
Visual
Basic
6.0
Programación Orientada a Objetos
Desarrollando Grupo Experto Bucarelly

VISUAL BASIC 6.0

Programación Orienta a Objetos
Desarrollando Grupo Experto Bucarelly

Primera edición

VISUAL BASIC 6.0

Programación Orienta a Objetos
Desarrollando Grupo Experto Bucarelly

Primera edición

CARLOS M. RODRÍGUEZ BUCARELLY
Autor de este libro
Maestro actual del Centro Educativo Divina Providencia
Ing. en Sistemas

PABLO A. RODRÍGUEZ BUCARELLY
Coautor de este libro
Encargado del departamento de monitoreo en Codetel
Ing. en Sistemas

Diseño de páginas
EDGAR H. SÁNCHEZ TAVERAS
Técnico en Informática

Revisión técnica
CARLOS A. MOREL PICHARDO
Lic. en Informática

Impresor
JOSÉ ENRIQUE GARCÍAS
Lic. en Informática

Agradecimientos

Agradezco primero a **Dios**, ser supremo, creador del universo y todas las cosas dentro de él.

Muchas personas ayudaron en la preparación de esta edición, pero agradezco principalmente los importantes comentarios de mi hermano Pablo A. Rodríguez Bucarelly.

Los siguientes revisores aportaron comentarios y sugerencias cuidadosas para el mejoramiento del manuscrito de esta edición: Ing. Carlos A. Morel Pichardo, Tec. Edgar H. Sánchez Taveras y Adm. Nelson M. Rodríguez Bucarelly.

Un agradecimiento especial a mi compañero de trabajo Lic. Juan Fco. Mena Mañon, que más que mi compañero lo considero como un padre, por sus grandes consejos y la gran confianza que ha depositado en mí para la elaboración de sus proyectos matemáticos.

Otros agradecimientos que son también muy especial son para nuestros soles Hna. Miledis y Hna. Mónica directora y subdirectora del centro educativo Divina Providencia quienes han permitido la distribución de este libro en su distinguido centro educativo.

C O N T E N I D O
R E D U C I D O

2004

Visual Basic 6.0

Siguiente página >

Siguiente página >

Siguiente página >

Siguiente página >

 Este libro ha sido creado con la finalidad de proporcionar la información necesaria para el manejo del programa **Visual Basic 6.0**, por motivo al alto costo de los libros de esta materia.

Santo Domingo, Republica Dominicana.

Por: Ing. Carlos Manuel Rodríguez Bucarelly
Enero del 2004.

A medida del paso de los años los lenguajes de programación han evolucionado considerablemente dando lugar a nuevos métodos de diseño de programas que facilitan al programador la tarea de diseñar aplicaciones complejas, que requieren de una gran cantidad de codificación y diseños de algoritmos para su desarrollo.

Existen distintos tipo de método de diseño de programas. El primer método de diseño que se empleaba para la creación de programas es el **Tipo Secuencial (bath)**. Un programa *secuencial* se desarrolla de forma ordenada, cada línea de código se ejecuta una por una, es decir, cada línea de código esta precedida por otra línea de código que ya fue ejecuta. A este tipo de programas se les llaman también *programas orientados a procedimientos* o *algoritmos* **(Procedural Languages)**.

Otros tipos de programas son los interactivos que exigen la intervención del usuario en tiempo de ejecución, ya sea para suministrar datos, o bien, para indicar al programa lo que debe hacer por medio de menús.

Por su parte los programas *orientados a eventos* son los programas típicos de **Windows**, tales como **Word**, **Excel**, **PowerPoint**, etc. Cuando uno de estos programas ha arrancado, lo único que hace es quedarse a la espera de alguna *acción* del usuario, que en este caso a dicha acción en la programación *orientada a event*os se le llama *evento.* Un *evento* es una acción que realiza el usuario hacia un objeto, por ejemplo, cuando el usuario hace clic en un botón de comando, esa acción de hacer clic en el botón se le llama ***evento Click.*** También cabe mencionar el evento **MouseMove** *(movimiento del ratón)* que ocurre cuando el usuario mueve el puntero del *mouse (ratón)* por cualquier objeto sobre una *ventana*.

Por ser considerada la *programación orientada a eventos* el método más fácil de programar, se ha seleccionado para la elaboración de este libro uno de los programas más popular para el diseño de aplicaciones orientas a eventos que es **Microsoft Visual Basic 6.0**.

Capitulo I

CONTENIDO

Capitulo I

1.1 ¿Qué es la programación orientada a objetos?

En **Visual Basic 6.0** y en otros programas, se le llama *objeto* a todo lo que se ve en una ventana típica de Windows; los objetos son por ejemplo un botón de comando, una caja de texto, una imagen, en general todo objeto visible que puedas ver en la pantalla.

Se les llaman *objetos* porque cada uno de ellos poseen *propiedades*, *eventos* y *métodos*. Un *botón de comando* tiene *propiedades* tales como: **Caption** (Titulo) que indica el texto que tiene el botón, también tiene las *propiedades***Width** (Anchura) y **Height** (Altura) que establecen la anchura y altura del botón.

Todos los objetos con que trabajemos en **Visual Basic 6.0** poseen *propiedades*, *métodos* y *eventos*, aunque algunos objetos pueden tener *propiedades*, *métodos* y *eventos* que otros objetos no pueden tener, por ejemplo, un objeto **TextBox** (caja de texto) tiene la propiedad **Text** (texto) que indica el texto que contiene la caja, de tal manera, es obvio que un botón de comando no pueda tener esta propiedad, así como el evento **Change** (Cambio) que poseen las cajas de textos que indican cuando el usuario esta cambiando el contenido de la caja, también esta claro que un botón de comando **(CommandButton)** no pueda tener este evento.

Resumiendo todo lo dicho anteriormente, se puede definir la **Programación Orientada a Objetos (POO)** como aquella en la que trabajamos con objetos visibles cada uno de los cuales poseen sus propios *eventos*, *métodos* y *propiedades*.

1.2 Programas orientados a eventos

Es lógico que para que un programa se pueda llamar *orientado a eventos* debe haber sido creado en un lenguaje de programación *orientado a objetos*, ya que cada *objeto* espera a algún evento que realice el usuario sobre él.

Los *programas orientados a eventos* son los programas típicos de Windows, Linux, Beos, que esperan a que el usuario realice alguna acción, ya sea con el *mouse* o con el *teclado* para realizar alguna función, por ejemplo, la calculadora de **Windows** espera a que el usuario haga clic (*evento clic*) con el mouse sobre uno de los botones que contienen los números para ponerlo en la caja de texto, o bien, espera a que el usuario pulse un número desde el teclado para ponerlo en la caja de texto.

1.3 Programas para el entorno de Windows

Visual Basic 6.0 está orientado a la realización de programas para **Windows**, pudiendo incorporar todos los elementos de este entrono informático: ventanas, botones, cajas de diálogo y de texto, botones de opción y de selección, barras de desplazamiento, gráficos, menús, etc.

Prácticamente todos los elementos de interacción con el usuario de los que dispone **Windows 95/98/XP/2000/NT** pueden ser programados en **Visual Basic 6.0** de un modo muy sencillo. En ocasiones bastan unas pocas operaciones con el ratón y la introducción a través del teclado de algunas sentencias para disponer de aplicaciones con todas las características de **Windows 95/98/XP/2000/NT**.

- 1.3.1 Modo de diseño y modo de ejecución

La aplicación **Visual Basic** de Microsoft puede trabajar de dos modos distintos: En *modo diseño* y en *modo de ejecución*. En *modo diseño* el usuario construye interactivamente la aplicación, colocando *controles* en el formulario, definiendo sus *propiedades*, y desarrollando funciones para gestionar los *eventos*.

La aplicación se prueba en *modo de ejecución*. En este caso el usuario actúa sobre el programa (introduce eventos) y prueba cómo responde el programa. Hay algunas *propiedades* de los *controles* que deben establecerse en modo de diseño, pero muchas otras pueden cambiarse en tiempo de ejecución desde el programa escrito en **Visual Basic 6.0**.

- 1.3.2 Formularios y Controles

Cada uno de los elementos gráficos que pueden formar parte de una aplicación típica de **Windows** es un tipo de *control*: botones, cajas de dialogo y de texto, cajas de selección desplegables, los botones de selección y de opción, las barras de desplazamiento horizontales y verticales, los gráficos, los menús, y muchos otros elementos son *controles* para **Visual Basic 6.0**. Cada *control* debe tener un **nombre** a través del cual se puede hacer referencia a el en el programa. **Visual Basic 6.0** asigna **nombres** por defecto a los *controles*. **Visual Basic** permite al usuario cambiar los *nombres por defecto*.

En **Visual Basic** 6.0 un *formulario* es una ventana. Un *formulario* puede ser considerado como una especie de contenedor para los *controles*. Una aplicación puede tener uno o varios *formularios* (ventanas), pero un único *formulario* puede ser suficiente para la creación de una aplicación sencilla. Los *formularios* deben también tener un *nombre* que permita hacerse referencia a él ó del él.

- 1.3.3 Objetos, Propiedades y Clase

A los *controles* que colocamos en un *formulario* que poseen *propiedades*, *métodos* y *eventos* se les llaman *objetos* y a las características propias de esos *objetos* se les llaman *propiedades*.

La **clase** es la entidad genérica a la que pertenece un *control*, por ejemplo, en un programa puede haber varios botones, cada uno de los cuales es un *objeto* que pertenece a una *clase* de los controles **(CommandButton)**. Cada *formulario* y cada tipo de *control* tienen un conjunto de *propiedades* que definen su aspecto gráfico (tamaño, color, posición en la ventana, tipo y tamaño de letra, etc.) y su forma de responder a las acciones (*eventos*) del usuario. Cada *propiedad* tiene un nombre que viene ya definido por el lenguaje.

Por lo general, las *propiedades* de un *objeto* son datos que tienen valores lógicos **(true, false)** o numéricos concretos, propios de ese *objeto* y distintos de las de otros *objetos* de su *clase*. Así pues, cada *clase*, tipo de *objeto* o *control* tienen su conjunto de propiedades, y cada *objeto* o *control* tienen valores determinados para las *propiedades* de su *clase*.

Casi todas las *propiedades* de los *objetos* pueden establecerse en tiempo de *diseño* y también casi siempre en tiempo de *ejecución*. En este segundo caso se accede a sus valores por medio de las sentencias del programa en forma análoga a como se accede a cualquier variable en un lenguaje de programación. Para ciertas *propiedades* ésta es la única forma de acceder a ellos. Por supuesto **Visual Basic 6.0** permite crear distintos tipos de variables, como verá más adelante.

Para acceder a una *propiedad* de un *objeto* se hace por medio del *nombre del objeto* **(name),** seguido de un *punto* (**.**) y el *nombre de la propiedad*. Por ejemplo, para cambiar el color de fondo de una caja de texto (TextBox) llamada Text1 se haría de la siguiente manera:

Ejemplo:
Text1**.BackColor** = vbRed

Donde **Text1** es el nombre del *control*, **BackColor** es el nombre de la propiedad que permite cambiar el color del fondo de la caja de texto y **vbRed** es el valor que se le ha asignado a la *propiedad* **BackColor**, que en este caso es el color rojo de **Visual Basic 6.0**.

- 1.3.4 Nombre de objetos

En principio cada *objeto* de **Visual Basic 6.0** debe tener un nombre, por medio del cual se hace referencia a dicho *objeto*. El nombre **(name)** puede ser el que el usuario desee, e incluso **Visual Basic 6.0** proporciona *nombres por defecto* para los diversos *controles*. Estos *nombres por defecto* hacen referencia al tipo de *control* y van seguido de un número que se incrementa a medida que se van introduciendo más *controles* de ese mismo tipo en el formulario, por ejemplo, **Text1** para una caja de texto, **Text2** para otra caja de texto, **Command1** para un botón de comando, **Command2** para otro botón de comando, etc.

Los *nombres por defectos* no son adecuados porque sólo hacen referencia al tipo de *control*, pero no al uso que de dicho control está haciendo el programador. Por ejemplo, si se agregan dos botones a una aplicación **(Command1, Command2)** uno para salir de la aplicación y otro para guardar los cambios hechos en la aplicación, sería recomendable que el botón de **cerrar** lleve por nombre "cmdCerrar" y el botón de **guardar** "cmdGuardar" y no los *nombres por defecto* **Command1** y **Command2**, ya que en ocasiones no sabremos para que utilizamos uno y para que utilizamos el otro.

Para asignar los nombres a los *controles* existe una convención ampliamente aceptada que es la siguiente: se utilizan siempre tres letras en minúscula que indican el tipo de *control*, seguido de otras letras (la primera en mayúscula) libremente escogidas por el usuario, que tienen que hacer referencia al uso que se va a dar a ese *control*.

La **tabla 1.1** muestra las abreviaturas de los *controles* más usuales, junto con la nomenclatura inglesa de la que se derivan.

Abreviatura	Control	Abreviatura	Control
chk	CheckBox	cbo	Combo y Drop-ListBox
cmd	CommandButton	dir	DirListBox
drv	DriveListBox	fil	FileListBox
frm	Form	fra	Frame
hsb	HorizontalScrollBar	img	Image
lbl	Label	lin	Line
lst	List	mnu	Menu
opt	OptionButton	pct	PictureBox
shp	Shape	txt	TExtEditBox
tmr	Timer	vsb	VerticalScrollBar

Tabla 1.1. Abreviaturas para los controles más usuales.

- 1.3.5 Eventos

Ya se ha dicho que las acciones del usuario sobre un programa se llaman *eventos*. Son *eventos* típicos: hacer clic sobre un botón, el hacer doble clic sobre un fichero para abrirlo, el arrastrar un icono, el pulsar una tecla o combinación de teclas, el elegir una opción de un menú, el escribir en una caja de texto o simplemente mover el mouse.

Cada vez que se produce un *evento* sobre un determinado tipo de *control*, **Visual Basic 6.0** arranca una determinada *función* o *procedimiento* que realiza la acción programada por el usuario para ese *evento* concreto. Estos *procedimientos* se llaman con un nombre que se forma a partir del nombre del *objeto* y el nombre del *evento*, separados por el carácter (_) underscore. Por ejemplo, el *evento* clic de un botón de comando:

```
Private Sub Command1_Click ( )

End Sub
```

Donde **Private Sub** indica la declaración del procedimiento, **Command1** especifica el nombre del *control*, el carácter (_) underscore indica la separación entre el nombre del *control* y el nombre del *evento*, **Click** es el nombre del *evento* que especifica que la acción de ese procedimiento se ejecutará cuando el usuario haga clic sobre el botón, los paréntesis **( )** se utilizan para otras declaraciones que veremos más adelante y **End Sub** indica el fin del *procedimiento*.

- 1.3.6 Métodos

Los *métodos* son funciones que también son llamadas desde el programa, pero a diferencia de los *procedimientos* no son programadas por el usuario, sino que vienen ya pre-programadas con el lenguaje de programación. Los *métodos* realizan tareas típicas, previsibles y comunes para todas las aplicaciones. Cada tipo de *objeto* o de *control* tienen sus propios *métodos*. Por ejemplo, los formularios poseen un *método* llamado **Hide** que permite ocultar el formulario y otro *método* llamado **Show** que permite mostrarlo después de haber sido ocultado.

Para hacer referencia a un *método* basta con indicar el *nombre del objeto* **(name)** y el *nombre del método* separados por un *punto.*

Ejemplo: Para ocultar un formulario

Form1.Hide

Dónde **Form1** indica el nombre del *control* que en este caso es un formulario y **Hide** especifica el nombre del *procedimiento.*

1.5 Preguntas para contestar

1. ¿Qué es la programación orientada a objetos?
2. ¿Qué son los objetos, eventos y propiedades?
3. Diga la diferencia entre eventos y métodos.
4. ¿Qué indica la diferencia entre un tipo de control y otro tipo de control?
5. Mencione algunos programas de Windows orientado a eventos.
6. Mencione algunos eventos de algunos controles.
7. ¿Cuál es la diferencia entre el modo de diseño y el modo de ejecución?
8. ¿Qué son los formularios?
9. ¿Qué son los controles?
10. Para que se utiliza el punto en una línea de comando.
11. ¿Porque no es recomendable dejar los nombres por defectos a los controles de Visual Basic 6.0?
12. Diga las abreviaturas para los controles: CheckBox, CommandButton, Label, Shape, List, Timer, Image.

Capitulo II

CONTENIDO

Capitulo II

2.1 ¿Qué es Visual Basic 6.0?

Visual Basic 6.0 es una excelente herramientas de programación que permite crear aplicaciones para **Windows 95/98/2000/XP/NT**. Con ella se puede crear desde una simple calculadora hasta una hoja de cálculo de la talla de Excel, o un procesador de texto como Word o bien, cualquier aplicación que se le ocurra al programador.

Este programa permite crear ventana, botones, menús y cualquier otro elemento de Windows de una forma fácil e intuitiva.

2.2 El entorno de Visual Basic 6.0

Cuando se arranca **Visual Basic 6.0** aparece en la pantalla una configuración similar a la mostrada en la siguiente figura:

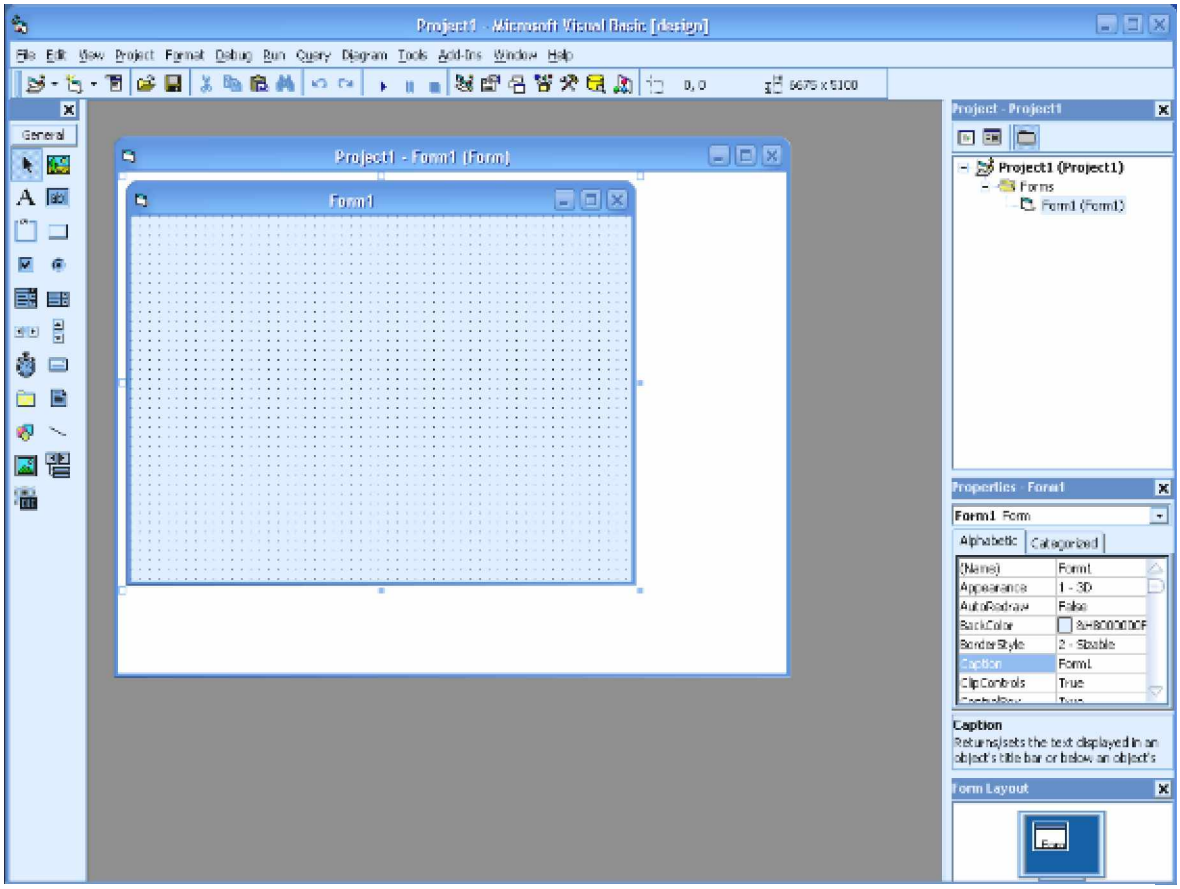


Figura 2.1. Entorno de Desarrollo de Visual Basic 6.0

En ella se pueden distinguir los siguientes elementos:

1.- **La barra de titulo**, la **barra de menús** y la **barra de herramientas estándar**.
2.- **Barra de herramientas no estándar** (Toolbox) con los *controles* más comunes.
3.- ***Formulario*** (Form) en gris, en el que se colocan los *controles*. Está dotado de una rejilla (grid) para facilitar la alineación de los controles en el formulario.
4.- **Ventana de proyecto** (Project), que muestra los formularios y otros módulos de programas que forman parte de la aplicación.
5.- **Ventana de propiedades** (Properties), en la que se pueden ver las propiedades de un objeto seleccionado sobre el formulario.
6.- Ventana **Form Layout**, que permite determinar la forma en que se vera la aplicación cuando comience a ejecutarse.

Existen otros elementos tales como: la ventana para **Edición de Códigos (Code Editor)** y la ventana **Depurador** ó **Debugger** para ver valores en variables en tiempo de ejecución. Todo este conjunto de herramientas y de ventanas es lo que se llama un Entorno Integrado de Desarrollo o IDE (Integrated Development Environment).

- 2.2.1 La barra de menús y la barra de herramientas estándar

Figura 2.2. Barra de menús de Visual Basic 6.0

La barra de menús de **Visual Basic 6.0** resulta similar a la de cualquier otra aplicación de **Windows**, tal y como aparece en la figurar 2.2.

Figura 2.3. Barra de herramientas estándar

La **Barra de Herramientas Estándar** aparece debajo de la barra de menús, que permite acceder a las opciones más importantes de los menús. En **Visual Basic 6.0** existen cuatro barras de herramientas: *Debug*, *Edit*, *FormEditor* y *Estándar*, por defecto sólo aparece la **barra de herramientas estándar**, aunque en la **Figura 2.1** se muestran las cuatro. Haciendo clic con el botón derecho del mouse sobre cualquier parte de la *barra de herramientas*, aparece un menú contextual con el que se puede hacer aparecer y ocultar cualquiera de las barras.

Algunos de los menús de la *barra de menús* tienen muy pocas novedades, es decir, algunos incluyen las opciones típicas de los menús de cualquier aplicación.

El menú **File** tiene pocas novedades. Lo más importante es la distinción entre *proyectos*. Un *proyecto* reúne y organiza todos los ficheros que componen el programa o aplicación. Estos ficheros pueden ser *formulario*, *módulos*, *clases*, *recursos*, etc. **Visual Basic 6.0** permite tener más de un proyecto abierto simultáneamente, lo cual puede ser útil en ocasiones. Con el comando **Add Project…** se añade un nuevo proyecto en la ventana **Project Manager**. Con los comandos **Open Proyect…** o **New Project** se abre o se crea un nuevo proyecto, pero cerrando el o los *proyectos* que estuvieran abiertos previamente. En este menú está el comando **Make ProjectName.exe…**, que permite crear ejecutables de los *proyectos*.

Tampoco el menú **Edit** aporta cambios importantes sobre lo que es lo habitual. Por el contrario el menú **View**, generalmente de poca utilizada, es bastante propio de **Visual Basic 6.0**. Este permite hacer aparecer en pantalla las distintas ventanas del entorno de desarrollo, así como acceder a un *formulario* o al código relacionado con un *control* (que también aparece al hacer doble clic sobre él), y manejar *funciones* y *procedimientos*.

El menú **Proyect** permite añadir distintos tipos de elementos a un proyecto. **Con Project Properties…** se puede elegir el tipo de proyecto y determinar el formulario con el que se arrancará la aplicación (*Startup Object*). Con el comando **Components** se pueden añadir nuevos controles a la *barra de herramientas no estándar* (Toolbox) que aparece a la izquierda de la pantalla.

En menú **Tools** se encuentran los comandos para arrancar el **Menu Editor** y para establecer opciones del programa. En **Tools/Opcion…** se encuentras opciones relativas al proyecto en el que se trabaja.

Por ultimo, la **ayuda (help)** (siempre imprescindible y en el cado de **Visual Basic 6.0** particularmente muy bien hecha) que se encuentra en el menú **Help**, se basa fundamentalmente en una clasificación temática ordenada de la información disponible **(Contents),** en una clasificación alfabética de la información **(Index)** y en la búsqueda de información sobre algún tema por el nombre **(Search).**

- 2.2.2 La barra de herramientas no estándar (Toolbox)

Figura 2.3

La figura 2.3 muestra la *barra de herramientas no estándar* (Toolbox), que incluye los *controles* con los que se puede diseñar la pantalla de la aplicación. Estos *controles* son por ejemplo, botones, etiquetas, cajas de texto, zonas gráficas, etc.

Para introducir un *control* en el *formulario* simplemente hay que hacer clic con el botón izquierdo del mouse sobre el *control* deseado y colocarlo en el formulario con la posición y el tamaño deseado. Haciendo doble clic sobre el *control* es también otra forma de colocar el *control* en el *formulario*, quedando este ubicado en el centro del *formulario*.

El número de *controles* que pueden aparecer en esta ventana varían con la configuración del sistema. Para introducir nuevos componentes se utiliza el comando **Components…** del menú **Proyect**, con el cual se abre el cuadro de dialogo mostrado a continuación en la figura 2.4.

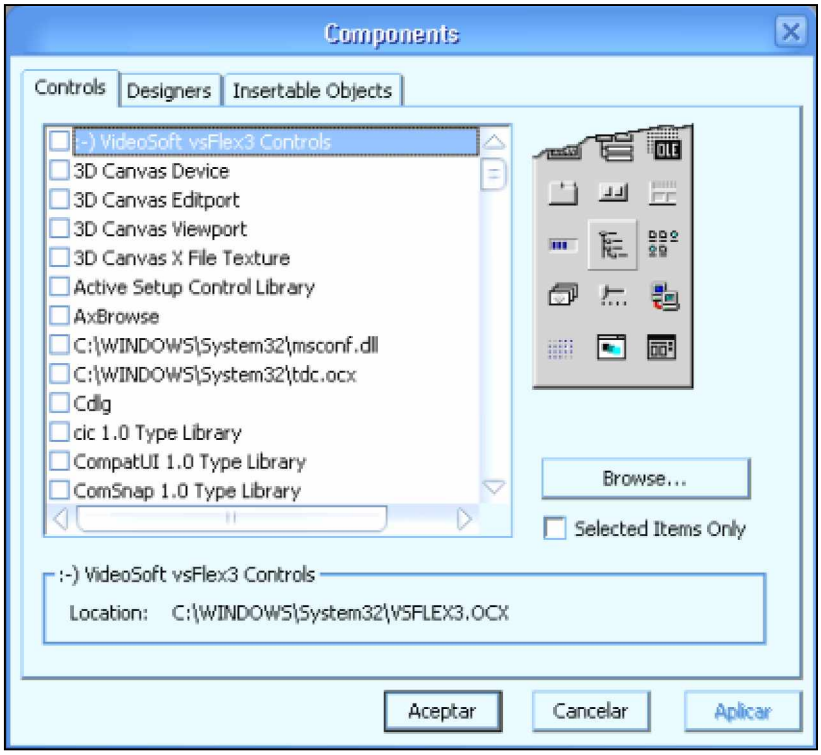


Figura 2.4 Cuadro de dialogo **Components**

- 2.2.3 Formularios (Forms)

Los *formularios* son las zonas de la pantalla sobre las que se diseña el programa y sobre los que se sitúan los *controles* o *herramientas* del **Toolbox**. Al ejecutar el programa, el *Form* se convertirá en la ventana de la aplicación donde aparecerán los botones, las cajas de texto, los gráficos, etc. En la **Figura 2.5** se muestra un típico *formulario* de **Visual Basic 6.0**.

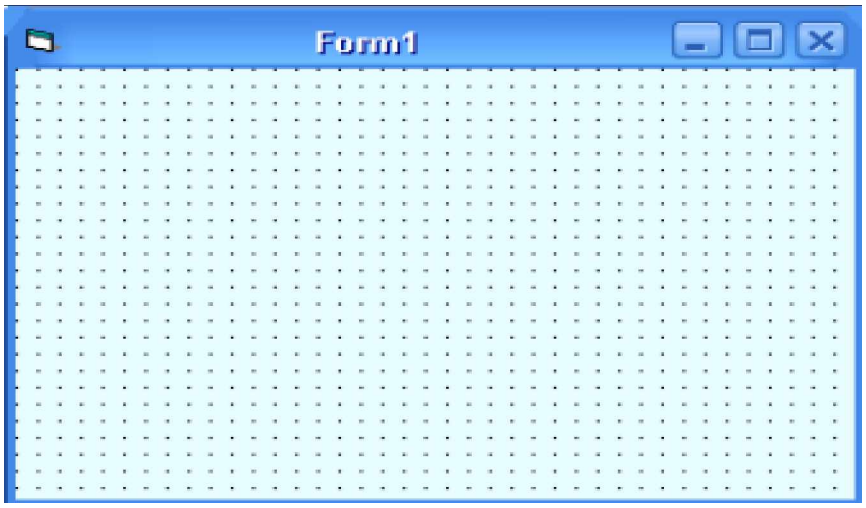


Figura 2.5. Formulario de Visual Basic 6.0

El conjunto de puntos que aparecen sobre el formulario se llama *malla* o *retícula* **(grid)** que permite alinear los *controles* manualmente de una forma precisa, evitando tener que introducir coordenadas continuamente. Esta *malla* sólo será visible en el proceso de diseño del programa; al ejecutarlo la *malla* automáticamente desaparece.

- 2.2.4 La ventana de proyecto (Project)

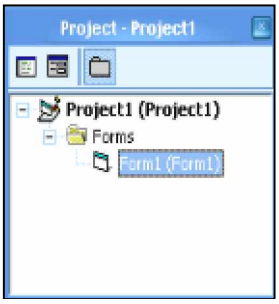


Figura 2.6 Ventana de proyecto

La *ventana de proyect* permite acceder a los distintos *formularios* y *módulos* que componen el *proyecto*. Desde ella se puede ver el diseño gráfico de dichos formularios, y también permite editar el código que contienen.

- 2.2.5 La ventana de propiedades (Properties)

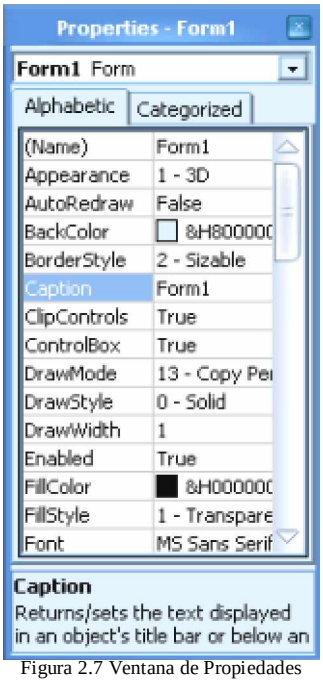


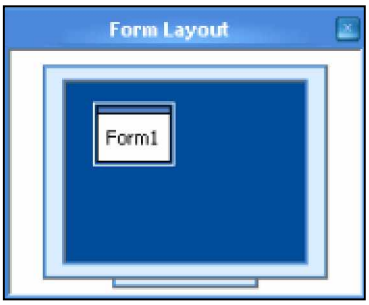
Figura 2.7 Ventana de Propiedades

En la *ventana de propiedades* se muestran todas las propiedades de un *control* seleccionado en la aplicación. Mediante esta ventana es posible cambiar los valores de las *propiedades* de cada uno de los *controles* indicando así sus características antes de ejecutar la aplicación.

A medida que se seleccionan los *controles* en la aplicación, se puede apreciar que las propiedades que antes mostraba la ventana desaparecen y muestran las *propiedades del control* que ha sido seleccionado actualmente.

La cantidad de *propiedades* que aparecen en la *ventana de propiedades*, depende de las características propias de ese *control*.

- 2.2.6 La ventana Form Layout



En esta ventana se muestran todos los *formularios* de una aplicación, donde es posible seleccionar cada uno de ellos y ubicarlos en la posición deseada de la pantalla. El *formulario* actual, es decir, con el cual se trabaja, es el *formulario* que aparece seleccionado en la ventana de **Form Layout**.

2.3 Creación de programas ejecutables

Una vez finalizada la programación de la nueva aplicación, la siguiente tarea suele consistir en la creación de un programa ejecutable para su distribución e instalación en cuantos ordenadores se desee, incluso aunque en ellos no este instalado **Visual Basic 6.0**.

Para crear un programa ejecutable se utiliza el comando **Make ProjectName.exe…** del menú **File**. De esta manera se genera un fichero cuya extensión será (.EXE). Para que este programa funcione en un ordenador solamente se necesita que el fichero MSVBVM60.DLL esté instalado en el directorio de **C:\Windows\System o C:\WinNT\System32**.

En el caso de proyectos más complejos en los que se utilicen muchos *controles* pueden ser necesarios más ficheros, la mayoría de ellos con extensiones **.ocx**, .**vbx** o **.dll**. Para saber en cada caso cuales son los ficheros necesarios, se puede consulta el fichero **.vbp** que contiene la descripción completa del proyecto. Casi todos esos ficheros se instalan automáticamente al instalar el compilador de **Visual Basic 6.0** en el ordenador.

