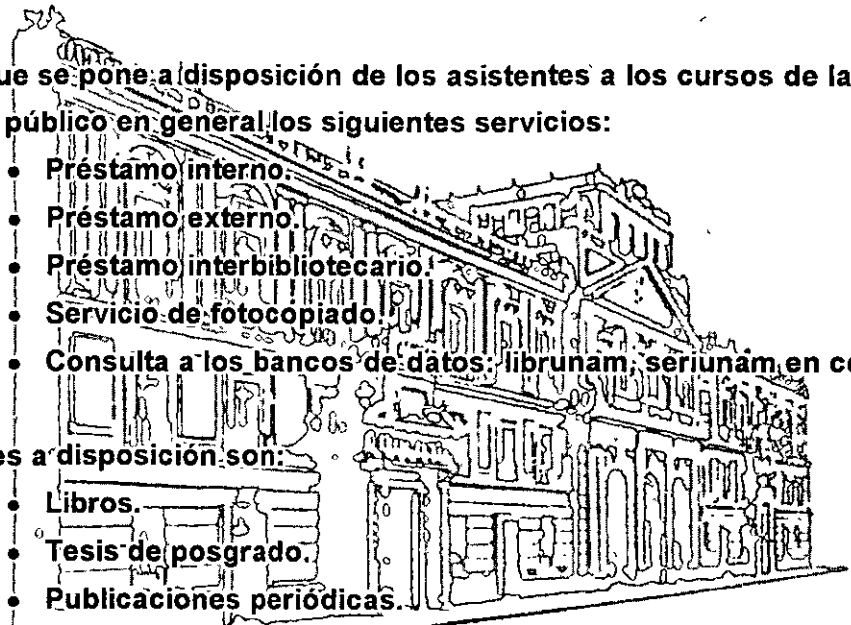
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN
" ING. BRUNO MASCANZONI "

El Centro de Información y Documentación Ing. Bruno Mascanzoni tiene por objetivo satisfacer las necesidades de actualización y proporcionar una adecuada información que permita a los ingenieros, profesores y alumnos estar al tanto del estado actual del conocimiento sobre temas específicos, enfatizando las investigaciones de vanguardia de los campos de la ingeniería, tanto nacionales como extranjeras.



Es por ello que se pone a disposición de los asistentes a los cursos de la DECFI, así como del público en general los siguientes servicios:

- **Préstamo interno.**
- **Préstamo externo.**
- **Préstamo interbibliotecario.**
- **Servicio de fotocopiado.**
- **Consulta a los bancos de datos: librunam, seriunam en cd-rom.**

Los materiales a disposición son:

- **Libros.**
- **Tesis de posgrado.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1988 a la fecha.**

En las áreas de ingeniería industrial, civil, electrónica, ciencias de la tierra, computación y, mecánica y eléctrica.

El CID se encuentra ubicado en el mezzanine del Palacio de Minería, lado oriente.

El horario de servicio es de 10:00 a 14:30 y 16:00 a 17:30 de lunes a viernes.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

