Comencemos a programar con

VBA - Access

Entrega 10

Estructuras de Control II

Eduardo Olaz

Estructuras de Control, segunda parte

Las Instrucciones While - - - Wend

La estructura de bucle

```
For Contador = ValorInicial To ValorFinal Step Salto
    - -
Next Contador
```

que hemos analizado en la entrega anterior, realiza una iteración del código un número de veces que resulta previsible en función de los valores **ValorInicial**, **ValorFinal** y **Salto**.

En las sucesivas iteraciones, la variable **Contador** va tomando valores que varían de forma constante entre un ciclo y otro.

El código incluido en el bucle se ejecutará al menos una vez, aunque fuera de forma incompleta si en su camino se tropezara con una sentencia **Exit For**.

Supongamos que necesitamos una estructura que se vaya ejecutando mientras el valor que va tomando una variable cumpla determinadas características, y además que esa variable pueda cambiar en forma no lineal.

Para realizar esta tarea podemos contar con la clásica estructura **While - - Wend**.

Digo lo de clásica porque es un tipo de estructura que ha existido desde las primeras versiones de Basic.

Esta estructura tiene la siguiente sintaxis

```
While condición
[intrucciones]
Wend
```

Condición es una expresión numérica o de tipo texto, que puede devolver **True**, **False** ó **Null**. Si devolviera **Null**, **While** lo consideraría como **False**.

Las ***instrucciones*** de código se ejecutarán mientras ***condición*** de cómo resultado **True**.

Supongamos que queremos crear un procedimiento que nos muestre los sucesivos valores que va tomando una variable, mientras esta variable sea menor que 100.

Los valores que irá tomando la variable serán cada vez el doble que la anterior.

Podríamos realizarlo de esta forma

```
Public Sub PruebaWhile()
    Dim lngControl As Long
    lngControl = 1
    While lngControl < 100
        Debug.Print lngControl
        lngControl = lngControl * 2
    Wend
End Sub
```

Este código nos mostrará en la ventana inmediato:

```
1
2
4
8
16
32
64
```

Tras efectuar el 7º ciclo, la variable `lngControl` tomará el valor **128**, por lo que la expresión **lngControl < 100** devolverá **False**.

Esto hará que el código pase a la línea siguiente a **Wend**, con lo que el procedimiento de prueba finalizará.

Una utilización tradicional para **While - - Wend** ha sido la lectura de ficheros secuenciales de texto, utilizando la función **Eof**, ficheros de los que de entrada no se conoce el número de líneas,.

Esta función, mientras no se llega al final del fichero devuelve el valor **False**.

Cuando llega al final devuelve el valor **True**.

Por ello el valor **Not Eof**, mientras no se haya llegado al final del fichero, devolverá lo contrario, es decir **True**.

Veamos el siguiente código:

```
Public Sub MuestraFichero( _
            ByVal Fichero As String)
    Dim intFichero As Integer
    Dim strLinea As String

    intFichero = FreeFile
    Open Fichero For Input As #intFichero
    While Not EOF(intFichero)
        Line Input #intFichero, strLinea
        Debug.Print strLinea
    Wend
End Sub
```

Éste es el clásico código para leer el contenido de un fichero secuencial.

Vamos a fijarnos en la estructura **While - - Wend**.

Traducido a lenguaje humano quiere decir:

Mientras no llegues al final del fichero **#intFichero**

Lee la línea del fichero, hasta que encuentres un retorno de carro y asígnaselo a la variable **strLinea**.

Imprime el contenido de la variable en la ventana inmediato

Vuelve a la línea de **While** para repetir el proceso

Las Instrucciones Do - - - Loop

El conjunto de instrucciones **While - - Wend** nos permite crear bucles que se ejecuten sólo si una variable, o expresión toma determinados parámetros.

While - - Wend no posee ninguna expresión que permita salir desde dentro del bucle en un momento dado, sin antes haberlo completado.