2.4 La ayuda (help) de Visual Basic 6.0

Visual Basic 6.0 dispone de un **Help** excelente, como la mayoría de las aplicaciones de Microsoft. En esta nueva versión la ayuda se ofrece a través de una interfaz de usuario similar a la de *Internet Explorer*. Estando seleccionado un *control*, una *propiedad* o un *formulario*, o estando seleccionada una *palabra clave* en la *ventana de código*, esta ayuda se puede utilizar de modo sensible al contexto pulsando la tecla [F1].

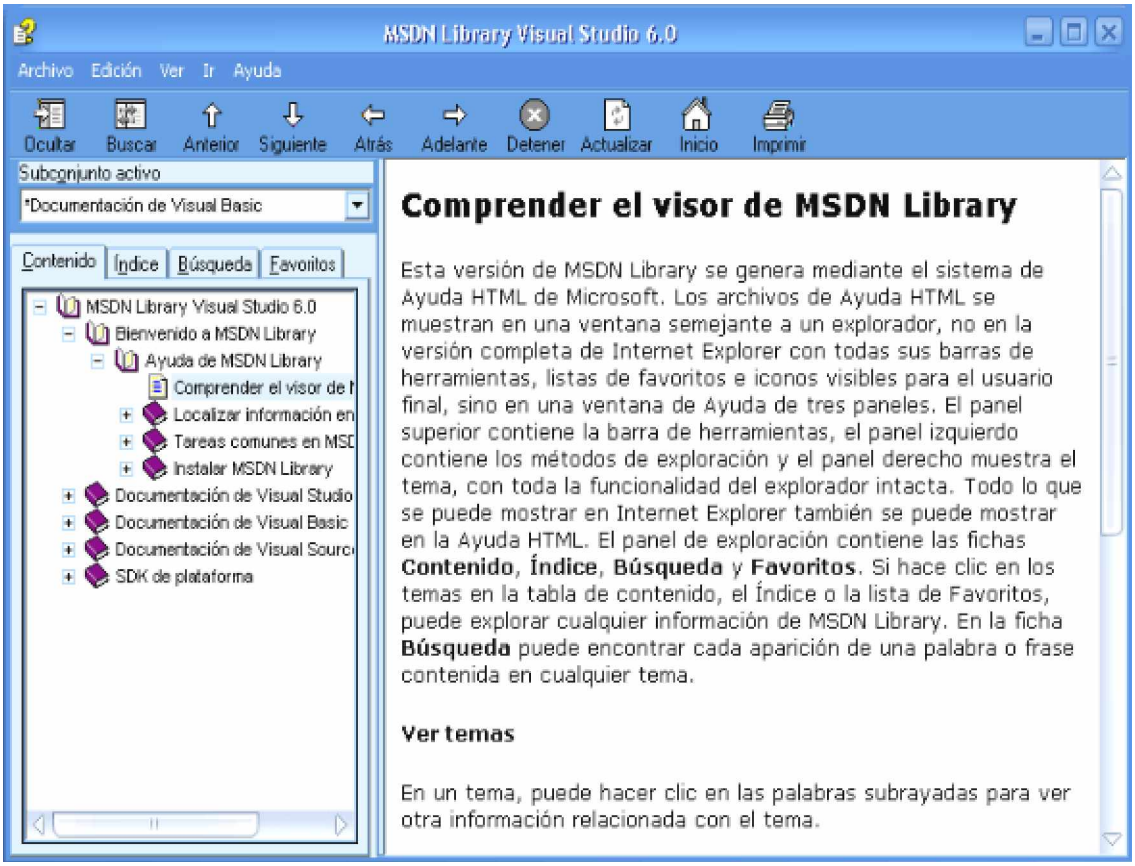


Figura 2.9 Ayuda de Visual Basic 6.0

Otra forma de acceder a la ayuda es mediante las opciones del *menú* **Help**. Así, mediante la opción *Index* se puede obtener información sobre muchos términos relacionados con **Visual Basic 6.0**.

Una vez obtenida la ayuda sobre un término seleccionado se pueden encontrar temas relacionados con ese término en la opción **See Also**. En caso de que se haya solicitado ayuda sobre un determinado tipo de *control*, se podría acceder también a la ayuda obtener todos y cada uno de sus *propiedades, eventos y métodos* con las opciones **Properties, Methods y Events**, respectivamente.

2.5 Utilización del Code Editor

El *Editor de Código* de **Visual Basic 6.0** es la ventana en la cual se escriben las sentencias del programa. Esta ventana presenta algunas características muy interesantes que conviene conocer para sacar el máximo partido de la aplicación.

Para abrir la ventana del *editor de código* se elige **Code** en el menú **View**. También se abre haciendo clic en el botón **View Code** de la *ventana de proyecto* **(Project)**, o haciendo doble clic en el *formulario* o cualquiera de sus *controles*. La **Figura 2.10** muestra un aspecto típico de la *ventana de código.*

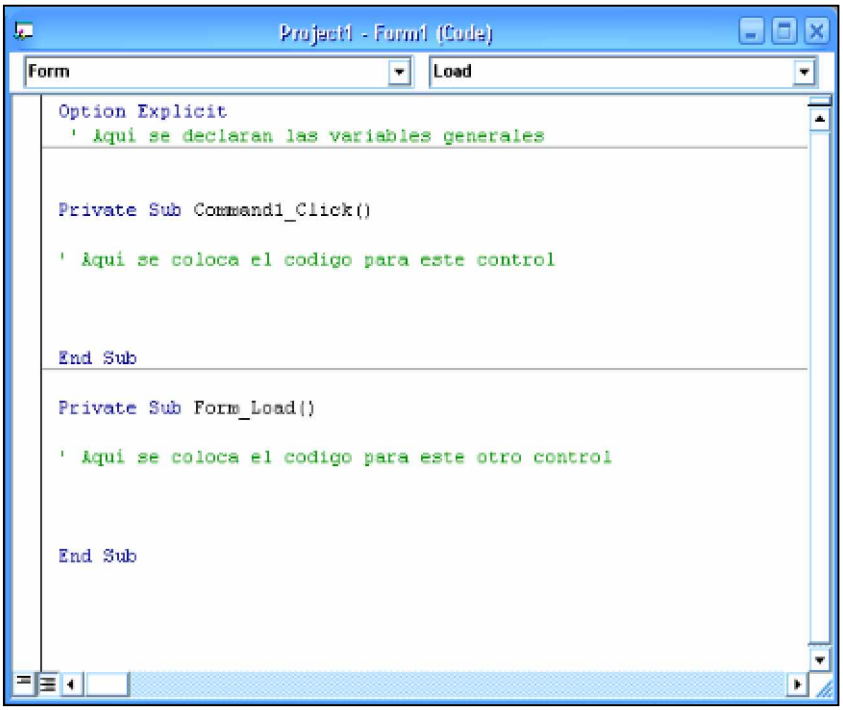


Figura 2.10. Ventana de Code Editor

En primer lugar, el **Code Editor** utiliza un código de colores para hacer diferencia entre líneas de código. Así, el código escrito por el usuario aparece en negro, las palabras clave de **Basic** en azul, los comentarios en verde, los errores en rojo, etc. Esta simple ayuda visual permite detectar y corregir problemas con más facilidad.

En la parte superior de la ventana aparecen dos *listas desplegables*. La de la izquierda corresponde a los distintos controles del formulario, el cual puede seleccionarse desde esta lista y de esta manera modificar su código. La *lista desplegable* de la derecha muestra los distintos *procedimientos* que se corresponden con el elemento seleccionado en la lista de la izquierda.

En la parte superior mostrada en la figura 2.10 encontramos la declaración **Option Explicit** que permite obliga al programador a declarar todas las variables que valla a usar, si utiliza una variable que no ha declarado el programa abortará con un mensaje de error.

También encontramos dos *procedimientos*, uno para uno para el **evento Click** de un botón de comando y otro para el **evento Load** del formulario. Dentro de estos *procedimientos* he agregado algunos comentarios que aparecen de color verde para indicar que esta es la zona donde debe escribirse el código para cada *procedimiento*.

En esta ventana aparecen dos barras de desplazamiento, una vertical y una horizontal que permiten desplazar el contenido de la ventana para observar los códigos no visibles debido al reducido tamaño de la ventana y al la gran cantidad de código proporcionado por el usuario para la aplicación.

2.6 Utilización del Debugger

Cualquier programador con un mínimo de experiencia sabe que una parte muy importante del tiempo destinado a la elaboración de un programa se destina a la **detección** y **corrección de errores**. Casi todos los entornos de desarrollo disponen hoy en día de potentes herramientas que facilitan la depuración de los programas realizados. La herramienta más utilizada para ellos es el *Depurador* o **Debugger**. La característica principal del **Debugger** es que permite ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando cada momento el valor de cada una de las variables. De esta manera se facilita enormemente el descubrimiento de las fuentes de errores.

- 2.6.1 Ejecución controlada de un programa

Para ejecutar **parcialmente** un programa se pueden utilizar varias formas. Una de ellas consiste en incluir **breakpoints** (puntos de parada de la ejecución) en determinadas líneas de código. Los **breakpoints** se indican con un punto grueso en el margen y un cambio de color de línea, tal como se ve en la figura 2.11. El colocar un **breakpoint** en una línea de código implica que la ejecución del programa se detendrá al llegar a esa línea. Para insertar **breakpoint** en una línea del código se utiliza la opción **Toggle Breakpoint** del menú **Debug**, con el botón del mismo nombre o pulsando la tecla [F9], estando el cursos posicionado sobre la línea de código. Para borrarlo se repite la misma operación.

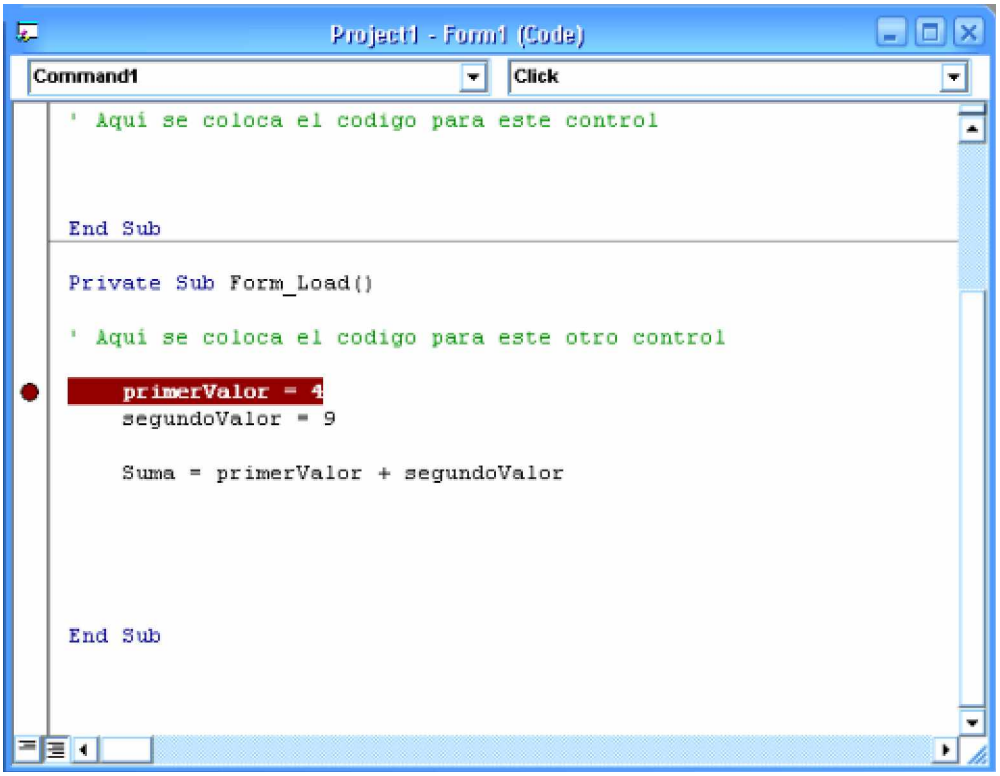


Figura 2.11. Utilización del Debugger

2.7 Preguntas para contestar

1.- Mencione los sistemas para los cuales **Visual Basic 6.0** puede crear aplicaciones.

2.- Mencione los elementos del entorno de programación **Visual Basic 6.0**.

3.- ¿Cuál es la diferencia entre la barra de herramientas estándar y la no estándar?

4.- ¿Qué encontramos en la barra de herramientas no estándar (ToolBox)?

5.- Mencione las barras de herramientas que existen en **Visual Basic 6.0** y diga cual de ella aparece por defecto.

6.- ¿Cuáles comandos se destacan en el menú **File**?

7.- ¿Cuál de todos los menús posee características generalmente propias de **Visual Basic 6.0**?

8.- ¿Qué son los formularios?

9.- ¿Qué encontramos en la ventana de proyecto (**Project**)?

10.- ¿Qué presenta la ventana de propiedades (**Properties**)?

11.- ¿En que consiste la creación de programas ejecutables en **Visual Basic 6.0**?

12.- Hable del archivo **MSVBVM60.DLL**.

13.- ¿Qué es el editor de código (**Code Editor**)?

14.- ¿Qué representa el color verde en la ventana del editor de código de **Visual Basic 6.0**?

15.- ¿Qué representa el color rojo en la ventana del editor de código de **Visual Basic 6.0**?

16.- ¿Qué representa el color azul en la ventana del editor de código de **Visual Basic 6.0**?

17.- ¿A que se refiere la depuración o **Debugger**?

18.- Hable de la ejecución controlada de un programa con **Debugger**.

Capitulo III

CONTENIDO

Capitulo III

3.1 Introducción a los controles más usuales

Los *controles* más usuales son aquellos que usamos con mayor frecuencia al momento de crear una aplicación. Estos *controles* son: los botones de comando, los botones de opción, las cajas de texto, las etiquetas, las barras de desplazamiento, las listas, las cajas combinadas, los botones de comprobación, etc.

- 3.1.1 Botón de comando (CommandButton)

Los *botones de comando* son aquellos botones típicos que vemos siempre en las aplicaciones de Windows que realizan una operación en especifico, por ejemplo, un botón para "Cancelar", un botón para "Salir", un botón para "Imprimir", etc.

Estos botones poseen una gran cantidad de propiedades, métodos y eventos que definiremos a continuación. Es importante recordar que todos los valores de las propiedades de un *control* seleccionado en la aplicación pueden ser observados y modificados desde la *ventana de propiedades* (Properties).

- 3.1.1.1 Propiedades de los botones de comando (CommandButton)

Antes de describir las propiedades más usadas de los *botones de comando*, abra un nuevo proyecto desde el menú **File** e inserte un *botón de comando* (CommandButton)  de la *Barra de herramientas no estándar* (ToolBox).

Propiedad	Descripción
Name	Se utiliza para asignarle el nombre al *control*. Este nombre permite hacer referencia al *control*.
	Ejercicio: Haga clic en el *control* o en el *botón de comando*, busque la propiedad **(Name)** en la *ventana de propiedades* y borre el valor por defecto que en este caso es *Command1* y escriba "cmdSalir" que será el nuevo *nombre del control*.

Propiedad	Descripción
BackColor	Cambia el color del botón. Para que el botón tome el color seleccionado debe establecer el valor "**1- Graphical**" en la propiedad **Style** del botón de comando.
	Ejercicio: Busque la propiedad **Style** del botón de comando y seleccione el valor "**1- Graphical**", luego ubíquese en la propiedad **BackColor** y seleccione el color deseado para el botón. El botón toma el color seleccionado.
Caption	Esta propiedad permite establecer el texto que aparece escrito en el botón de comando. Si utiliza el carácter **(&) ampersand** delante de cualquier carácter del texto escrito, ese carácter permite acceder a la función del botón con solo presionar la tecla *control (ctrl.)* más el carácter que tiene colocado el **ampersand** como si hubiese hecho clic sobre el.
	Ejercicio: Seleccione la propiedad **Caption** del botón de comando y escriba **&Salir**. El ampersand utilizado delante de la letra "**S**" indica el acceso directo del botón. Inmediatamente el botón aparece con el texto "**Salir**"
DisabledPicture	Establece la imagen que aparecerá en el control cuando este esté deshabilitado. Es decir, cuando la propiedad **Enabled** este establecida a **False**.
Enabled	Habilita o deshabilita el *control*, es decir, indica si el botón responderá a los eventos del usuario. Si el valor de esta propiedad esta en **False**, el botón no responderá a ninguna acción que el usuario haga sobre él.
Font	Permite cambiar el tipo de fuente del texto que aparece en el botón.
Height y Width	Permite cambiar la altura y anchura del botón. Donde **Height** representa la altura y **Width** la anchura expresada en *Twips* (unidad de medida de la pantalla).
Left y Top	Establece la posición izquierda y superior del control sobre su contenedor. Donde **Left** indica la posición a la izquierda y **Top** la posición superior del control.
Picture	Asigna un Bitmap (Imagen) al botón de comando siempre y cuando el valor de la propiedad **Style** del botón de comando este establecido a "**1- Graphical**"

Propiedad	Descripción
TabIndex	Indica el orden o el número de orden a que le corresponde el *control* recibir el foco (focus) a mediada que se pulse la tecla **Tab** sobre los *controles* de la aplicación.
Visible	Establece si el botón estará o no visible cuando se ejecute la aplicación. Puede tomar los valores **True** o **False** (verdadero o falso).
ToolTipText	Establece el texto contextual que aparece cuando se coloca el puntero del mouse sobre el *control*.

- 3.1.1.2 Eventos sobre los botones de comando

Se ha dicho que los *eventos* son las acciones que espera el *control* que el usuario realice sobre el. Los *eventos* deben ser seleccionados desde la ventana del *Editor de Código* que aparece cuando se hace doble Clic sobre el control. Los *eventos* para el control seleccionado aparecen en la esquina superior derecha del *Editor de Código* (Code Editor) en una lista desplegable.

Evento	Descripción
Click	Ocurre cuando el usuario presiona y suelta el botón izquierdo del mouse (ratón) sobre el *control*.
	Ejercicio: Haga doble Clic en el botón de comando y seleccione el *evento click*, por defecto aparece seleccionado. Escriba la línea de código siguiente: MsgBox (“Usando los eventos de los controles más usuales.”) Ejemplo: **Private Sub** cmdSalir_Click( ) ‘ La sintaxis End permite salir de la aplicación **End** **End Sub** ---------- **Nota:** Después de haber escrito el código corra la aplicación presionando la tecla **[F5]** y haga clic sobre el botón. El programa abortará de inmediato.
DragDrop	Ocurre cuando se completa una operación de arrastrar y colocar como resultado de arrastrar un control sobre un objeto y soltar el botón del mouse o usar el método **Drag** con su argumento de acción establecido a 2 (**Drop**).

Evento	Descripción
DragOver	Ocurre cuando una operación de arrastrar y colocar está en curso. Puede usar este *evento* para controlar el puntero del mouse a mediada que entra, sale o descansa directamente sobre un destino dado.
GotFocus	Ocurre cuando el objeto recibe el enfoque, ya sea mediante una acción del usuario, como tabular o hacer clic en el objeto, o cambiar el enfoque en el *código* mediante el método **SetFocus**.
KeyDown, KeyUp	Ocurre cuando el usuario presiona o suelta una tecla mientras en objeto tiene el enfoque.
KeyPress	Ocurre cuando el usuario presiona y suelta una tecla al igual que KeyDown y KeyUp.
LostFocus	Ocurre cuando el control pierde el enfoque.
MouseDown, MoseUp	Ocurren cuando el usuario presiona (MouseDown) o suelta (MouseUp) un botón del mouse.
MouseMove	Ocurre mientras el usuario mueve el puntero del mouse sobre el *Control*.

- 3.1.1.3 Métodos de los botones de comando

Los *métodos* son las acciones que el *control* puede realizar sobre si mismo sin la necesidad de la intervención del usuario, por medio de códigos o algún evento.

Método	Descripción
Drag	Inicia, termina o cancela una operación de arrastre de cualquier control.
Refresh	Fuerza el volver a dibujar un control completo.
SetFocus	Hace que el control reciba el enfoque.

- 3.1.1.4 Algunos ejercicios prácticos

1-) Aplicación que muestra un mensaje de bienvenida cuando se hace clic sobre un botón de comando.

Pasos a seguir:

a) Abra una nueva aplicación y agregue un botón de comando.

b) Seleccione el botón de comando y en la propiedad **Caption** escriba **&Hola**.

c) Haga doble clic en el botón de comando y escriba dentro del evento **Click** lo siguiente:

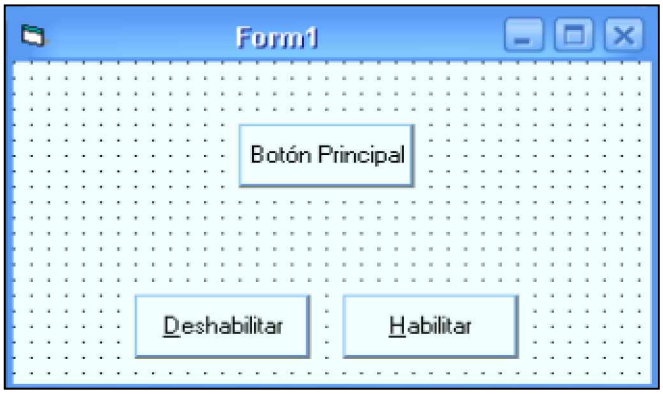
```
Private Sub Command1_Click ( )
    MsgBox ("Bienvenido a Visual Basic 6.0")
End Sub
```

d) Ejecute la aplicación con la tecla [F5] y haga clic luego sobre el botón.

2-) Aplicación que permite habilitar y deshabilitar un botón de comando.

Pasos a seguir:

a) Abra una nueva aplicación e inserte tres botones de comando.



b) Escriba al primer botón en la propiedad **Caption** "Botón Principal" y en la propiedad **Name** escriba "cmdBotonPrincipal". Al segundo botón escriba en la propiedad **Caption** "&Deshabilitar" y en la propiedad **Name** "cmdDeshabilitar". Al tercer botón escriba en la propiedad **Caption** "&Habilitar" y en la propiedad **Name** "cmdHabilitar".

c) En el *evento* ***Click*** del segundo botón escriba lo siguiente:

```
Private Sub cmdDeshabilitar_Click ( )
        cmdBotonPrincipal.Enabled = False
End Sub
```

d) En el *evento* **Click** del tercer botón escriba lo siguiente:

```
Private Sub cmdHabilitar_Click ( )
        cmdBotonPrincipal.Enabled = True
End Sub
```

e) Corra la aplicación y haga clic sobre los botones de habilitar y deshabilitar.

f) Guarde la aplicación desde **Save Project** del menú **File**.

Ejercicios Propuestos:

1.- Crear una aplicación con tres botones de comando uno para mostrar el mensaje “Hola a todos los programadores.”, otro para mostrar el mensaje “Visual Basic 6.0” y el otro para Salir de la aplicación.

2.- Crear una aplicación con tres botones de comando uno de los botones será para ocultarse y mostrarse mediante el evento **Click** de los otros dos botones. Uno de los *botones* debe ocultar el botón principal y el otro mostrarlo. Utilizar la *propiedad* **Visible**.

3.- Crear una aplicación que cambie el Texto que muestra el botón cuando se hace clic sobre el mismo.

4.- Crear una aplicación que cambie el Texto de un botón de comando mediante el evento **Click** de otro botón de comando.

5.- Hacer una aplicación que muestre el Texto (Caption) del botón que se pulsa en un cuadro de dialogo. Utilice la sintaxis **MsgBox**.

- 3.1.2 Cajas de texto (TextBox)

Un *control* **TextBox** también denominado *control* de campo de edición o *control* de edición, muestra información introducida en tiempo de ejecución introducida por el usuario o asignada al control en código en tiempo de ejecución.

- 3.1.2.1 Propiedades de las cajas de texto

Las cajas de texto poseen las propiedades comunes (Name, BackColor, Enabled, Font, Height, Width, Left, Top, TabIndex, Visible y ToolTipText) ya vistas anteriormente en el *control* **CommandButton**. Aparte de estas propiedades las cajas de texto poseen características especiales, es decir, muy propias de ellas. Estas propiedades se detallan a continuación:

Propiedad	Descripción
BorderStyle	Cambia el estilo de borde del *control*. Esta propiedad puede tomar los valores **0-None** (ningún borde ni elemento relacionado con el.) o **1- FixedSingle** (con borde simple fijo). Ejercicio: En un formulario inserte una caja de texto y establezca el valor de la propiedad **BorderStyle** a **0-None** o a **1- FixedSingle** y observe como el borde de la caja de texto cambia.
DataField	Devuelve o establece el dato de un campo contenido en una base de datos apuntada por un control establecido en la propiedad **DataSource** de la caja de texto.
DataSource	Devuelve o establece el origen de datos mediante el cual un receptor de datos enlaza con una base de datos.
ForeColor	Fija el color de texto que contendrá la caja de texto. Ejercicio: Seleccione la caja de texto del formulario, ubíquese en la **ForeColor** y seleccione el color deseado. Corra la aplicación y escriba en la caja de texto. El texto que digite o escriba en el área de edición de la caja de texto aparece con el color seleccionado en la propiedad **ForeColor**.
HideSelection	Determina si el texto seleccionado con la propiedad **SelLength**, aparece resaltado. Esta propiedad puede tomar los valores **True** o **False**.

Propiedad	Descripción
SelStart	No disponible en la *ventana de propiedades*, pero si en la ventana del *Code Editor* y también en modo de ejecución de la aplicación. Esta propiedad devuelve o indica el comienzo de la selección en una cadena de caracteres, donde el valor indicado es la posición de un carácter de la cadena.
SelLength	Devuelve o establece el número de caracteres seleccionados. No disponible en la *ventana de propiedades*
SelText	Devuelve o establece una cadena con el texto seleccionado actualmente o es una cadena de longitud cero si no hay caracteres seleccionados.
Text	Si duda la propiedad más importante. Devuelve o establece el texto contenido en el área de edición. Ejercicio: Haga Clic sobre la caja de texto. Busque la propiedad **Text** y escriba "Contenido de la caja". El texto aparecerá en el área de edición de la caja.
Locked	Devuelve o establece un valor que indica si un *control* se puede modificar. Esta propiedad puede tomar los valores **True** o **False**.
MaxLenth	Devuelve o establece un valor que indica el número de caracteres que puede tener la caja de texto en el área de edición.
MultiLine	Devuelve o establece un valor que indica si el control acepta más de una línea de texto. Puede tomar los valores **True** y **False** y solo de lectura en tiempo de ejecución.
ScrollBars	Devuelve o establece un valor que indica si un *objeto* tiene barras de desplazamiento horizontal o vertical. Puede tomar los valores **0- None** (si barra), **1- Horizontal** (barra horizontal), **2- Vertical** (barra vertical) y **3- Both** (ambas barras).
PasswordChar	Devuelve o establece un valor que se muestra cada vez que se escribe en la caja de texto. Esta propiedad solo admite un carácter.

- 3.1.2.2 Eventos sobre las cajas de texto

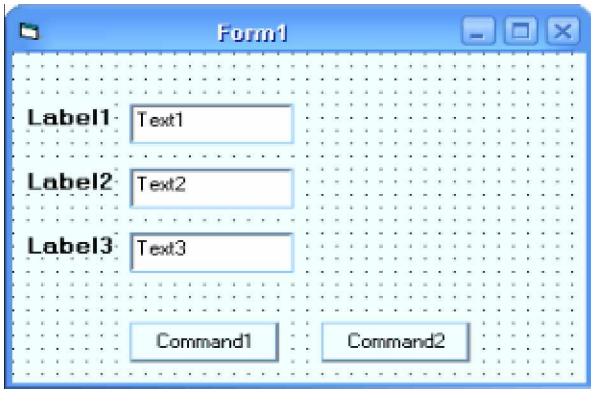
Las cajas de texto incluyen los eventos típicos de los *controles* (Clic, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseMove) ya estudiados anteriormente. Aparte de estos eventos las *cajas de texto* incluyen un evento muy propio de él que es el evento **Change** que ocurre cuando se modifica el texto en el área de edición de la *caja de texto*.

- 3.1.2.3 Métodos de las cajas de texto

Incluye los métodos más usuales de los *controles* (Drag, Refresh, SetFocus) aparte de otros métodos que no so usándoos con tanta frecuencia en una aplicación de **Visual Basic**.

- 3.1.2.4 Algunos ejercicios prácticos

1.) Aplicación que suma el contenido de dos cajas de texto y muestra el resultado en otra caja de texto.



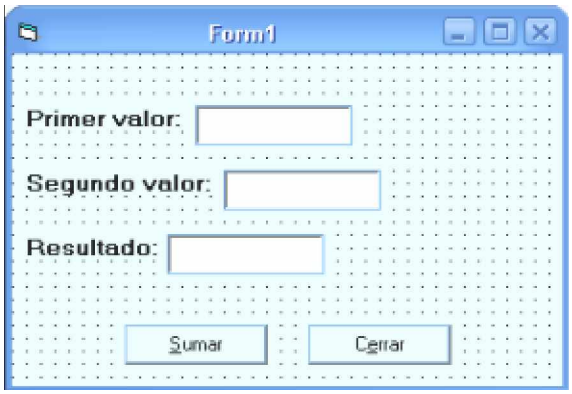
Pasos a seguir:

a) Inserte tres Labels (Etiquetas) tal y como se ven en la imagen anterior. Seleccione la primera etiqueta (Label1) y en la propiedad **Caption** de esta escriba "**Primer valor:**" y en la propiedad **Font** seleccione "*Negrita*", seleccione la segunda etiqueta (Label2) y en la propiedad **Caption** escriba "**Segundo Valor:"** y al igual establezca en la propiedad **Font** "*Negrita*" y en la tercera etiqueta (Label3) escriba "**Resultado:**" y establezca "Negrita" en la propiedad **Font**.

b) Inserte tres cajas de texto (Text1, Text2, Text3) tal y como se ven en la imagen y en la propiedad **Text** de cada caja de texto borre su el valor por defecto (Text1, Text2 y Text3).

c) Inserte dos botones de comando como se muestra en la imagen. En la propiedad **Caption** del primer botón escriba "&Sumar" y en el segundo botón de comando escriba "C&errar".

El aspecto de los controles sobre el formulario debe ser el siguiente:



d) Haga doble clic sobre el botón **[Sumar]** y en el *evento Click* escriba la siguiente línea de código dentro del procedimiento:

```
Private Sub Sumar_Click ( )
        Text3.Text = Val(Text1.Text) + Val(Text2.Text)
End Sub
```

Nota: La sintaxis **Val** indica que el contenido de la caja de texto será tratado como números y no como cadena de texto.

e) Haga doble clic en *Form1* de la **ventana de proyecto** (Project) para volver al formulario.

f) Haga doble clic en el botón **[Cerrar]** y en el *evento Click* escriba:

```
Private Sub Cerrar_Click ( )
        End
End Sub
```

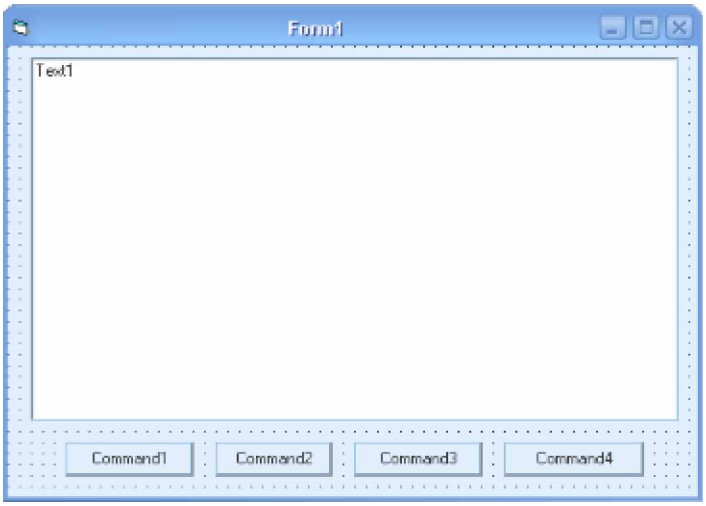
g) Corra la aplicación pulsando la tecla [F5].

h) Introduzca valores a las cajas de texto y haga clic en el botón **[Sumar]**.

i) Para salir haga clic en el botón **[Cerrar]**.

j) Guarde la aplicación desde **Save Project** del menú **File**.

2.) Programa que permite cambiar el tipo de fuente de un texto introducido en una caja de texto.