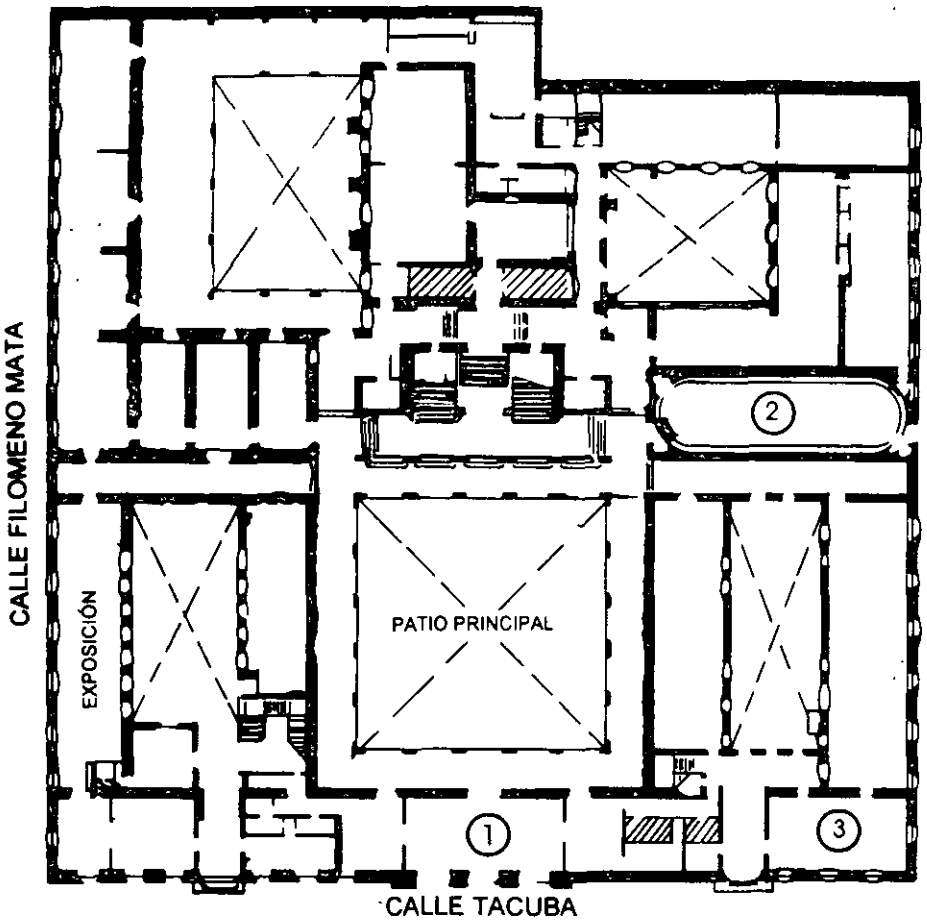
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

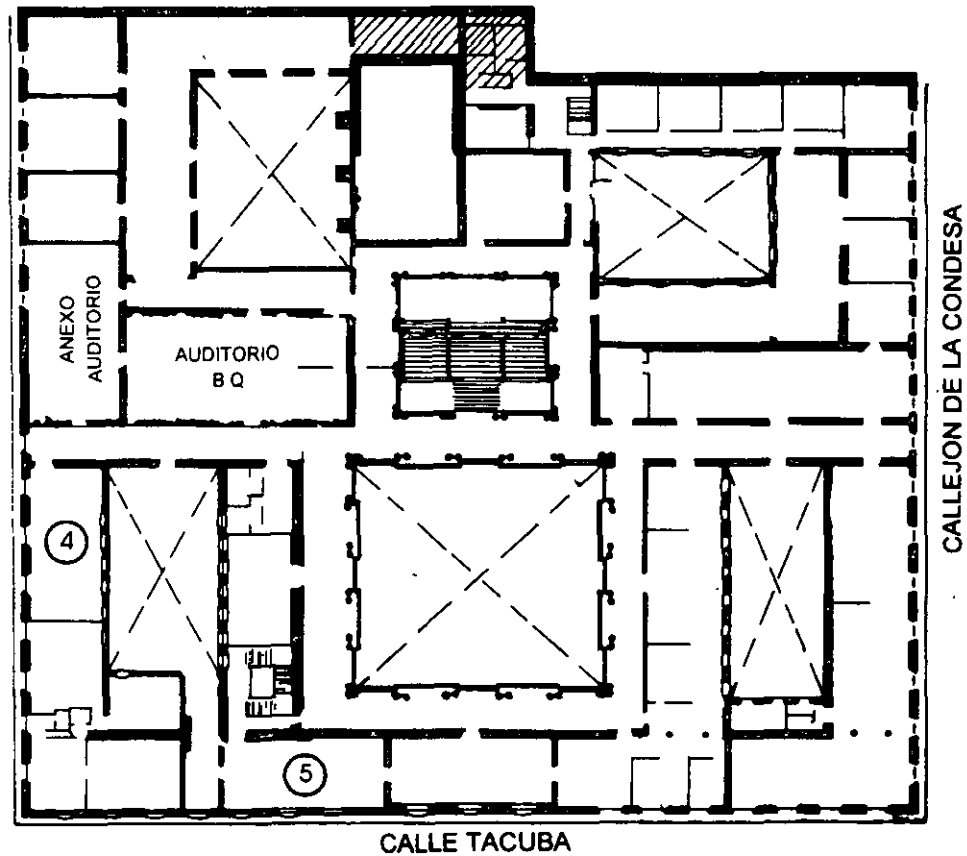
División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO Postal M-2285
Teléfonos: 5512-8955 5512-5121 5521-7335 5521-1987 Fax 5510-0573 5521-4021 AL 25

