Sentencias de control

Las sentencias de control, denominadas también estructuras de control, permiten tomar decisiones y realizar un proceso repetidas veces. Visual Basic dispone de las siguientes estructuras:

- If ... Then
- If ... Then ... Else
- Select Case
- For ... Next
- While ...Wend
- Do ...Loop
- Goto

Vemos a continuación la sintaxis correspondiente a cada una de ellas; cualquier expresión entre corchetes – [ ] – es opcional. De las expresiones que figuran entre llaves – { } – se puede elegir una, la necesaria en cada caso.

If ...Then

Permite tomar una decisión referente al camino a seguir o acción a ejecutar en un proceso basándose en el resultado (verdadero o falso) de una condición. Su sintaxis es:

If condición Then acción 1 [Else acción 2]

donde *condición* debe ser una expresión numérica, relacional o lógica. El resultado que se obtiene al evaluar la condición es verdadero (*True*) o falso (*False*); acción 1 o 2 son una o más sentencias (cada debe estar separada por dos puntos).
Si la *condición* es verdadera, se ejecuta la *acción 1*, y si la condición es falsa, se ejecuta la *acción 2*, lógicamente si *Else* ha sido especificada. En cualquier caso, la ejecución continúa con la siguiente sentencia ejecutable. Por ejemplo:

```
If x Then b = a / x
  b = b + 1
  '. . .
```

En este ejemplo, la condición es una expresión numérica *x*, y *b = a / x*, que equivale a la *acción 1*, se ejecutará si la condición es cierta (*x* distinta de *0*) y no se ejecutará si la condición es falsa (*x* igúal a *0*). En cualquier caso, se continúa la ejecución en la línea siguiente, *b = b + 1*. Otro ejemplo:

```
If a > b Then a = a + 1
  '. . .
```

En este otro ejemplo, la condición es una expresión de relación, *a < b*. La acción *a = a + 1* sólo se ejecutará si *a* es menor que *b*. En cualquier caso, se continúa en la línea siguiente. Un ejemplo más:

```
If a > b And b > c Then Print a
  '. . .
```

Aquí la condición es una expresión lógica. Se ejecutará la sentencia *Print a* sólo si *a* es mayor que *b* y *b* mayor que *c*. En cualquier otro caso, se continúa en la línea siguiente. Otro ejemplo:

```
If a = b * 5 Then x = 4 : a = a + x Else b = 0
  '. . .
```

En este ejemplo, si se cumple la condición *a = b * 5*, se ejecutan las sentencias *x = 4* y *a = a + x*. En otro caso, se ejecuta la sentencia *b = 0*. En ambos casos se continúa en la siguiente línea de programa. Un ejemplo más:

```
If r$ = "sí" Then End
```

En este otro ejemplo, la sentencia *End* se ejecutará cuando *r$* sea igual a la cadena de caracteres *"sí "*.
El siguiente código comprueba cómo es un número a con respecto a otro b.

```
If a > b Then Print a; "es mayor que"; b
If a < b Then Print a; "es menor que"; b
If a = b Then Print a; "es igual a"; b
```

Cuando se comparan dos valores fraccionarios, hay que tomar precauciones en el sentido de que es difícil obtener la igualdad. Por ejemplo:

```
If a# = 0# Then Print "a es igual a 0"
```

En el ejemplo anterior es posible que no se llegue a dar nunca la condición, ya que *a* (valor real expresado en precisión doble) puede tomar valores muy próximos a cero, como *1 E-23* (1 por 10 elevado a -23), pero no exactamente cero. Para evitar este problema podríamos escribir:

```
If Abs(a#) <= 1E-23 Then Print "a es igual a 0"
```

La sentencia *If ... Then ... Else*, con sus sentencias asociadas, debe aparecer en una sola línea. Por ejemplo:

```
If a = b Then x = 0: y = 0 Else a = 1: c = c + a
```

Cuando lo anterior no sea posible, se utilizará la estructura *If* que veremos a continuación. Aun siendo posible utilizar la sentencia *If*, es recomendable utilizar estructura *If*, sobre todo por su facilidad de interpretación.

