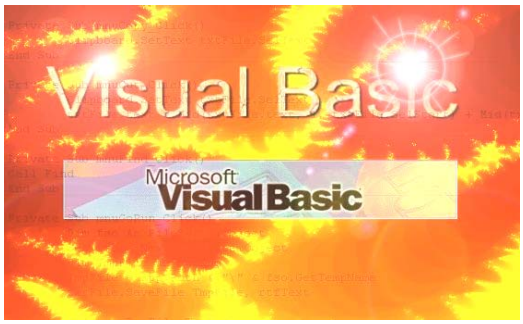
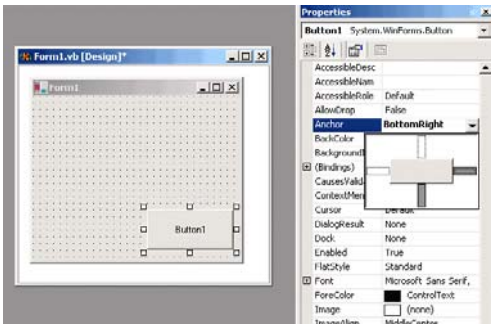
CURSO

Curso Completo de Visual Basic 6.0





Escuela Superior de Ingenieros Industriales
UNIVERSIDAD DE NAVARRA
Javier García de Jalón · José Ignacio Rodríguez
Alfonso Brazález · Patxi Funes · Eduardo Carrasco · Jesús Calleja

3. LENGUAJE BASIC

3.5 OPERADORES

La **Tabla 3.3** presenta el conjunto de operadores que soporta ***Visual Basic 6.0***.

Tipo	Operación	Operador en Vbasic
Aritmético	Exponenciación	^
	Cambio de signo (operador unario)	-
	Multiplicación, división	*, /
	División entera	\
	Resto de una división entera	**Mod**
	Suma y resta	+, -
Concatenación	Concatenar o enlazar	&
Relacional	Igual a	=
	Distinto	<>
	Menor que / menor o igual que	< <=
	Mayor que / mayor o igual que	> >=
Otros	Comparar dos expresiones de caracteres	**Like**
	Comparar dos referencias a objetos	**Is**
Lógico	Negación	**Not**
	And	**And**
	Or inclusivo	**Or**
	Or exclusivo	**Xor**
	Equivalencia (opuesto a Xor)	**Eqv**
	Implicación (*False* si el primer operando es *True* y el segundo operando es *False*)	**Imp**

Cuando en una expresión ***aritmética*** intervienen operandos de diferentes tipos, el resultado se expresa, generalmente, en la misma precisión que la del operando que la tiene más alta. El orden, de menor a mayor, según la precisión es ***Integer***, ***Long***, ***Single***, ***Double*** y ***Currency***.

Los operadores ***relacionales***, también conocidos como operadores de *comparación*, comparan dos expresiones dando un resultado ***True (verdadero)***, ***False (falso)*** o ***Null (no válido)***.

El operador **&** realiza la concatenación de dos operandos. Para el caso particular de que ambos operandos sean cadenas de caracteres, puede utilizarse también el operador +. No obstante, para evitar ambigüedades (sobre todo con variables de tipo ***Variant***) es mejor utilizar **&.**

El operador ***Like*** sirve para comparar dos cadenas de caracteres. La sintaxis para este operador es la siguiente:

```
Respuesta = Cadena1 Like Cadena2
```

donde la variable ***Respuesta*** será ***True*** si la ***Cadena1*** coincide con la ***Cadena2***, ***False*** si no coinciden y ***Null*** si ***Cadena1*** y/o ***Cadena2*** son ***Null***.

Para obtener más información se puede consultar el ***Help*** de ***Visual Basic***.