Pasos a seguir:

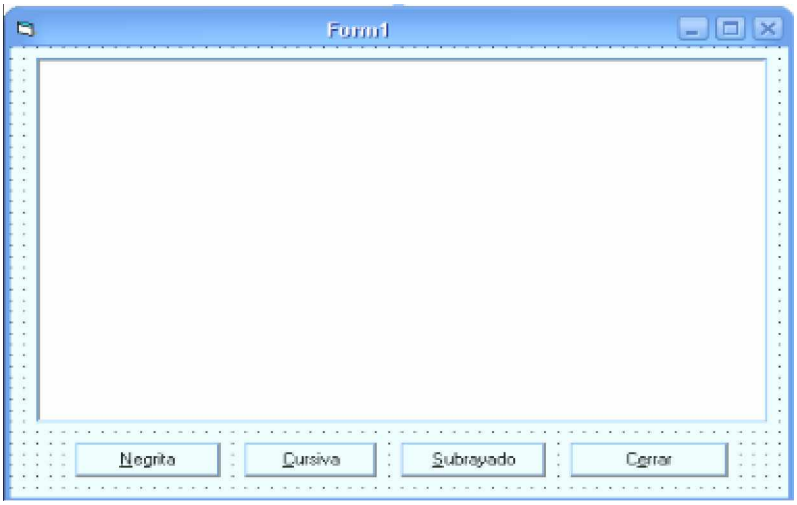
a) Abra un nuevo proyecto y en la propiedad **Height** del formulario establezca el valor **5430** y en la propiedad **Width** establezca el valor **7290**.

b) Inserte una caja de texto y cuatro botones de comando tal y como se ve en la imagen.

c) Seleccione la caja de texto y en la propiedad **Name** escriba "**txtContenido**" y en la propiedad **Text** borre el valor por defecto (Text1) y en la propiedad **MultiLine** seleccione el valor "**True**".

d) Seleccione el primer botón de comando y en la propiedad **Name** escriba "cmdNegrita" y en la propiedad **Caption** escriba "&Negrita". Seleccione el segundo botón y en la propiedad **Name** escriba "cmdCursiva" y en la propiedad **Caption** escriba "&Cursiva". Seleccione el tercer botón y en la propiedad **Name** escriba "cmdSubrayado" y en la propiedad **Caption** escriba "&Subrayado". En el cuarto botón (*command4*) escriba en la propiedad **Name** "cmdCerrar" y en la propiedad **Caption** escriba "C&errar".

e) El aspecto de los *controles* sobre el formulario debe ser el siguiente:



f) Escriba en el **evento Click** de cada botón el código correspondiente mostrado a continuación.

```
Private Sub cmdNegrita_Click ( )
        If TxtContenido.FontBold = False Then
            TxtContenido.FontBold = True
        Else
            TxtContenido.FontBold = False
        End If
End Sub
```

```
Private Sub cmdCursiva_Click ( )
        If TxtContenido.FontItalic = False Then
            TxtContenido.FontItalic = True
        Else
            TxtContenido.FontItalic = False
        End If
End Sub
```

```
Private Sub cmdSubrayado_Click ( )
   If TxtContenido.FontUnderline = False Then
      TxtContenido.FontUnderline = True
   Else
      TxtContenido.FontUnderline = False
   End If
End Sub
```

```
Private Sub cmdCerrar_Click ( )
        End
End Sub
```

g) Corra la aplicación pulsando la tecla [F5].

h) Escriba en la caja de texto y utilice los botones para cambiar el tipo de fuente.

i) Salga de la aplicación haciendo clic en el botón **[Cerrar]** de la aplicación.

j) Guarde la aplicación desde **Save Project** del menú **File**.

- **Aplicaciones para realizar:**

1.- Hacer una aplicación que calcule la suma y la resta del contenido de cuatro cajas de texto y muestra la suma en una caja de texto y la resta en otra caja de texto. La aplicación debe tener botón para salir.

2.- Hacer una aplicación que permita cambiar el tamaño de letra del texto contenido en una caja de texto. El programa debe tener una caja de texto donde se indique el tamaño de la letra y mediante ese valor se realizará la codificación necesaria para aplicárselo a la caja de edición que contiene el texto.

3.- Hacer una aplicación que permita cambiar el color de fondo y el color de letra del contenido de una caja de texto. El programa debe tener los botones con el color deseado por el programador, una parte de los botones será para cambiar el color del fondo de la caja de texto y otra parte será para cambiar el color de la letra.

4.- Hacer una aplicación que muestre en una ventana de dialogo la longitud o la cantidad de caracteres que posee una caja de texto. El programa debe tener un botón para mostrar la ventana con la longitud de la caja de texto. Utilizar la sintaxis **MsgBox** para la ventana de dialogo.

5.- Hacer una aplicación que traduzca al idioma ingles el contenido de una caja de texto y muestre la traducción en otra caja de texto. El programa debe tener un botón para **Traducir** y otro para **Salir** de la aplicación.

6.- Hacer una aplicación que realice las cuatros operaciones básicas: suma, resta, multiplicación, división, de tres valores contenidos en cajas de texto y muestre los resultados en cajas distintas.

- 3.1.3 Botones de opción (OptionButton)

Un botón de opción muestra una opción que se puede activar o desactivar. Generalmente, los *controles* **OptionButton** se utilizan en un grupo de opciones para mostrar opciones entre las cuales el usuario solo puede seleccionar una, los *controles* **OptionButton** se agrupan si los dibuja dentro de un contenedor como un *control* **Frame**, un control **PictureBox** o un **Formulario**. Para agrupar controles **OptionButton** en un **Frame** o **PictureBox**, dibuje en primer lugar el **Frame** o el **PictureBox** y, a continuación, dibuje dentro los controles **OptionButton**.

- 3.1.3.1 Propiedades de los botones de opción

Los botones de opción poseen las mismas propiedades de los botones de comando de la cual se destaca la propiedad **Caption** que muestra el texto indicador de la función de ese botón en la aplicación y la propiedad **Value** que indica si el control esta seleccionado o no, puede tomar los valores **True** (seleccionado) y **False** (no seleccionado).

- 3.1.3.2 Eventos sobre los botones de opción

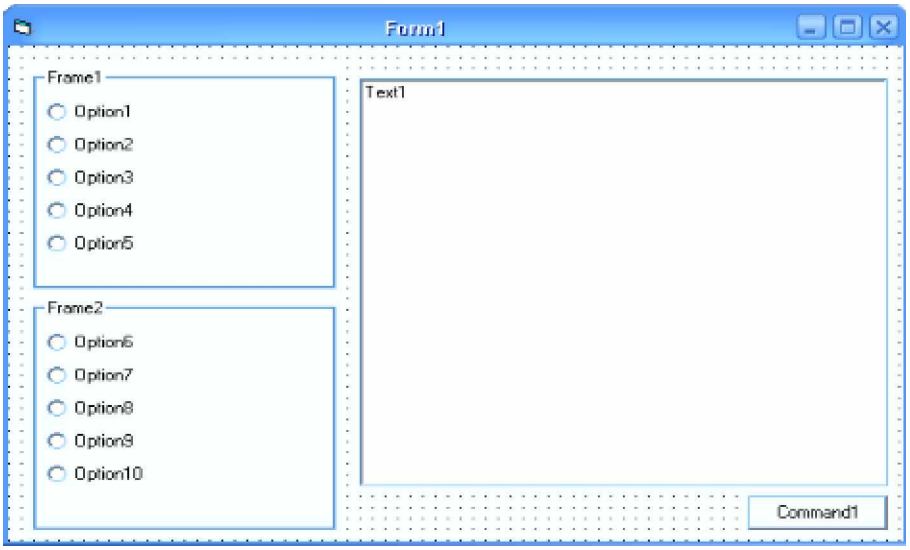
Los botones de opción poseen al igual que los *controles* ya estudiados los mismos eventos típicos.

- 3.1.3.3 Métodos de los botones de opción

Los botones de opción también se complementan con los métodos típicos ya estudiados anteriormente. Los métodos son muy poco usados en los controles que usamos frecuentemente, por tal razón, nos limitaremos en hacer énfasis en cada uno de ellos.

- 3.1.3.4 Algunos ejercicios prácticos

1.) Aplicación que posee un conjunto de botones de opción que indican los colores que se le aplicaran al fondo y a la letra de una caja de texto (TextBox). Cada conjunto de colores estará por separado en *controles* **Frame** que tendrán por encabezado la función que realizará ese conjunto de botones dentro del control **Frame**.



Pasos a seguir:

a) Abra un nuevo proyecto desde el menú **File**.

b) En la propiedad **Height** del formulario escriba el valor 5925 y en la propiedad **Width** escriba 8685 para fijar el tamaño adecuado en el formulario.

c) Inserte dos **Frame**  en el formulario tal y como se ve en la imagen.

d) Dentro del primer **Frame** inserte cinco botones de opción y dentro del segundo también inserte cinco botones de opción.

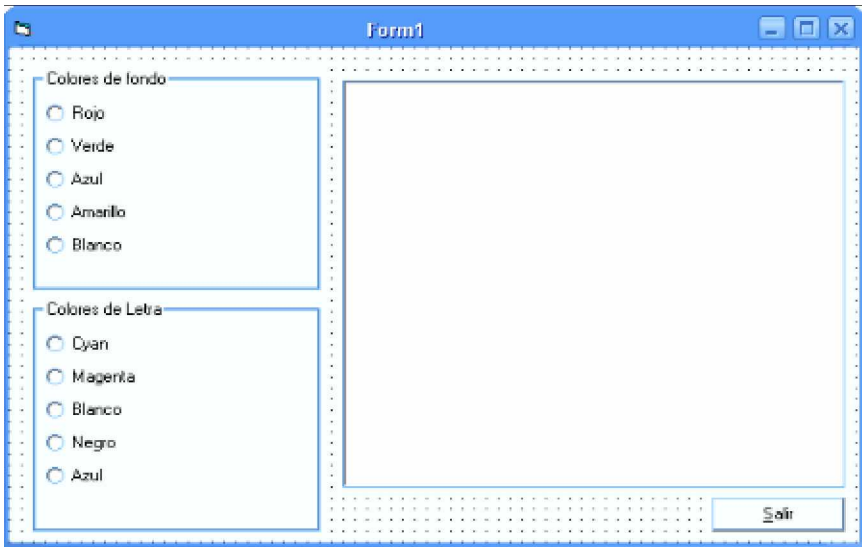
e) Inserte una caja de texto a la derecha de los *controles* **Frame** y un botón de comando en la esquina inferior izquierda del formulario tal y como se puede apreciar en la imagen anterior.

f) Aplique las siguientes características a los controles del formulario:

Control	Propiedades	Valor
Frame1	*Name*	frmColorFondo
	Caption	Colores de fondo
Frame2	*Name*	frmColorLetra
	Caption	Colores de letra
Text1	*Name*	txtContenido
	Text	(vacío)

Control	Propiedades	Valor
Command1	*Name*	cmdSalir
	Caption	&Salir
Option1	*Name*	optRojoFondo
	Caption	Rojo
Option2	*Name*	optVerdeFondo
	Caption	Verde
Option3	*Name*	optAzulFondo
	Caption	Azul
Option4	*Name*	optAmarilloFondo
	Cantion	Amarillo
Option5	*Name*	optBlancoFondo
	Caption	Blanco
Option6	*Name*	optCyanLetra
	Caption	Cyan
Option7	*Name*	optMagentaLetra
	Caption	Magenta
Option8	*Name*	optBlancoLetra
	Caption	Blanco
Option9	*Name*	optNegroLetra
	Caption	Negro
Option10	*Name*	optAzulLetra
	Caption	Azul

g) El aspecto del formulario debe ser el que se muestra en la siguiente página:



g) A cada *control* escriba las líneas de código correspondiente:

```
Private Sub optRojoFondo_Click ( )
   txtContenido.BackColor = vbRed
End Sub
```

```
Private Sub optVerdeFondo_Click ( )
   txtContenido.BackColor = vbGreen
End Sub
```

```
Private Sub optAzulFondo_Click ( )
   txtContenido.BackColor = vbBlue
End Sub
```

```
Private Sub optAmarilloFondo_Click ( )
   txtContenido.BackColor = vbYellow
End Sub
```

```
Private Sub optBlancoFondo_Click ( )
   txtContenido.BackColor = vbWhite
End Sub
```

```
Private Sub optCyanLetra_Click ( )
   txtContenido.ForeColor = vbCyan
End Sub
```

```
Private Sub optMagentaLetra_Click ( )
   txtContenido.ForeColor = vbMagenta
End Sub
```

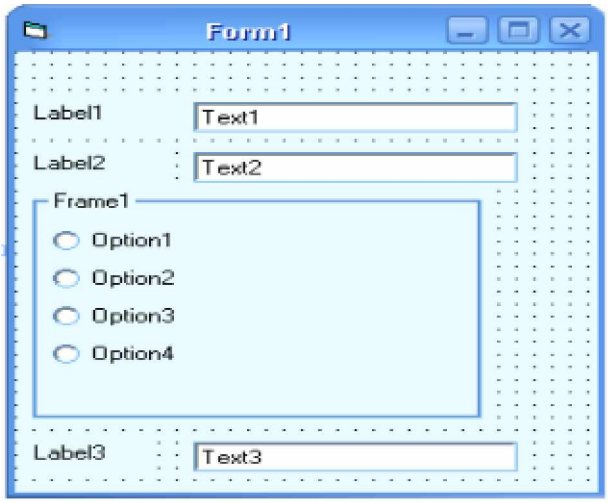
```
Private Sub optBlancoLetra_Click ( )
   txtContenido.ForeColor = vbWhite
End Sub
Private Sub optNegroLetra_Click ( )
   txtContenido.ForeColor = vbBlack
End Sub
Private Sub optAzulLetra_Click ( )
   txtContenido.ForeColor = vbBlue
End Sub
Private Sub cmdSalir_Click ( )
   End
End Sub
```

h) Corra la aplicación pulsando la tecla [F5].

i) Después que el programa esta en ejecución utilice los botones de opción para cambiar el color de la caja y de la letra. Para ver el color de la letra debe escribir en la caja de texto.

j) Salga de la aplicación haciendo clic en el botón **[Salir]**.

2.) Aplicación que realiza las cuatro operaciones básicas de matemática mediante cuatro botones de opción:



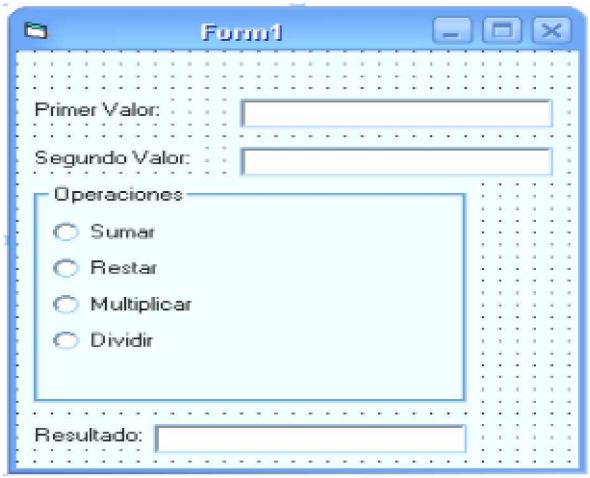
Pasos a seguir:

a) Inserte primero dos *etiquetas* y dos *cajas de texto*, luego inserte un *control* **Frame** y dentro de ese **Frame** inserte cuatro *botones de opción*. Luego inserte una tercera etiqueta y una tercera caja de texto tal y como se ve en la imagen.

b) Aplique las siguientes características a los controles del formulario:

Control	Propiedades	Valor
Label1	*AutoSize* *Caption*	True Primer valor:
Label2	*AutoSize* *Caption*	True Segundo valor:
Text1	*Name* *Text*	txtPrimerValor (vacío)
Text2	*Name* *Text*	txtSegundoValor (vacío)
Frame1	*Caption*	Operaciones
Option1	*Name* *Caption*	optSumar Sumar
Option2	*Name* *Caption*	optRestar Restar
Option3	*Name* *Caption*	optMultiplicar Multiplicar
Option4	*Name* *Caption*	optDividir Dividir
Label3	*AutoSize* *Caption*	True Resultado:
Text3	*Name* *Text*	txtResultado (vacío)

c) La apariencia de los controles sobre el formulario debe ser la siguiente:



d) Escriba el código correspondiente en los **eventos Click** de cada control:

```
Private Sub optSumar_Click ( )
   txtResultado.Text = Val(txtPrimerValor.Text) + Val(txtSegundoValor.Text)
End Sub
```

```
Private Sub optRestar_Click ( )
   txtResultado.Text = Val(txtPrimerValor.Text) - Val(txtSegundoValor.Text)
End Sub
```

```
Private Sub optMultiplicar_Click ( )
   txtResultado.Text = Val(txtPrimerValor.Text) * Val(txtSegundoValor.Text)
End Sub
```

```
Private Sub optDividir_Click ( )
   If Val(txtSegundoValor.Text) = 0 Then
         MsgBox ("No se puede dividir por cero.")
   Else
         txtResultado.Text = Val(txtPrimerValor.Text) / Val(txtSegundoValor.Text)
   End If
   End Sub
```

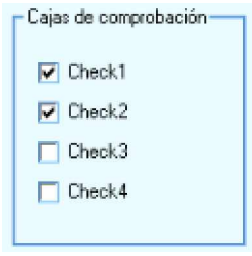
e) Corra la aplicación pulsando la tecla [F5].

f) Introduzca valores a las cajas de texto y utilice los botones de opción para realizar la operación deseada.

g) Salga de la aplicación desde el botón cerrar **[X]** de la ventana.

h) Guarde la aplicación desde **Save Project** del menú **File**.

- 3.1.4 Cajas de comprobación (CheckBox)



La única diferencia entre los botones de opción (OptionButton) y las cajas de comprobación (CheckBox) es que dentro de un mismo contenedor se pueden seleccionar más de una caja de comprobación, es decir, pueden haber varias cajas de comprobación con la propiedad **Value** a **1- Checked** establecida.

- 3.1.4.1 Propiedades de las cajas de comprobación

Al igual que en los botones de opción, las propiedades más importantes son las propiedad **Caption** y la propiedad **Value** que indica si el elemento esta seleccionado o no.

- 3.1.4.2 Eventos sobre las cajas de comprobación

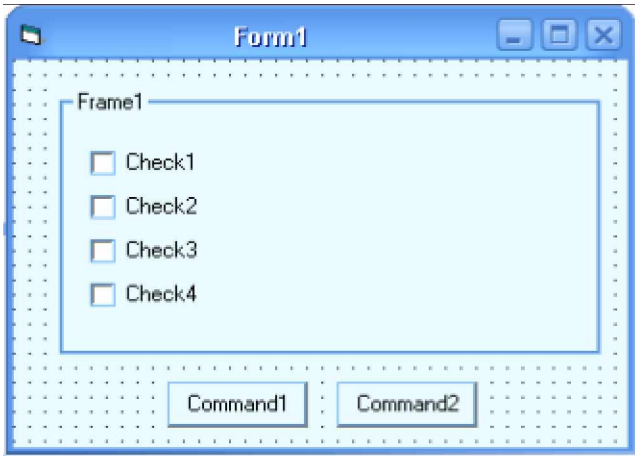
Las cajas de comprobación no se diferencian en nada a los demás *controles* con respecto a los eventos típicos de los *controles*. Puede utilizar los eventos ya vistos anteriormente.

- 3.1.4.3 Métodos de las cajas de comprobación

Las cajas de comprobación poseen todos los métodos ya estudiados anteriormente (Drag, Refresh, SetFocus, Etc).

- 3.1.4.4 Algunos ejercicios prácticos

1.) Aplicación que muestra en ventanas de dialogo los elementos seleccionados en un contenedor con un conjunto de *CheckBox* que representan libros de programación.



Pasos a seguir:

a) Abra un nuevo proyecto desde el menú **File**.

b) En la propiedad **Height** establezca el valor **3690** y en la propiedad **Width** establezca el valor **4905**. En la propiedad **Caption** del formulario escriba “Libros de programación”.

c) Inserte un **Frame** y dentro de este agregue cuatro *cajas de comprobación*, tal y como se ve en la imagen.

d) Inserte dos *botones de comando* (Command1, Command2) en la parte inferior del formulario.

e) Establezca las siguientes características a los *controles* del formulario:

Control	Propiedades	Valor
Frame1	*Caption*	Libros de programación
	Height	2175
	Width	4215
Check1	*Name*	chkLibro1
	Caption	Visual Basic 6.0, Tomo I. Autor: Microsoft.
	Width	3855
Check2	*Name*	chkLibro2
	Caption	Pascal estructurado 7.0, Tomo II. Autor: Borland.
	Width	3855
Check3	*Name*	chkLibro3
	Caption	Delphi 6.0, Tomo I. Autor: Borland.
	Width	3855
Check4	*Name*	chkLibro4
	Caption	Cobol 2.0, Autor: Borland.
	Width	3855
Command1	*Name*	cmdMostrarLibro
	Caption	&Mostrar libro
Command2	*Name*	cmdSalir
	Caption	&Salir

f) El aspecto de los *controles* sobre el formulario debe ser el siguiente:

g) Agregue el código correspondiente a cada procedimiento de los *controles*:

```
Private Sub cmdMostrarLibro_Click()
   If chkLibro1.Value = 1 Then
      MsgBox (chkLibro1.Caption)
   End If

   If chkLibro2.Value = 1 Then
      MsgBox (chkLibro2.Caption)
   End If

   If chkLibro3.Value = 1 Then
      MsgBox (chkLibro3.Caption)
   End If

   If chkLibro4.Value = 1 Then
      MsgBox (chkLibro4.Caption)
   End If

   If chkLibro1.Value = 0 And chkLibro2.Value = 0 And chkLibro3.Value = 0 And _
     chkLibro4.Value = 0 Then
        MsgBox ("Seleccione un libro de la lista.")
    End If
End Sub
```

```
Private Sub cmdSalir_Click()
          End
End Sub
```

h) Corra la aplicación con la tecla [F5].

i) Seleccione algunos libros de la lista, y a continuación, haga clic en el botón **Mostrar libro**.

j) Salga de la aplicación haciendo clic en el botón salir.

k) Guarde la aplicación desde el menú **File**.

- 3.1.5 Barras de desplazamiento (ScrollBars)

En este tipo de control las propiedades más importantes son **Max** y **Min**, que determinan el rango en el que está incluido su valor, **LargeChange** y **SmallChange** que determinan lo que se modifica su valor al hacer clic en la barra o en el botón con la flecha del *control*. Otra de las propiedades importantes es la propiedad **Value** que determina el valor actual de la barra de desplazamiento. Las barras de desplazamiento no incluyen la propiedad **Caption**.

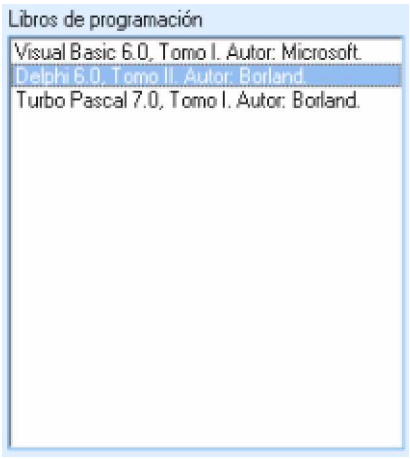
El evento que se programa habitualmente es **Change**, que se activa cuando la barra de desplazamiento modifica su valor. Para mas información sobre este control consulte la ayuda de **Visual Basic 6.0**.

- 3.1.6 A Las etiquetas (Labels)

Un control Label es un control gráfico que se puede usar para mostrar texto que el usuario no podrá cambiar directamente. En las etiquetas la propiedad más importante es **Caption**, que contiene el texto que aparece sobre el control. Esta propiedad puede ser modificada desde el programa, pero no interactivamente sino, mediante líneas de código.

Las etiquetas tienen las propiedades **AutoSize** y **WordWrap**. La primera, cuando está a **True**, ajusta el tamaño del control al del texto en él contenido. La segunda hace que el texto se distribuya en varias líneas cuando no cabe en una sola.

- 3.1.7 Las cajas de lista (ListBox)



Una **lista** es un *control* en el que se pueden tomar varios varios registros de líneas, teniendo uno o varios de ellos seleccionado. Si en la lista hay más registros de los que se pueden mostrar al mismo tiempo, se añade automáticamente una ***ScrollBar***.

Para añadir o eliminar registros de la lista en modo de Ejecución se utilizan los métodos **AddItem** y **RemoveItem**.

El contenido de un ListBox suele inicializarse desde el *evento* **Form_Load** de los formularios, de tal manera, las listas obtienen sus elementos antes de que la aplicación cargue totalmente.

- 3.1.7.1 Métodos y propiedades de las cajas de lista

A continuación se detallan los métodos y las propiedades más importantes de un **ListBox**.

Método	Descripción
AddItem	Se utiliza para agregar registros a la *lista* cuando la aplicación esta en ejecución. Su formato es: **ObjetoList**.**AddItem Elemento, Posición** Donde **ObjetoList** representa el *control* **ListBox**, **AddItem** es el método que agrega el elemento al ***ListBox***, **Elemento** es el texto que se muestra en la posición indicada, y **Posición** es el lugar donde se ubicará el elemento, comenzando desde la posición 0. Ejercicio: a) Abra un nuevo proyecto desde el menú **File**. b) Inserte un **ListBox**. c) Haga doble clic en cualquier parte del *formulario* menos donde se encuentre el cuadro de lista. d) En el evento **Load** del formulario escriba: **Private Sub** Form_Load ( ) List1.AddItem "Visual Basic 6.0, Tomo I. Autor: Microsoft.", 0 List1.AddItem “Delphi 6.0, Tomo II. Autor: Borland.”, 1 List1.AddItem “Turbo Pascal 7.0, Tomo I. Autor: Borland.”, 2 **End Sub** e) Corra la aplicación pulsando la tecla [F5]. f) Observe que se han añadido los elementos al cuadro de lista. g) Cierre la aplicación desde el botón cerrar [X].
RemoveItem	Este método se utiliza para eliminar un elemento o registro del **ListBox**. Su Formato es: **RemoveItem (Posición)** Donde **Posición** es el lugar de la lista donde se encuentra el elemento que desea borrar.

Método	Descripción
Clear	Borra todo el contenido del *control* ListBox. Su formato es: **ObjetoList.Clear**

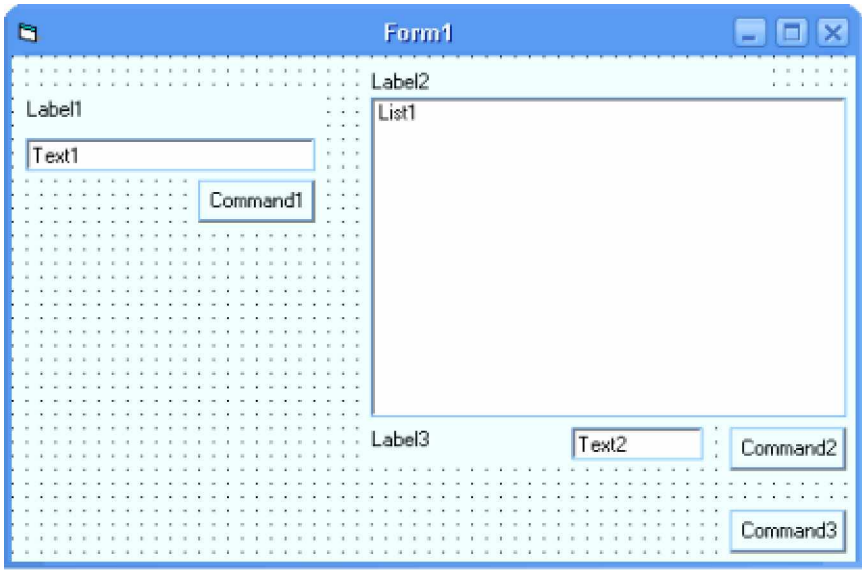
Propiedades:

Propiedad	Descripción
List	Se utiliza para agregar elementos o registros en tiempo de diseño al *control* ***ListBox***. Se recomienda utilizar el método **AddItem** para agregar los elementos en vez de la propiedad **List** que es menos práctica y menos especifica.
ListCount	Devuelve el número de elementos que contiene un *control* **ListBox**. Ejemplo: MsgBox(List1.ListCount)
Index	Devuelve o establece el número que identifica un *control* de forma exclusiva en una *matriz de controles*. Sólo está disponible si el *control* forma parte de una matriz de *controles*.
ListIndex	Devuelve o establece el índice del elemento seleccionado actualmente en el *control*. No está disponible en *tiempo de diseño*. **Ejemplo (a):** 'Muestra en una ventana la posición de un elemento seleccionado en un control ListBox. **MsgBox** (List1.ListIndex) **Ejemplo (b):** 'Muestra en una ventana el texto de un elemento seleccionado en un control ListBox. **MsgBox** (List1.List(List1.ListIndex))
MultiSelect	Devuelve o establece un valor que indica si el usuario puede realizar selecciones múltiples en un *control* **FileListBox** o **ListBox**, y la forma de llevarlas a cabo. Es de sólo lectura en tiempo de ejecución.

Propiedad	Descripción
SelCount	Devuelve el número de elementos seleccionados en un *control* **ListBox**.
Selected	Devuelve o establece el estado de selección de un elemento de un *control* **FileListBox** o **ListBox**. Esta propiedad es una matriz de valores booleanos con el mismo número de elementos que la propiedad **List**. No está disponible en **tiempo de diseño.** **Sintaxis** objeto**.Selected(**índice**)** [= booleano]
Sorted	Devuelve un valor que indica si los elementos de un *control* se colocan automáticamente en orden alfabético.

- 3.1.7.2 Algunos ejercicios prácticos

1.) Aplicación que permite agregar y remover países en un *control* ListBox.



Pasos a seguir:

a) Abra un nuevo proyecto desde el menú **File**.

b) En la propiedad **Height** agregue **4920**, en la propiedad **Width** establezca el valor **7170** y en la propiedad **Caption** escriba "Países".

c) Inserte primero una *etiqueta* (Label1) y debajo de esta inserte un *control* **TextBox** (Text1) y un *botón de comando* (Command1) tal y como se puede apreciar en la imagen.

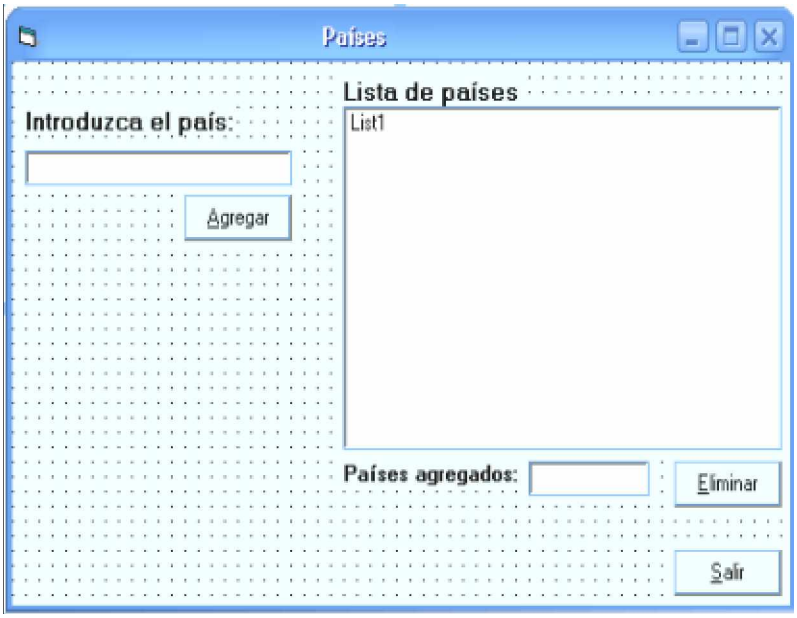
d) Insertar una segunda etiqueta (Label2) y debajo de esta un *control* ListBox (List1).

e) Debajo de estos inserte una *tercera etiqueta* (Label3), una segunda *caja de texto* (Text2) y dos *botones de comandos* (Command2, Command3).

f) Aplique las siguientes características a los *controles* sobre el formulario:

Control	Propiedad	Valor
Label1	*AutoSize* *Caption* *Font*	True Introduzca el país: Tamaño 10, Estilo Negrita.
Text1	*Name* *Text*	txtPais (vacío)
Command1	*Name* *Caption*	cmdAgregar &Agregar
Label2	*AutoSize* *Caption* *Font*	True Lista de países: Tamaño 10, Estilo Negrita.
List1	*Name*	lstPaises
Label3	*AutoSize* *Caption* *Font*	True Países agregados: Tamaño 8, Estilo Negrita.
Text2	*Name* *Text* *Locked*	txtCantidadPaises (vacío) True
Command2	*Name* *Caption*	cmdEliminar &Eliminar
Command2	*Name* *Caption*	cmdSalir &Salir

g) La apariencia de los controles sobre el formulario debe ser como se muestra en la página siguiente:



h) Dentro de cada procedimiento escriba el código correspondiente:

```
Private Sub cmdAgregar_Click()
   'Verifica que la caja no se deje vacía
   If Len(txtPais.Text) = 0 Then
            MsgBox ("No puede dejar la caja vacía.")
   Else
            lstPaises.AddItem txtPais.Text   ' Agrega el país en el control ListBox
            txtPais.Text = ""    ' Limpia la caja de texto
            txtPais.SetFocus     ' Hace que el curso se mantenga sobre la caja
            txtCantidadPaises.Text = lstPaises.ListCount   'Pone el número de países agregados
   End If
End Sub
```

```
Private Sub cmdEliminar_Click()
   On Error GoTo Error   'Verificar si ocurre un error tratar de borrar un elemento.
   lstPaises.RemoveItem (lstPaises.ListIndex)  'Borra el elemento
   txtCantidadPaises.Text = lstPaises.ListCount
    Exit Sub     'Indica que lo que esta debajo solo ocurrirá cuando pase algún error.
Error:
        MsgBox ("No existen elementos seleccionados.")
End Sub
```

```
Private Sub cmdSalir_Click()
    End        ‘ Finaliza la aplicación
End Sub
```

i) Corra la aplicación pulsando la tecla [F5].

