

    Debug.Print
    n = 5
    ReDim Referencias(1 To n)
    Referencias(5) = "Referencia 05"
    Debug.Print Referencias(5)

    n = 8
    ReDim Referencias(1 To n)
    Debug.Print
    Debug.Print "Tras Redim"
    Debug.Print "Los datos se han borrado"
    Debug.Print """ & Referencias(5) & """
    Debug.Print
    Referencias(5) = "Referencia 05"
    Referencias(8) = "Referencia 08"
    Debug.Print "Los datos se han cargado"
    Debug.Print """ & Referencias(5) & """
    Debug.Print """ & Referencias(8) & """
    Debug.Print
    n = 10
    ReDim Preserve Referencias(1 To n)
    Debug.Print "Tras Redim con Preserve"
    Debug.Print "los datos se han conservado"
    Debug.Print """ & Referencias(5) & """
```

```
    Debug.Print """" & Referencias(8) & """"
End Sub
```

Antes de seguir, voy a hacer unas aclaraciones.

La línea de código `Debug.Print` *realiza solamente un salto de línea en la ventana Inmediato.*

Las cuatro comillas seguidas equivalen a una comilla en modo texto. Es decir

```
Debug.Print """" & "MiTexto" & """"
```

hace que se imprima `"MiTexto"` *en la ventana Inmediato.*

En el caso del ejemplo: `"Referencia 05"`.

Si ejecutamos el procedimiento `PruebaRedim`, nos imprimirá en la ventana Inmediato:

```
Referencia 05

Tras Redim
Los datos se han borrado
""

Los datos se han cargado
"Referencia 05"
"Referencia 08"

Tras Redim con Preserve
los datos se han conservado
"Referencia 05"
"Referencia 08"
```

Tras las líneas

```
n = 5
ReDim Referencias(1 To n)
```

Redimensiona la matriz Referencias como si hubiéramos hecho

```
ReDim Referencias(1 To 5)
```

Posteriormente asigna un valor al elemento de la matriz de índice 5 y lo imprime.

Lo siguiente que hace es redimensionar la matriz a 8 elementos.

Tras ello el elemento 5º de la matriz ha desaparecido

Vuelve a asignar valores, en este caso a los elementos 5º y 8º y los imprime.

Redimensiona otra vez la matriz, esta vez con preserve, y se comprueba que no han desaparecido los valores anteriores.

Nota:

Lógicamente, aunque usemos **`Preserve`**, si redimensionamos una matriz a un número menor de elementos que la matriz anterior, los elementos superiores al nuevo índice máximo desaparecerán.

Instrucción Erase

Si tenemos declarada una matriz dinámica, VBA reserva una zona de memoria para guardar sus datos.

Si quisiéramos dejar libre, de forma explícita esa memoria una vez utilizada esa matriz, podemos usar la instrucción **Erase**.

Si consultamos la ayuda de VBA indica que **Erase** Vuelve a inicializar los elementos de matrices de tamaño fijo y libera el espacio de almacenamiento asignado a matrices dinámicas.

Esto quiere decir que si tenemos declarada una matriz de tamaño fijo por ejemplo:

```
Dim MatrizFija(10) As String
Dim MatrizFija2(10) As Long
```

La instrucción

```
Erase MatrizFija, MatrizFija2
```

No liberará la memoria ocupada, sólo reinicializará la matriz `MatrizFija` a cadenas vacías y la matriz `MatrizFija2` a ceros.

En cambio si tenemos `ReDim Referencias(1 To 5)`

`Erase Referencias`, libera la memoria ocupada por la matriz `Referencias`.

Redim con varias dimensiones

Supongamos ahora que además del caso de las viviendas del ejemplo tuviéramos que controlar otro grupo de viviendas compuesto de 4 bloques de 8 plantas (1 a 8) y con 6 puertas por plantas (1 a 6).

Para ello podríamos haber declarado inicialmente la matriz como