3.6 SENTENCIAS DE CONTROL

Las ***sentencias de control***, denominadas también *estructuras de control,* permiten tomar decisiones y realizar un proceso repetidas veces. Son los denominados bifurcaciones y bucles. Este tipo de estructuras son comunes en cuanto a concepto en la mayoría de los lenguajes de programación, aunque su sintaxis puede variar de un lenguaje de programación a otro. Se trata de unas estructuras muy importantes ya que son las encargadas de controlar el *flujo* de un programa según los requerimientos del mismo.

Visual Basic 6.0 dispone de las siguientes estructuras de control:

If ... Then ... Else
Select Case
For ... Next
Do ... Loop
While … Wend
For Each … Next

3.6.1 Sentencia IF ... THEN ... ELSE ...

Esta estructura permite ejecutar condicionalmente una o más sentencias y puede escribirse de dos formas. La primera ocupa sólo una línea y tiene la forma siguiente:

If condicion **Then** sentencia1 [**Else** sentencia2]
La segunda es más general y se muestra a continuación:

```
If condicion Then
sentencia(s)
[Else
sentencia(s)]
End If
```

Si ***condicion*** es ***True (verdadera)***, se ejecutan las sentencias que están a continuación de ***Then***, y si ***condicion*** es ***False (falsa)***, se ejecutan las sentencias que están a continuación de ***Else***, si esta cláusula ha sido especificada (pues es opcional). Para indicar que se quiere ejecutar uno de varios bloques de sentencias dependientes cada uno de ellos de una condición, la estructura adecuada es la siguiente:

```
If condicion1 Then
sentencias1
ElseIf condicion2 Then
sentencias2
Else
sentencia-n
End If
```

Si se cumple la ***condicion1*** se ejecutan las ***sentencias1***, y si no se cumple, se examinan secuencialmente las condiciones siguientes hasta ***Else***, ejecutándose las sentencias correspondientes al primer ***ElseIf*** cuya condición se cumpla. Si todas las condiciones son falsas, se ejecutan las sentencias-n correspondientes a ***Else***, que es la opción por defecto.

La **Figura 3.2** presenta esquemáticamente ambas formas de representar estas sentencias:

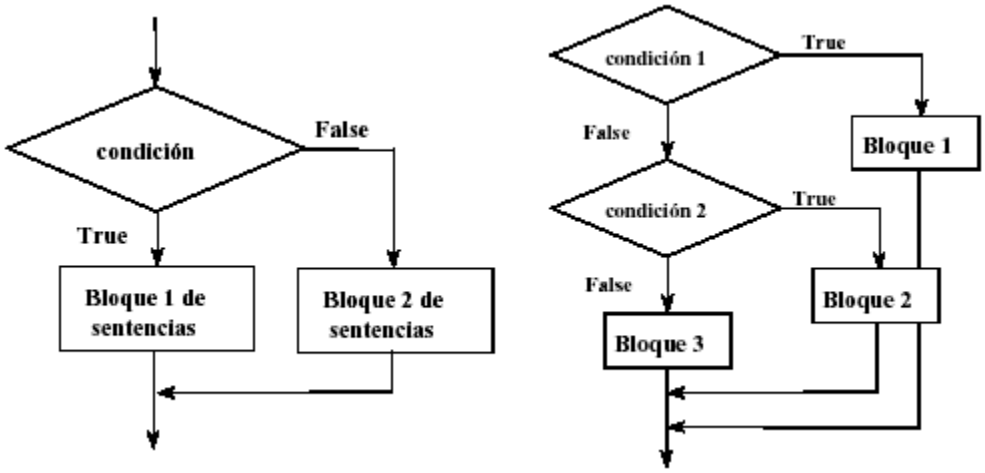


Figura 3.2. Bifurcaciones *If* e *If...Else*.