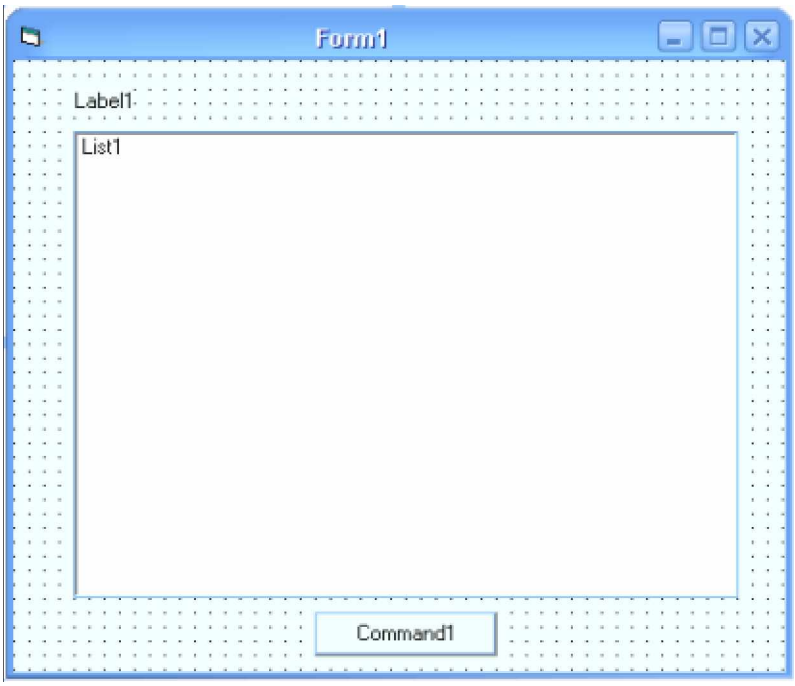
j) En la caja de texto de la aplicación introduzca algún país, y a continuación, haga clic en el botón **Agregar**. Agregue todos los países que desee y podrá observar que todos los países se agregan al *control* **ListBox**.

k) Seleccione algunos de los países ya agregados y luego, haga clic en el botón **Eliminar**.

l) Salga de la aplicación.

m) Guarde la aplicación desde el menú **File**.

2.-) Aplicación que te muestra en un cuadro de dialogo la capital de un país seleccionado en un *control* **ListBox**.



Pasos a seguir:

a) Abra un nuevo proyecto desde el menú **File**.

b) En la propiedad **Height** establezca el valor **5700** y en la propiedad **Width** el valor **6270**.

c) En la propiedad **Caption** del formulario escriba “Países y capitales”.

d) Inserte una *etiqueta* en la parte superior del formulario.

e) Inserte un *control* **ListBox** como se ve en la imagen.

f) Inserte un botón de comando (Command1) debajo del *control* **ListBox**.

g) Agregue las siguientes características a los *controles* sobre el formulario:

Control	Propiedad	Valor
Label1	*AutoSize* *Caption*	True Seleccione un país:
List1	*Name*	lstPaises
Command1	*Name* *Caption*	cmdSalir &Salir

h) Haga doble clic en cualquier zona libre del formulario, es decir, en una parte que no resida algún control, y en el evento **Load** del formulario escriba la siguiente línea de código:

```
Private Sub Form_Load()
'Agrega los países al control
   lstPaises.AddItem "República Dominicana"
   lstPaises.AddItem "Perú"
   lstPaises.AddItem "Salvador"
   lstPaises.AddItem "México"
   lstPaises.AddItem "Puerto Rico"
   lstPaises.AddItem "Ecuador"
End Sub
```

i) En el *evento* **Click** del *control* **ListBox** escriba lo siguiente:

```
Private Sub lstPaises_Click()
   If lstPaises.List(lstPaises.ListIndex) = "República Dominicana" Then
      MsgBox ("Santo Domingo")
   ElseIf lstPaises.List(lstPaises.ListIndex) = "Perú" Then
      MsgBox ("Lima")
   ElseIf lstPaises.List(lstPaises.ListIndex) = "Salvador" Then
      MsgBox ("San Salvador")
   ElseIf lstPaises.List(lstPaises.ListIndex) = "México" Then
      MsgBox ("México")
   ElseIf lstPaises.List(lstPaises.ListIndex) = "Puerto Rico" Then
      MsgBox ("San Juan")
   ElseIf lstPaises.List(lstPaises.ListIndex) = "Ecuador" Then
      MsgBox ("Quito")
   End If
End Sub
```

j) Corra la aplicación pulsando la tecla [F5].

k) Seleccione cualquier país de la lista para que el programa muestre su capital en una ventana de dialogo.

l) Salga de la aplicación.

m) Guarde la aplicación desde el menú **File**.

- Ejercicios propuestos

a) Crear una aplicación que permita agregar y eliminar libros en un *control* **ListBox** mediante una *caja de texto*. El programa debe tener una *etiqueta* donde muestre la cantidad de libros que contiene el **ListBox**.

b) Crear una aplicación que contenga dos *controles* **ListBox** que permitan cambiar el color de fondo y el color de letra de una *caja de texto* que se encuentre sobre un formulario. Uno de los **ListBox** debe tener la lista de los colores de fondo que se le aplicará a la *caja de texto* y el otro *control* **ListBox** los colores para la letra de la *caja de texto*.

c) Crear una aplicación que en un *control* **ListBox** contenga 20 números cuales quiera. Cuando un número de lo de la lista sea seleccionado debe mostrarse ese número en una ventana de dialogo.

d) Crear una aplicación que permita agregar y eliminar nombres de personas en un *control* **ListBox** y que permita organizarlos alfabéticamente.

e) Crear una aplicación que mediante una lista de colores en un *control* **ListBox** permita cambiar el color de la ventana de la aplicación.

- 3.1.8 Cajas combinadas (ComboBox)

Un **ComboBox** no tiene muchas diferencias en relación con un *control* **ListBox**. La diferencia que existe entre ambos *controles* es que un **ComboBox** oculta la lista de elementos y solo se muestra cuando se hace clic en el *botón flecha abajo* **[▼]** que contiene el *control*, mientras que el *control* **ListBox** muestra la lista de elementos sin ocultarla.

Otra de la diferencia principal es que un **ComboBox** tiene una propiedad llamada **Style**, que puede adoptar tres valores (1, 2 ó 3) que corresponden con tres distintas formas de presentar una lista:

1.) **0- DropDownCombo**: Éste es el valor más habitual y corresponde con el caso en el que sólo se muestra el registro seleccionado, que es editable por el usuario, permaneciendo el resto de los elementos oculto hasta que el usuario despliega la lista completa haciendo clic sobre *el botón flecha abajo* **[▼]**.

2.) **1- Simple Combo**: En este caso el registro seleccionado también es editable, y se muestra una lista no desplegable dotada si es necesario de una **ScrollBar**.

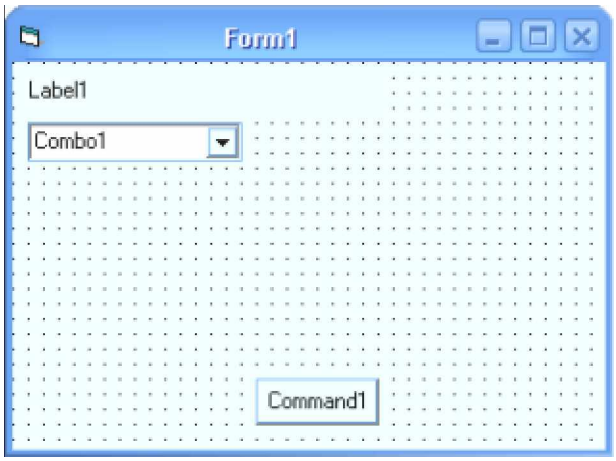
3.) **2- DropDown List**: En este último caso el registro seleccionado no es editable y la lista es desplegable.

A pesar de estas dos grandes diferencias, existen muchas relaciones con respecto a los métodos y las propiedades entre ambos *controles*. Por ejemplo los *métodos* **AddItem**, **RemoveItem** o **Clear** y las *propiedades* **List**, **ListIndex** o **ListCount**.

La *propiedad* **Text** corresponde con lo que aparece en el *área de edición* del **ComboBox** que es por lo general el primer elemento de la *lista desplegable*.

- 3.1.8.1 Algunos ejercicios prácticos

- Aplicación que muestra los número del 1 a 30 en un *control* **ComboBox**.



b) Inserte una *etiqueta* en la esquina superior izquierda del formulario.

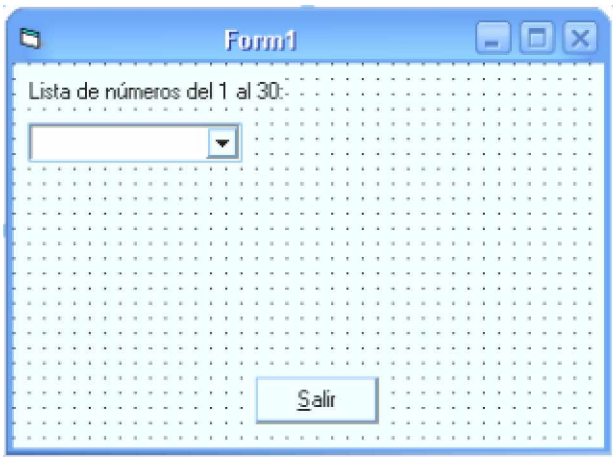
c) Debajo de la etiqueta inserte un *control* **ComboBox**.

d) Inserte un *botón de comando* en la parte inferior del formulario tal y como se ve en la imagen anterior.

e) Establezca las siguientes características a los *controles* sobre el formulario:

Control	Propiedad	Valor
Label1	*AutoSize* *Caption*	True Lista de números del 1 al 30:
Combo1	*Name* *Text*	cboNumeros (vacío)
Command1	*Name* *Caption*	cmdSalir &Salir

f) El aspecto de los controles sobre el formulario debe ser el siguiente:



g) A cado *control* escriba el código correspondiente:

```
Private Sub Form_Load ( )
   Dim i As Integer                 'Declara una variable para un bucle
   For i = 1 To 30                  ' Inicia el bucle del 1 hasta 30
        cboNumeros.AddItem i        ' Agrega el elemento
   Next i                           ' Repite hasta que el bucle no termina.
End Sub
```

```
Private Sub cmdSalir_Click()
   End
End Sub
```

h) Corra la aplicación pulsando la tecla [F5].

i) Despliegue la lista haciendo clic en el *botón flecha abajo* **[▼]** del *control* **ComboBox**.

j) Cierra la aplicación.

k) Guarde la aplicación desde el menú **File**.

- 3.1.9 Controles relacionados con ficheros

En una aplicación de Windows es habitual tener que abrir y cerrar ficheros para leer datos, guardar un documento, cambiar entre unidades de disco, etc. Hay tres *controles* básicos que resultan de suma utilidad para realizar cualquiera de estas tareas. El *control* **DriveListBox** que muestra las unidades lógicas de discos de un computador, El *control* **DirListBox** que muestra la lista de directorios de la unidad actualmente seleccionada y el *control* **FileListBox** que muestra la lista de ficheros que contiene un directorio o una unidad de disco. Estos tres *controles* trabajan mayormente en conjunto como veremos en el **Capitulo 6** de este libro.

- 3.1.10 Control tiempo (Timer)

Un *control* **Timer** puede ejecutar código a intervalos periódicos produciendo un *evento* **Timer**, que ocurre cuando ha transcurrido un *Intervalo* preestablecido para un *control* **Timer**. La frecuencia del intervalo se almacena en la propiedad **Interval** del *control* que especifica el tiempo en milisegundos.

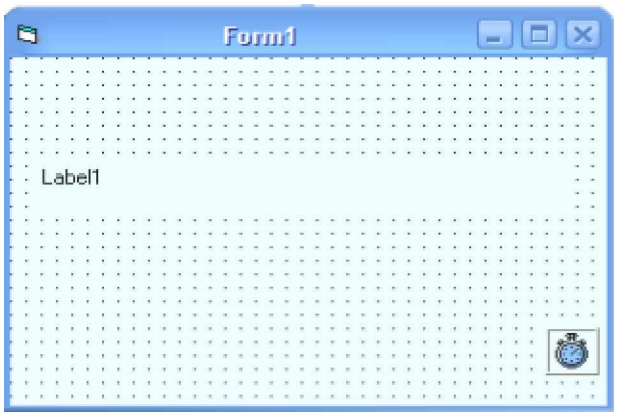
- 3.1.10.1 Propiedades del control tiempo

De todas las propiedades de un *control tiempo* son **Enabled** para habilitar o deshabilitar el *control* y la propiedad **Interval** para especificar el intervalo de tiempo en que el *control* realizará el *evento*.

Para representar los segundos en *milisegundos* solo debe multiplicar la cantidad de segundos por mil. Por ejemplo, 2 segundos sería 2 x 1000 = 2000 *milisegundos*, un minuto sería 60 * 1000 = 60000 *milisegundos*, así sucesivamente.

- 3.1.10.2 Algunos ejercicios prácticos

1.-) Aplicación que simula una barra de progreso sin usar un contenedor.



Pasos a seguir:

a) Abra un nuevo proyecto.

b) Inserte una etiqueta y un *control tiempo* tal y como se ve en la imagen.

c) Aplique las siguientes características a los *controles* sobre el formulario.

Control	Propiedades	Valor
Label1	*Name*	BarraProgreso
	Caption	(vacío)
	BackColor	Seleccione el color que prefiera.
	Left	240
	Top	960
	Height	375
	Width	15
Timer	*Interval*	100

d) Dentro del *control tiempo* agregue las siguientes líneas de código:

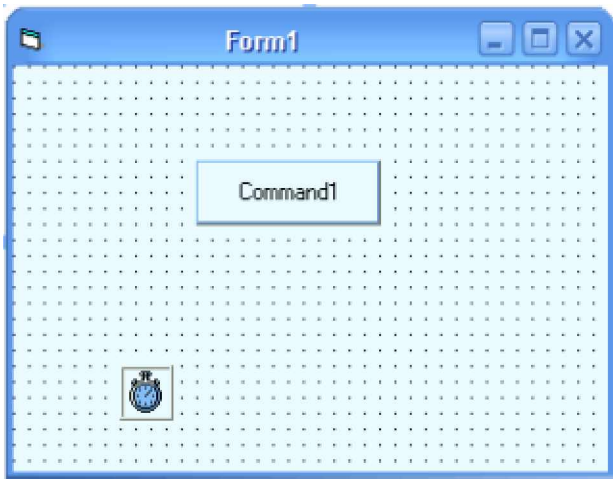
```
Private Sub Timer1_Timer ( )
   If BarraProgreso.Width < 4215 Then
      BarraProgreso.Width = BarraProgreso.Width + 100    'Suma 100 twip a la anchura del control
   Else
      BarraProgreso.Width = 4215
      Timer1.Enabled = False     'Deshabilita el tiempo
   End If
End Sub
```

e) Corra la aplicación pulsando la tecla [F5].

f) Para cerrar la aplicación haga clic en el botón cerrar **[X]** de la ventana.

g) Guarde la aplicación.

2.-) Aplicación que mueve un *botón de comando* por distintas zonas de un formulario.



Pasos a seguir:

a) Abra un nuevo proyecto.

b) Inserte un *botón de comando* y un control tiempo en cualquier parte del formulario.

c) En la propiedad **Interval** del control tiempo escriba el valor **100**.

d) Haga doble clic sobre el *control* tiempo y en el evento **Timer** escriba:

```
Private Sub Timer1_Timer ( )
    Randomize                          'Inicia el generador de números aleatorios.
    Command1.Top = Int((3000 * Rnd) + 1)  'Genera valores aleatorios entre 3000 y 1 para Top.
    Command1.Left = Int((3000 * Rnd) + 1)  'Genera valores aleatorios entre 3000 y 1 para Left.
End Sub
```

e) Corra la aplicación pulsando la tecla [F5].

f) Cierra la aplicación desde el botón cerrar **[X]**de la ventana.

g) Guarde la aplicación desde el menú **File**.

3.-) Aplicación que pone la hora del sistema en un *control* **Label**.

Pasos a seguir:

a) Abra un nuevo proyecto.

b) Inserte una etiqueta un *control* **Tiempo** tal y como se ve en la imagen.

c) En la propiedad **Interval** del *control* **Tiempo** escriba el valor **1000**.

d) Haga doble clic sobre el *control* **Tiempo** y escriba la siguiente línea de código:

```
Private Sub Timer1_Timer ( )
   Label1.Caption = Time
End Sub
```

e) Corra la aplicación pulsando la tecla [F5].

f) Para cerrar la aplicación haga clic en el botón cerrar **[X]** de la ventana.

g) Guarde la aplicación desde el menú **File**.

3.2 Algunas propiedades comunes a varios controles

Como pudimos observar al trabajar con los *controles* de **Visual Basic 6.0** que hay algunas *propiedades* que son comunes a muchos *controles*. A continuación se hace una lista de estos *controles*:

- **Appearance:** Devuelve o establece el estilo de dibujo o la apariencia de los *controles* de un objeto.
- **BackColor:** Establece el color de fondo de un objeto.
- **Caption:** Establece el texto que aparece dentro o junto al objeto.

- **Enabled**: Establece si un objeto es accesible o modificable.
- **Font**: Establece las características del tipo de letra del objeto.
- **ForeColor**: Establece el color del texto y/o gráficos de un objeto.
- **Height** y **Width**: Establecen la altura y anchura de un objeto.
- **Left** y **Top**: Establecen la distancia horizontal y vertical entre el origen del *control* y el origen del objeto que lo contiene, que puede ser un *formulario*, un *control* **Frame** o un **PictureBox**.
- **MousePointer**: Establece la forma que adoptará el puntero del ratón al posicionarse sobre el objeto.
- **Name**: Indica el nombre del objeto. Todos los objetos sobre un formulario deben tener su nombre ya sea el *nombre por defecto* que le asigna **Visual Basic** o un *nombre definido* por el usuario, que permite hacer referencia al objeto.
- **Visible**: Establece si el objeto es visible o invisible en el momento que se ejecuta la aplicación.

3.3 Cajas de dialogo estándar (CommondDialog)

 El *control* **CommonDialog** proporciona un conjunto de cuadros de diálogo estándar para realizar operaciones como abrir y guardar archivos, establecer las opciones de impresión y seleccionar colores y fuentes. El control también tiene la posibilidad de presentar Ayuda ejecutando el motor de Ayuda de Windows.

Sintaxis

CommondDialog.Metodo

El *control* **CommonDialog** proporciona una interfaz entre **Visual Basic** y las rutinas de la biblioteca de vínculos dinámicos **Commdlg.dll** de Microsoft Windows. Para crear un cuadro de diálogo utilizando este control, **Commdlg.dll** debe encontrarse en el directorio **SYSTEM** de Microsoft Windows.

Para usar el *control* **CommonDialog** en una aplicación, agréguelo a un formulario y establezca sus propiedades. El cuadro de diálogo presentado por el control está determinado por los métodos del control.

En *tiempo de ejecución* se presenta un cuadro de diálogo o se ejecuta el motor de Ayuda, cuando se invoca el método apropiado; en *tiempo de diseño*, el control **CommonDialog** se presenta como un icono dentro de un formulario. No se puede cambiar el tamaño de dicho icono.

El *control* **CommonDialog** puede presentar los cuadros de diálogo siguientes utilizando el método especificado.

Método	**Cuadro de diálogo presentado**
ShowOpen	Cuadro de diálogo **Abrir**.
ShowSave	Cuadro de diálogo **Guardar como**.
ShowColor	Cuadro de diálogo **Color**.
ShowFont	Cuadro de diálogo **Fuente**.
ShowPrinter	Cuadro de diálogo **Imprimir u Opciones de impresión**.
ShowHelp	Cuadro de diálogo Invoca el motor de **Ayuda de Windows**.

A continuación se muestran las ventanas que puede abrir el *control* **CommondDialog**:

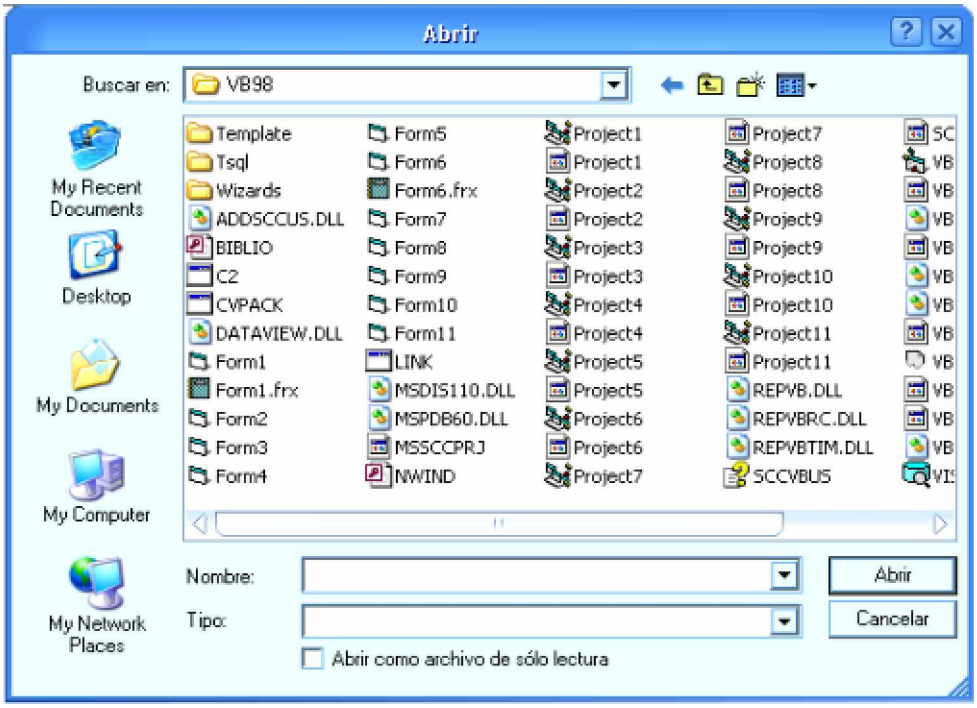


Figura 3.1. Cuadro de dialogo Abrir (Open)

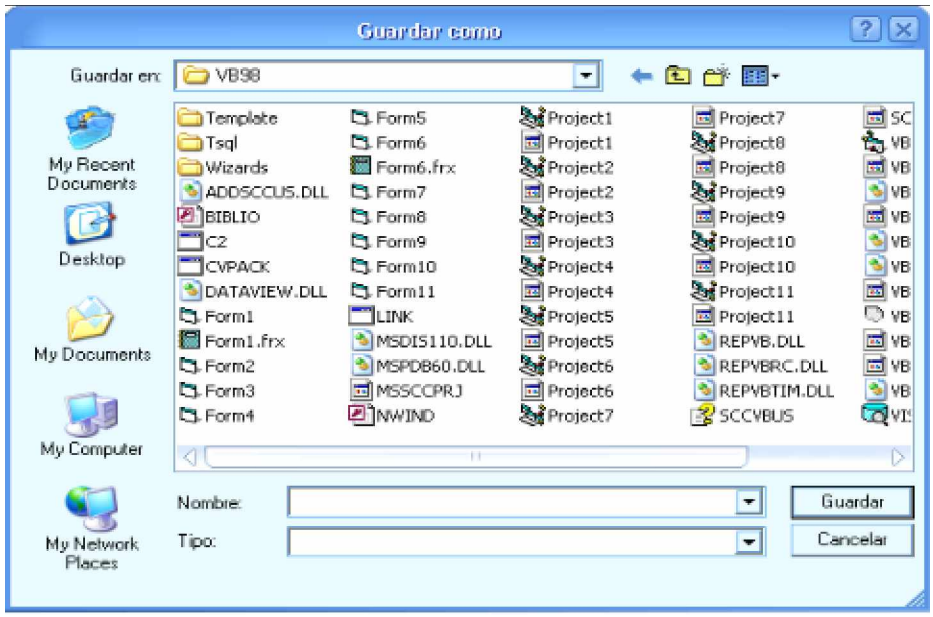


Figura 3.2. Cuadro de dialogo Guardar (Save).

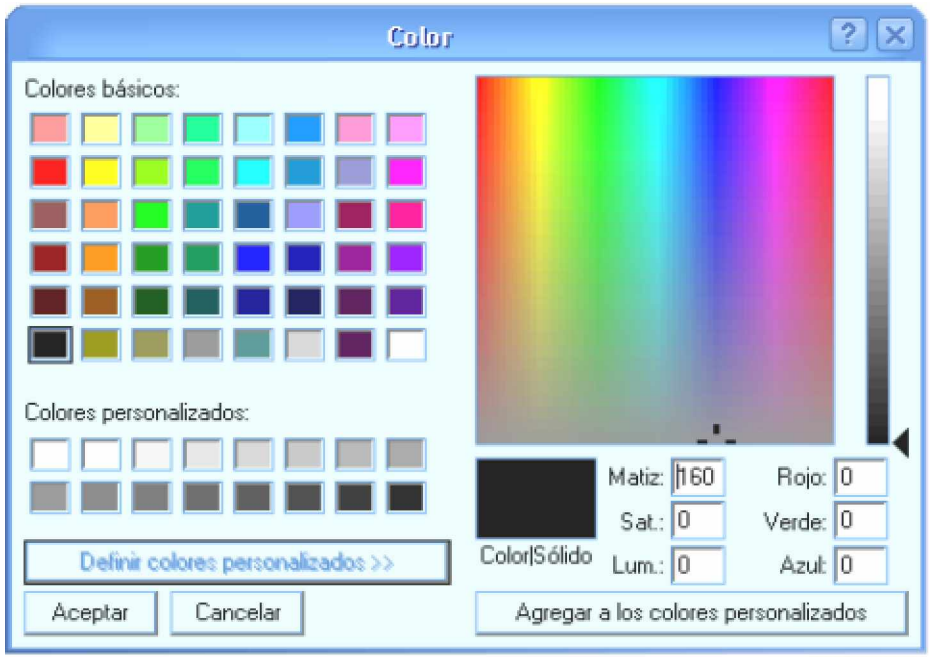


Figura 3.3. Cuadro de dialogo color.

Figura 3.4. Cuadro de dialogo Fuente.

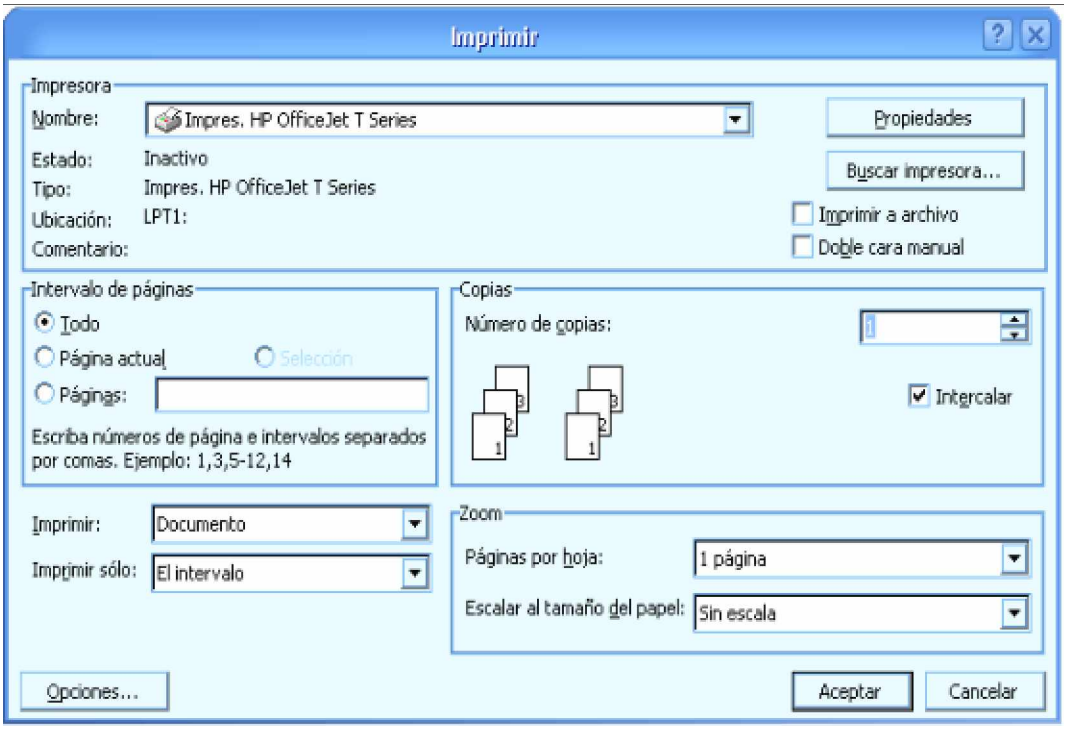
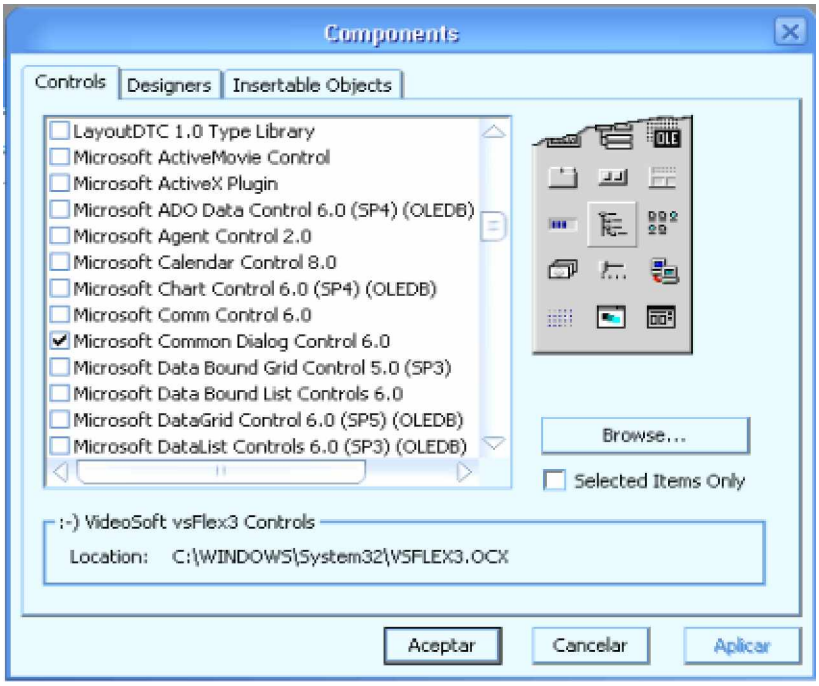


Figura 3.5. Cuadro de dialogo de Impresión.

En las figuras anteriores se pueden observar distintos tipos de ventanas de dialogo que puede proporcionar el *control* **CommonDialog**. Por ejemplo, si se desea visualizar una de las ventanas de dialogo o cuadro de dialogo, solo debe especificar el nombre del *control* y el *método* que corresponde a esa ventana:

```
CommonDialog1.ShowOpen     ‘ Abre el cuadro de dialogo Abrir.
```

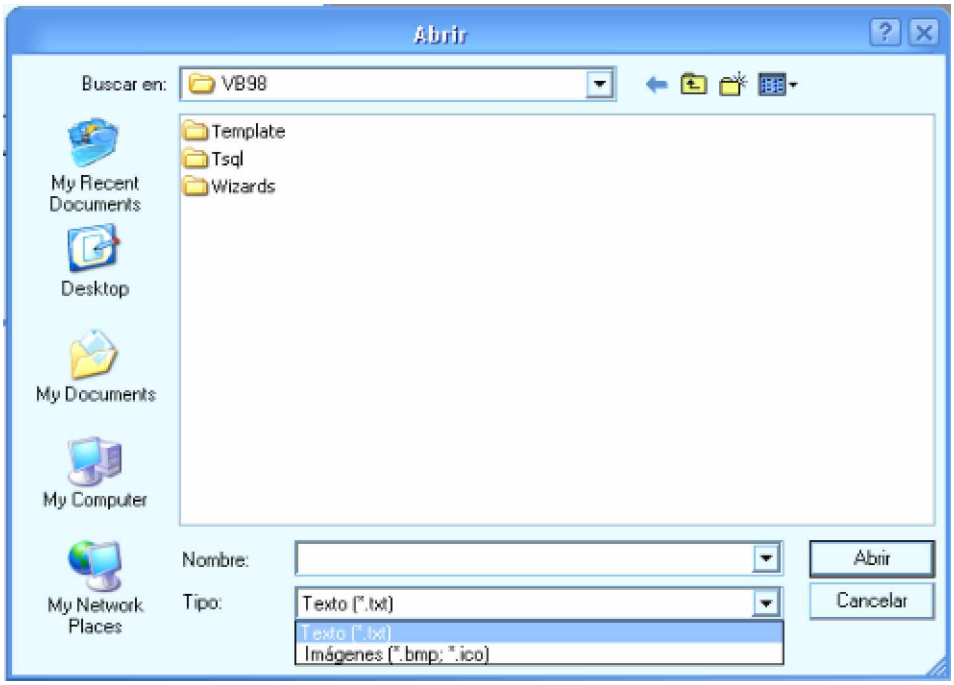
El *control* **CommonDialog** no aparece por defecto en la barra de herramientas no estándar. Para agregar este *control* a la barra de *herramientas no estándar*, haga clic en el menú **Project** y luego seleccione la opción **Components…** o simplemente presione **Ctrl + T**. Aparecerá el siguiente cuadro de dialogo:



En la ventana **Components** seleccione el elemento ***Microsoft Commond Dialog Control 6.0*** tal y como se puede apreciar en la imagen y luego, haga clic en el *botón* **Aceptar**. El control **CommonnDialog** se agregará a la barra de *herramientas no estándar*.