Anidamiento de sentencias If

Las sentencias *If ... Then ... Else* pueden estar anidadas, con la única limitación de la longitud de una línea lógica. Esto quiere decir que a continuación de las palabras reservadas *Then* o *Else* puede escribirse otra sentencia *If*. Por ejemplo:

```
If a > b Then Print a; "es mayor que" : b; Else _
If a < b Then Print a; "es menor que"; b; Else _
Print a; "es igual a"; b
```

Si en una línea lógica no hay e1 mismo número de cláusulas *Then* y *Else*, la regla es que cada *Else* se corresponde con el *If* más próximo que no haya sido emparejado. Por ejemplo:

```
If a = b Then If b = c Then Print "a = b = c" Else Print "b <> c"
```

En este ejemplo aparecen dos cláusulas *If* y una *Else*. Pues bien, aplicando la regla anterior, la cláusula *Else* se corresponde con el segundo *If*. Otro ejemplo:

```
If a = 0 Then If b <> 0 Then s = s + b Else s = s + a
```

En este ejemplo, cuando *a <> 0* se pasa a ejecutar la siguiente línea de programa. Si lo que desea es que se ejecute *s = s + a* cuando *a < > 0*, entonces se tendría que escribir:

```
If a = 0 Then If b <> 0 Then s = s + b Else Else s = s + a
```

El código siguiente da como resultado el menor de tres números *a*, *b*, *c*.

```
If a < b Then If a < c Then menor = a Else menor = c E1se _
If b < c Then menor = b Else menor = c _
Print menor
```

La interpretación de las sentencias *If* anidadas tal cual las acabamos de ver puede presentarle dificultades, por lo que en la mayoría de los casos, por no decir en todos, es preferible utilizar la estructura *If* que vemos a continuación.

If ... Then ... Else

Cuando utilice una estructura *If* tiene que escribirla de la forma que se presenta a continuación, de lo contrario ocurrirá un error de sintaxis:

```
If condición-1 Then
sentencias-1
[ElseIf condición-2 Then
```

```
  sentencias-2] ...
[Else
  sentecias-n]
End If
```

La ejecución de esta estructura sucede de la siguiente forma: si se cumple la *condición-1* , se ejecutan las *sentencias-1*, y si no se cumple, se examinan secuencialmente las condiciones siguientes hasta *Else*, ejecutándose las sentencias correspondientes al primer *ElseIf* cuya condición sea cierta. Si todas las condiciones son falsas, se ejecutan las *sentencias-n* correspondientes a *Else*. En cualquier caso, se continúa en la sentencia que sigue a *End If*.
La estructura *If* proporciona varias ventajas sobre la sentencia *If* de una sola línea expuesta anteriormente. Algunas de ellas son:

- El número de sentencias después de las cláusulas *Then* o *Else* puede ser cualquiera.
- Las condiciones complejas son fácilmente codificadas.
- El anidamiento resulta fácil y claro.

Por ejemplo, supongamos que al efectuar una compra en un almacén, si adquirimos más de 100 unidades de un mismo artículo, nos hacen un descuento de un 40 %; entre 25 y 100, un 20 %; entre 10 y 24, un 10 %, y no hay descuento para una adquisición de menos de l0 unidades. Calcular el importe a pagar.

```
If Cantidad > 100 Then
        Importe = Cantidad * PrecioUnidad * 0.6
ElseIf Cantidad >= 25 Then
        Importe = Cantidad * PrecioUnidad * 0.8
ElseIf Cantidad >= 10 Then
        Importe = Cantidad * PrecioUnidad * 0.9
Else
        Importe = Cantidad * PrecioUnidad
End If
```