Por ejemplo,

```
Numero = 53 ’        Se inicializa la variable.
If Numero < 10 Then
Digitos = 1
ElseIf Numero < 100 Then
' En este caso la condición se cumple (True) luego se ejecuta lo
siguiente.
Digitos = 2
Else 'En el caso que no se cumplan los dos anteriores se asigna 3
Digitos = 3
End If
```

3.6.2 Sentencia SELECT CASE

Esta sentencia permite ejecutar una de entre varias acciones en función del valor de una expresión.
Es una alternativa a ***If ... Then ... ElseIf*** cuando se compara la misma expresión con diferentes valores. Su forma general es la siguiente:

```
Select Case expresion
Case etiq1
[ sentencias1]
Case etiq2
[ sentencias2]
Case Else
sentenciasn
End Select
```

donde ***expresion*** es una expresión numérica o alfanumérica, y ***etiq1***, ***etiq2***, ... pueden adoptar las formas siguientes:

1. *expresion*
2. *expresion* **To** *expresion*
3. **Is** *operador-de-relación expresion*
4. *combinación de las anteriores separadas por comas*

Por ejemplo,

```
Numero = 8 ’ Se inicializan las variable.
Select Case Numero ’ Se va a evaluar la variable Numero.
Case 1 To 5 ' Numero está entre 1 y 5.
Resultado = "Se encuentra entre 1 y 5"
' Lo siguiente se ejecuta si es True la expresión.
Case 6, 7, 8 ' Numero es uno de los tres valores.
Resultado = "Se encuentra entre 6 y 8"
Case Is = 9 , Is = 10 ' Numero es 9 ó 10.
Resultado = "El valor es 9 o 10"
Case Else ' Resto de valores.
Resultado = "El número no se encuentra entre 1 y 10"
End Select
```

Cuando se utiliza la forma *expresion* ***To*** *expresion*, el valor más pequeño debe aparecer en primer lugar.

Cuando se ejecuta una sentencia ***Select Case***, ***Visual Basic*** evalúa la ***expresion*** y el control del programa se transfiere a la sentencia cuya etiqueta tenga el mismo valor que la expresión evaluada, ejecutando a continuación el correspondiente bloque de sentencias. Si no existe un valor igual a la ***expresion*** entonces se ejecutan las sentencias a continuación de ***Case Else***.

3.6.3 Sentencia FOR ... NEXT

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

```
For variable = expresion1 To expresion2 [Step expresion3]
[sentencias]
Exit For
[sentencias]
Next [variable]
```

Cuando se ejecuta una sentencia ***For***, primero se asigna el valor de la ***expresion1*** a la variable y se comprueba si su valor es mayor o menor que la ***expresion2***. En caso de ser menor se ejecutan las sentencias, y en caso de ser mayor el control del programa salta a las líneas a continuación de ***Next***. Todo esto sucede en caso de ser la ***expresion3*** positiva. En caso contrario se ejecutarán las sentencias cuando la variable sea mayor que ***expresion2***. Una vez ejecutadas las sentencias, la variable se incrementa en el valor de la ***expresion3***, o en 1 si ***Step*** no se especifica, volviéndose a efectuar la comparación entre la variable y la ***expresion2***, y así sucesivamente.

La sentencia ***Exit For*** es opcional y permite salir de un bucle ***For ... Next*** antes de que éste finalice. Por ejemplo,

```
MyString="Informática "

For Words = 3 To 1 Step -1               ' 3 veces decrementando de 1 en 1.
      For Chars = Words To Words+4       '5 veces.
            MyString = MyString & Chars ' Se añade el número Chars al string.
      Next Chars                         ' Se incrementa el contador
      MyString = MyString & " "          ' Se añade un espacio.
Next Words
'El valor de MyString es: Informática 34567 23456 12345
```

3.6.4 Sentencia DO ... LOOP

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

```
' Formato 1:

Do [{While/Until} condicion]
   [ sentencias]
   [Exit Do]
   [ sentencias]
 Loop


' Formato 2:
Do
  [ sentencias]
  [Exit Do]
  [ sentencias]
Loop [{While/Until} condicion]
```