```
Dim Vivienda() As Integer
```

Posteriormente cuando vallamos a utilizarla con el primer conjunto de bloques de viviendas haríamos

```
Redim Vivienda(1 To 20, 0 To 3, 1 To 4)
```

Cuando tengamos que utilizarla con la segunda urbanización

```
Redim Vivienda(1 To 4, 1 To 8, 1 To 6)
```

Índices superior e inferior de una matriz.

En un punto del código nos puede ocurrir que necesitemos saber qué indices tiene como máximo y mínimo una matriz, ya sea ó no dinámica.

Para ello tenemos las funciones **UBound** y **LBound**.

UBound devuelve el índice máximo y **LBound** devuelve el índice mínimo.

Para probar estas funciones vamos a hacer lo siguiente

```
Public Sub PruebaIndicesMaxYMin()
    Dim Matriz(1 To 8) As Long
    Dim MatrizDinamica() As String
    Dim MultiDimensional(1 To 4, 1 To 8, 10) As Long
```

```
        ReDim MatrizDinamica(-2 To 4) As integer

        Debug.Print "Valor mínimo de Matriz()"
        Debug.Print LBound(Matriz)
        Debug.Print "Valor máximo de Matriz()"
        Debug.Print UBound(Matriz)
        Debug.Print "Número de elementos"
        Debug.Print
        Debug.Print UBound(MatrizDinamica) - LBound(Matriz) + 1
        Debug.Print "Valor mínimo de MatrizDinamica()"
        Debug.Print LBound(MatrizDinamica)
        Debug.Print "Valor máximo de MatrizDinamica()"
        Debug.Print UBound(MatrizDinamica)
        Debug.Print "Número de elementos"
        Debug.Print UBound(MatrizDinamica) - LBound(Matriz) + 1
        Debug.Print
        Debug.Print "Valor máximo índice 1° MultiDimensional()"
        Debug.Print UBound(MultiDimensional, 1)
        Debug.Print "Valor mínimo índice 2° MultiDimensional()"
        Debug.Print LBound(MultiDimensional, 2)
        Debug.Print "Valor mínimo índice 2° MultiDimensional()"
        Debug.Print LBound(MultiDimensional, 3)
    End Sub
```

Para obtener el valor máximo ó mínimo de los índices en una matriz de varias dimensiones, como se puede ver en el código, hay que hacer lo siguiente

```
UBound(NombreDeLaMatriz, N°DeIndice)
LBound(NombreDeLaMatriz, N°DeIndice)
```

Registros (Estructuras de variables definidas por el usuario)

Un registro ó estructura definida por el usuario está compuesta por un conjunto de datos, del mismo ó diferente tipo, que están relacionadas.

Supongamos que en un programa quisiéramos controlar los datos de diferentes personas.

Los datos a controlar son Nombre, Apellido1, Apellido2, Fecha de nacimiento y Teléfono.

Podríamos definir 5 variables, por ejemplo de esta forma:

```
Public strNombre As String
Public strApellido1 As String
Public strApellido2 As String
Public datNacimiento As Date
Public strTelefono As String
```

Incluso si tuviéramos que manejar varias personas simultáneamente podríamos crear esas variables como matrices dinámicas.

Pero ¿no sería una ventaja agrupar todos los datos en una misma variable?

Supongamos que lo pudiéramos hacer, y que esa variable se llamara `Amigo`.
Sería interesante que

`Amigo.Nombre` nos devolviera el nombre, ó que
`Amigo.Apellido1` nos devolviera su apellido.

Esto puede hacerse mediante la siguiente estructura:

```
Public Type Persona
    Nombre As String
    Apellido1 As String
    Apellido2 As String
    FechaNacimiento As Date
    Telefono As String
End Type
```

Tras esto podríamos hacer