VBA posee una instrucción más potente, es la instrucción **Do - - - Loop**.

Su sintaxis posee dos formas distintas de utilización

```
Do [{While | Until} condición]
      [instrucciones]
      [Exit Do]
      [instrucciones]
Loop
```

O con esta otra sintaxis:

```
Do
      [instrucciones]
      [Exit Do]
      [instrucciones]
Loop [{While | Until} condición]
```

Veamos la primera forma:

Después de **Do** nos permite seleccionar **While *condición***, ó **Until *condición***.

Si ponemos **While**, después de **Do** el bucle se ejecutaría mientras la condición sea cierta.

Si escribimos **Until**, el bucle se ejecutaría hasta que la condición sea cierta.

Si la condición no fuese cierta no se ejecutaría el bucle tanto si hemos puesto **While**, como si hubiéramos escrito **Until** después de **Do**.

Por lo tanto podría ocurrir, tanto con **While** como con **Until** en función del resultado de ***Condición***, que no se llegara a ejecutar el bucle ni una sola vez.

Si deseáramos que siempre se ejecutara al menos una vez el bucle, deberíamos usar **While** ó **Until** después de **Loop**.

Supongamos que queremos escribir una función a la que pasándole un número entero positivo, nos indique si ese número es ó no primo.

Supongo que no hará falta recordaros que un número primo es aquél que sólo es divisible por 1 ó por sí mismo.

Este es el método que voy a emplear. – Sí. Ya se que no es el óptimo:

- Dividir el número entre valores enteros, empezando por el dos, y a continuación por los sucesivos valores impares, hasta que encontremos un valor que divida de forma exacta al número a probar (su resto = 0).

  Si el resto de la división da cero indica que el número es divisible por ese valor, por lo que el número no será primo y deberemos salir del bucle.
- Seguir con el ciclo mientras el valor por el que se va a dividir el número no sea mayor que la raíz cuadrada del número.

Necesitáis saber que en VBA, el operador que devuelve el resto de una división es **Mod**.

Si dividimos 17 entre 3 da de resto 2 `17 Mod 3 → 2`

Ya sé que este código es manifiestamente mejorable, pero funciona y me viene bien para el ejemplo con **`Do Loop`**.

Funciona si el número que probamos es menor ó igual que **2.147.483.647**

Este es el máximo número Long positivo. Este número también es primo.

```
Public Function EsPrimo( _
                    ByVal Numero As Long _
                    ) As Boolean
    Dim lngValor As Long
    Dim dblRaiz As Double

    Select Case Numero
    Case Is < 1
        MsgBox (Numero & " está fuera de rango")
        EsPrimo = False
        Exit Function
    Case 1, 2
        EsPrimo = True
        Exit Function
    Case Else
        dblRaiz = Numero ^ 0.5
        lngValor = 2
        ' Comprobamos si Numero es divisible por lngValor
        If Numero Mod lngValor = 0 Then
            EsPrimo = False
            Exit Function
        End If
        lngValor = 3
        EsPrimo = True
        Do While lngValor <= dblRaiz
            If Numero Mod lngValor = 0 Then
                EsPrimo = False
                Exit Function
            End If
            lngValor = lngValor + 2
        Loop
    End Select
End Function
```

Nota:

En este código he usado para calcular la raíz cuadrada de un número, elevar éste a **0,5**.

En VBA hay una función que calcula la raíz cuadrada directamente: **Sqr(Número)**.

Es equivalente a **Número^0.5**

Habiendo escrito la función **EsPrimo**, en un módulo estándar, vamos a crear un formulario en el que introduciendo un número en un cuadro de texto, tras pulsar un botón, nos diga si es primo ó no.

Cerramos el editor de código y creamos un nuevo formulario y lo ponemos en Vista Diseño.

Añadimos al formulario una etiqueta, un cuadro de texto y un botón.