La sentencia opcional ***Exit Do*** permite salir de una bucle ***Do ... Loop*** antes de que finalice éste. Por ejemplo,

```
Check = True                    ' Se inicializan las variables.
Counts = 0
Do                              ' Empieza sin comprobar ninguna condición.
      Do While Counts < 20      ' Bucle que acaba si Counts>=20 o con Exit Do.
            Counts = Counts + 1 ' Se incrementa Counts.
            If Counts = 10 Then ' Si Counts es 10.
               Check = False    ' Se asigna a Check el valor False.
               Exit Do          ' Se acaba el segundo Do.
            End If
      Loop
Loop Until Check = False        ' Salir del "loop" si Check es False.
```

En el ejemplo mostrado, se sale de los bucles siempre con ***Counts = 10***. Es necesario fijarse que si se inicializa ***Counts*** con un número mayor o igual a 10 se entraría en un bucle infinito (el primer bucle acabaría con ***Counts = 20*** pero el segundo no finalizaría nunca, bloqueándose el programa y a veces el ordenador).

3.6.5 Sentencia WHILE … WEND

Esta sentencia es otra forma de generar bucles que se recorren mientras se cumpla la condición inicial. Su estructura es la siguiente:

```
While condicion
    [ sentencias]
Wend
```

Por ejemplo,

```
Counts = 0              ' Se inicializa la variable.
While Counts < 20       ' Se comprueba el valor de Counts.
    Counts = Counts + 1 ' Se incrementa el valor de Counts.
Wend                    ' Se acaba el bucle cuando Counts > 19.
```

En cualquier caso se recuerda que la mejor forma de mirar y aprender el funcionamiento de todas estas sentencias es mediante el uso del ***Help*** de ***Visual Basic***. Ofrece una explicación de cada comando con ejemplos de utilización.

3.6.6 Sentencia FOR EACH ... NEXT

Esta construcción es similar al bucle ***For***, con la diferencia de que la variable que controla la repetición del bucle no toma valores entre un mínimo y un máximo, sino a partir de los elementos de un array (o de una colección de objetos). La forma general es la siguiente:

```
For Each variable In grupo
      [ sentencias]
Next variable
```

Con arrays *variable* tiene que ser de tipo ***Variant***. Con colecciones *variable* puede ser ***Variant*** o una variable de tipo ***Object***. Esta construcción es muy útil cuando no se sabe el número de elementos que tiene el array o la colección de objetos.

3.7 ALGORITMOS

3.7.1 Introducción

Un ***algoritmo*** es en un sentido amplio una "*secuencia de pasos o etapas que conducen a la realización de una tarea*". Los primeros algoritmos nacieron para resolver problemas matemáticos.

Antes de escribir un programa de ordenador, hay que tener muy claro el algoritmo, es decir, cómo se va a resolver el problema considerado. Es importante desarrollar buenos algoritmos (correctos y eficientes). Una vez que el algoritmo está desarrollado, el problema se puede resolver incluso sin entenderlo.

Ejemplo: Algoritmo de Euclides para calcular el m.c.d. de dos números enteros A y B

1. Asignar a M el valor de A, y a N el valor de B.

2. Dividir M por N, y llamar R al resto.

3. Si R distinto de 0, asignar a M el valor de N, asignar a N el valor de R,volver a comenzar la etapa 2.

4. Si R = 0, N es el m.c.d. de los números originales.

Es muy fácil pasar a ***Visual Basic*** este algoritmo:

```
Dim a, b As Integer
a = 45: b = 63    ' Estos son los valores M y N
If a < b Then     ' Se permutan a y b
      temp = a : a = b : b = temp
End If
m = a : n = b : resto = m Mod n ' Mod devuelve el valor del resto
While resto <> 0 'Mientras el resto sea distinto de 0
  m = n: n = resto:
  resto = m Mod n
Wend
' La solución es la variable n. En este caso el resultado es 9
```

Si son necesarios, deben existir criterios de terminación claros (por ejemplo, para calcular seno(x) por desarrollo en serie se deberá indicar el número de términos de la serie). No puede haber etapas imposibles (por ejemplo: "imprimir el conjunto de todos los números enteros").