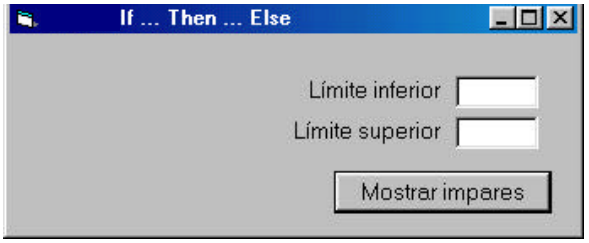
En el ejemplo que acabamos de exponer, hay que resaltar que las condiciones van puestas de más a menos unidades, ya que, como se ha indicado, las condiciones se examinan secuencialmente a partir de la primera.
Volvamos al ejemplo de realizar una rutina que dé como resultado el menor de tres números *a*, *b*, *c*. Para evitar ambigüedades, utilizaremos ahora la estructura *If* en vez de la sentencia *If* convencional. El resultado es:

```
If a < b Then
   If a < c Then
      menor = a
   Else
      menor = c
   End If
ElseIf b < c Then
   menor = b
Else
   menor = c
End If
Print menor                                    ,
```

Realizaremos, ahora, una aplicación que imprima los números impares que hay en un intervalo *(a, b)*. Una vez introducido el intervalo, verificar si *a* es menor que *b*; si no es así invertir sus respectivos valores.
Un número es impar cuando no es múltiplo de 2, y un número es múltiplo de 2 cuando al dividirlo por 2 el resto es igual a 0. Esto se puede especificar preguntando si número Mod 2 = 0.
La solución de este ejemplo puede ser así:

1. Se solicita el límite inferior y superior del intervalo, mediante las cajas de texto *txtLimteInferior* y *txtLimiteSuperior*.
2. Se verifica si el límite inferior es mayor que el superior, en cuyo caso se intercambian.

3. Se localiza el primer número impar del intervalo y se imprimen todos los números impares a partir de éste.

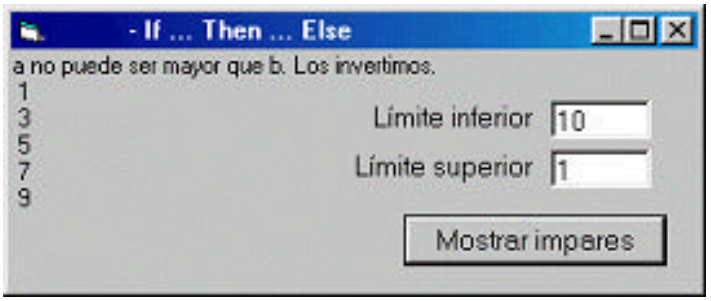


Para comenzar confeccionemos el formulario el cual deberá ser similar al que muestra la figura de la izquierda.
El espacio que aparece a la derecha es destinado a la impresión, luego de que pulse *Mostrar impares*.
El código que debemos incluir en el evento Clic del botón de comando *Mostrar impares* es el siguiente:

```
Private Sub cmdMostrar_Click()
  Dim a As Integer, b As Integer
  Dim x As Integer, n As Integer
  a = txtLimiteInferior.Text
  b = txtLimiteSuperior.Text
  'Verificar si a es menor que b
  If a > b Then
    Print "a no puede ser mayor que b. Los invertimos."
    x = a: a = b: b = x
  End If
  'Localizar el primer número impar
  If a Mod 2 = 0 Then a = a + 1
  'Escribir todos los números impares
  For n = a To b Step 2
    Print n
  Next n
End Sub
```

Ejecute la aplicación y observe los resultados. Por ejemplo, si ingresa el rango 1 – 10, al hacer clic en el botón de pulsación *Mostrar impares*, el formulario deberá mostrarse como sigue:

En el caso de ingresar el límite inferior mayor que el inferior, en el formulario se imprimirá:



Select Case

Esta sentencia permite ejecutar una de varias acciones en función del valor de una expresión. Es una alternativa a *If ... Then ..ElseIf* cuando lo que se necesita es comparar la misma expresión con diferentes valores. Su sintaxis es

Select Case expr-test
Case lista 1 [sentencias 1]
[Case lista 2 [sentencias 2]
[Case Else [sentencias n]]
End Select

donde *expr-test* es una expresión numérica o alfanumérica, y *lista 1* , *lista 2* presentan una lista que puede tener cualquiera de las formas siguientes:

expresión[, expresión] . . .
expresión To expresión
Is operador-de-relación expresión
combinación de las anteriores separadas por coma

Aquí, *expresión* es cualquier expresión numérica o de caracteres del mismo tipo que *expr-test*. Por ejemplo:

```
Case Is < x       'expr-test < x
Case 3              'expr-test = 3
Case x To 20     'expr-test = x, x+1, ...,20
Case 3, x           'expr-test = 3, x
Case -1, x To 5  'expr-test = -1, x, x+1,....5
Case "sí", "SI"  'expr-test = "sí", "SI"
Case Is >= 10     'expr-test >= 10
```

Cuando se utiliza la forma *expresión To expresión*, el valor más pequeño debe aparecer en primer lugar.
Cuando se ejecuta una sentencia *Select Case*, Visual Basic evalúa la *expr-test* y busca el primer *Case* que incluya el valor evaluado, ejecutando a continuación el correspondiente bloque de sentencias. Si no existe un valor igual a la *expr-test* entonces se ejecutan las sentencias a continuación de *Case Else*. En cualquier caso, el control pasa a la siguiente sentencia a *End Select*. Por ejemplo:

```
Select Case X
  Case 1
    Text1.Text = "1"
  Case 2, 3
    Text1.Text = "2 o 3"
  Case 4 To 9
    Text1.Text = "4 a 9"
  Case Else
    Text1.Text = "X<1 o X>9"
End Select
```

En este ejemplo, si *X* vale *1*, se asigna *"1"* a la caja de texto *Text1*; si vale *2 o 3*, se asigna *"2 o 3"* a la caja de texto *Text1*; si vale *4, 5, 6, 7, 8 o 9*, se asigna *"4 a 9"* a la caja de texto *Text1*; y en cualquier otro caso, se asigna *"X<1 o X>9"* a la caja de texto *Textl*. Cuando se produce una coincidencia, se ejecuta sólo el código que hay hasta el siguiente *Case*, o hasta *End Select* si se trata del último *Case*.

Realizaremos ahora una aplicación que cambie el color de fondo del formulario a partir del color ingresado en la caja de texto *txtColor*.
El formulario tendrá el siguiente aspecto:

Utilizando el procedimiento *cmdCambiarColor_Clic* que se ejecuta cuando se pulsa el botón *Cambiar color del formulario*, el proceso puede escribirse así:

```
Private Sub cmdCambiarColor_Click()
  Select Case LCase(txtColor.Text)
    Case "rojo"
       frmCase.BackColor = vbRed              'color rojo
    Case "amarillo"
       frmCase.BackColor = vbYellow    'color amarillo
    Case "azul"
       frmCase.BackColor = vbBlue             'color azul
    Case "verde"
       frmCase.BackColor = vbGreen            'color verde
    Case Else
       frmCase.BackColor = &H8000000F         'color cara del botón
  End Select
End Sub
```

La función *Lcase* devuelve un tipo *String* que se ha convertido a minúsculas. Este nos sirve cuando no sabemos el formato en el cual se ingresará la cadena de caracteres. En este caso estamos comparando el color escrito en la caja de texto *txtColor* con "rojo", "amarillo", etc, las cuales están escritas en minúsculas por lo que si el usuario ingresa "Rojo", no correspondería al *Case "rojo"* sino al *Case Else.*
Al ejecutar la aplicación, el formulario se mostrará del color escrito en la caja de texto:

For ... Next

La sentencia *For* da lugar a un lazo o bucle y permite ejecutar un conjunto de sentencias cierto número de veces. Su sintaxis es:

```
For variable = expresión 1 To expresión 2 [Step expresión 3]
[sentencias]
  [Exit For]
  [sentencias]
Next [variable[, variable]…]
```

Cuando se ejecuta una sentencia *For* en la que el valor de la *expresión 3* es positivo o no se ha especificado, primero se asigna el valor de la *expresión 1* a la *variable* y a continuación se comprueba si la *variable* es mayor que la *expresión 2*, en cuyo caso se salta el cuerpo del bucle y se continúa en la línea que esté a continuación de la sentencia *Next*. En otro caso, se ejecutan las líneas de programa que haya entre la sentencia *For* y la sentencia *Next*. Por último, la variable se incrementa en el valor de la *expresión 3*, o en *1* si *Step* no se especifica, volviéndose a efectuar la comparación entre la *variable* y la *expresión 2*, y así sucesivamente.
La sentencia *Exit For* permite salir de un bucle *For ... Next* antes de que éste finalice.
Un bucle *For ... Next* se ejecuta más rápidamente cuando la variable es entera y las *expresiones 1, 2, 3,* constantes. Por ejemplo:

```
Sub Form_Click()
  Dim I As Integer, Suma As Integer
  For I = 1 To 99 Step 2 'Para I = 1, 3, 5,...hasta 99
    Suma = Suma + 1
```

```
  Next I
  Print Suma
End Sub
```

Este ejemplo indica que cuando hagamos clic sobre el formulario, se visualizará sobre el mismo la suma de los números impares entre 1 y 99. Recuerde que 1a propiedad *AutoRedraw* del formulario debe valer *True*.
Si el valor de la *expresión 3* es negativo, se comprueba si la *variable* es menor que la *expresión 2*, en cuyo caso se pasa a ejecutar la siguiente sentencia a *Next* En otro caso, se ejecuta el bucle y se decrementa la *variable*. Este proceso se repite hasta que la *variable* sea menor que la *expresión 2*. Por ejemplo, el siguiente procedimiento conduce al mismo resultado que el anterior.

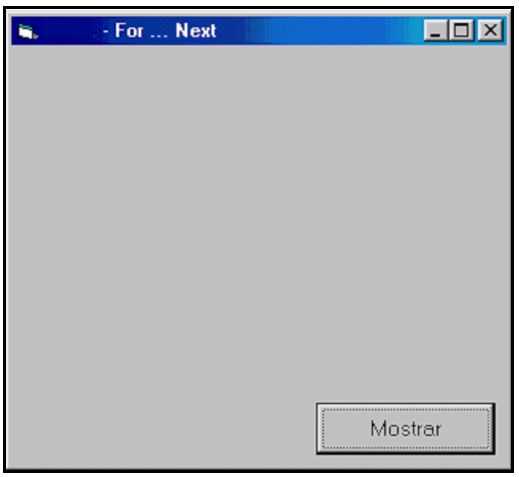
```
Sub Form_Click()
  Dim I As Integer, Suma As Integer
  For I = 99 To 1 Step –2 'Para I = 99, 97,...hasta 1
    Suma = Suma + 1
  Next I
  Print Suma
End Sub
```

El siguiente ejemplo da lugar a que se impriman los valores l, 1.5, ... 3.5, 4. Para ello debe definir la variable de algún tipo fraccionario.

```
For h = 1 To 4 Step 0.5
  Print h,
Next h
```

Siendo la *expresión 1* <= *expresión 2*, si el valor de la *expresión 3* es cero, se creará un bucle infinito.
Un cambio en el valor de la *expresión 3* para finalizar el bucle mientras está dentro del mismo, hace difícil su lectura y depuración. Para este fin utilice la sentencia *Exit For*.