- 3.3.1 Ventana abrir y gravar (Open/Save) del Dialog Control

A continuación se muestra una tabla con las *propiedades* más importantes para los *métodos* **ShowOpen** y **ShowSave**.

Propiedad	Descripción
DefaultExt	Es la extensión por defecto a utilizar para **abrir/salvar** archivos. Con **Save**, si el nombre del fichero se teclea sin extensión, se añade esta extensión por defecto.
DialogTitle	Devuelve o da valor al título de la caja de diálogo.
FileName	Nombre completo del archivo a **abrir/salvar**, incluyendo el **path**.
FileTitle	Nombre del archivo a abrir/salvar sin la ruta de acceso correspondiente.
Filter	Contiene los filtros de selección que aparecerán indicados en la parte inferior de la pantalla en la lista de tipos de archivo. Pueden indicarse múltiples tipos de archivo, separádolos mediante una barra vertical **"|"** que se puede obtener pulsando las teclas Ctrl + Alt + 1. **Su sintaxis es la siguiente:** Objeto.Filter = "(descripción a aparecer en la caja de lista) | filtro" **Ejemplo:** CommonDialgo1.ShowOpen CommonDialog1.Filter = "Texto (*.txt) | *.txt | Imágenes (*.bmp; *.ico) | *.bmp; *.ico" Aparecerá la siguiente ventana con la lista **Tipo** de la siguiente manera: 

Propiedad	Descripción
FilterIndex	Indica el índice de los elementos del filtro. Por defecto empieza a enumerar por "1".
InitDir	Contiene el nombre del directorio por defecto. Si no se especifica, se utiliza el directorio actual.
Flags	Devuelve o establece las opciones de los cuadros de dialogo que muestra el control **CommonDialog**.

- 3.3.2 Ventana de imprimir (Print) del Dialog Control

A continuación se muestra una tabla con las propiedades más importantes para el *método* de impresión (**ShowPrint**).

Propiedad	Descripción
Copies	Determina el número de copias a realizar por la impresora.
FromPage	Selecciona el número de página a partir del cual comienza el rango de impresión.
ToPage	Selecciona el número de página hasta la cual llega el rango de impresión.
PrinterDefault	Cuando es **True** se imprime en el objeto **Visual Basic Printer**. Además las opciones actuales de impresión que se cambien serán asignadas como las opciones de impresión por defecto del sistema.

- 3.3.3 Ventana de fuente (Font) del Dialog Control

Las propiedades más importantes para el *método* **ShowFont** son:

Propiedad	Descripción
Color	Color de impresión. Para usar esta propiedad hace falta establecer la propiedad **Flags** al valor de la constante **cdlCFEffects**.
FontBold, FontItalic, FontStrikethru, FontUnderline	Devuelve o asigna los valores de los estilos de la fuente actual.
FontSize	Devuelve o asigna el tamaño de la fuente en uso.
FontName	Devuelve o asigna el nombre de la fuente en uso.
Min y Max	Asigna o lee los valores del tamaño de fuente mínimo y máximo respectivamente que aparecerán en la lista de selección del tamaño de la fuente.

3.4 Los formularios

Se sobre entiende que un *formulario* es la ventana de máximo nivel en la que aparecen los distintos *controles* de la aplicación. Pero se debe tener en cuenta que los *formularios* también poseen propiedades, métodos y eventos sobre los cuales el usuario trabaja para propósitos específicos en la aplicación. A continuación se muestra una tabla con las propiedades más importantes de los *formularios*.

- 3.4.1 Propiedades de los formularios

Propiedad	Descripción
BorderStyle	Establece el estilo de borde del formulario. Puede tomar entre los valores: **0- None** (Sin borde), **1- FixedSingle** (Simple fijo), **2- Sizable** (Predeterminado. Tamaño ajustable), **3- FixedDouble** (Dialogo fijo), **4- FixedToolWindow** (Ventana fija), **5- SizableToolWindow** (Ventana de tamaño ajustable). Para más información acerca de los estilos puede consultar la ayuda (**help**) de **Visual Basic 6.0**.
Caption	Muestra el título del formulario.
BackColor	Establece el color de fondo del formulario.
MaxButton	Determina si el botón de maximizar estará activo o no. Puede tomar los valores **True** o **False**.
MinButton	Determinar si el botón de minimizar estará activo o no. Puede tomar los valores **True** o **False**.
MDIChild	Devuelve o establece un valor que indica si un *formulario* debe mostrarse como **formulario secundario MDI** dentro de un **formulario MDI**. Es de sólo lectura en *tiempo de ejecución*. Puede tomar los valores **True** o **False**.
ShowInTaskBar	Determina si el icono de la aplicación se muestra cuando el formulario se minimiza. Puede tomar los valores **True** o **False**.
StartUpPosition	Establece la posición inicial del *formulario*. Puede tomar los valores: **0- StartUpManual** (Ninguna posición inicial), **1- StartUpOwner** (Centrado en el elemento que pertenece), **2- StartUpScreen** (Centrado en toda la pantalla) y **3- StarUpWindowsDefault** (Esquina superior izquierda de la pantalla).
WindowState	Establece el estado del *formulario* al iniciar la aplicación. El estado puede ser **0- Normal**, **1- Minimized** (el formulario aparece minimizado) ó **2- Maximized** (el formulario aparece maximizado).

- 3.4.2 Métodos sobre los formularios

A continuación se muestra una tabla con los *métodos* más usuales sobre los formularios.

Método	Descripción
Hide	Oculta un *formulario* pero no lo descarga.
Show	Muestra un formulario ya sea que este se halla ocultado o que nunca se halla ocultado.
Print	Imprime el formulario con todos sus *controles*. Si no quiere imprimir un *control* en especifico solo debe establecer el valor **False** a la propiedad **Visible** de ese *control*.

- 3.4.3 Eventos de los formularios

A continuación se muestra una tabla con los *eventos* más usuales de los *formularios*.

Evento	Descripción
Load	Ocurre al momento que el *formulario* se *carga*. Una *carga* ocurre cuando un formulario es mostrado o al momento que la aplicación se ejecuta. **Load** también puede ser utilizado para cargar un formulario, es decir, ponerlo en funcionamiento a el y a todos sus *controles* sobre el pero sin mostrar el formulario.
Unload	Ocurre cuando el formulario se descarga. Una *descarga* ocurre cuando el *formulario* o la ventana son cerradas, ya sea por medio de códigos o por el botón cerrar de la aplicación. **Unload** también puede ser utilizado para cerrar un *formulario*. Ejemplo: **Unload Me** 'Descarga el formulario actual.
Resize	Ocurre cuando un objeto se muestra primero o cuando cambia el estado de una ventana. (Por ejemplo, cuando se maximiza, minimiza o restaura un formulario.)
Terminate	Se produce cuando todas las referencias a una instancia de un *formulario*, un *formulario* MDI, un control **User**, una página de propiedades, una clase de Web, un diseñador de páginas DHTML o una clase se quitan de la memoria estableciendo a **Nothing** todas las variables que hacen referencia al objeto o cuando la última referencia al objeto queda fuera del *alcance*.

- 3.4.4 Formularios múltiples

Un programa puede contener más de un *formulario.* De hecho, habitualmente los programas contienen múltiples *formularios.*

Sin embargo, un programa siempre debe tener un ***formulario principal***, que es el que aparece al arrancar el programa. Se puede indicar cuál debe ser el *formulario* principal en el menú **Project/Project Properties**, en la ficha **General**, en la sección **Startup Form**. Por defecto, el programa considera como *formulario* principal el primero que se haya creado. El resto de *formularios* que se incluyan en el programa serán cargados en su momento, a lo largo de la ejecución del programa.

Para añadir en tiempo de diseño nuevos *formularios* al programa, hay que acudir al menú **Project/Add Form**. La forma de *cargar* y *descargar* estos *formularios* se ha explicado con anterioridad. Es importante sin embargo recordar que conviene descargar aquellos sub-formularios que ya no sean de utilidad, ya que así se ahorran recursos al sistema.

- 3.4.4.1 Formularios MDI (Multiple Document Interface)

En algunos casos puede ser interesante establecer una jerarquía entre las *ventanas* o *formularios* que van apareciendo sucesivamente en la pantalla del ordenador, de tal manera que al cerrar una que se haya establecido como principal, se cierren también todas las que se han abierto desde ella y dentro de ella. De esta forma una misma aplicación puede tener varios documentos abiertos, uno en cada ventana hija. Así trabajan por ejemplo los programas **Word, Excel, PowerPoint,** etc. A este conjunto de *documentos* o *ventanas* que trabajan a conjunto y dependen de una *ventana* principal se le llama **MDI** (**Multiple Document Interface**).

Un formulario **MDI** (**interfaz de múltiples documentos**) es una ventana que actúa como fondo de una aplicación y es el contenedor de formularios que tienen su propiedad **MDIChild** establecida a **True**.

Sintaxis

MDIForm

Para crear un objeto **MDIForm**, elija **Agregar formulario MDI** en el menú **Proyecto**.

Una aplicación sólo puede tener un objeto **MDIForm**, pero varios formularios *secundarios MDI*. Si un formulario secundario MDI tiene menús, la barra de menús del formulario secundario reemplazará automáticamente a la barra de menús del objeto **MDIForm** cuando el formulario secundario MDI esté activo. Un formulario secundario MDI minimizado se mostrará como un icono en el **MDIForm**.

Un objeto **MDIForm** sólo puede contener controles **Menu** y **PictureBox**, y controles personalizados que tengan una propiedad **Align**. Para colocar otros controles en un **MDIForm**, puede dibujar un cuadro de imagen en el formulario y después dibujar otros controles dentro del cuadro de imagen. Puede usar el método **Print** para mostrar texto en un cuadro de imagen de un **MDIForm**, pero no puede usar este método para mostrar texto en el **MDIForm** propiamente dicho.

Un objeto **MDIForm** no puede ser *modal*.

Los formularios secundarios MDI se diseñan de forma independiente del **MDIForm**, pero siempre están contenidos en el **MDIForm** en *tiempo de ejecución*.

Puede tener acceso a la colección de controles de un **MDIForm** mediante la colección **Controls**. Por ejemplo, para ocultar todos los controles de un **MDIForm** puede usar código similar a éste:

```
For Each Control in MDIForm1.Controls
   Control.Visible = False
Next Control
```

La propiedad **Count** del **MDIForm** indica el número de controles de la colección **Controls**.

3.5 Controles basados en arreglos (arrays)

Un **array** de *controles* esta formado por *controles* del mismo tipo que comparten el nombre y los procedimientos o funciones para gestionar los *eventos*. Para identificar a cada uno de los controles pertenecientes al **array** se utiliza **Index** o **Indice**, que es una propiedad más del control.

La utilidad principal de los **arrays** se presenta en aquellos casos en los que el programa debe responder de forma semejante a un mismo evento sobre varios *controles* del mismo tipo. Los botones más claro son los botones de opción y los menús.

En estos casos el programa responde de manera semejante independientemente de cuál es la opción seleccionada. Los **arrays** de *controles* comparten código, lo cual quiere decir que sólo hay que programar una función para responder a un evento de un determinado tipo sobre cualquier *control* del **array**. Las funciones que gestionan los eventos de un **array** tienen siempre un argumento adicional del tipo **Index As Integer** que indica qué *control* del **array** ha recibido el evento.

3.6 Imagen con todos los *controles* más usuales en Visual Basic 6.0

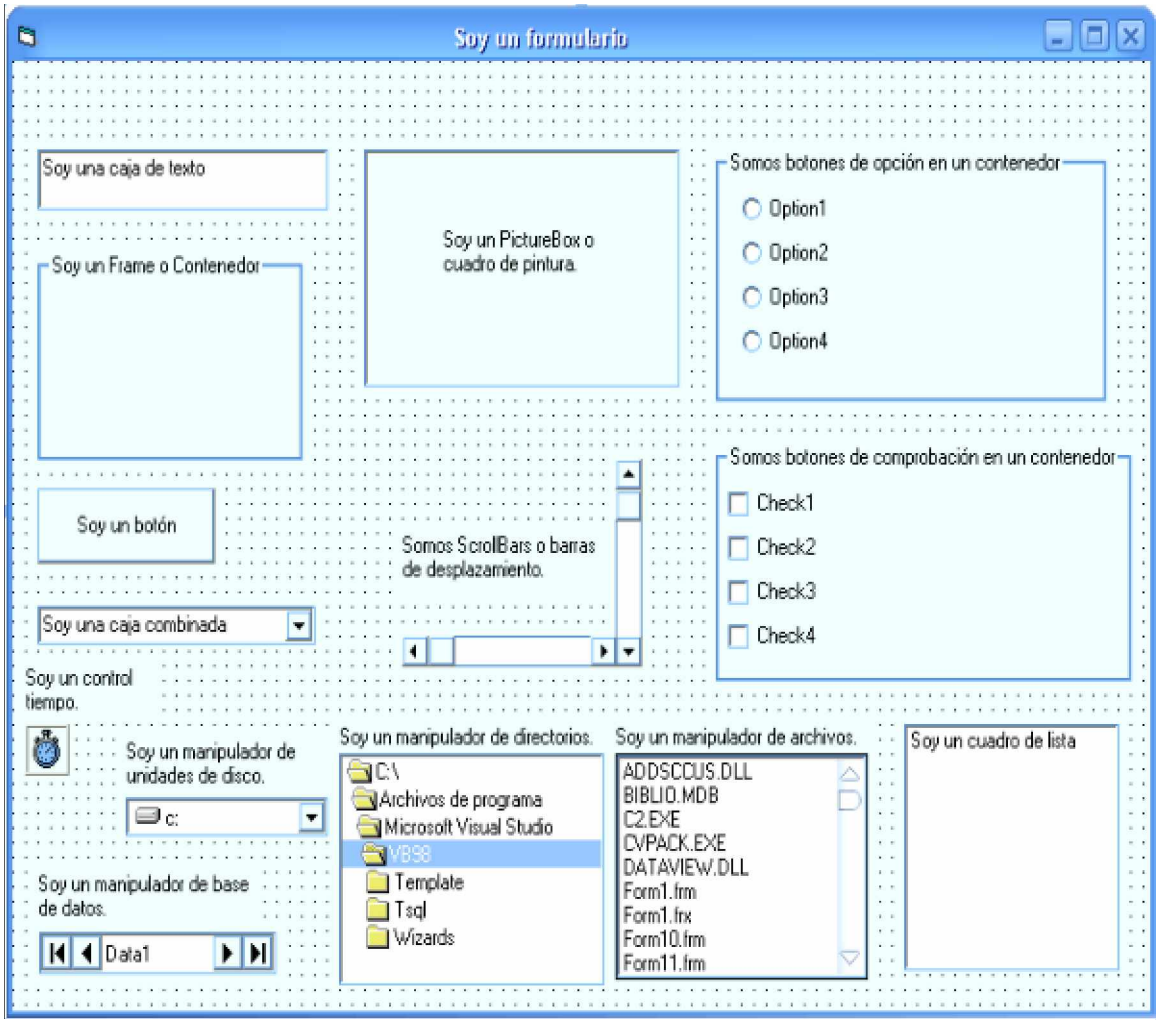


Figura 3.6. Los controles más usados en Visual Basic 6.0

Capitulo IV

CONTENIDO

4.19 El lenguaje Basic

- 4.1.1 Introducción

Después de haber conocido todos los elementos y *controles* más importantes de Visual Basic 6.0 es imprescindible conocer los fundamentos del lenguaje de programación **Basic**.

Un **programa** de computadora está constituido en sentido general por **variables** que contienen los datos con los que se trabaja y por **algoritmos** que son las sentencias que operan sobre estos datos. Estos datos y **algoritmos** suelen estar incluidos dentro de *funciones* y *procedimientos*.

Desde un principio los distintos tipos de *lenguaje de programación* han manejado variables y sentencias que permiten realizar operaciones simples y complejas que de una forma u otra proporcionan *valores* que juegan un papel muy importante en la aplicación (programa). De tal manera, **Visual Basic 6.0** a pesar de ser un lenguaje de programación orientado a objetos y no estructurado, proporciona una gran cantidad de herramientas y sintaxis para la manipulación de *valores* y *variables* que son de suma utilidad en cualquier aplicación que se realice en **Visual Basic**.

En este capitulo trataremos el uso de los *comentarios*, las *variables*, *constantes*, los *distintos tipos de operadores* y los *distintos tipos de datos* para las *variables*.

4.20 Comentarios y otros elementos en el Lenguaje Basic

Visual Basic 6.0 interpreta que todo lo que está a la derecha de una comilla simple (**‘**) en una línea cualquiera del programa es un **comentario** y no lo tiene en cuenta al momento de compilación. El **comentario** puede empezar al comienzo de la línea o a continuación de una instrucción que debe ser ejecutada.

Una de las características principales de los **comentarios** es que toman el color verde por defecto al momento de ser declarados, por ejemplo:

```
‘Este proceso que esta debajo de mi, suma dos valores contenidos en A y B.
S = A + B          ‘ Almacena la suma en la variable S.
```

Los *comentarios* son bastante útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. En programas que no contengan muchas líneas de código puede no parecer demasiado importante, pero cuando se trata de proyectos realmente complejos, o desarrollados por varias personas su utilización es imprescindible. En el caso de que el código no esté *comentado* este trabajo de actualización y revisión puede resultar una tarea muy complicada.

Otro aspecto práctico en la programación es la posibilidad de escribir una *sentencia* en más de una línea. En el caso de *sentencias* bastantes largas es conveniente cortar la línea para que entre en la pantalla. Para ello es necesario dejar un *espacio en blanco* al final de la línea y escribir un underscore (_) o carácter de subrayado tal y como se muestra a continuación.

```
If (a > b) And (a > c) And (a > d) And (a > e) And (a > f) And (a > g) And (a > h) _
And (b <> 0) And (c <> 0) And (d <> 0) And (e <> 0) And (f <> 0) And (g <> 0) Then
        MsgBox ("El valor de a es: " & a)
End If
```

Visual Basic 6.0 permite también incluir varias sentencias en una misma línea. Para ello las sentencias deben ir separadas por el carácter *dos puntos* (**:**). Por ejemplo:

```
MsgBox ("Bienvenido") : MsgBox ("a Visual Basic 6.0")
```

4.21 Objeto de un programa

Los objetos son las unidades significativas más pequeñas de texto de una aplicación **Visual Basic** y se dividen en: símbolos especiales, identificadores, etiquetas, números, constantes, variables, etc.

- 4.3.1 Identificadores

Los *identificadores* representan los objetos de un programa (constantes, variables, tipos de datos, procedimientos, funciones, programas, campos de registros, etc.). Un *identificador* es una secuencia de caracteres que puede ser de cualquier longitud, pero sólo 255 primeros caracteres son significativos.

Un identificador se caracteriza por estas reglas:

1. Debe comenzar con una letra (A a Z, mayúsculas o minúsculas) y no puede contener blancos.

2. Letras, dígitos y caracteres subrayados (_) están permitidos después del primer carácter excepto como último carácter.

3. No se puede utilizar una palabra reservada como identificador.

4. Los caracteres %, &, !, #, @, $ solo pueden ser usados al final del identificador(variable) para indicar el tipo de datos que almacenará dicha variable.

Estos pueden ser identificadores válidos:

Nombre

Area_Rectangulo

Lado1

Impuesto_Sobre_La_Renta

ImpuestoTotales

Estos son idenficiadores no validos:

Area&Rectangulo	El ampersand (&) solo puede ser usado al final
Lado 1	Contiene espacios en blanco
Impuesto@Renta	Tiene un @ después del primer carácter
For	Es una palabra reservada en **Visual Basic 6.0**

- 4.3.2 Palabras reservadas en Visual Basic 6.0

Las palabras reservadas en **Visual Basic 6.0** no se pueden utilizar como identificadores, ya que tienen significado especial en **Visual Basic** y no se utilizan para otros propósitos. A continuación se muestra una lista con las palabras reservados en **Visual Basic 6.0**.

Abs	Add	And	App
Activate	Array	Ascatn	Base
Beep	Cbool	Cbyte	Ccur
Cdate	Cdbl	Cdec	Cint
Clng	Csng	Cstr	Cvar
Cverr	Call	Case	Cdh
Choose	Chdir	Chdrive	Clear
Close	Collection	Command	Compare
Const	Cos	CreateObject	Curdir
Dateserial	Datevalue	Day	Ddb
Defftype	Dim	Deletesetting	Dir
Do	Doevents	Each	End
Environ	Eof	Eqv	Erase
Err	Error	Exit	Exp
Explicit	Fileattr	Filecopy	Filedatetime
Filelen	Fix	For	Format
Freefile	Function	Fv	Get
Getattr	GetObject	Getsetting	Getallsetting
Gosub	Goto	Hex	Hour
Imp	Input	Instr	Int
Integer	Ipmt	Irr	Is
Isarray	Isdate	Isempty	Iserror
Ismissing	Isnull	Isnumeric	Isobject
Item	Kill	Lbound	Lcase
Left	Let	Like	Loc
Lock	Lof	Log	Loop
Lset	Ltrim	Me	Mid
Minute	Mirr	Mkdir	Mod
Month	Name	New	Next
Not	Nper	Npv	Oct
On	Onerror	Open	Or
Option	Print	Ppmt	Print#
Private	Property	Public	Put
Pv	Qbcolor	Raise	Randomize
Rate	Redim	Remove	Reset
Resume	Return	Rgb	Right
Rmdir	Rnd	Rset	Rtrim
Savesettings	Second	Selectcase	Seek
Shell	SendKeys	Setattr	Sgn
Sin	Single	Sln	Space
Spc	Sqr	Static	Stop
Str	Strcomp	Strconv	String
Sub	Syd	Switch	Tab
Tan	Timer	TimeSerial	TimeValue
Trim	Typename	Ubound	Ucase
Unlock	Val	Vartype	Weekday
Wend	While	Width	Write#
Xor	Yeqr	#if	#else

Palabras del propio lenguaje de **Visual Basic 6.0.**

4.22 Tipos de datos de variables

Los tipos de datos de variables son los distintos objetos de información con los que trabaja una aplicación en Visual Basic. Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como un 'B', un valor entero tal como 90 o un número real tal como 5.16.

- 4.4.1 Clasificación de los tipos de datos

Los tipos de datos de variables se pueden clasificar de acuerdo a su almacenamiento en: tipos enteros (Byte, Integer, Long), tipos reales (Single, Double, Currency), tipos cadena (String), tipos lógicos (Boolean), tipos fecha (Date), tipos variados (Variant).

- 4.4.1.1 Tipos enteros (Byte, Integer, Long)

Visual Basic tiene tres tipos de datos predefinidos para representar los números enteros: Byte, Integer y Long.

1) Byte: Las variables tipo **Byte** se almacenan como números de 8 bits (1 byte) sencillos sin signo con un intervalo de valores entre 0 y 225. El tipo de datos **Byte** es útil para almacenar datos binarios.

2) Integer: Las variables **Integer** se almacenan como números de 16 bits (2 bytes) con valores que van de -32.768 a 32.767. El carácter de declaración de tipo para el tipo **Integer** es el signo de porcentaje (**%**).

Las variables tipo **Integer** también se pueden utilizar para representar valores enumerados. Un valor enumerado puede contener un conjunto finito de números enteros únicos, cada uno de los cuales tiene un significado especial en el contexto en el que se utiliza. Los valores enumerados proporcionan una forma cómoda de seleccionar entre un número conocido de opciones. Por ejemplo, cuando se pregunta al usuario que elija un color de una lista, se podría tener 0 = negro, 1 = blanco y así sucesivamente. Es una buena práctica de programación definir constantes utilizando la instrucción **Const** para cada valor enumerado.

3) Long: Las variables **Long** (enteros largos) se almacenan como números con signo de 32 bits (4 bytes) con un valor comprendido entre -2.147.483.648 y 2.147.483.647. El carácter de declaración de tipo para **Long** es el signo **&**.

- **4.4.1.2 Tipos reales (Single, Double, Currency)**

Visual Basic también posee tres tipos de datos para representar los números reales: Single, Double y Currency.

1) Single: Las variables **Single** (punto flotante de precisión simple) se almacenan como números IEEE de coma flotante de 32 bits (4 bytes) con valores que van de -3,402823E38 a -1,401298E-45 para valores negativos y de 1,401298E-45 a 3,402823E38 para valores positivos. El carácter de declaración de tipo para Single es el signo de exclamación (**!**).

2) Double: Las variables dobles (punto flotante de doble precisión) se almacenan como números IEEE de coma flotante de 64 bits (8 bytes) con valores de -1,79769313486232E308 a -4,94065645841247E-324 para valores negativos y de 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos. El carácter de declaración de tipo para **Double** es el signo de número (**#**).

3) Currency: Las variables tipo **Currency** se almacenan como números de 64 bits (8 bytes) en un formato de número entero a escala de 10.000 para dar un número de punto fijo con 15 dígitos a la izquierda del signo decimal y 4 dígitos a la derecha. Esta representación proporciona un intervalo de -922.337.203.685.477,5808 a 922.337.203.685.477,5807. El carácter de declaración de tipo para **Currency** es el signo **@**.

El tipo de datos **Currency** es útil para cálculos monetarios y para cálculos de punto fijo, en los cuales la precisión es especialmente importante.

- **4.4.1.3 Tipos cadena (String)**

Hay dos clases de cadenas: cadenas de longitud variable y cadenas de longitud fija.

- Las cadenas de longitud variable pueden contener hasta 2.000 millones de caracteres (2^31).
- Las cadenas de longitud fija que pueden contener de 1 a 64 KB (2^16) caracteres.

Nota: No se puede usar una cadena **Public** de longitud fija en un *módulo de clase*.

Los códigos para caracteres de tipo **String** varían desde 0 a 255. Los primeros 128 caracteres (0–127) del juego de caracteres corresponden a las letras y los símbolos de un teclado estándar de EE.UU. Estos primeros 128 caracteres son los mismos que los definidos por el juego de caracteres **ASCII**. Los siguientes 128 caracteres (128–255) representan caracteres especiales, como letras de alfabetos internacionales, acentos, símbolos de moneda y fracciones. El carácter de declaración de tipo para **String** es el signo de dólar (**$**).

- 4.4.1.4 Tipos lógicos (Boolean)

Las variables tipo **Boolean** se almacenan como números de 16 bits (2 bytes), pero sólo pueden ser **True** o **False**. Las variables tipo **Boolean** se presentan como `True` o `False`(cuando se utiliza **Print**) o #TRUE# o #FALSE# (cuando se utiliza **Write #**). Utilice las *palabras clave* **True** y **False** para asignar uno de los dos estados a las variables tipo **Boolean**.

Cuando se convierten a tipo **Boolean** otros *tipos numéricos*, 0 se convierte en **False**, y el resto de los valores se convierten en **True**. Cuando los valores tipo **Boolean** se convierten a *otros tipos de datos* numéricos, **False** se convierta en 0 y **True** se convierte en -1.

- 4.4.1.5 Tipos variados (Variant)

Una variable **Variant** es capaz de almacenar todos los tipos de datos definidos en el sistema. No tiene que convertir entre esos tipos de datos si los asigna a una variable **Variant**; **Visual Basic** realiza automáticamente cualquier conversión necesaria.

4.23 Constantes

A menudo verá que el código contiene *valores constantes* que reaparecen una y otra vez. O puede que el código dependa de ciertos números que resulten difíciles de recordar (números que, por sí mismos, no tienen un significado obvio).

En estos casos, puede mejorar mucho la legibilidad del código y facilitar su mantenimiento si utiliza constantes. Una *constante* es un nombre significativo que sustituye a un número o una cadena que no varía. Aunque una *constante* recuerda ligeramente a una variable, no puede modificar una constante o asignarle un valor nuevo como ocurre con una variable. Hay dos orígenes para las *constantes*:

Constantes intrínsecas o definidas por el sistema: proporcionadas por aplicaciones y controles. Las constantes de **Visual Basic** se muestran en **Visual Basic (VB)** y **Visual Basic** para aplicaciones **(VBA)** y las bibliotecas de objetos en el Examinador de objetos. Otras aplicaciones que proporcionan bibliotecas de objetos, como Microsoft Excel y Microsoft Project, también proporcionan una lista de constantes que puede usar con sus objetos, métodos y propiedades. También se definen constantes en la biblioteca de objetos de cada control **ActiveX**. Para obtener más detalles acerca de la utilización del Examinador de objetos, vea "Programar con objetos".

Las constantes simbólicas o definidas por el usuario: se declaran mediante la instrucción **Const**. Las constantes definidas por el usuario se describen en la próxima sección, "Crear sus propias constantes".

En **Visual Basic**, los nombres de *constantes* tienen un formato que combina mayúsculas y minúsculas, con un prefijo que indica la biblioteca de objetos que define la constante. Las *constantes* de las bibliotecas de objetos de **Visual Basic** y **Visual Basic** para aplicaciones tienen el prefijo "vb"; por ejemplo, **vbTileHorizontal**.

- 4.5.1 Declaración de constantes

La sintaxis para declarar una *constante* es la siguiente:

[Public | Private] Const NombreConstante **[As Tipo]** = Expresión

El *argumento* ***NombreConstante*** es un nombre simbólico valido (Las reglas son las mismas que para crear variables), y **Expresión** está compuesta por *constantes* y operadores de cadena o números. Sin embargo, no puede usar llamadas a funciones en expresión.

La declaración de una *constante* puede ser de distintos tipos: *Pública* (dentro de un módulo), *Privada* (en el área de declaraciones general de un formulario) o *Local* (dentro de un procedimiento).

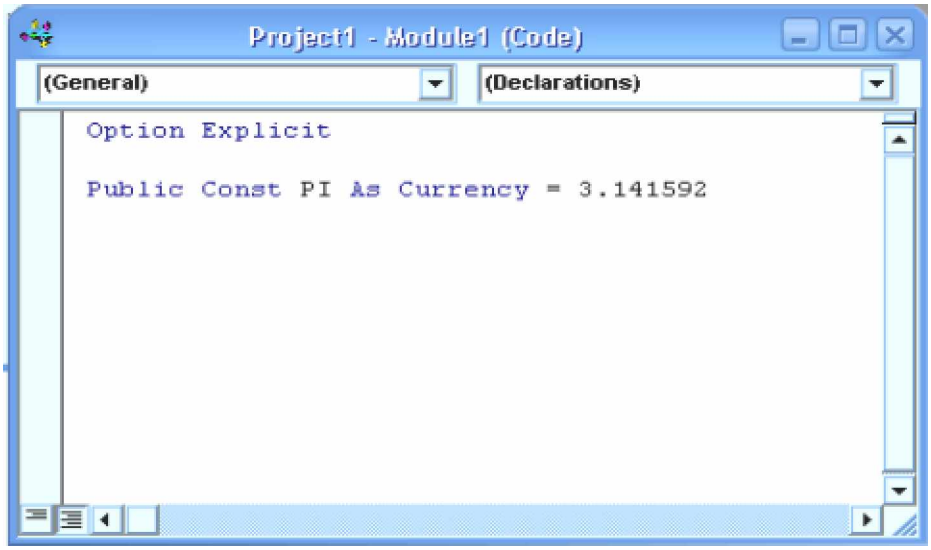
1.- Declaración de una constante pública:

Declarar una *constante pública* significa que esa constante podrá ser utilizada desde cualquier

procedimiento, formulario o módulo. Para declarar una *constante pública* siga los siguientes pasos:

a) Inserte un módulo desde el menú **Project/Add Module**.
b) Dentro de ese modulo escriba la instrucción **Public** seguida de la instrucción **Const** y a continuación, el *nombre de la constante* con su *tipo asociado*.

Ejemplo:

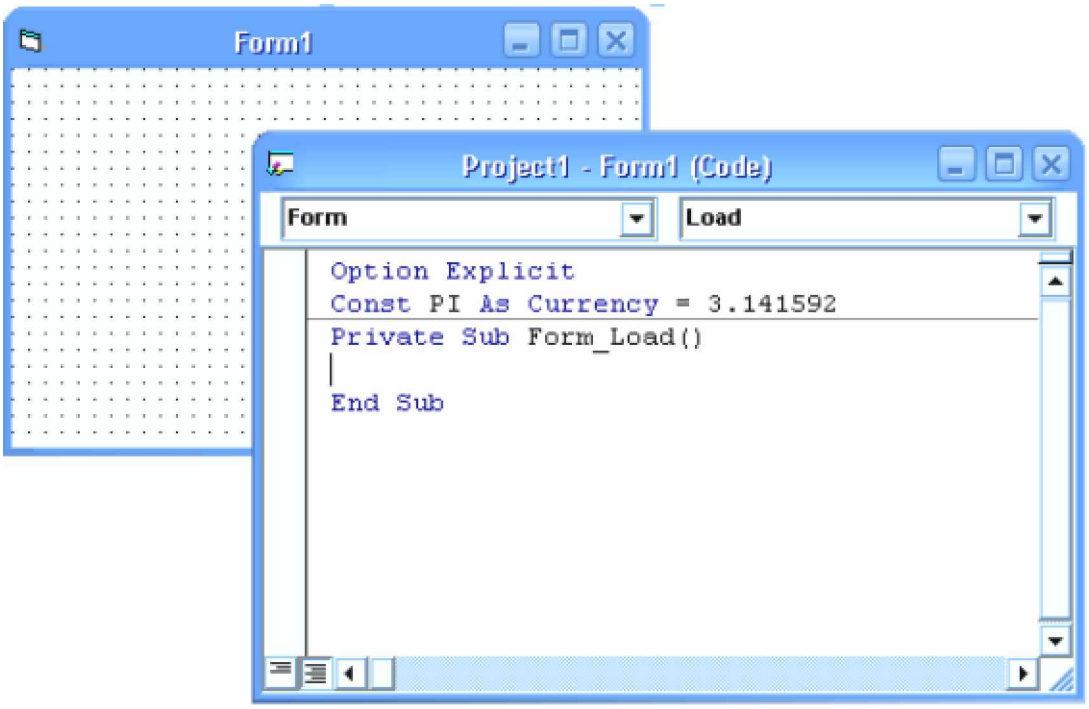


c) Después de hacer esta declaración puede usar la *constante* PI dentro de cualquier procedimiento (un botón de comando, un control ListBox, un formulario, etc.) sin la necesidad de referenciar al módulo que la contiene.