```
Public Sub PruebaRegistro()
    Dim Cliente As Persona
    Dim Vendedor As Persona
    Cliente.Nombre = "Antonio"
    Cliente.Apellido1 = "Vargas"
    Cliente.Apellido2 = "Giménez"
    Cliente.Telefono = "979111111"
    Debug.Print
    Debug.Print Cliente.Nombre
    Debug.Print Cliente.Apellido1
    Debug.Print Cliente.Apellido2
    Debug.Print Cliente.Telefono
    Debug.Print
    'Ahora usando With
    With Vendedor
        .Nombre = "Pedro"
        .Apellido1 = "Jaizquíbel"
        .Apellido2 = "Gorráiz"
        .Telefono = "979222222"

        Debug.Print.Nombre _
            & " "; .Apellido1 _
            & " "; .Apellido2
        Debug.Print "Teléfono " _
            & .Telefono
    End With
End Sub
```

Si ejecutamos este procedimiento, nos mostrará lo siguiente en la ventana Inmediato:

```
Antonio
Vargas
Giménez
979111111

Pedro Jaizquíbel Gorráiz
Teléfono 979222222
```

Para crear la estructura `Persona`, he utilizado el par de sentencias

```
Type
     - - -
     - - -
End Type
```

Entre estas dos sentencias se declaran las variables que van a actuar como campos, ó miembros de ese registro, con el tipo al que pertenecen.

La estructura de tipo Registro puede ser declarada como `Public` ó `Private` en la cabecera de los módulos nunca dentro de un procedimiento.

Dentro de un módulo de clase ya sea propio, o asociado a un formulario ó informe, tanto las estructuras como las variables que las van a manejar sólo pueden declararse como de tipo `Private`.

Por ejemplo si hemos declarado en un módulo la estructura `Persona`, no puedo declarar una variable pública, dentro de un formulario ó informe, del tipo `Persona`

```
Public Comprador As Persona (daría error)
```

Pero sí podría hacer

```
Private Comprador As Persona
```

La instrucción With

Si observamos el código del procedimiento `PruebaRegistro` de la página anterior, vemos que, tanto para asignar valores a los atributos de la variable `Vendedor`, he utilizado la instrucción **`With`**, acompañada de su complemento **`End With`**.

Esto nos permite evitarnos tener que repetir de reiterativamente el nombre `Vendedor` en cada línea de asignación ó lectura.

Entre **`With Vendedor`** y **`End With`** al escribir el punto se supone que estamos utilizando atributos del registro `Vendedor`.

Este tipo de sintaxis ayudados por la instrucción **`With`**, se puede utilizar no sólo con Variables **`Registro`**, sino también con todo tipo de objetos, como veremos posteriormente.

Matrices de Registros

Además de los tipos de datos estudiados también podemos declarar matrices de Estructuras tipo Registro, incluso Matrices Dinámicas.

Por ejemplo podríamos declarar en la cabecera de un formulario lo siguiente:

```
Private Trabajadores() As Persona
```

Y en alguno de los procedimientos del formulario podríamos reinicializarla; por ejemplo en el evento **Al cargar**.

```
Private Sub Form_Load()
    ReDim Trabajadores(100)
End Sub
```

A partir de este momento podríamos cargar y leer datos en la matriz desde cualquiera de los procedimientos del formulario.

Para ello deberemos pasarle el índice de la matriz a la variable Trabajadores, y el nombre del atributo.

```
Trabajadores(1).Nombre = "Pepe"
Trabajadores(1).Apellido1 = "Gotera"
Trabajadores(2).Nombre = "Otilio"
```

E incluso podríamos utilizar la instrucción `With`

```
With Trabajadores(1)
     Debug.Print .Nombre
     Debug.Print .Apellido1
End With
```

En el siguiente capítulo veremos unos Objetos muy utilizados internamente por Access.

Son las **Colecciones**.

Estas estructuras del tipo **`Collection`** aparecen en muchos elementos de **Access**:

- Formularios
- Informes
- Controles
- ADO
- DAO
- Etc…