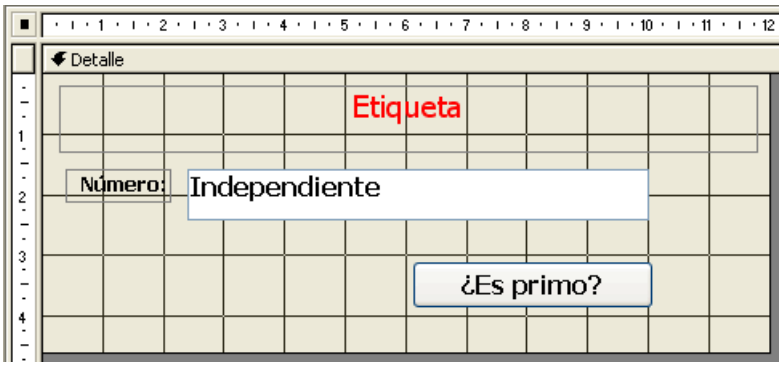
Nombres aplicados a los controles:

Etiqueta	**lblMensaje**
Cuadro de texto	**txtNumero**
Etiqueta del cuadro de texto	**lblNumero**
Botón	**cmdPrimo**

Ajustamos algunas de las propiedades del formulario, por ejemplo para quitar los separadores de registro, botones, etc.…

Ya que va a ser un formulario con muy pocos controles, ponemos los textos algo mayores que lo normal, e incluso podemos jugar con los colores.

A mí me ha quedado así

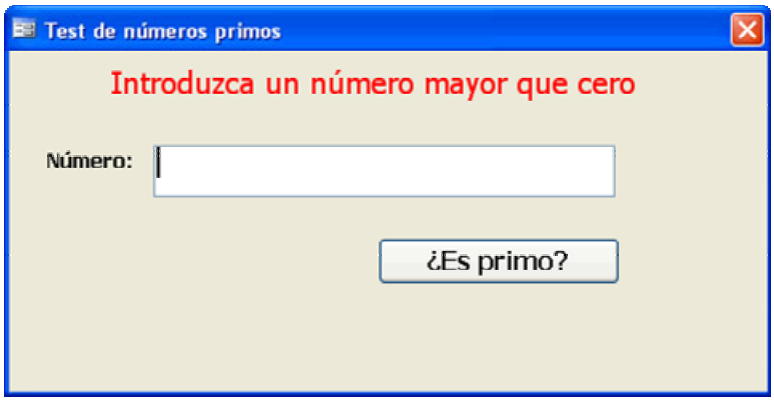


Abrimos la ventana de propiedades y teniendo seleccionado el formulario, vamos a la página de Eventos.

Hacemos que al abrir el formulario ponga como título del mismo **"Test de números primos"**, y como texto de la etiqueta `lblMensaje`, **"Introduzca un número entero"**.

```
Private Sub Form_Open(Cancel As Integer)
    Caption = "Test de números primos"
    lblmensaje.Caption = _
              "Introduzca un número mayor que cero"
End Sub
```

Al abrir el formulario quedará así:



Para que el formulario tenga este aspecto, he modificado algunas de sus propiedades:

Propiedad	Valor
Selectores de registro	**No**
Botones de desplazamiento	**No**
Separadores de registro	**No**
Estilo de los bordes	**Diálogo**

Vamos a hacer ahora que tras introducir un número en el cuadro de texto, y presionar el botón, nos diga en la etiqueta si el número es primo.

Volvemos a la hoja de propiedades y seleccionamos **Eventos**.

Teniendo seleccionado el botón, activamos el evento **Al hacer clic**, pulsamos en el botoncito que aparece con los tres puntos y seleccionamos **Generador de código**, y a continuación **Aceptar**.

Vamos a escribir el código:

Os recuerdo que detrás de la comilla simple lo que se escriba es un comentario (líneas en verde). Estas líneas VBA las ignora, sirviendo sólo como ayuda al usuario.

También os recuerdo que el espacio en blanco seguido de la barra inferior, al final de una línea, hace que la línea siguiente se considere como la misma línea.