3.7.2 Representación de algoritmos

Existen diversas formas de representar algoritmos. A continuación se presentan algunas de ellas:

- ***Detallada***: Se trata de escribir el algoritmo en un determinado lenguaje de programación (lenguaje de máquina, ensamblador, fortran, basic, pascal, C, Matlab, Visual Basic, ...).

- ***Simbólica***: Las etapas son descritas con lenguaje próximo al natural, con el grado de detalle adecuado a la etapa de desarrollo del programa.

- ***Gráfica***: por medio de diagramas de flujo.

La sintaxis (el modo de escribir) debe representar correctamente la semántica (el contenido). La sintaxis debe ser clara, sencilla y accesible.

En cualquier caso e independientemente del tipo de representación utilizada lo importante es tener muy claro el algoritmo a realizar y ponerlo por escrito en forma de esquema antes

de ponerse a programarlo. Merece la pena pasar unos minutos realizando un esquema sobre papel antes de ponerse a teclear el código sobre un teclado de ordenador.

3.8 FUNCIONES Y PROCEDIMIENTOS

3.8.1 Conceptos generales sobre funciones

Las aplicaciones informáticas que habitualmente se utilizan, incluso a nivel de informática personal, suelen contener decenas y aún cientos de miles de líneas de código fuente. A medida que los programas se van desarrollando y aumentan de tamaño, se convertirían rápidamente en sistemas poco manejables si no fuera por la *modularización*, que es el proceso consistente en dividir un programa muy grande en una serie de módulos mucho más pequeños y manejables. A estos módulos se les suele denominar de distintas formas (*subprogramas*, *subrutinas*, *procedimientos*, *funciones*, etc.) según los distintos lenguajes. Sea cual sea la nomenclatura, la idea es sin embargo siempre la misma: dividir un programa grande en un conjunto de subprogramas o funciones más pequeñas que son llamadas por el programa principal; éstas a su vez llaman a otras funciones más específicas y así sucesivamente.

La división de un programa en unidades más pequeñas o funciones presenta –entre otras– las ventajas siguientes:

1. ***Modularización***. Cada función tiene una misión muy concreta, de modo que nunca tiene un número de líneas excesivo y siempre se mantiene dentro de un tamaño manejable. Además, una misma función (por ejemplo, un producto de matrices, una resolución de un sistema de ecuaciones lineales, ...) puede ser llamada muchas veces en un mismo programa, e incluso puede ser reutilizada por otros programas. Cada función puede ser desarrollada y comprobada por separado.

2. ***Ahorro de memoria y tiempo de desarrollo***. En la medida en que una misma función es utilizada muchas veces, el número total de líneas de código del programa disminuye, y también lo hace la probabilidad de introducir errores en el programa.

3. ***Independencia de datos y ocultamiento de información***. Una de las fuentes más comunes de errores en los programas de computador son los *efectos colaterales* o perturbaciones que se pueden producir entre distintas partes del programa. Es muy frecuente que al hacer una modificación para añadir una funcionalidad o corregir un

error, se introduzcan nuevos errores en partes del programa que antes funcionaban correctamente. Una función es capaz de mantener una gran independencia con el resto del programa, manteniendo sus propios datos y definiendo muy claramente la *interfaz* o comunicación con la función que la ha llamado y con las funciones a las que llama, y no teniendo ninguna posibilidad de acceso a la información que no le compete.

Continuará.....

Nota de Radacción: El lector puede descargar este capítulo y capítulos anteriores del curso desde la sección ***"Artículos Técnicos"*** en el sitio web de **EduDevices (www.edudevices.com.ar)**