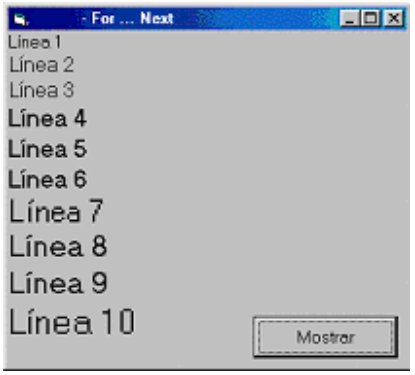
Escribiremos una aplicación que conste de un botón de comando cuyo título sea *Mostrar*, el mismo contendrá un bucle *For ... Next* que utiliza el método *Print* para mostrar 10 veces en el formulario la palabra *Línea*, seguida del contador del bucle. Visual Basic permitirá modificar propiedades y actualizar variables clave en un bucle. Así que adicionaremos a este ejercicio el cambio de la fuente con la que se escribe en el formulario a un tamaño cada vez mayor a medida que se recorra el bucle. La propiedad *FontSize* especifica el tamaño del texto mostrado en un formulario; podrá utilizarla como alternativa a la modificación del tamaño con la propiedad *Font*.
Utilice el control *CommandButton* para crear un botón de orden en la parte inferior del formulario y asigne a la propiedad *Caption* del botón de orden la palabra "*Mostrar*".



Pulse dos veces el botón *Mostrar* del formulario. En la ventana código aparecerá el procedimiento de suceso *cmdMostrar_Click*. Introduzca las siguientes sentencias de programa en dicho procedimiento:

```
Private Sub cmdMostrar_Click()
  For i = 1 To 10
    FontSize = 10 + i
    Print "Línea"; i
  Next i
End Sub
```

Este bucle *For...Next* utiliza el método *Print* para mostrar 10 veces en el formulario la palabra Línea, seguida del contador del bucle. El punto y coma (;) de la sentencia *Print* hace que Visual Basic muestre la variable contador junto a la cadena "*Línea*", sin espacio adicional en medio. Sin embargo, cuando ejecute el programa podrá ver un espacio en blanco entre "*Línea*" y la variable contadora. Cuando se imprimen valores numéricos, el método *Print* reserva un espacio en blanco para el signo menos, aunque dicho signo no sea necesario en todas las ocasiones.
Al ejecutar el programa, cuando pulse el botón *Mostrar* en el formulario se mostrará:



Bucles anidados

Un bucle *For ... Next* puede colocarse dentro de otro bucle *For ... Next* y entonces se dice que están anidados. En este caso, cada bucle debe tener un nombre de variable único. La sentencia *Next* para el bucle interior debe aparecer ante que la del bucle exterior. Esto es:

```
For var1 = ...
  For var2 = ...
    ' sentencias
  Next var2
Next var1
```

Si los bucles anidados tienen el mismo punto de terminación, puede utilizarse para todos ellos una sola sentencia *Next*, la cual tomará la forma:

```
Next var1, var2
```

En el ejemplo siguiente las sentencias *Next j*, *Next i* podrían ser sustituidas por la sentencia Nextj, i:

```
Dim aster As String, guion As String, resu As String
Dim i As Integer, j As Integer

aster = "*": guión = "-" : resu = ""
For I = 1 To 5
  resu = resu & aster : Print resu
  For j = 1 To 2
    resu = resu & guion : Print resu
  Next j
Next i
```

La variable o variables en la sentencia *Next* pueden ser omitidas, en cuyo caso la sentencia *Next* actuará sobre la sentencia *For* más reciente. Incluir las variables es, en general, muy útil para evitar confusiones, pero es absolutamente necesario cuando realice bifurcaciones fuera de los bucles.
Una sentencia *For* sin emparejar da lugar a un error *For sin Next* y una sentencia *Next* sin emparejar da lugar a un error *Next sin For*.