El dividir así las líneas lo hago como ayuda para la composición de este texto y para ordenar el código.

```
Private Sub cmdPrimo_Click()
    Dim strNumero As String
    Dim lngNumero As Long

    ' Pasamos a la variable el contenido _
      de txtNumero, sin blancos en las esquinas
    ' Nz(txtNumero, "") devuelve una cadena vacía _
      si txtNumero contuviera Null
    ' Trim (Cadena) quita los "Espacios en blanco" _
      de las esquinas de la Cadena
    strNumero = Trim(Nz(txtNumero, ""))
```

```
        ' IsNumeric(strNumero) devuelve True _
          si strNumero representa a un número
        If IsNumeric(strNumero) Then

            ' La función EsPrimo() _
              funciona con números long positivos _
              entre 1 y 2147483647
            If Val(strNumero) > 2147483647# _
                        Or Val(strNumero) < 1 Then
                        lblmensaje.Caption = _
                "El número está fuera de rango"
                txtNumero.SetFocus
                Exit Sub
            End If
            lngNumero = Val(strNumero)
            ' Format(lngNumero, "#,##0") _
              devuelve una cadena con separadores de miles
            strNumero = Format(lngNumero, "#,##0")
            If EsPrimo(lngNumero) Then
                 lblmensaje.Caption = _
                    "El número " _
                    & strNumero _
                    & " es primo"
            Else
                lblmensaje.Caption = _
                "El número " _
                    & strNumero _
                    & " no es primo"
            End If
        Else
            lblmensaje.Caption = _
                "No ha introducido un número"
        End If
        ' El método SetFocus _
          hace que el control txtNumero tome el foco
        txtNumero.SetFocus
    End Sub
```

Tras presionar el botón **cmdPrimo** se produce el evento **clic**, por lo que se ejecuta el procedimiento **cmdPrimo_Click()** que maneja ese evento

Este procedimiento lo primero que hace es declarar dos variables, **strNumero** de tipo **string** y **lngNumero** de tipo **Long**.

A continuación asigna el contenido del cuadro de texto **txtNumero**, procesado primero con la función **Nz**, que devuelve una cadena vacía si tiene el valor **Null**, y a continuación le quita los posibles espacios en blanco de los extremos mediante la función **Trim**.

Seguidamente pasa por la primera estructura de decisión **If**, controlando si la cadena **strNumero** es de tipo numérico.

Si no lo fuera muestra en la etiqueta el mensaje **"No ha introducido un número"**.

Si lo fuera, primero comprueba si la expresión numérica de **strNumero** está entre **1** y **214748364**, rango de valores válidos en el rango de los **Long**, para la función **EsPrimo**.

Si no fuera así, muestra el mensaje **" El número está fuera de rango"**, lleva el cursor al control **txtNumero** y sale del procedimiento.

Supongamos que el contenido de **strNumero** ha logrado pasar todos estos controles.

Mediante la función **Val(strNumero)** asigna el valor a la variable **lngNumero**.

Como ya no vamos a utilizar la cadena **strNumero** para más cálculos, para mostrar el número, le asignamos el resultado de la función **Format(lngNumero, "#,##0")**.

Con esta utilización, la función **Format** devuelve una cadena formada por el número con los separadores de miles.

> *La función **Format** tiene un amplio abanico de posibilidades en la conversión de números y fechas a cadenas de texto.*
>
> Merece *por* sí misma un tratamiento más extenso.
>
> Se lo daremos en una próxima entrega.

El siguiente paso es comprobar si el número **lngNumero** es primo, utilizando la función **EsPrimo** que escribimos anteriormente.

Si lo fuera, escribiríamos en la etiqueta **"El número "** seguido del contenido de la cadena **strNumero**, y el texto **" es primo"**.

Si no lo fuera, escribiríamos lo mismo, pero indicando **" no es primo"**.