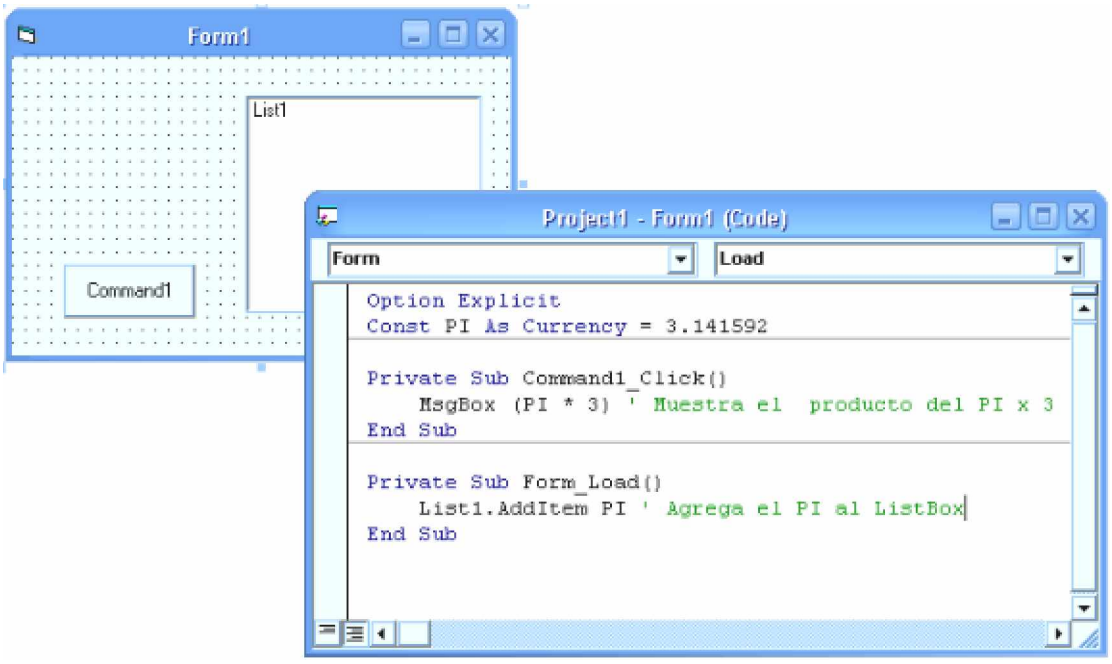
2.- Declaración de una constante privada:

Declarar una *constante privada* significa que esa *constante* puede ser usada dentro de todos los procedimientos de un mismo formulario o dentro del formulario donde se declara la *constante*. Para declarar una *constante privada* siga los siguientes pasos:

a) En la declaración **(General)** de un formulario escriba la instrucción **Const** seguida del nombre de la constante, su tipo asociado y expresión. Como se muestra a continuación:



b) Al hacer esto puede utilizar la *constante* PI desde cualquiera de los procedimientos dentro de ese formulario, es decir, puede agregar un botón de comando, un *control* ListBox, una etiqueta, cualquier control en general y hacer referencia a esta constante solo por su nombre desde cualquiera de los *eventos* de dicho control.



3.- Declaración de una constante local:

Declarar una *constante local* significa que esa *constante* solo puede ser usada dentro del procedimiento donde se declara. Para declarar una *constante local* escriba la instrucción **Const** seguida del nombre de la *constante*, su tipo asociado y su expresión dentro del mismo procedimiento.

Ejemplo:

4.6 Variables

En **Visual Basic** puede usar *variables* para almacenar valores temporalmente durante la ejecución de una aplicación. Las *variables* tienen un nombre (la palabra que utiliza para referirse al valor que contiene la *variable*) y un tipo de dato (que determina la clase de datos que la *variable* puede almacenar).

Puede considerar una *variable* como un marcador de posición en memoria de un valor desconocido. Por ejemplo, suponga que está creando un programa para una frutería que haga un seguimiento del precio de las manzanas. No sabe el precio de una manzana o la cantidad que se ha vendido hasta que no se produce realmente la venta. Puede usar dos *variables* para almacenar los valores desconocidos (vamos a llamarlos PrecioManzanas y ManzanasVendidas). Cada vez que se ejecuta el programa, el usuario especifica los valores de las dos *variables*. Para calcular las ventas totales y mostrarlas en un cuadro de texto llamado txtVentas, el código debería parecerse al siguiente:

txtVentas.text = PrecioManzanas * ManzanasVendidas

La expresión devuelve un total distinto cada vez, dependiendo de los valores que indique el usuario. Las *variables* le permiten realizar un cálculo sin tener que saber antes cuáles son los valores especificados.

4.6.1 Declaraciones de variables

Declarar una *variable* es decirle al programa algo de antemano. Se declara una variable mediante la instrucción **Dim**, proporcionando un nombre a la *variable*:

Dim nombreVariable **[As tipo]**

Las *variables* que se declaran en un procedimiento mediante la instrucción **Dim** sólo existen mientras se ejecuta el procedimiento. Cuando termina el procedimiento, desaparece el valor de la variable. Además, el valor de una *variable* de un procedimiento es local de dicho procedimiento; es decir, no puede tener acceso a una variable de un procedimiento desde otro procedimiento. Estas características le permiten usar los mismos nombres de *variables* en distintos procedimientos sin preocuparse por posibles conflictos o modificaciones accidentales.

La cláusula opcional **As** *tipo* de la instrucción **Dim** le permite definir el tipo de dato o de objeto de la *variable* que va a declarar. Los tipos de datos definen el tipo de información que almacena la *variable*. Algunos ejemplos de tipos de datos son **String, Integer** y **Currency**. Las *variables* también pueden contener objetos de **Visual Basic** u otras aplicaciones. Algunos ejemplos de tipos de objeto de **Visual Basic**, o clases, son *Object*, *Form1* y *TextBox*.

- 4.6.2 Nombres descriptivos de las variables

Los nombres de las variables deben estar sometidos a las siguientes reglas:

- Deben comenzar con una letra.
- No pueden incluir un punto o un carácter de declaración de tipo.
- No debe exceder de 255 caracteres.
- Deber ser única en el mismo alcance, es decir, o es pública, privada o solo local.

- 4.6.3 Almacenar y recuperar datos en variables

Utilice instrucciones de asignación para realizar cálculos y asignar el resultado a una *variable*:

```
ManzanasVendidas = 10              ' Se pasa el valor 10 a la variable.
ManzanasVendidas = ManzanasVendidas + 1   ' Se incrementa la variable.
```

Observe que el signo igual del ejemplo es un operador de asignación, no un operador de igualdad; el valor (10) se asigna a la variable (ManzanasVendidas).

4.7 Expresiones y operadores

Una *expresión* es un conjunto de datos o funciones unidos por operadores aritméticos. Las *expresiones* aritméticas están representados por: una *constante, una variable* o una *combinación de constantes* o *variables* unidas por operadores aritméticos.

Ejemplos de algunas expresiones aritméticas:

1) Suma = a + b
2) Area = (base * altura)/2

- 4.7.1 Operadores aritméticos: +, -, *, /

Son aquellos que se utilizan para realizar las operaciones básicas de las matemáticas. En las operaciones básicas tenemos: suma, resta, multiplicación y división.

Operador	Significado	Ejemplo	Resultado
+	Suma	a + b	Suma de a y b.
-	Resta	a – b	Diferencia entre a y b.
*	Multiplicación	a * b	Producto de a por b.
/	División	a / b	Cociente de a sobre b.
\	División entera	a \ b	Cociente entero de a sobre b.
Mod	Módulo o Resto	a mod b	Resto de a sobre b.
^	Exponenciación	a ^ b	Potencia de a elevado a b.

- 4.7.2 Operador Mod

Divide dos números y devuelve sólo el resto.

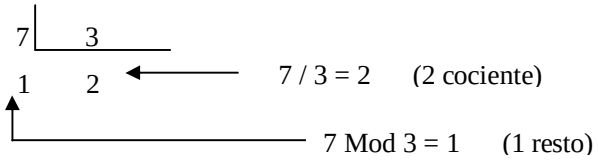
Sintaxis

Resultado = número1 Mod número2

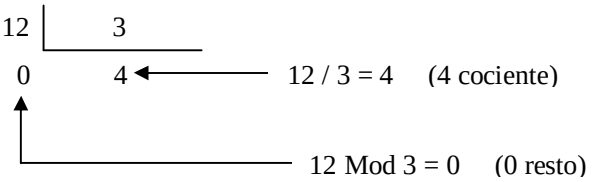
Donde *Resultado* es cualquier *variable* numérica, número1 y número2 es cualquier expresión numérica.

Ejemplos:

7 Mod 3 = 1



12 Mod 3 = 0



- 4.7.3 Operador lógico

Se utilizan para combinar las expresiones lógicas, es decir, que nos permiten evaluar más de una expresión lógica a la vez. Los *operadores* **and**, **or** y **not** trabajan con operando que son expresiones lógicas.

Su formato es:

[Operando1] **operador lógico** [Operando2]

Donde *operando1* y *operando2* pueden ser cualquier expresión lógica.

Ejemplos:

5 > 6 And 7 > 20 Falso.

3 > 1 And 6 < 20 Verdadero.

- El operador lógico And (y):

El *operador* **And (y)** combina dos o más condiciones simples y produce un resultado verdadero sólo si todos los operando son verdaderos.

Operador **and**		
Operando 1 **and** *Operando 2*		Valor
True	True	True
True	False	False
False	True	False
False	False	False

- El operador lógico or (o):

Proporciona un valor verdadero si uno de los operando es verdadero.

Operador **or**		
Operando 1 **or** *Operando 2*		Valor
True	True	True
True	False	True
False	True	True
False	False	False

El operador lógico not (no):

Niega el valor original de una expresión, si es verdadero será falso, si es falso será verdadero.

Operador **not**	
not *Operando 1*	Valor
True	False
False	True

- 4.7.4 Operadores de concatenación

La *concatenación* en la programación consiste en la unión de dos o más *cadenas de caracteres* mediante los símbolos (+) y (**&**). Unir dos o más cadenas de caracteres mediante uno de estos operadores es bastante sencillo y se asemeja al proceso realizado en una suma.

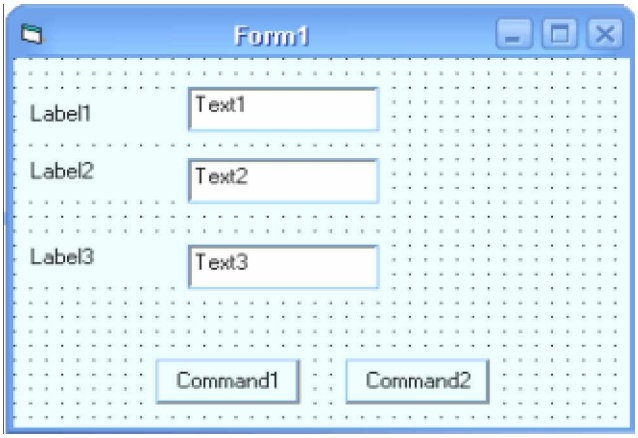
Ejemplo:
Cadena1 = “Buenos “
Cadena2 = “Días.”
Cadena3 = Cadena1 + Cadena2
MsgBox (Cadena3)
Cadena3 = Cadena1 & Cadena2
MsgBox (Cadena3)

4.8 Algunos ejercicios prácticos

● Se solicita calcular el área de un rectángulo donde: area = lado1 * lado2. El lado1 y el lado2 deben introducidos mediante dos cajas de texto y el resultado (area) debe ser mostrado en otra caja de texto.

Pasos a seguir:

a) Abra un nuevo proyecto desde el Menú **File/New Project**.
b) Inserte tres etiquetas, tres cajas de texto y dos botones de comando tal y como se muestra a continuación:



c) Aplica las siguientes características a los *controles* sobre el formulario:

Control	Propiedad	Valor
Form	*Caption*	Área de un rectángulo
Label1	*AutoSize* *Caption* *Font*	True Lado 1: Tamaño 10, Negrita.
Label2	*AutoSize* *Caption* *Font*	True Lado 2: Tamaño 10, Negrita.
Label3	*AutoSize* *Caption* *Font*	True Area: Tamaño 10, Negrita.
Text1	*Name* *Text*	txtLado1 (vacío)
Text2	*Name* *Text*	txtLado2 (vacío)
Text3	*Name* *Text*	txtArea (vacío)
Command1	*Name* *Caption*	cmdCalcular &Calcular
Command2	*Name* *Caption*	cmdSalir &Salir

d) La apariencia de los *controles* sobre el formulario debe ser la siguiente:

Ing. Carlos Manuel Rodríguez Bucarelly

e) Dentro de cada procedimiento escriba el código correspondiente:

```
Private Sub cmdCalcular_Click ( )
   Dim lado1, lado2, area As Long          ' Declara las variables
   lado1 = Val(txtLado1.Text)              ' Almacena el lado1
   lado2 = Val(txtLado2.Text)              ' Almacena el lado2
   area = lado1 * lado2                    ' Calcula el área
   txtArea.Text = area                     ' Muestra el área
End Sub
```

```
Private Sub cmdSalir_Click ( )
   End
End Sub
```

f) Corra la aplicación pulsando la tecla **[F5].**

g) Introduzca valores en las cajas de los lados del rectángulo y haga clic en el botón **Calcular**.

h) Salga de la aplicación y guárdela.

- Ejercicios propuestos

1) Se requiere calcular el área de un triángulo: ***area = (base * altura)/2***, la base y la altura deben ser introducidas mediante cajas de texto. El programa debe mostrar el área del triángulo en una caja de texto.

2) Se requiere calcular el volumen de una esfera: ***volumen = 4/3 * PI * radio³***

3) Se requiere evaluar la siguiente formula: X1 = $\frac{-b+\sqrt{b^2-4ac}}{2a}$ (es la solución positiva de una ecuación de segundo grado). Los datos deben ser proporcionados mediante cajas de texto.

4) Se requiere calcular el área y el volumen de un cilindro: **area = (2 * (PI * radio²)) + ((2 * PI * radio) * h)** y **volumen = (PI * radio²) * h**.

5) Se requiere calcular las raíces de una ecuación de segundo grado: **x1= (-b + raiz(b² - (4 * a * c)) / 2 * a** y **x2= (-b - raiz(b² - (4 * a * c)) / 2 * a**. Los datos deben ser proporcionados mediante cajas de texto.

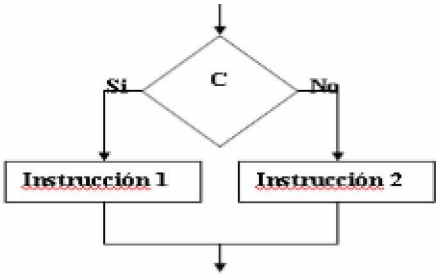
4.1 Sentencias de control

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 un conjunto de estructuras muy importantes ya que se encargan 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**

4.9.1 Sentencia IF … THEN … ELSE …

Dado que una condición produce un valor verdadero o falso, se necesita una sentencia de control que ejecute determinada sentencia si la condición es verdadera, y otra si es falsa. En Pascal esta alternativa se realiza con la sentencia **IF-THEN-ELSE**. A continuación se describe el diagrama de flujo y el formato de la sentencia.



Formatos de la sentencia IF:

```
If <condición> Then
<Sentencias1>
Else
<Sentencias2>
End If
```

```
If <condición> Entonces
<Sentencias1>
Sino
<Sentencias2>
End If
```

Si **condición** es **True (verdadera)**, se ejecutan las sentencias que están a continuación de **Then**, y si **condición** es **False (falsa)**, se ejecutan las sentencias que están a continuación de **Else**, si esta cláusula ha sido especificada.

Ejemplo:

```
numero = 10
If numero < 20 Then
        MsgBox ("El número es menor de 10.")
Else
        MsgBox ("El número es mayor de 10.")
End If
```

Es lógico que la sentencia especificada después de la cláusula **Else** nunca se ejecutará, esto es porque se ha especificado que la *variable* **numero** tiene por valor diez y por ende la condición seguida a **If** es verdadera. 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
        Sentencias-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** correspondiente a la cláusula **Else**, que es la opción por defecto.

- 4.9.2 Sentencia SELECT CASE

La sentencia **case** se utiliza para elegir entre diferentes alternativas. Una sentencia **case** se compone de varias sentencias simples. Cuando **case** se ejecuta, una de las sentencias simples se selecciona y ejecuta.

Su formato es:

Select Case Expresión **Case** vpe1 [sentencias1] **Case** vpe2 [sentencias2] **Case** vpe…N [Sentencias…N] **Case Else** [Sentencias-sino] **End Select**	**Seleccione según** expresión **Sea** vpe1 [sentecias1] **Sea** vpe2 [sentencias2] **Sea** vpe…N [Sentencias…N] **No sea** vpe1, vpe2, vpe…N [Sentencias-sino] **Fin selección**

Donde **expresión** es una expresión numérica o alfanumérica que puede proporcionar una serie de valores distintos y uno de ellos puede o no encontrarse en la lista. A estos valores en una *sentencia* **Case** se les podrían llamar "**valores proporcionados por la expresión (vpe)**". Las *etiquetas* **vp1**, **vpe2**, **vpe…N** representan valores que puede o no proporcionar la expresión, según sea el valor se ejecutan las *sentencias* seguidas a la *etiqueta* (**vpeN**) correspondiente. La clausula *opcional* **Case Else** indica que si los valores proporcionados por la **expresión** no se encuentran en la listas de las etiquetas (**vpeN**) entonces se ejecutarán las [**Sentencias-sino**].

Ejemplo:

```
Numero = X
Select Case Numero
Case 1
        MsgBox ("Este es el número uno.")
Case 2
```

MsgBox (“Este es el número dos.”)

Case 3

MsgBox (“Este es el número tres.”)

Case 4

MsgBox (“Este es el número cuatro.”)

Case Else

MsgBox (“No se a proporcionado los valores 1, 2, 3 o 4”)

End Select

La *variable* **X** en la variable **numero** representa un valor cualquiera que puede estar comprendido en el intervalo 1...4 o puede no ser uno de estos valores. Si **X** toma el valor de **1** entonces se ejecuta el caso uno (**vpe1**) que en este caso se ejecuta la sintaxis **MsgBox (“Este es el número uno.”)**, si **X** toma el valor de **2** entonces se ejecuta el caso dos (**vpe2**), y a si sucesivamente.

- 4.9.3 Sentencia FOR … NEXT

La sentencia **For … Next** repite una determinada serie de sentencias dado un valor inicial y un valor final. Este **bucle** se utiliza cuando se conoce por anticipado el número de repeticiones requerida por el programa. Si el número de repeticiones no se conoce por anticipado entonces debe utilizar las sentencia **While … Wend** o **Do … Loop** en lugar de **for**.

Su formato es:

a)

For *variable = valor inicial* **To** *valor final*

[Sentencias…]

Next *variable*

b)

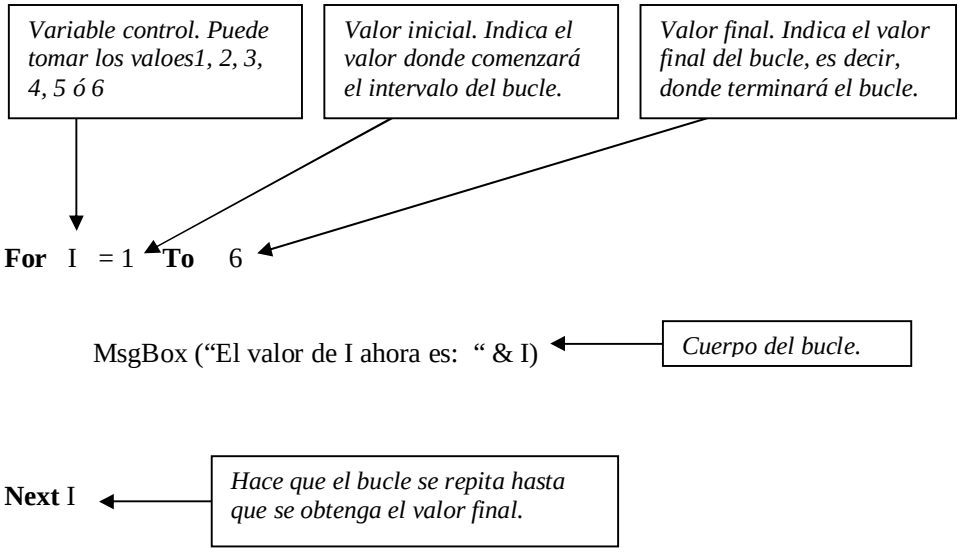
For *variable = valor inicial* **To** *valor final* **Step** –X

[Sentencias…]

Next *variable*

Al ejecutarse la sentencia **For** por primera vez, el *valor inicial* se asigna a *variable* que se denomina *variable de control*, y a continuación se ejecuta la sentencia del interior del bucle hasta que la *variable de control* toma el *valor final.*

Ejemplo:



En el segundo formato vemos la cláusula condicional **Step – X** que indica que el bucle se decrementa en X.

Ejemplo:

```
For I = 6 To 1 Step - 1
        MsgBox ("El valor de I ahora es:  " & I)
Next I
```

En este caso **I** tomará valores comenzando desde 6 hasta llegar a 1 (6, 5, 4, 3, 2, 1). Si en caso de haber especificado **-2** en vez de **-1**, entonces se decrementa de dos en dos (6, 4, 2, 0).

Al usar la cláusula **Step** debe tener en cuenta que si el *valor inicial* del bucle es menor que el *valor final* del bucle, el bucle nunca se ejecutará.

- 4.9.4 Sentencia DO … LOOP

Utilice el bucle **Do** para ejecutar un bloque de instrucciones un número indefinido de veces. Hay algunas variantes en la instrucción **Do...Loop**, pero cada una evalúa una condición numérica para determinar si continúa la ejecución. Como ocurre con **If...Then**, la *condición* debe ser un valor o una expresión que dé como resultado **False** (cero) o **True** (distinto de cero).

En el ejemplo de **Do...Loop** siguiente, las *instrucciones* se ejecutan siempre y cuando *condición* sea **True**:

Do While *condición*
instrucciones

Loop

Cuando Visual Basic ejecuta este bucle **Do**, primero evalúa *condición*. Si *condición* es **False** (cero), se salta todas las instrucciones. Si es **True** (distinto de cero), Visual Basic ejecuta las instrucciones, vuelve a la instrucción **Do While** y prueba la condición de nuevo.

Por tanto, el bucle se puede ejecutar cualquier número de veces, siempre y cuando *condición* sea distinta de cero o **True**. Nunca se ejecutan las instrucciones si *condición* es **False** inicialmente.

Ejemplo:

```
Dim contador As Integer
contador = 0

Do While contador < 10
   contador = contador + 1
   MsgBox (contador)
Loop
```

Mientras *contador* sea menor de diez el bucle se repetirá. Cuando el cuerpo del bucle se ejecuta la variable contador se incrementa en uno.

- 4.9.5 Sentencia WHILE … WEND

La estructura repetitiva **while** (mientras) es aquella en la que el número de iteraciones no se conoce por anticipado y el cuerpo del bucle se repite *mientras* se cumple una determinada condición. Por esta razón, a estos bucles se les denomina *bucles condicionales.*

Su formato es:

a)

While *condición*

[Sentencias…]

Wend

b)

Mientras *condición* sea verdadera

[Sentencia…]

Wend

Las reglas de funcionamientos de estos bucles son las siguientes:

1. La condición se evalúa antes y después de cada ejecución del bucle. Si la condición es verdadera, se ejecuta el bucle, y si es falsa, el bucle no se ejecuta.

2. Si la condición se evalúa a falso cuando se ejecuta el bucle por primera vez, el cuerpo del bucle no se ejecutará nunca. En este caso se dice que el bucle se ha ejecutado cero veces.

3. Mientras la condición sea verdadera el bucle se ejecutará. Esto significa que el bucle se ejecutará indefinidamente a menos que “algo” en el interior del bucle modifique la condición haciendo que su valor pase a falso.

Ejemplo:

```
Dim Contador
Contador = 0          ' Inicializa la variable.

While Contador < 20          ' Comprueba el valor del Contador.
   Contador = Contador + 1   ' Incrementa Contador.
Wend                         ' Finaliza el bucle End While cuando Contador > 19.
```

- 4.9.6 Sentencia FOR EACH … NEXT

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

Sintaxis

For Each *elemento* **In** *grupo*

[*instrucciones*]

[**Exit For**]

[*instrucciones*]

Next [*elemento*]

La sintaxis de la instrucción **For Each ... Next** consta de las siguientes partes:

Parte	**Descripción**
elemento	Requerido. **Variable** que se utiliza para iterar por los elementos del conjunto o matriz. Para conjuntos, *elemento* solamente puede ser una variable del tipo **Variant**, una variable de objeto genérica o cualquier variable de objeto específica. Para matrices, *elemento* solamente puede ser una variable tipo **Variant**.
grupo	Requerido. Nombre de un conjunto de objetos o de una matriz (excepto una matriz de *tipos definidos por el usuario*).
instrucciones	Opcional. Una o más instrucciones que se ejecutan para cada elemento de un *grupo*.

La entrada al bloque **For Each** se produce si hay al menos un elemento en *grupo*. Una vez que se ha entrado en el bucle, todas las instrucciones en el bucle se ejecutan para el primer elemento en *grupo*. Después, mientras haya más elementos en *grupo*, las instrucciones en el bucle continúan ejecutándose para cada elemento. Cuando no hay más elementos en el *grupo*, se sale del bucle y la ejecución continúa con la instrucción que sigue a la instrucción **Next**.

Se pueden colocar en el bucle cualquier número de instrucciones **Exit For**. La instrucción **Exit For** se utiliza a menudo en la evaluación de alguna condición (por ejemplo, **If...Then**) y transfiere el control a la instrucción que sigue inmediatamente a la instrucción **Next**.

Puede anidar bucles **For Each...Next**, colocando un bucle **For Each...Next** dentro de otro. Sin embargo, cada *elemento* del bucle debe ser único.

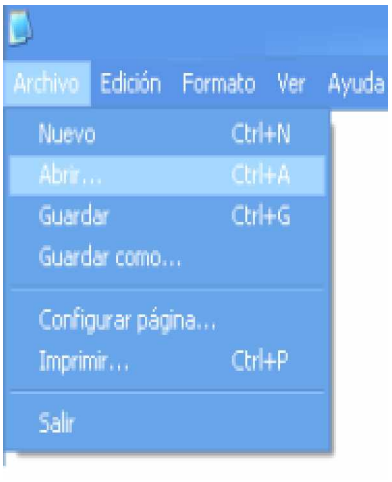
Nota Si omite *elemento* en una instrucción **Next**, la ejecución continúa como si se hubiera incluido. Si se encuentra una instrucción **Next** antes de su instrucción **For** correspondiente, se producirá un error.

No se puede utilizar la instrucción **For Each...Next** con una matriz de tipos definidos por el usuario porque un tipo **Variant** no puede contener un tipo definido por el usuario.

Capitulo V

CONTENIDO

5.7 ¿Qué son los menús?



Al conjunto de opciones presentadas al usuario para su selección en una zona determinada de la pantalla se les llaman **menús**. La gran mayoría de las aplicaciones de Windows poseen menús que contienen todas las herramientas necesarias que hacen posible la utilidad de la aplicación.

Los programas tales como Word, Excel, WordPad, incluyen **Barras de Menús** y dentro de estas *barras* se encuentran los menús y dentro de los menús las opciones para cada menú.

Por ejemplo, el menú **Archivo (File)** de Microsoft Word incluye opciones o comandos tales como: Nuevo, Abrir, Cerrar, Guardar, Guardar como, Imprimir, etc.

Los menús presentan sobre los demás *controles* la ventaja de que ocupan menos espacio en pantalla, pero tienen la limitante de que las opciones o comandos de los menús no están visibles hasta que se despliega totalmente el menú.

5.1 Elementos de los menús

Entre los elementos principales de los menús tenemos los *accesos directos*, *los indicadores de cuadro de dialogo* (**…**), el *indicador de submenú* (►), las *líneas divisoras* y *las imágenes*. Todos estos elementos permiten una mejor legibilidad al trabajar con los menús.

● *Los accesos directos:* son aquellos que mediante combinaciones de teclas nos permiten acceder a un menú o a una opción de un menú. Por ejemplo, para desplegar el menú **Archivo (File)** de Microsoft Word basta con pulsar las combinaciones de teclas **Alt + A** (en español) ó **Alt + F** (en inglés), o para activar la *ventana de dialogo* **Abrir** se pulsan las teclas **Ctrl + A** (en español) ó **Ctrl + O** (en inglés).

● *Los indicadores de cuadro de dialogo (…)*: Estos están representados por *tres puntos suspensivos* (**…**) que indican que al seleccionar esta opción se mostrará una ventana de dialogo dónde se requerirá de algún *evento* proporcionado por el usuario.

● *El indicador de submenú (►):* El indicador de submenú esta representado por una *flecha a la derecha*, que indica que el elemento de ese menú posee otras opciones de menú llamado submenú.

● *Las líneas divisoras:* Así como su nombre lo indica, son *líneas divisoras* que dividen entre un conjunto de opciones y otro conjunto de opciones dentro de un mismo menú. Las líneas divisoras no tienen ninguna función especial dentro de un menú, solo indican la división entre un conjunto de opciones dentro de un mismo menú.

● Las imágenes: Las *imágenes* en los menús juegan un papel muy importante, ya que ilustran gráficamente la función de un elemento dentro de ese menú. Por ejemplo, la opción **Imprimir** del menú **Archivo** de Microsoft Word posee la imagen de una impresora  , que indica que es la opción de imprimir.

5.2 El Editor de Menú (Menu Editor)

Visual Basic 6.0 posee una potente herramienta para la creación de menús y todos los elementos que estos los componen. El **Editor de Menú** (**Menu Editor**) es la herramienta que permite la creación de estos. Para activar o mostrar la ventan del **Menu Editor**, haga clic en la opcion **Menu Editor** del menú **Tools** (*herramientas*) o bien, haga clic en el botón  correspondiente al **Menu Editor** de la *barra de herramientas estándar.* Al seleccionar esta opción se mostrará la siguiente ventana:

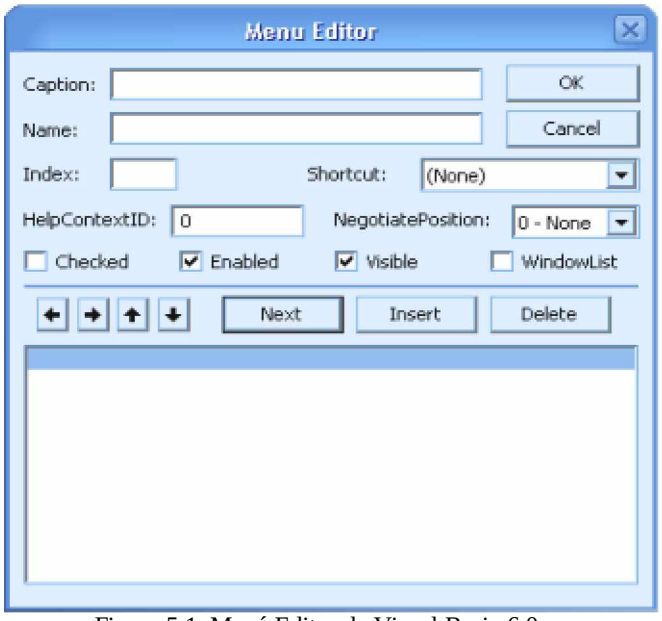


Figura 5.1. Menú Editor de Visual Basic 6.0

- 5.3.1 Descripción de los elementos del Editor de Menús

En la **figura 5.1** se muestra la ventana del *Editor de Menús* que posee todas las herramientas necesarias para la creación de estos. A continuación se describen cada uno de los elementos del *Editor de Menús.*

● *Caption (Título):* En esta *caja de texto* se escribe el nombre del menú o elemento de un menú o submenú. En esta *caja de texto*, el carácter (**&**) **ampersand** tiene un significado especial, ya que indica que el carácter o la letra a la derecha de él será el acceso directo a ese menú o elemento de un menú o submenú. Por ejemplo, si un menú denominado **Archivo** tiene el carácter (**&**) **ampersand** posicionado a la izquierda, es decir, delante de la letra **A** significa que sólo basta con pulsar las combinaciones de las teclas **Alt + A** para tener acceso a ese menú, elemento de un menú o submenú.

● *Name (Nombre):* En esta *caja de texto* se especifica el nombre del menú, elemento de un menú o submenú, que se utiliza para referenciar a ese menú en el *editor de códigos*.