While ... Wend

Un bucle *While* repite la ejecución de un conjunto de sentencias mientras una condición dada sea cierta. La condición se verifica antes de ejecutarse el conjunto de sentencias.

```
While condición
  [sentencias]
Wend
```

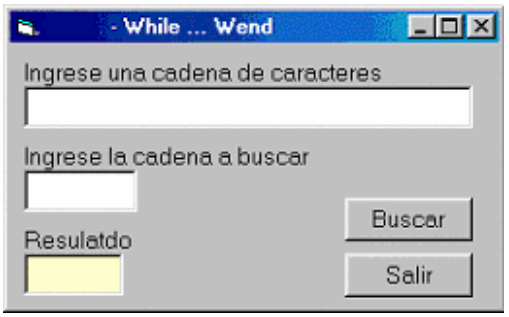
donde *condición* es cualquier expresión numérica, relacional o lógica.
La ejecución de la sentencia While sucede así:

1. Se evalúa la *condición*.
2. Si el resultado de la evaluación es falso, las *sentencias* que forman el cuerpo de *While* no se ejecutan y se pasa el control a la siguiente sentencia en el código a *Wend*.
3. Si el resultado de la evaluación es verdadero, se ejecutan las *sentencias* que forman el cuerpo de *While* y el proceso descrito se repite desde el punto 1.

Por ejemplo, el código siguiente imprime los valores enteros entre 1 y 10:

```
Dim n As Integer
n = 1              'valor inicial de n
While n <= 10          'mientras n sea menor o igual que 10...
  Print n;             'escribir el valor de n
  n = n + 1            'siguiente valor de n
Wend
```

Los bucles *While ... Wend* pueden anidarse a cualquier nivel. Cada *Wend* emparejará con el *While* más reciente. Una sentencia *While* sin emparejar da 1ugar a un error *While sin Wend* y una sentencia *Wend* sin emparejar da lugar a un error *Wend sin While*.
Como ejemplo de lo expuesto, vamos a realizar un programa cuente las veces que se repite una cadena dentro de otra cadena repitiendo el bucle tantas veces como se encuentre la *cadena2*. Dichas cadenas serán ingresadas en las cajas de texto *txtCadena1* y *txtCadena2*. El formulario será similar al siguiente:



La solución de este problema puede ser de la forma siguiente:

- Primero definimos las variables que vamos a utilizar en los cálculos.

  ```
  Dim posición As Integer, contador As Integer
  Dim cadena1 As String, cadena2 As String
  ```

- Después, asignamos valores a las variables definidas

  ```
  cadena1 = txtCadena1.Text
  ```

```
cadena2 = txtCadena2.Text
posición = 1
```

- Realizamos la comprobación de la existencia de la *cadena2* en la cadena *cadena1* usando la sentencia *While*.

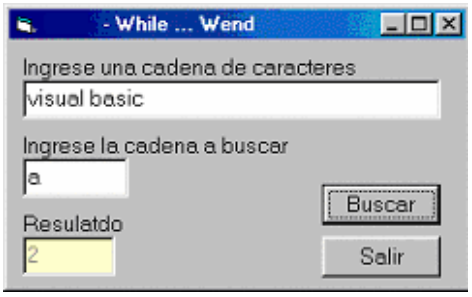
```
While InStr(posición, cadena1, cadena2)
      posición = InStr(posición, cadena1, cadena2) + 1
      contador = contador + 1
Wend
```

La función *InStr* devuelve un tipo *Variant* (*Long*) que especifica la posición de la primera aparición de una cadena en otra. Si la *cadena2* no está en la otra cadena *cadena1*, *InStr* devuelve 0 y no se ejecuta el bucle.
Luego mostramos el valor de la variable contador en la caja de texto *txtResultado*.