Terminado todo esto llevamos el cursor al cuadro de texto **txtNumero** mediante su método **SetFocus**.

> *Todo muy bien.*
> *El cliente está contento y el programa responde a lo que nos pedía, pero…*
> *Casi siempre hay un pero…*
> *Viendo lo efectivos y rápidos que hemos sido, al cliente se le ocurre que sería muy interesante poner dos botoncitos que al presionarlos, dado un número cualquiera, nos muestre el número primo inmediatamente mayor ó menor al número que hemos mostrado.*
> *-Tiene que ser fácil, total ya has hecho lo más importante y éste es un pequeño detalle adicional, que no te costará prácticamente nada de tiempo y supongo que no tendrás problemas* para hacérmelo sin aumentar el importe presupuestado…

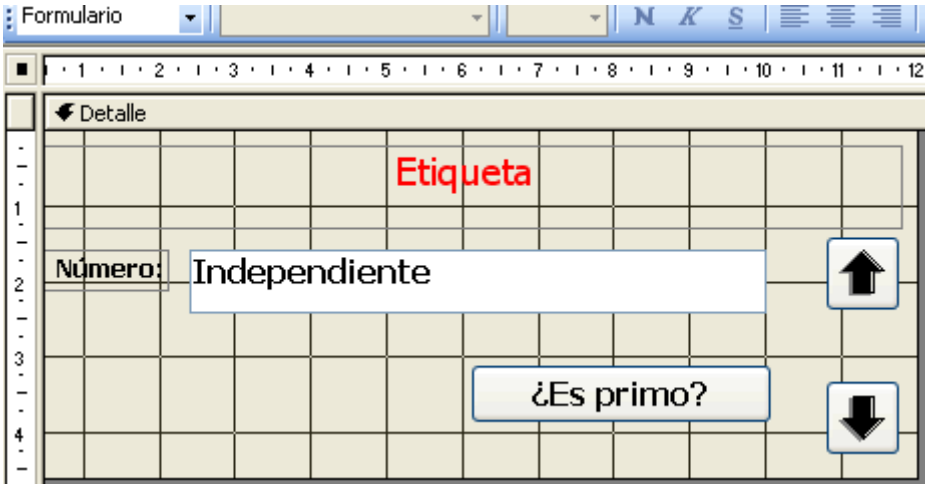
¿A alguno le suena esta conversación?.

Y además, aunque ya has terminado lo que te pedían, como hay que añadirle este “pequeño detalle…” no te pagan hasta que no lo termines…

Decido añadir dos botones con unas flechas en su interior.

Al primero, con una flecha hacia arriba lo llamo **cmdPrimoSiguiente**, y al segundo, con una flecha hacia abajo, **cmdPrimoAnterior**.

Este es el diseño que le doy al formulario:



Los eventos clic de los dos botones los escribo así:

```
Private Sub cmdPrimoSiguiente_Click()
    ' La siguiente línea hace que se ignoren _
      los posibles errores en la ejecución.
    On Error Resume Next
    Dim strNumero As String
    Dim lngNumero As Long
    Dim blnPrimo As Boolean
    strNumero = Trim(Nz(txtNumero, ""))
    If IsNumeric(strNumero) Then
        lngNumero = Val(strNumero)
        ' Si lngNumero está entre 0 y 2147483646
        If lngNumero < 2147483647# And lngNumero >= 0 Then

            ' Mientras blnPrimo no sea Cierto _
              Es decir Mientras lngNumero no sea primo.
            Do While Not blnPrimo
                lngNumero = lngNumero + 1
                blnPrimo = EsPrimo(lngNumero)
            Loop
            txtNumero = CStr(lngNumero)
            cmdPrimo_Click
        Else
            txtNumero = "1"
            cmdPrimo_Click
        End If
    Else
        txtNumero = "1"
        cmdPrimo_Click
    End If
```

```
End Sub
```

En el código anterior podemos ver algunas cosas interesantes.