● *Index (Índice):* La *caja de texto* **Índice** hace referencia a la posibilidad de crear *arrays* de menús.

● *ShortCut (Acceso directo):* Permite asignar acceso directo a los elementos de cada menú. En esta lista se muestran una serie de combinaciones, el cual el usuario puede seleccionar la más adecuada para ese elemento de menú.

● *Checked (Verificación):* Permite agregar un cotejo de verificación (√) a un elemento de un menú.

● *Enabled (Habilitado):* Indica si el menú o elemento del menú responderá a los eventos del usuario.

● *Visible (Visible):* Indica si el menú o elemento del menú estará visible o no.

Las demás opciones *HelpContextID*, *NegotiatePosition* y *WindowList* son pocas usadas, por tal razón sean limitado su descripción. Para más información acerca de estás opciones, consulte la ayuda (help) de **Visual Basic 6.0**.

- Descripción de los botones del Editor de Menús

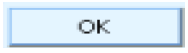
 El botón *flecha izquierda* se utiliza para eliminar cuatro puntos suspensivos a la izquierda, que indican si ese elemento es un comando de un menú o submenú.

 El botón *flecha derecha* agrega cuatro puntos suspensivos (**….**) a la izquierda de un elemento de un menú o submenú. Si este botón se pulsa dos veces se agregan ocho puntos suspensivos (**……..**) a la izquierda de un elemento, indicando que es un elemento de un submenú. Puede pulsar este botón las veces que se necesario siempre y cuando sepa establecer el orden jerárquico entre los elementos del menú.

 Los botones *flecha arriba y flecha abajo* se utilizan para desplazarse entre los menús, elementos de menú o submenú.

El botón *siguiente* se utiliza para agregar un menú, un elemento de menú o submenú. Al pulsar este botón sobre un elemento ya agregado se inserta otra nueva línea en el editor de menú con el mismo formato de la línea interior, es decir, si el elemento sobre el cual se pulsa este botón es un elemento de un menú, entonces la línea que se agrega también será un elemento de menú para ese mismo menú.

El botón *insertar* se utiliza para insertar un elemento o ítem en la posición de un elemento seleccionado quedando este debajo del nuevo elemento.



El botón *guardar* guarda todas las modificaciones echas en el *Editor de Menús*.

El botón *cancelar* omite cualquier modificación echa en el *Editor de Menús* y al mismo tiempo cierra la ventana del editor.

- 5.3.2 Creación de menús en Visual Basic 6.0

Como su nombre lo indica, la creación de menús es un proceso práctico y por tal razón vamos a crear el siguiente menú:

Creación del menú Archivo

Pasos a seguir:

1.- Abra un nuevo proyecto desde el menú **File**.

2.- Haga clic en el icono del Editor de Menús .

3.- En la *caja de texto* **Caption (título)** escriba **&Archivo** y en la *caja de texto* **Name (nombre)** escriba **menuArchivo**.

4.- Haga clic en el botón **Next (siguiente).**

5.- Haga clic en el botón *flecha derecha* **(→)** para agregar cuatro puntos suspensivos (**….**).

6.- Haga clic en la caja de texto **Caption (título)** y escriba **&Nuevo** y en la caja de texto **Name (nombre)** escriba **elementoNuevoMenuArchivo**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + N**, y a continuación, haga clic en el botón **Next (Siguiente)**.

7.- En la *caja de texto* **Caption (título)** escriba **&Abrir…** y en la *caja de texto* **Name (nombre)** escriba **elementoAbrirMenuArchivo**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + A**, y a continuación, haga clic en el botón **Next (Siguiente)**.

8.- En la *caja de texto* **Caption (título)** escriba **&Guardar** y en la *caja de texto* **Name (nombre)** escriba **elementoGuardarMenuArchivo**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + G**, y a continuación, haga clic en el botón **Next (Siguiente)**.

9.- En la *caja de texto* **Caption (título)** escriba **&Guardar como…** y en la *caja de texto* **Name (nombre)** escriba **elementoGuardarComoMenuArchivo**, y a continuación, haga clic en el botón **Next (Siguiente)**.

10.- En la *caja de texto* **Caption (título)** escriba un guión o el símbolo de resta "**-**" y en la *caja de texto* **Name (nombre)** escriba **Linea1MenuArchivo**, y a continuación, haga clic en el botón **Next (Siguiente)**.

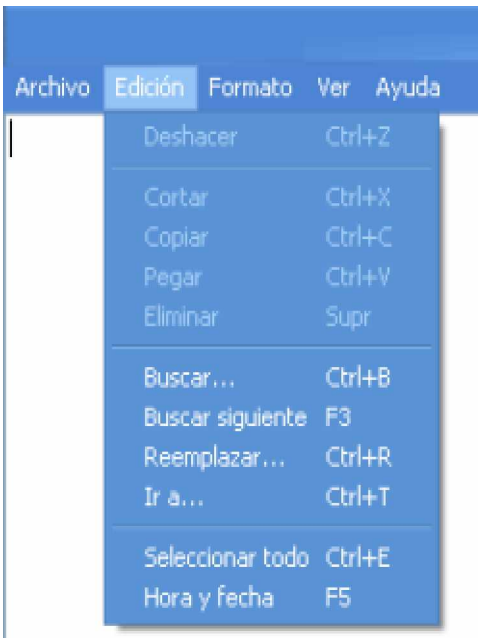
11.- En la *caja de texto* **Caption (título)** escriba **&Configurar página…** y en la *caja de texto* **Name (nombre)** escriba **elementoConfigurarPaginaMenuArchivo**, y a continuación, haga clic en el botón **Next (Siguiente)**.

12.- En la *caja de texto* **Caption (título)** escriba **&Imprimir…** y en la *caja de texto* **Name (nombre)** escriba **elementoImprimirMenuArchivo**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + P**, y a continuación, haga clic en el botón **Next (Siguiente)**.

13.- En la *caja de texto* **Caption (título)** escriba un guión o el símbolo de resta "**-**" y en la *caja de texto* **Name (nombre)** escriba **Linea2MenuArchivo**, y a continuación, haga clic en el botón **Next (Siguiente)**.

14.- En la *caja de texto* **Caption (título)** escriba **Sa&lir** y en la *caja de texto* **Name (nombre)** escriba **elementoSalirMenuArchivo**, y a continuación, haga clic en el botón **Next (Siguiente)**.

Creación del menú Edición



Pasos a seguir:

1.-Haga clic en el botón *flecha izquierda* (←) para eliminar los cuatro puntos suspensivos a la izquierda.

2.- En la *caja de texto* **Caption (título)** escriba **&Edición** y en la *caja de texto* **Name (nombre)** escriba **MenuEdicion**, y a continuación, haga clic en el botón **Next (Siguiente)**.

3.-Haga clic en el botón *flecha derecha* (→) para agregar los cuatro puntos suspensivos a la izquierda.

4.- En la *caja de texto* **Caption (título)** escriba un guión o el símbolo de resta "**-**" y en la *caja de texto* **Name (nombre)** escriba **Linea1MenuEdicion**, y a continuación, haga clic en el botón **Next (Siguiente)**.

5.- En la *caja de texto* **Caption (título)** escriba **&Deshacer** y en la *caja de texto* **Name (nombre)** escriba **elementoDeshacerMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + Z**, y a continuación, haga clic en el botón **Next (Siguiente)**.

6.- En la *caja de texto* **Caption (título)** escriba un guión o el símbolo de resta "**-**" y en la *caja de texto* **Name (nombre)** escriba **Linea2MenuEdicion**, y a continuación, haga clic en el botón **Next (Siguiente)**.

7.- En la *caja de texto* **Caption (título)** escriba **&Cortar** y en la *caja de texto* **Name (nombre)** escriba **elementoCortarMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + X**, y a continuación, haga clic en el botón **Next (Siguiente)**.

8.- En la *caja de texto* **Caption (título)** escriba **C&opiar** y en la *caja de texto* **Name (nombre)** escriba **elementoCopiarMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + C**, y a continuación, haga clic en el botón **Next (Siguiente)**.

9.- En la *caja de texto* **Caption (título)** escriba **&Pegar** y en la *caja de texto* **Name (nombre)** escriba **elementoPegarMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + V**, y a continuación, haga clic en el botón **Next (Siguiente)**.

10.- En la *caja de texto* **Caption (título)** escriba **&Eliminar** y en la *caja de texto* **Name (nombre)** escriba **elementoEliminarMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Del** ó **Supr**, y a continuación, haga clic en el botón **Next (Siguiente)**.

11.- En la *caja de texto* **Caption (título)** escriba un guión o el símbolo de resta "**-**" y en la *caja de texto* **Name (nombre)** escriba **Linea3MenuEdicion**, y a continuación, haga clic en el botón **Next (Siguiente)**.

12.- En la *caja de texto* **Caption (título)** escriba **&Buscar…** y en la *caja de texto* **Name (nombre)** escriba **elementoBuscarMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + B**, y a continuación, haga clic en el botón **Next (Siguiente)**.

13.- En la *caja de texto* **Caption (título)** escriba **B&uscar siguiente** y en la *caja de texto* **Name (nombre)** escriba **elementoBuscarSiguienteMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **F3**, y a continuación, haga clic en el botón **Next (Siguiente)**.

14.- En la *caja de texto* **Caption (título)** escriba **Re&mplazar** y en la *caja de texto* **Name (nombre)** escriba **elementoRemplazarMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + R**, y a continuación, haga clic en el botón **Next (Siguiente)**.

15.- En la *caja de texto* **Caption (título)** escriba **&Ir a…** y en la *caja de texto* **Name (nombre)** escriba **elementoIrASiguienteMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + T**, y a continuación, haga clic en el botón **Next (Siguiente)**.

16.- En la *caja de texto* **Caption (título)** escriba un guión o el símbolo de resta "**-**" y en la *caja de texto* **Name (nombre)** escriba **Linea4MenuEdicion**, y a continuación, haga clic en el botón **Next (Siguiente)**.

17.- En la *caja de texto* **Caption (título)** escriba **Seleccionar &todo** y en la *caja de texto* **Name (nombre)** escriba **elementoSeleccionarTodoMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **Ctrl + E**, y a continuación, haga clic en el botón **Next (Siguiente)**.

18.- En la *caja de texto* **Caption (título)** escriba **&Hora y fecha** y en la *caja de texto* **Name (nombre)** escriba **elementoHoraFechaMenuEdición**. En la *caja combinada* de **Shortcut** seleccione **F5**, y a continuación, haga clic en el botón **Next (Siguiente)**.

Creación del menú Formato

Pasos a seguir:

1.-Haga clic en el botón *flecha izquierda* (←) para eliminar los cuatro puntos suspensivos a la izquierda.

2.- En la *caja de texto* **Caption (título)** escriba **&Formato** y en la *caja de texto* **Name (nombre)** escriba **MenuFormato**, y a continuación, haga clic en el botón **Next (Siguiente)**.

3.-Haga clic en el botón *flecha derecha* (→) para agregar los cuatro puntos suspensivos a la izquierda.

4.- En la *caja de texto* **Caption (título)** escriba **Ajus&te de línea** y en la *caja de texto* **Name (nombre)** escriba **ElementoAjusteDeLineaMenuFormato**, y a continuación, haga clic en el botón **Next (Siguiente)**.

5.- En la *caja de texto* **Caption (título)** escriba **&Fuente** y en la *caja de texto* **Name (nombre)** escriba **ElementoFuenteMenuFormato**, y a continuación, haga clic en el botón **Next (Siguiente)**.

Creación del menú Ver

Pasos a seguir:

1.-Haga clic en el botón *flecha izquierda* (←) para eliminar los cuatro puntos suspensivos a la izquierda.

2.- En la *caja de texto* **Caption (título)** escriba **&Ver** y en la *caja de texto* **Name (nombre)** escriba **MenuVer**, y a continuación, haga clic en el botón **Next (Siguiente)**.

3.-Haga clic en el botón *flecha derecha* (→) para agregar los cuatro puntos suspensivos a la izquierda.

4.- En la *caja de texto* **Caption (título)** escriba **Barra &de estado** y en la *caja de texto* **Name (nombre)** escriba **ElementoBarraEstadoMenuVer**, y a continuación, haga clic en el botón **Next (Siguiente)**.

Creación del menú Ayuda

Pasos a seguir:

1.-Haga clic en el botón *flecha izquierda* (←) para eliminar los cuatro puntos suspensivos a la izquierda.

2.- En la *caja de texto* **Caption (título)** escriba **Ay&uda** y en la *caja de texto* **Name (nombre)** escriba **MenuAyuda**, y a continuación, haga clic en el botón **Next (Siguiente)**.

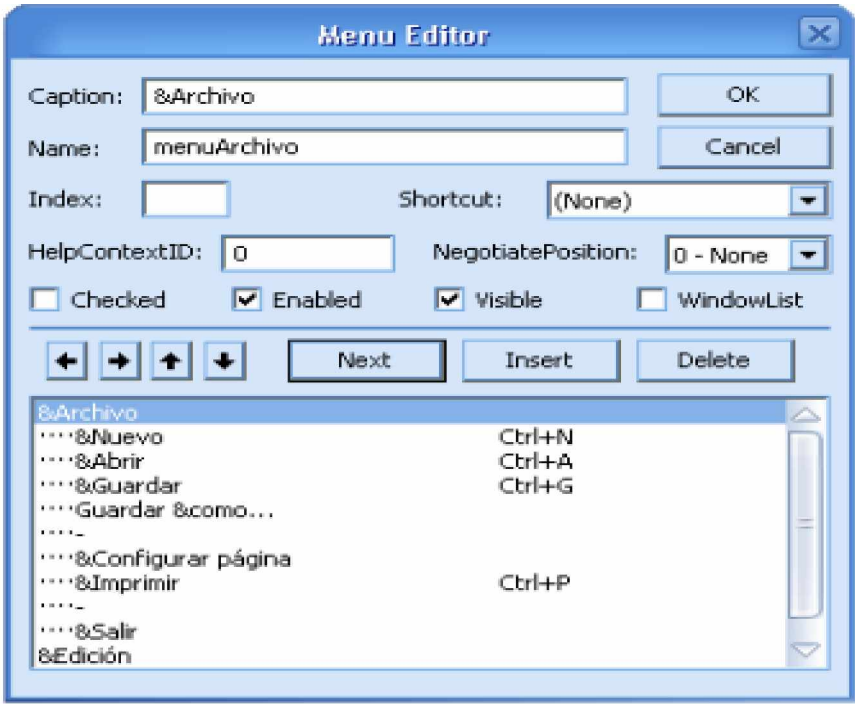
3.-Haga clic en el botón *flecha derecha* (→) para agregar los cuatro puntos suspensivos a la izquierda.

4.- En la *caja de texto* **Caption (título)** escriba **&Temas de Ayuda** y en la *caja de texto* **Name (nombre)** escriba **ElementoTemasDeAyudaMenuAyuda**, y a continuación, haga clic en el botón **Next (Siguiente)**.

5.- En la *caja de texto* **Caption (título)** escriba un guión o el símbolo de resta "**-**" y en la *caja de texto* **Name (nombre)** escriba **Linea1MenuAyuda**, y a continuación, haga clic en el botón **Next (Siguiente)**.

6.- En la *caja de texto* **Caption (título)** escriba **&Acerca del Bloc de notas** y en la *caja de texto* **Name (nombre)** escriba **ElementoAcercaMenuAyuda**, y a continuación, haga clic en el botón **Next (Siguiente)**.

7.- Haga clic en el botón **Ok** para finalizar.



NOTA: En caso de algún error, verifique si ha escrito correctamente los nombres **(name)** de los menús, elementos de los menús y submenús propuestos. También verifique si ha establecido el orden jerárquico adecuado entre los ítems o elementos de menús y submenús. Si el problema persiste consulte con su maestro, o bien, envié un E-Mail a twinsmaster02@hotmail.com explicando el problema.

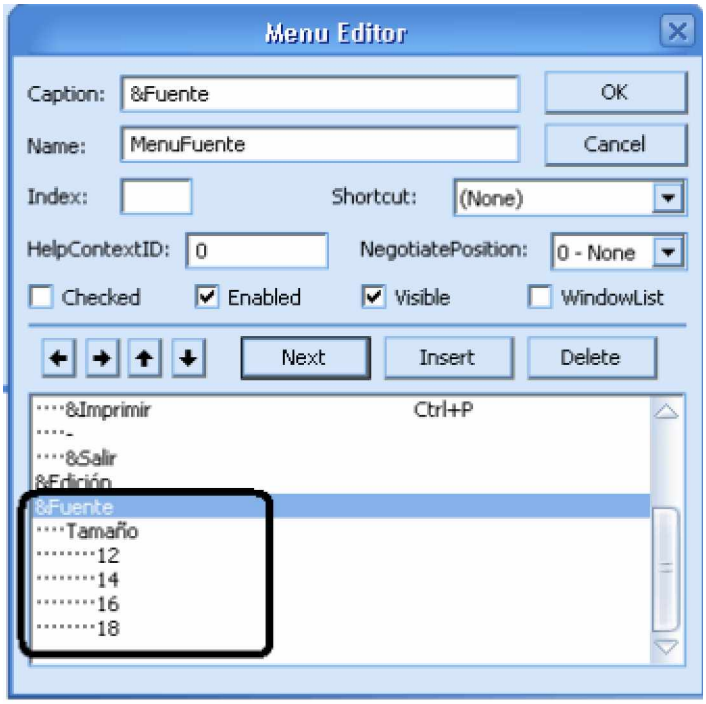
Después de haber realizados todos estos pasos, corra la aplicación pulsando la tecla **F5** y desplácese por todos los menús ya creado.

- 5.3.3 Creación de submenús

Los submenús representan a los menús que se encuentran dentro de un menú, es decir, dentro de un elemento de un menú o un elemento de un submenú. Un submenú esta representado por medio de una flecha a la derecha (►). A continuación se muestra un submenú típico en un menú:

Se puede observar en la que el menú **Fuente** contiene un *elemento* o *ítem* con el título **Tamaño** y este elemento esta precedido por una *pequeña flecha* (►), que indica que existe un submenú dentro de ese elemento.

La creación de submenús consiste en establecer un orden jerárquico de menor a mayor, dónde el elemento que contendrá el submenú debe tener cuatro puntos suspensivos menos a la izquierda, como se muestra a continuación:



El recuadro resaltado es la zona donde se especifica el menú **Fuente** con sus elementos. Se puede observar que el elemento **Tamaño** posee cuatro *puntos suspensivos (....)* menos que los *elementos* 12, 14, 16, 18. Esto indica que estos elementos pertenecen al elemento **Tamaño** que esta dentro del menú **Fuente**.

- 5.3.4 Evento principal de los elementos de los menús

Es lógico, que **Visual Basic 6.0** permita agregar líneas de código a cada uno de los elementos de los menús y submenús que se agreguen a una aplicación. Esto lo hace mediante el *evento general* de los menús, el *evento* **Click**.

Para agregar código a algún elemento de un menú o submenú, sólo tiene que hacer clic sobre el menú en modo de diseño y luego hacer otro clic, sobre el elemento al cual quiere agregar el código. El *evento* **Click** debe parecerse a esto:

```
Private Sub ElementoX_Click ( )
        [Sentencias…]
End Sub
```

Capitulo VI

CONTENIDO

Capitulo VI

6.1 Cajas de diálogo MsgBox e InputBox

Estas cajas de diálogo son aquellas cajas típicas de Windows, que en ocasiones proporcionan o requieren información. La caja de mensajes **MsgBox** muestra un mensaje en un cuadro de diálogo, espera a que el usuario haga clic en un botón y devuelve un valor tipo **Integer** correspondiente al botón elegido por el usuario.

Sintaxis

MsgBox (*prompt*[, *buttons*][, *title*][, *helpfile*, *context*])

Donde:

Parte	Descripción
prompt	Requerido. *Expresión de cadena* que representa el ***prompt*** en el cuadro de diálogo. La longitud máxima de ***prompt*** es de aproximadamente 1024 caracteres, según el ancho de los caracteres utilizados. Si ***prompt*** consta de más de una línea, puede separarlos utilizando un carácter de retorno de carro (**Chr(**13**)**) o un carácter de avance de línea (**Chr(**10**)**), o una combinación de caracteres de retorno de carro – avance de línea (**Chr(**13**)** y **Chr(**10**)**) entre cada línea y la siguiente.
buttons	Opcional. *Expresión numérica* que corresponde a la suma de los valores que especifican el número y el tipo de los botones que se pretenden mostrar, el estilo de icono que se va a utilizar, la identidad del botón predeterminado y la modalidad del cuadro de mensajes. Si se omite este argumento, el valor predeterminado para ***buttons*** es 0.
title	Opcional. Expresión de cadena que se muestra en la barra de título del cuadro de diálogo. Si se omite ***title***, en la barra de título se coloca el nombre de la aplicación.
helpfile	Opcional. Expresión de cadena que identifica el archivo de Ayuda que se utiliza para proporcionar ayuda interactiva en el cuadro de diálogo. Si se especifica ***helpfile***, también se debe especificar ***context***.
context	Opcional. Expresión numérica que es igual al número de contexto de Ayuda asignado por el autor al tema de Ayuda correspondiente. Si se especifica ***context***, también se debe especificar ***helpfile***.

El *argumento* ***buttons*** tiene los siguientes valores:

Constante	Valor	Descripción
VbOKOnly	0	Muestra solamente el botón **Aceptar**.
VbOKCancel	1	Muestra los botones **Aceptar** y **Cancelar**.
VbAbortRetryIgnore	2	Muestra los botones **Anular**, **Reintentar** e **Ignorar**.
VbYesNoCancel	3	Muestra los botones **Sí**, **No** y **Cancelar**.
VbYesNo	4	Muestra los botones **Sí** y **No**.
VbRetryCancel	5	Muestra los botones **Reintentar** y **Cancelar**.
VbCritical	16	Muestra el icono de **mensaje crítico**.
VbQuestion	32	Muestra el icono de **pregunta de advertencia**.
VbExclamation	48	Muestra el icono de **mensaje de advertencia**.
VbInformation	64	Muestra el icono de **mensaje de información**.
VbDefaultButton1	0	El primer botón es el predeterminado.
VbDefaultButton2	256	El segundo botón es el predeterminado.
VbDefaultButton3	512	El tercer botón es el predeterminado.
VbDefaultButton4	768	El cuarto botón es el predeterminado.
VbApplicationModal	0	Aplicación modal; el usuario debe responder al cuadro de mensajes antes de poder seguir trabajando en la aplicación actual.
VbSystemModal	4096	Sistema modal; se suspenden todas las aplicaciones hasta que el usuario responda al cuadro de mensajes.
VbMsgBoxHelpButton	16384	Agrega el botón Ayuda al cuadro de mensaje.
VbMsgBoxSetForeground	65536	Especifica la ventana del cuadro de mensaje como la ventana de primer plano.
VbMsgBoxRight	524288	El texto se alínea a la derecha.
VbMsgBoxRtlReading	1048576	Especifica que el texto debe aparecer para ser leído de derecha a izquierda en sistemas hebreo y árabe.

El primer grupo de valores (0 a 5) describe el número y el tipo de los botones mostrados en el cuadro de diálogo; el segundo grupo (16, 32, 48, 64) describe el estilo del icono, el tercer grupo (0, 256, 512) determina el botón predeterminado y el cuarto grupo (0, 4096) determina la modalidad del cuadro de mensajes. Cuando se suman números para obtener el valor final del argumento ***buttons***, se utiliza solamente un número de cada grupo.

Valores devueltos por los botones

Constante	Valor	Descripción
vbOK	1	**Aceptar**
vbCancel	2	**Cancelar**
vbAbort	3	**Anular**
vbRetry	4	**Reintentar**
vbIgnore	5	**Ignorar**
vbYes	6	**Sí**
vbNo	7	**No**

Cuando se proporcionan tanto ***helpfile*** como ***context***, el usuario puede presionar F1 para ver el tema de Ayuda correspondiente al ***context***. Algunas *aplicaciones host*, por ejemplo Microsoft Excel, también agregan automáticamente un botón **Ayuda** al cuadro de diálogo.

Si el cuadro de diálogo cuenta con un botón **Cancelar**, presionar la tecla ESC tendrá el mismo efecto que hacer clic en este botón. Si el cuadro de diálogo contiene un botón **Ayuda**, se suministra ayuda interactiva para ese cuadro de diálogo. Sin embargo, no se devuelve valor alguno hasta que se hace clic en uno de estos botones.

Nota Si desea especificar más que el primer argumento con nombre, debe utilizar **MsgBox** en una *expresión*. Si desea omitir algún *argumento* de posición, debe incluir el delimitador de coma correspondiente.

Ejemplo:

En este ejemplo se utiliza la función **MsgBox** para mostrar un mensaje de error crítico en un cuadro de diálogo con botones Sí y No. El mensaje que aparecerá es **¿Desea continuar?**. El valor devuelto por la función **MsgBox** cuando se pulsa un botón, se almacena en una variable llamada *respuesta*. Este ejemplo lo puede probar dentro de un botón de comando.

```
Private Sub Command1_Click ( )

Dim Mensaje, Estilo, Título, Respuesta          'Se declaran las variables
Mensaje = "¿Desea continuar?"                   'Define el mensaje.
Estilo = vbYesNo + vbCritical + vbDefaultButton2    'Define los botones.

 'Muestra el mensaje.
Respuesta = MsgBox (Mensaje, Estilo, Título)

'Verifica cual fue el botón pulsado por el usuario.

If Respuesta = vbYes Then                       ' El usuario eligió el botón Sí.
    MsgBox ("Se hizo clic en el botón Sí.")
Else                                            ' El usuario eligió el botón No.
  MsgBox ("Se hizo clic en el botón No.")
End If

End Sub
```

6.2 Método Print

Mediante este método es posible mostrar texto en **formularios**, cajas **PictureBox** y otros *controles* que poseen dicho *método*.

Sintaxis

Objeto**.Print** [*Salida*]

Donde:

Objeto: Representa un *expresión objeto*. Este puede ser un **formulario**, un control **PictureBox**, u otros elementos que poseen este método.

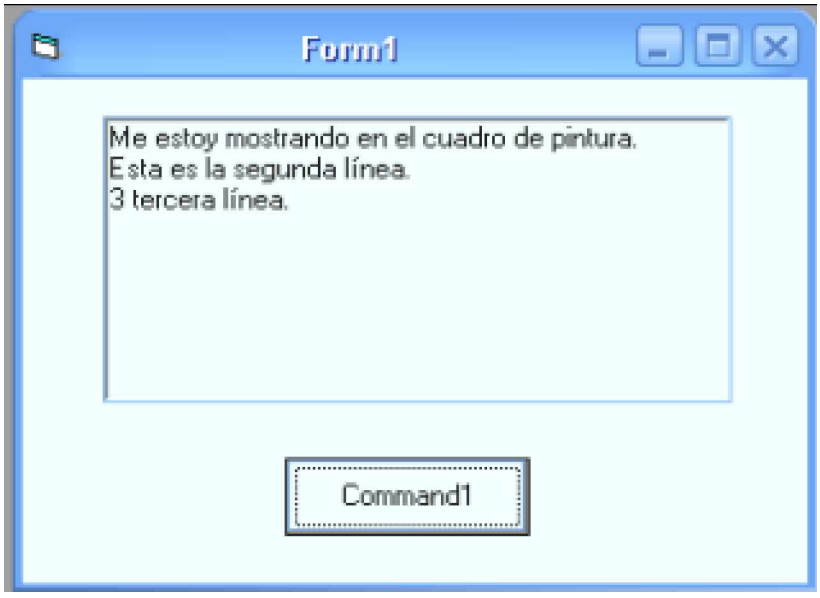
Salida: Se refiere a la lista de expresiones que se van a imprimir. Esta puede ser el contenido de una *variable*, una misma cadena de caracteres entre comillas, un valor numérico, el valor de una propiedad de un *control*, etc.

Ejemplo:

'Agregue un *control PictureBox* y un *botón de comando* a un nuevo formulario, y escriba:

```
Private Sub Command1_Click ( )
        Picture1.Print "Me estoy mostrando en el cuadro de pintura."
        Picture1.Print "Esta es la segunda línea."
        Picture1.Print 2 + 1 & "  tercera línea."
End Sub
```

Cuando se corra la aplicación y se haga clic en el *botón de comando* **(Command1),** los resultados serán los siguientes:



- 6.2.1 Características generales

El *argumento* **Salida** tiene las sintaxis y los componentes siguientes:

{**Spc(***n***)** | **Tab(***n***)**} *expresión, posicióncarácter*

Parte	**Descripción**
Spc(*n***)**	Opcional. Se utiliza para insertar caracteres de espacio en la salida, donde *n* es el número de espacios que se vana insertar.
Tab(*n***)**	Opcional. Se utiliza para situar el punto de inserción en un número de columna absoluto, donde *n* es el número de columna. Utilice **Tab** sin argumentos para situar el punto de inserción al principio de la siguiente *zona de impresión*. Las *zonas de impresión* comienzan cada 14 columnas. El ancho de cada columna es un promedio del ancho de todos los caracteres, medido con el tamaño de punto de la fuente elegida.
expresión	Opcional. *Expresión numérica* o *expresión de cadena* que se va a imprimir.
posicióncarácter	Opcional. Especifica el punto de inserción del carácter siguiente. Utilice un punto y coma (;) para situar el punto de inserción inmediatamente a continuación del último carácter mostrado. Utilice **Tab(***n***)** para situar el punto de inserción en un número de columna absoluto, o **Tab** sin argumentos para situarlo al principio de la siguiente zona de impresión. Si se omite *posicióncarácter*, el carácter siguiente se imprimirá en la línea siguiente.

- 6.2.2 Función Format

Devuelve un tipo **Variant** (**String**) que contiene una *expresión* formateada de acuerdo a las instrucciones contenidas en una expresión de formato.

Sintaxis

Format (*expresión*[, *formato*[, *primerdíadesemana*[, *primerdíadeaño*]]])

La sintaxis de la función **Format** consta de las siguientes partes:

Parte	Descripción
expresión	Requerido. Cualquier expresión válida.
formato	Opcional. Una expresión de formato definida por el usuario o con nombre válida.
primerdíadesemana	Opcional. Una *constante* que especifica el primer día de la semana.
primerdíadeaño	Opcional. Una *constante* que especifica la primera semana del año.

Valores

El argumento *primerdíadesemana* tiene estos valores:

Constante	Valor	Descripción
vbUseSystem	0	Utiliza el valor de API NLS.
vbSunday	1	Domingo (predeterminado)
vbMonday	2	Lunes
vbTuesday	3	Martes
vbWednesday	4	Miércoles
vbThursday	5	Jueves
vbFriday	6	Viernes
vbSaturday	7	Sábado

El argumento *primerdíadeaño* tiene estos valores:

Constante	Valor	Descripción
vbUseSystem	0	Utiliza el valor de API NLS.
vbFirstJan1	1	Comienza con la semana donde está el 1 de enero (predeterminado).
vbFirstFourDays	2	Comienza con la primera semana del año que tenga cuatro días como mínimo.
vbFirstFullWeek	3	Comienza con la primera semana completa del año.

Para dar formato haga lo siguiente:

- Utilice formatos numéricos con nombre predefinidos o cree formatos numéricos definidos por el usuario.
- Utilice formatos de fecha/hora con nombre predefinidos o cree formatos de fecha/hora definidos por el usuario.
- Utilice formatos de fecha y hora o formatos numéricos.
- Cree sus propios formatos de cadena definidos por el usuario.

Ejemplos:

```
Dim MiHora, MiFecha, MiCadena
MiHora = #17:04:23#
MiFecha = #27 enero 1993#

' Devuelve la hora actual del sistema en el formato largo de hora definido por el sistema.

MiCadena = Format(Time, "Long Time")

' Devuelve la fecha actual del sistema en el formato largo de fecha definido por el sistema.

MiCadena = Format(Date, "Long Date")

MiCadena = Format(MiHora, "h:m:s")                    ' Devuelve "17:4:23".

MiCadena = Format(MiHora, "hh:mm:ss AMPM")     ' Devuelve "05:04:23 PM".

MiCadena = Format(MiFecha, "dddd, d mmm aaaa")  ' Devuelve "Miércoles,
' 27 de Ene de 1993".

' Si no se suministra el formato, devuelve una cadena.

MiCadena = Format(23)          ' Devuelve "23".

' Formatos definidos por el usuario.

MiCadena = Format(5459.4, "##,##0.00")   ' Devuelve "5.459,40".

MiCadena = Format(334.9, "###0.00")   ' Devuelve "334,90".

MiCadena = Format(5, "0.00%")   ' Devuelve "500,00%".

MiCadena = Format("HOLA", "<")   ' Devuelve "hola".

MiCadena = Format("Esto es", ">")   ' Devuelve "ESTO ES".
```

6.3 Utilización de impresoras

Visual Basic 6.0 permite obtener por la impresora gráficos y texto similares a los que se pueden obtener por la pantalla, aunque con algunas diferencias de cierta importancia. Existen tres formas de imprimir aunque solo mencionaremos las dos más usada: La primera mediante el método **PrintForm** de los formularios, y la segunda utilizando el objeto **Printer**, que es un objeto similar al objeto **PictureBox**. Ambos métodos tienen puntos fuertes y débiles que se comentarán a continuación.