El procedimiento completo se muestra a continuación:

```
Private Sub cmdBuscar_Click()
  Dim posición As Integer, contador As Integer
  Dim cadena1 As String, cadena2 As String
  cadena1 = txtCadena1.Text
  cadena2 = txtCadena2.Text
  posición = 1
    While InStr(posición, cadena1, cadena2)
       posición = InStr(posición, cadena1, cadena2) + 1
       contador = contador + 1
    Wend
  txtResultado.Text = contador
End Sub
```

Antes de entrar en la repetitiva *While ... Wend* es necesario preparar la condición (iniciar las variables) para que la evaluación de la misma tenga sentido.



La sentencia *Do ... Loop* que se expone a continuación, proporciona una manera más flexible y estructurada de realizar los bucles.

Do ... Loop

Un *Loop* (bucle) repite la ejecución de un conjunto de sentencias mientras condición dada sea cierta, o hasta que una condición dada sea cierta. La puede ser verificada antes o después de ejecutarse el conjunto de sentencias:

Formato 1:

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

Formato 2:

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

donde *condición* es cualquier expresión que se evalúe a *True* o a *False*.
Esta sentencia permite realizar varias estructuras diferentes. Permite, como se aprecia en los formatos, crear bucles con la condición de terminación al final o al principio del bloque de sentencias. Por ejemplo:

```
Sub Form_Click()
  Dim I As Integer, Suma As Integer
  I = 1
  Do While I <= 99      'Hacer mientras I <= 99
        Suma = Suma + 1
        I = I + 2
  Loop
  Print Suma
End Sub
```

Este ejemplo indica que cuando hagamos clic sobre el formulario se verá sobre el mismo la suma de los números impares entre 1 y 99. El mismo resultado se obtiene con el ejemplo que se presenta a continuación:

```
Sub Form_Click()
  Dim I As Integer, Suma As Integer
  I = 99
  Do
        Suma = Suma + 1
        I = I - 2
  Loop Until I < 1
  Print Suma
End Sub
```

La sentencia *Exit Do* permite salir de un bucle *Do ... Loop* antes de que finalice éste.

Goto

Transfiere el control a una línea específica del código, identificada por una etiqueta o por un número de línea. Su sintaxis es:

Goto {etiqueta | número de línea}

Si la línea a la que se transfiere el control es una sentencia ejecutable, se ejecuta esa sentencia y las que le siguen. Si no es ejecutable, la ejecución se inicia en la primera sentencia ejecutable que se encuentra tras dicha línea. Por ejemplo, el siguiente procedimiento calcula e imprime el área de uno o más círculos.

```
Private Sub Form_Click()
  Dim r As Single, a As Single
  Print "Escribir 0 para finalizar"
Comienzo:
  r = InputBox("Radio:")
  If r <= 0 Then
    Exit Sub
  Else
    a = 3.141592 * r ^ 2
    Print "Area ="; a
  End If
  Goto Comienzo
End Sub
```

Un uso abusivo de esta sentencia da lugar a aplicaciones difíciles de interpretar y de mantener. Por ello, en programación se utiliza solamente en ocasiones excepcionales. La función que vaya a desempeñar una sentencia *Goto* puede suplirse utilizando cualquiera de las sentencias de control estructuradas vistas anteriormente. Su utilización se restringe a la sentencia *On Error* que veremos más adelante.

For Each ... Next

Repite un grupo de sentencias para cada elemento de una matriz o de una colección.

Material de apoyo

Aprenda Microsoft Visual Basic 6.0 ya
Michael Halvorioel
Editorial: MC Graw Hill

Programación Orientada en ADO
David Sceppa
Editorial: MC Graw Hill

Bases de Datos con Visual Basic 6
Jeffrey P. Mc. Raws
Editorial: Prentice Hall

Edición Especial Visual Basic 6
Brian Siler & Jeff Sopotts
Editorial: Prentice Hall

Visual Basic 6 Curso de Programación
Fco Javier Cevallos
Editorial: Alfa Omega Ra-ma