Lo primero que nos puede llamar la atención es la sentencia:

```
On Error Resume Next
```

Esta es la forma más básica de efectuar un control de los errores que se puedan originar durante la ejecución de un programa en VBA.

Simplemente se le está indicando a VBA que si se produjera un error en algún punto del procedimiento lo ignore y vaya a la siguiente sentencia del código.

> *El ignorar los errores no es una verdadera forma de control.*
>
> *Aprenderemos en otra entrega diferentes formas de manejar los posibles errores, ya sean generados por el código, por datos inadecuados de los usuarios, etc.*

Más adelante nos encontramos con una sentencia If que evalúa una expresión doble

```
If lngNumero < 2147483647# And lngNumero >= 0 Then
```

Para que esta expresión sea cierta, lo tienen que ser a l vez las dos expresiones unidas por **And**; es decir **lngNumero** tiene que ser menor que **2147483647** y simultáneamente tiene que ser mayor ó igual que **0**.

Cuando varias expresiones estén unidas por el Operador Lógico **And**, para que la expresión total sea cierta, es necesario que lo sean cada una de esas expresiones. Con que haya una falsa, la expresión total será falsa.

Por el contrario, cuando varias expresiones estén unidas por el Operador Lógico **Or**, para que la expresión total sea cierta, es suficiente con que lo sea una cualquiera de las expresiones que la forman.

A continuación nos encontramos con otro Operador, es el operador negación **Not**.

```
Do While Not blnPrimo
```

Not hace que la expresión lógica que le sigue cambie su valor.

Así si **blnPrimo** contiene el valor **True**

```
Not blnPrimo
```

devolverá el valor **False**.

La expresión equivale a:

```
Mientras blnPrimo no sea cierto
```

Que es equivalente a

```
Mientras blnPrimo sea falso.
```

Con ello se ejecutará el código contenido entre la línea de **Do** y la línea del **Loop**.

Cuando `lngNumero` sea primo, la función EsPrimo asignará True a `blnPrimo`, con lo que se saldrá del bucle, pondrá la cadena de texto del número **txtNumero** en el cuadro de texto y ejecutará el procedimiento **cmdPrimo_Click**, como si se hubiera presionado en el botón [**cmdPrimo**].

Si el valor de **lngNumero** no hubiera cumplido con el rango de valores, pone un 1 en el cuadro de texto **txtNumero**, y ejecuta el procedimiento **cmdPrimo_Click**.

En el procedimiento que maneja la pulsación de la tecla [**cmdPrimoAnterior**] aunque tiene una estructura semejante, se introducen unos cambios que considero interesante remarcar.

```
Private Sub cmdPrimoAnterior_Click()
    ' Ignorar el error
    On Error Resume Next

    Dim strNumero As String
    Dim lngNumero As Long

    strNumero = Trim(Nz(txtNumero, ""))
    If IsNumeric(strNumero) Then
        lngNumero = Val(strNumero)
        If lngNumero < 2147483648# And lngNumero > 1 Then
            lngNumero = lngNumero - 1

            Do Until EsPrimo(lngNumero)
                lngNumero = lngNumero - 1
            Loop
            txtNumero = CStr(lngNumero)
            cmdPrimo_Click
        Else
            txtNumero = "2147483647"
            cmdPrimo_Click
        End If
    Else
        txtNumero = "2147483647"
        cmdPrimo_Click
    End If
End Sub
```

En primer lugar utilizamos una estructura del tipo **`Do Until`**, en vez de **`Do While`**.

Además, como condición no utiliza una variable como en el caso anterior, sino que lo compara directamente con el valor devuelto por la función `EsPrimo`, que devuelve **`True`** ó **`False`** según sea el caso:

```
Do Until EsPrimo(lngNumero)
```

Con esto nos evitamos utilizar una variable y una sentencia adicional. Además el código resulta algo más claro..

En este caso, si la variable no supera los filtros, pone el valor "`2147483647`" en el cuadro de texto.