- 6.3.1 Método PrintForm

El método **PrintForm** envía una imagen del formulario especificado a la impresora. Para imprimir información desde la aplicación con **PrintForm**, primero debe presentar dicha información en un formulario y después imprimir ese formulario con el método **PrintForm**. La sintaxis es la siguiente:

[*formulario.*]**PrintForm**

Si omite el nombre del formulario, Visual Basic imprime el formulario actual. **PrintForm** imprime todo el formulario, incluso si alguna parte del formulario no es visible en la pantalla. Sin embargo, si un formulario contiene gráficos, los gráficos sólo se imprimen si la propiedad **AutoRedraw** del formulario es **True**. Cuando termina la impresión, **PrintForm** llama al método **EndDoc** para dejar preparada la impresora.

Por ejemplo, podría enviar texto a una impresora si lo imprime en un formulario y llama después a **PrintForm** con las instrucciones siguientes:

```
Print "Esto es un texto."
PrintForm
```

El método **PrintForm** es, con mucho, la forma más sencilla de imprimir desde una aplicación. Como puede enviar información a la impresora con la resolución de la pantalla del usuario (normalmente 96 puntos por pulgada), los resultados pueden ser desalentadores en las impresoras con mayor resolución (normalmente 300 puntos por pulgada en las impresoras láser). Los resultados pueden variar según los objetos del formulario.

Este *control* tiene la propiedad llamada **Drive** que recoge la unidad seleccionada por el usuario (puede ser una unidad física como el disco **c:** o una unidad lógica asignada por el usuario a otro disco o directorio en un servidor o en otro ordenador de la red.)

 El *cuadro de lista de directorios* **(DirList)** nos permite mostrar los directorios del sistema de archivos del ordenador. Es conveniente que este control muestre tres o cuatro carpetas o directorios. En tiempo de diseño muestra la carpeta en la que se inicia la aplicación y en la que por defecto se guarda el proyecto. Este *control* posee la propiedad **Path** que determina y asigna la unidad que se mostrarán en dicha caja.

 El *cuadro de lista de archivos* **(FileList)** nos muestra los archivos de un determinado directorio o carpeta. Su propiedad más interesante es **Pattern** que nos permite especificar qué tipo de archivos son mostrados en dicho control. Para utilizar esta propiedad se pueden utilizar los comodines * y ? al establecer la propiedad. Estos caracteres tienen el mismo significado que en MS-DOS o Windows para especificar los nombres de los archivos. Si estableces la propiedad **Pattern** con la cadena *.txt, estás indicando que se muestren sólo los archivos que tengan esta extensión. Se pueden mostrar más de un tipo de archivos separándolos con punto y coma (**;**).

Para conectar los tres controles haga lo siguiente:

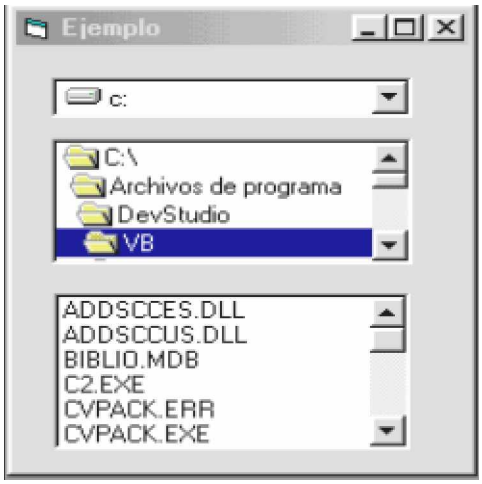
En tiempo de diseño, al dibujar los distintos controles del sistema de archivos, estos muestran la unidad y el directorio en la que se crea el proyecto como he comentado anteriormente. En tiempo de ejecución el usuario puede cambiar la unidad y el directorio o carpeta y esta situación no se verá reflejada si no se escribe código. Para que los controles estén sincronizados es necesario conectarlos. El evento predeterminado del control cuadro de lista de unidades es **Change**. Este evento sucede cuando el usuario despliega la lista de unidades y selecciona una unidad distinta a la actual, por lo que es el evento adecuado para actualizar la lista de directorios de la siguiente forma:

```
Private Sub Dir1_Change ( )
    Dir1.Parh = Drive1.Drive
End Sub
```

Para el control cuadro de lista de directorios deberemos hacer algo parecido, el código será el siguiente:

```
Private Sub Dir1_Change ( )
File1.Path = Dir1.Path
End Sub
```

De esta forma tenemos conectados los tres *controles* de acceso al sistema de archivos. A continuación, se muestra una imagen con los tres *controles* relacionados:



6.5 Introducción a los archivos

Un **archivo (fichero)** es un conjunto de información relacionada entre sí, almacenada como una unidad en un dispositivo de almacenamiento secundario (disquete, disco duro). Los datos almacenados en un archivo son de manera permanente de modo que pueden ser manipulados en cualquier momento. Cada archivo está referenciado por un identificador, su nombre.

6.6 Concepto de archivos bajo Windows/Visual Basic

Un **archivo** tiene un nombre almacenado en una carpeta junto con otros archivos de disco. Los nombres de los archivos en Windows y Visual Basic requieren de 1 a 215 caracteres (incluidos espacios en blanco), y pueden incluir también una extensión de 1 a 3 letras, normalmente son significativas y relativas al contenido del mismo. Por ejemplo:

LEAME.TXT	Archivo de texto
MISDATOS.DAT	Archivo de datos
PLANILLA.XLS	Archivo de MS-Excel
SHAKIRA.BMP	Archivo de mapa de bits

El nombre de un archivo para ser referenciado correctamente consta de:

- Unidad (Especificador de unidad, por ejemplo A, B, C)
- Camino (Especificador de ruta, por ejemplo \DATA\)
- Nombre (Especificador de archivo, por ejemplo DEMO.DAT)

Como ejemplo, suponga que el archivo DEMO.DAT está en la carpeta DATA de la unidad C (disco duro); la descripción completa del nombre del archivo es:

C:\DATA\DEMO.DAT

Ahora, suponga que el archivo CONSTANTES.DAT se encuentra en la carpeta LIB, que a su vez está contenida en la carpeta DATA de la unidad C. La descripción completa del nombre de dicho archivo está dada por:

C:\DATA\LIB\CONSTANTES.DAT

Desde el punto de vista de Visual Basic un archivo almacena los datos como un conjunto de registros, conteniendo todos ellos, generalmente, los mismos campos. Cada campo almacena un dato de tipo predefinido o de un tipo definido por el usuario. El elemento de información más simple estaría formado por un carácter.

6.7 Operaciones sobre el sistema de archivos

Para manipular el sistema de archivos de un disco, Visual Basic proporciona las sentencias que a continuación se describen.

- 6.7.1 Sentencia Kill

Permite eliminar un archivo almacenado en un disco. Su sintaxis es de la forma:

Kill NOMBRE_ARCHIVO

Donde NOMBRE_ARCHIVO es una cadena de caracteres que identifica al archivo que se desea borrar. Se permiten caracteres comodín (* y ?). Si el archivo no existe se produce un error. Por ejemplo:

Kill "C:\TEMP\TEMPO.DAT"

La instrucción anterior elimina el archivo TEMPO.DAT ubicado en la carpeta C:\TEMP.

- 6.7.2 Sentencia Name

Permite cambiar el nombre (renombrar) de un archivo del disco y moverlo a otro directorio si fuera preciso. Su sintaxis es la siguiente:

Name NOMBRE_ACTUAL **As** NOMBRE_NUEVO

Donde NOMBRE_ACTUAL es una cadena de caracteres que específica el nombre actual del archivo y NOMBRE_NUEVO es otra cadena de caracteres que indica el nuevo nombre que se desea dar al archivo. Este nombre no debe existir, de lo contrario se obtendrá un error. Por ejemplo:

Name "C:\TEMP\TEMPO.DAT" **As** "C:\DATA\CURSO.DAT"

La instrucción anterior cambia el nombre del archivo TEMPO.DAT por el nombre CURSO.DAT. El contenido del archivo permanece inalterado y su localización física se cambia de la carpeta C:\TEMP a C:\DATA.

- 6.7.3 Sentencia Mkdir

Permite crear una nueva carpeta. Su sintaxis es:

MkDir NUEVA_CARPETA

Donde NUEVA_CARPETA es una cadena de caracteres que identifica la carpeta que se va ha crear. Por ejemplo:

MkDir "C:\DATA\TEMP"

- 6.7.4 Sentencia RmDir

Elimina una carpeta vacía existente en el disco. Su sintaxis es de la forma:

RmDir NOMBRE_CARPETA

Donde NOMBRE_CARPETA es una cadena de caracteres que identifica la carpeta que se desea eliminar. Por ejemplo:

RmDir "C:\DATA\TEMP"

- 6.7.5 Sentencia ChDir

Permite cambiar la carpeta actual. Su sintaxis es:

ChDir NOMBRE_CARPETA

Donde NOMBRE_CARPETA es una cadena de caracteres que identifica la nueva ruta de acceso predeterminada. Por ejemplo:

ChDir "C:\DATA\GRAFICOS"

MsgBox App**.Path**

La propiedad **Path** del objeto **App** especifica la ruta de acceso actual.

- 6.7.6 Sentencia ChDrive

Permite cambiar la unidad de disco actual. Su sintaxis es:

ChDrive UNIDAD

Donde UNIDAD es un carácter que especifica la nueva unidad de disco. Si el parámetro UNIDAD es una cadena de múltiples caracteres sólo se lee la primera letra. Por ejemplo:

ChDrive "A"

6.8 Operaciones con archivos

Para realizar alguna operación sobre un archivo hay que referenciarlo mediante su nombre completo. Las operaciones básicas que se pueden realizar con los archivos son:

- Abrir, preparar un archivo para hacer referencia a él.
- Escribir, introducir un elemento de información a un archivo.
- Leer, obtener un elemento de información de un archivo.
- Modificar, alterar un elemento de información ya existente en un archivo.
- Cerrar, evitar cualquier otra referencia al archivo en tanto no se le abra otra vez.

6.9 Tipos de archivos

Los *tipos de archivos* dependen del modo como están organizados los registros y de la forma de accesar a la los datos contenidos en ellos. En Visual Basic existen tres tipos de archivos de datos, estos son:

- Archivos secuenciales (acceso secuencial).
- Archivos aleatorios (acceso aleatorio).
- Archivos binarios (acceso binario).

A continuación pasamos a describir cada uno de ellos en forma detallada.

- 6.9.1 Archivos de acceso secuencial

En un archivo de *acceso secuencial* los registros se almacenan siguiendo una a otro, según el orden en que son ingresados. Cuando se lee la información, se empieza por el primer registro y se continúa al siguiente hasta alcanzar el final. Las sentencias y funciones necesarias para manipular archivos de tipo secuencial se presentan a continuación.

- Sentencia Open

Permite abrir un archivo. La sintaxis para esta sentencia es la siguiente:

Open NOMBRE_ARCHIVO **For** MODO **As** # NÚMERO_ARCHIVO

Donde NOMBRE_ARCHIVO es una cadena que especifica el nombre del archivo que se debe ser abierto en MODO **Output**, **Append** o **Input**.

Modo	Descripción
Output	Escritura de datos. Si el archivo existe, su contenido actual se destruye. Si el archivo no existe, se crea.
Append	Añadir datos. Los datos son añadidos a partir de los últimos existentes. Si el archivo no existe, se crea.
Input	Lectura de datos. La lectura empieza desde el principio del archivo. Si el archivo no existe, se produce un error.

El parámetro NÚMERO_ARCHIVO es un entero cuyo valor debe estar comprendido entre 1 y 511. Este número será asociado al nombre del archivo mientras éste permanezca abierto. Para obtener el número del siguiente archivo disponible se utiliza la función **FreeFile()**.

Como ejemplo suponga que se requiere abrir el archivo DEMO.DAT ubicado en la carpeta C:\DATA, la instrucción sería la siguiente:

```
Dim N1 As Integer
N1 = FreeFile()
Open "C:\DATA\DEMO.DAT" For Output As # N1
```

- Sentencia Print

Permite escribir datos secuencialmente en un archivo. Su sintaxis es:

Print # NÚMERO_ARCHIVO, LISTA_DE_EXPRESIONES

Donde NÚMERO_ARCHIVO es el número utilizado cuando el archivo fue abierto. LISTA_DE_EXPRESIONES es un conjunto de expresiones (numéricas, de cadena, de fecha, etc.) separadas por punto y coma (**;**) que serán escritas en el archivo.

La sentencia **Print** escribe en el archivo una imagen de los datos tal y como se habrían visualizado sobre el formulario con la sentencia Print. Por ejemplo:

```
Dim N1 As Integer
N1 = FreeFile()
Open "C:\DATA\DEMO.DAT" For Output As # N1
Print # N1, "Visual Basic es fácil"; ", "; Date()
```

Al ejecutarse el código anterior se escribiría en el archivo la siguiente información:

Visual Basic es fácil, 24/04/2001

Como se observa, al utilizar la sentencia **Print** se deben delimitar los datos para que se impriman correctamente.

- Sentencia Write

Permite escribir datos secuencialmente en un archivo. Su sintaxis es:

Write # NÚMERO_ARCHIVO, LISTA_DE_EXPRESIONES

Donde NÚMERO_ARCHIVO es el número utilizado cuando el archivo fue abierto. LISTA_DE_EXPRESIONES es un conjunto de expresiones (numéricas, de cadena, de fecha, etc.) separadas por punto y coma (**;**) que serán escritas en el archivo.

La sentencia **Write** inserta comas (**,**) entre las expresiones de la LISTA_DE_EXPRESIONES, por tanto no es necesario poner delimitadores explícitamente como en el caso de la sentencia **Print**.

Cuando se utiliza la sentencia **Write** para escribir información en un archivo, se siguen distintas convenciones universales, de modo que los datos siempre se pueden leer e interpretar correctamente, independientemente de la configuración regional, estas convenciones son:

- Los datos numéricos siempre se escriben utilizando la coma (,) como separador decimal.
- Para datos de tipo Boolean se imprime **# TRUE #** o **# FALSE #.**
- Los datos de tipo Date se escriben en el archivo usando el formato de fecha universal (fechas como **# aaaa-mm-dd #** y horas como **# hh:mm:ss #).**

A manera de ejemplo considere el siguiente segmento de código:

```
Dim N1 As Integer
N1 = FreeFile()
Open "C:\DATA\DEMO.DAT" For Output As # N1
Write # N1, "Visual Basic es fácil"; Date()
```

La ejecución de este código escribiría en el archivo la siguiente información:

```
Visual Basic es fácil", # 2002-04-11 #
```

- Sentencia Close

Cierra uno archivo abierto mediante la sentencia Open. Su sintaxis es la siguiente:

```
Close # NÚMERO_ARCHIVO [, # NÚMERO_ARCHIVO, . . .]
```

Donde NÚMERO_ARCHIVO es el número con el cual se abrió el archivo. Por ejemplo:

```
Close # 1, # 2
```

La instrucción anterior cierra los archivos asociados con los números 1 y 2. La siguiente sentencia cierra todos los archivos abiertos.

```
Close
```

Sentencia Input

Permite leer datos de un archivo secuencial y los asigna a las variables especificadas. Su sintaxis es:

Input # NÚMERO_ARCHIVO, VARIABLE1 [, VARIABLE2, . . .]

Donde NÚMERO_ARCHIVO es el número utilizado cuando el archivo fue abierto. VARIABLE1, VARIABLE2, . . . son los nombres de las variables que han de recibir los correspondientes datos del archivo.

Los datos del archivo deben aparecer en el mismo orden que tienen las variables en la sentencia **Input** y deben coincidir con variables del mismo tipo de datos. Por ejemplo:

```
Dim N1 As Integer
N1 = FreeFile()
Open "C:\DATA\DEMO.DAT" For Input As # N1
Dim A As Integer
Dim B As Double
Dim S As String
Dim F As Date
Input # N1, A, B, S, F
```

El segmento de código anterior espera encontrar en el archivo un entero, un real, una cadena y una fecha, en ese orden (separados por comas o un retorno de carro).

- Sentencia Line Input

Permite leer una línea de un archivo secuencial ignorando los delimitadores (comas) y la asigna a una variable tipo cadena. Su sintaxis es:

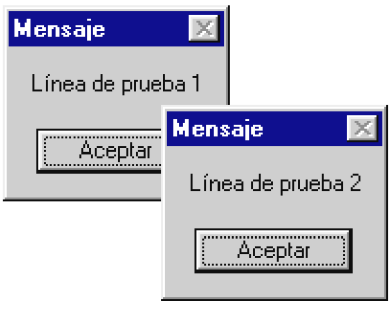
```
Line Input # NÚMERO_ARCHIVO, VARIABLE
```

Donde NÚMERO_ARCHIVO es el número utilizado cuando el archivo fue abierto. VARIABLE es el nombre de una variable tipo cadena de caracteres.

La sentencia **Line Input** se utiliza especialmente para leer un archivo de texto línea a línea, ya que esta sentencia lee todos los caracteres del archivo hasta que encuentra un retorno de carro, entonces continua en la siguiente línea y así sucesivamente. Por ejemplo:

```
Dim N1 As Integer, LINE1 As String, LINE2 As String
N1 = FreeFile()
Open "C:\DATA\DEMO.TXT" For Output As # N1
Print # N1, "Línea de prueba 1"
Print # N1, "Línea de prueba 2"
Close # N1
Open "C:\DATA\DEMO.TXT" For Input As # N1
Line Input # N1, LINE1
MsgBox LINE1
Line Input # N1, LINE2
MsgBox LINE2
```

La ejecución del código anterior produce la siguiente salida:



- Función Input

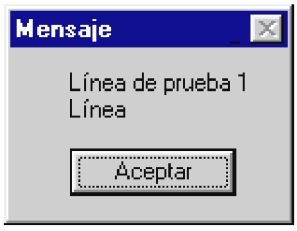
Retorna los siguientes *N caracteres* de un *archivo secuencial* y los asigna a una variable de cadena. Su sintaxis es de la forma:

VARIABLE = **Input**(*N* , # NÚMERO_ARCHIVO)

A diferencia de la sentencia **Input**, la función **Input()** retorna todos los caracteres que lee, incluyendo comas, retornos de carro, continuaciones de línea, etc. Por ejemplo:

```
Dim N1 As Integer, S As String
N1 = FreeFile()
Open "C:\DATA\DEMO.TXT" For Output As # N1
Print # N1, "Línea de prueba 1"
Print # N1, "Línea de prueba 2"
Close # N1
Open "C:\DATA\DEMO.TXT" For Input As # N1
S = Input(24, # N1)
MsgBox S
```

La ejecución del código anterior produce la siguiente salida:



Función EOF

Especifica si se ha llegado al final de un archivo. Su sintaxis es de la forma:

VARIABLE = **EOF**(NÚMERO_ARCHIVO)

Se utiliza **EOF()** para evitar producir un error al intentar obtener información más allá del final del archivo. **EOF()** retorna un valor **True** si se ha alcanzado el final del archivo y **False** en caso contrario. Por ejemplo:

```
Dim N1 As Integer
Dim CADENA As String
N1 = FreeFile()
Open "C:\DATA\DEMO.TXT" For Input As # N1
While Not EOF(N1)
    Line Input # N1, CADENA
    Print CADENA
Wend
Close # N1
```

Este segmento de código lee y visualiza cada línea del archivo de texto DEMO.TXT. El bucle finaliza cuando se detecta el final del archivo. Para que el código anterior funcione correctamente, no olvide poner la propiedad **AutoRedraw** del formulario a **True**.

Como ejemplo final del uso de archivos secuénciales vamos a desarrollar un sencillo editor de texto. Este editor aunque sus prestaciones son bastante limitadas va ha servir para poner en práctica lo aprendido recientemente.

- 6.9.2 Archivos de acceso aleatorio

En los archivo de *acceso aleatorio* el almacenamiento de los datos se hace mediante registros (todos de la misma longitud), lo cuales son identificados mediante un único número denominado índice. El primer registro de un archivo tiene como índice 1, el segundo tiene índice 2 y así sucesivamente. La información contenida en un archivo de este tipo puede ser accedida en cualquier secuencia, ya que cada registro individual se asocia con su respectivo índice y puede ser leído, escrito o actualizado.

Las sentencias y funciones necesarias para manipular archivos de tipo aleatorio se presentan a continuación.

- Sentencia Open

Permite abrir un archivo. La sintaxis para acceder aleatoriamente a un archivo es:

Open NOMBRE_ARCHIVO **For Random As** # NÚMERO_ARCHIVO **Len** = LON_REG

Donde NOMBRE_ARCHIVO es una cadena que especifica el nombre del archivo que se debe ser abierto en modo Random.

El parámetro NÚMERO_ARCHIVO es un entero cuyo valor está comprendido entre 1 y 511. Este número será asociado con el nombre del archivo mientras permanezca abierto. LON_REG es un entero que establece la longitud del registro para archivos aleatorios.

- Sentencia Put

Permite grabar un registro en un archivo abierto para acceso aleatorio. Su sintaxis es:

Put # NÚMERO_ARCHIVO, NÚMERO_REG, VARIABLE

Donde NÚMERO_ARCHIVO es el número bajo el cual se abrió el archivo, NÚMERO_REG es el número correspondiente al registro que se va ha grabar y VARIABLE contiene los datos a escribir en el archivo. Por ejemplo:

```
Dim N1 As Integer, REG As DISTRITO
REG.ID_DISTRITO = "L09"
REG.NOMBRE = "Chorrillos"
N1 = FreeFile()
Open "C:\DATA\RAND1.DAT" For Random As # N1 Len = Len(REG)
Put # N1, 1, REG
Close # N1
```

El segmento de código anterior utiliza una variable REG de tipo DISTRITO, cuya definición es la siguiente:

```
Private Type DISTRITO
    ID_DISTRITO As String * 3
    NOMBRE As String * 30
End Type
```

- Sentencia Get

Permite leer un registro procedente de un archivo de acceso aleatorio, almacenando los datos en una variable específica. Su sintaxis es de la forma:

Get # NÚMERO_ARCHIVO, NÚMERO_REG, VARIABLE

Donde NÚMERO_ARCHIVO es el número bajo el cual se abrió el archivo, NÚMERO_REG es el número correspondiente al registro que se va ha leer y VARIABLE almacena los datos del registro leído. Por ejemplo:

```
Dim N1 As Integer, I As Integer, REG As DISTRITO
N1 = FreeFile()
Open "C:\DATA\RAND1.DAT" For Random As # N1 Len = Len(REG)
I = 1
While Not EOF(# N1)
    Get # N1, I, REG
    MsgBox REG.ID_DISTRITO & " " & REG.NOMBRE
    I = I + 1
Wend
Close # N1
```

Cuando **EOF()** se utiliza con un archivo aleatorio, retorna un valor True si una sentencia Get intenta leer y no puede porque ha alcanzado el final del archivo.

- Función LOF

Retorna el número de bytes (caracteres) que ocupa un determinado archivo abierto mediante la sentencia Open. Su sintaxis es:

VARIABLE = **LOF**(# NÚMERO_ARCHIVO)

Donde NÚMERO_ARCHIVO es el número con el que se abrió el archivo.

Esta función es de utilidad, porque aplicada a un archivo de acceso aleatorio, permite conocer el número de registros almacenados en el archivo. Para ello debe dividir el valor retornado entre la longitud del registro. Como ejemplo, considere lo siguiente:

```
Dim N1 As Integer, REG As DISTRITO
Dim NUM_REGS As Integer, I As Integer
N1 = FreeFile()
Open "C:\DATA\RAND1.DAT" For Random As # N1 Len = Len(REG)
NUM_REGS = LOF(N1) / Len(REG)
For I = 1 To NUM_REGS
    Get # N1, I, REG
    MsgBox REG.ID_DISTRITO & " " & REG.NOMBRE
Next
Close # N1
```

- Función Loc

Esta función retorna la posición actual dentro de un fichero. Su sintaxis es:

VARIABLE = **Loc**(# NÚMERO_ARCHIVO)

La función **Loc()** aplicada a un archivo de acceso aleatorio retorna el número del último registro leído o grabado en el archivo especificado. Por ejemplo:

```
Dim N1 As Integer, REG As DISTRITO
Dim NUM_REGS As Integer, I As Integer
N1 = FreeFile()
Open "C:\DATA\RAND1.DAT" For Random As # N1 Len = Len(REG)
NUM_REGS = LOF(N1) / Len(REG)
I = 1
Do While True
    Get # N1, I, REG
    MsgBox REG.ID_DISTRITO & " " & REG.NOMBRE
    I = I + 1
    If Loc(N1) = NUM_REGS Then Exit Do
Loop
Close # N1
```

La sentencia **If** finaliza el bucle si se ha alcanzado el último registro.

Como ejemplo final vamos a desarrollar una aplicación que permita realizar el mantenimiento de los datos almacenados en el archivo C:\DATA\CURSO.DAT.

- 6.9.3 Archivos de acceso binario

Un *archivo binario* contiene más que simplemente texto. Puede contener imágenes, sonido, hojas de cálculo, o documentos concebidos para el procesamiento de texto.

El *acceso binario* permite la posibilidad de tratar cualquier archivo como una secuencia numerada de bytes, independientemente de la estructura del mismo. Los bytes ocupan las posiciones 1, 2, 3, etc. Por ejemplo, si se requiere recuperar un dato de tipo entero (Integer, 2 bytes) de la posición 3 del archivo, serían recuperados los bytes 3 y 4 para poder formar el valor del entero. Por tanto, antes de trabajar con archivos binarios es necesario conocer cómo fueron escritos los datos que contiene para poder recuperarlos correctamente.

- Sentencia Open

Permite abrir un archivo para acceso binario. Su sintaxis es de la forma:

Open NOMBRE_ARCHIVO **For Binary As** # NÚMERO_ARCHIVO

Donde NOMBRE_ARCHIVO es una cadena que especifica el nombre del archivo que se debe ser abierto en modo Binary. El parámetro NÚMERO_ARCHIVO es un entero cuyo valor está comprendido entre 1 y 511. Este número será asociado con el nombre del archivo mientras permanezca abierto.

- Sentencia Put

Permite grabar en un archivo binario tantos bytes como haya en una variable. Su sintaxis es:

Put # NÚMERO_ARCHIVO, POSICIÓN, VARIABLE

Donde NÚMERO_ARCHIVO es el número bajo el cual se abrió el archivo. POSICIÓN es el número de byte a partir del cual se han de grabar los datos contenidos en VARIABLE.

- **Sentencia Get**

Permite leer de un archivo binario tantos bytes como quepan en una variable. Su sintaxis es de la forma:

Get # NÚMERO_ARCHIVO, POSICIÓN, VARIABLE

Donde NÚMERO_ARCHIVO es el número bajo el cual se abrió el archivo. POSICIÓN es el número de byte a partir del cual se han de leer los datos almacenados en VARIABLE.

- **Sentencia Seek**

Permite situar la posición de lectura o de escritura en una posición determinada dentro del archivo. Su sintaxis es:

Get # NÚMERO_ARCHIVO, POSICIÓN, VARIABLE

Donde POSICIÓN es el número de byte a partir del cual queremos leer o escribir dentro del archivo. Como ejemplo vamos a desarrollar una aplicación que permite guardar "cifrados" los textos ingresados en un cuadro de texto. Para ello crear un nuevo proyecto y ubicar en el formulario un cuadro de texto (TxtMensaje) y un botón de comando (CmdCifrar). Luego, ingrese el siguiente código:

```
Const CLAVE As Integer = 3

Private Sub CmdCifrar_Click ( )
    Dim N1 As Integer, I As Integer
    Dim CAR As String * 1
    N1 = FreeFile()
    Open "C:\DATA\DEMO.BIN" For Binary As # N1
    For I = 1 To Len(TxtMensaje)
```

```
        CAR = Chr((Asc(Mid(TxtMensaje, I, 1)) + CLAVE) Mod 256)
            Put # N1, , CAR
        Next
        Close # N1
End Sub
```

En el código la función **Mid**() obtiene el carácter "I" de la caja de texto, la función Asc() obtiene su código ANSI, al que sumamos el valor de CLAVE, para después obtener el resto de la división entre 256, con el fin de mantenernos en el rango de 0 a 255 (rango de valores de la tabla de caracteres ANSI). Por último, la función **Chr**() retorna el carácter correspondiente al valor obtenido, el cual es almacenado en el archivo binario.

Por ejemplo si ingresa el mensaje "HOLA" se almacena en el archivo como "KROD" (lo puede comprobar mediante el Bloc de notas), ya que el valor **ANSI** de la "H" es 72, este carácter al sumarle el valor de CLAVE sería el 75 (72 + 3), que es la "K", y así sucesivamente (ver la tabla de caracteres **ANSI**).

El descifrado sería el proceso inverso, para ello crear un nuevo formulario y ubicar un botón de comando (CmdDescifrar), luego ingresar el código siguiente:

```
Const CLAVE As Integer = 3
Private Sub CmdDescifrar_Click()
        Dim N1 As Integer, I As Integer
        Dim CAR As String * 1, CADENA As String
        N1 = FreeFile()
        Open "C:\DATA\DEMO.BIN" For Binary As # N1
        Get # N1,  , CAR
```

```
    While Not EOF(N1)
        CAR = Chr((Asc(CAR) + (256 - Val(CLAVE))) Mod 256)
        CADENA = CADENA & CAR
        Get # N1, , CAR
    Wend
    MsgBox CADENA
End Sub
```

Códigos ASCII normales (códigos 0 - 127)

000	(nul)	016 ►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Códigos ASCII extendidos (códigos 128 - 255)

128	Ç	143	Å	158	×	172	¼	186	‖	200	╚	214	Í	228	õ	242	‗
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	Î	229	Õ	243	¾
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	Ï	230	µ	244	¶
131	â	146	Æ	161	í	175	»	189	¢	203	╦	217	┘	231	þ	245	§
132	ä	147	ô	162	ó	176	░	190	¥	204	╠	218	┌	232	Þ	246	÷
133	à	148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Ú	247	¸
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Û	248	°
135	ç	150	û	165	Ñ	179	│	193	┴	207	¤	221	¦	235	Ù	249	¨
136	ê	151	ù	166	ª	180	┤	194	┬	208	ð	222	Ì	236	ý	250	·
137	ë	152	ÿ	167	º	181	Á	195	├	209	Ð	223	▀	237	Ý	251	¹
138	è	153	Ö	168	¿	182	Â	196	─	210	Ê	224	Ó	238	¯	252	³
139	ï	154	Ü	169	®	183	À	197	┼	211	Ë	225	ß	239	´	253	²
140	î	155	ø	170	¬	184	©	198	ã	212	È	226	Ô	240		254	■
141	ì	156	£	171	½	185	╣	199	Ã	213	ı	227	Ò	241	±	255	

● **Shape:** Es un control gráfico que se muestra como un rectángulo, un cuadrado, una elipse, un círculo, un rectángulo redondeado o un cuadrado redondeado.

● **RichTextBox:** Es un control que permite al usuario escribir y modificar texto al tiempo que proporciona características de formato más avanzadas que el control **TextBox** convencional.

● **HScrollBar, VScrollBar (Controles):** Las barras de desplazamiento permiten explorar fácilmente una larga lista de elementos o una gran cantidad de información. Además, proporcionan una representación análoga de la posición actual. Puede usar una barra de desplazamiento como dispositivo de entrada o como indicador de velocidad o cantidad; por ejemplo, para controlar el volumen de un juego de PC o para ver el tiempo transcurrido en un proceso temporizado.

● **DataGrid (Control):** Muestra y permite la manipulación de datos de una serie de filas y columnas que corresponden a registros y campos de un objeto **Recordset**.

● **MSFlexGrid:** El control Microsoft FlexGrid (**MSFlexGrid**) muestra datos de tablas y efectúa operaciones en ellos. Proporciona una flexibilidad completa para ordenar, combinar y aplicar formato a tablas que contienen cadenas e imágenes. Cuando se enlaza a un control **Data**, el control **MSFlexGrid** muestra datos de sólo lectura.

● **Control de datos ADO:** es similar al control intrínseco **Data** y al **Control de datos remotos** (RDC). El **Control de datos ADO** permite crear rápidamente una conexión con una base de datos mediante **Objetos de datos ActiveX de Microsoft** (ADO). Es posible crear en tiempo de diseño una conexión al establecer la propiedad **ConnectionString** con una cadena de conexión válida y, a continuación, la propiedad **RecordSource** con una instrucción apropiada para el administrador de base de datos. Puede establecer también la propiedad **ConnectionString** con el nombre de un archivo que defina una conexión; el archivo se genera mediante el cuadro de diálogo **Vínculo de datos** que aparece cuando hace clic en **ConnectionString** en la ventana Propiedades y, después, en **Generar** o en **Seleccionar**.

● **Winsock:** El control **Winsock**, invisible para el usuario, proporciona un acceso sencillo a los servicios de red TCP y UDP. Pueden usarlo los programadores de Microsoft Access, Visual Basic, Visual C++ o Visual FoxPro. Para escribir aplicaciones de servidor o de cliente no necesita comprender los detalles de TCP ni llamar a las API de **Winsock** de nivel inferior. Si establece las propiedades y llama a los métodos del control, podrá conectar fácilmente con un equipo remoto e intercambiar datos en las dos direcciones.

► **MSDN Library Visual Studio 6.0a** ◄

► **Aprenda Visual Basic 6.0 (Como si estuviera en primero)** ◄
Javier García de Jalón * José Ignacio Rodríguez * Alfonso Brazález

► **Capítulo 7 (Los archivos).** ***Documento Electrónico*** ◄
Carlos Castillo Peralta