PALACIO DE MINERIA

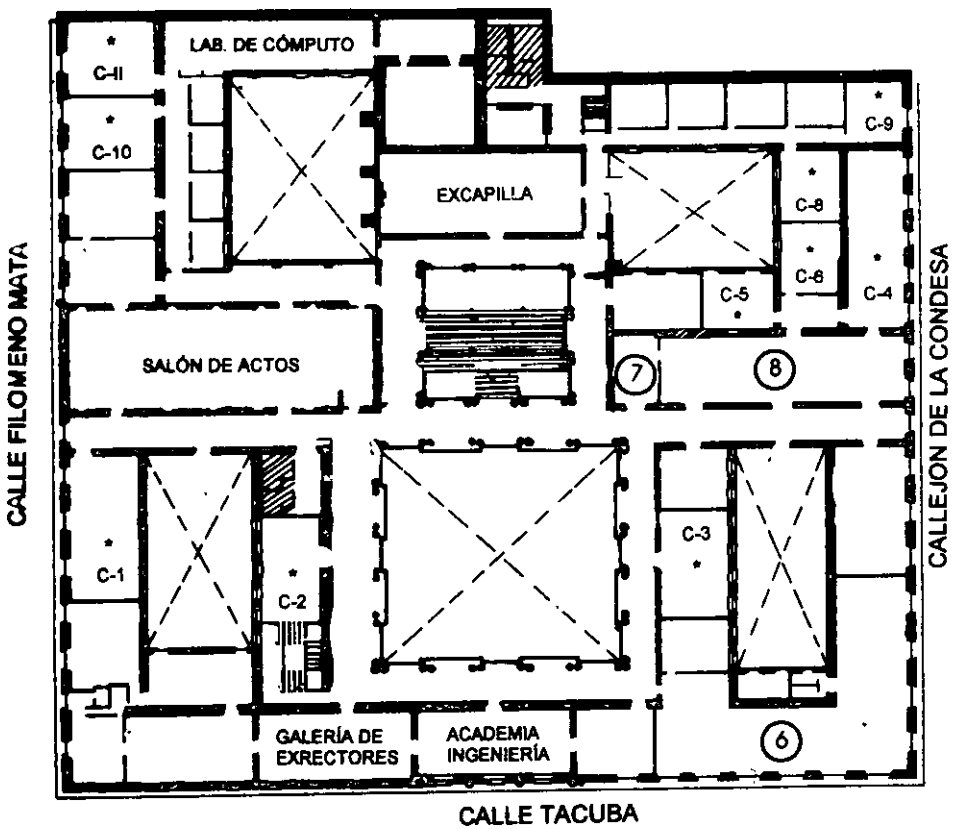


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



1er. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA

CURSO: ***CC056 Visual Basic Básico***
FECHA: ***2 al 13 de septiembre del 2002***

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN: 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
Ing. Oscar Martín del Campo Cárdenas				

Promedio _____

EVALUACIÓN DE LA ENSEÑANZA

CONCEPTO	CALIF.
ORGANIZACIÓN Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD DEL CURSO	
ACTUALIZACIÓN DEL CURSO	
APLICACIÓN PRACTICA DEL CURSO	

Promedio _____

EVALUACIÓN DEL CURSO

CONCEPTO	CALIF
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDÁCTICO UTILIZADO	

Promedio _____

Evaluación total del curso_____

Continúa...2

1. ¿Le agradó su estancia en la División de Educación Continua?

SI ☐ NO ☐

Si indica que "NO" diga porqué: _____

_____

2. Medio a través del cual se enteró del curso:

Periódico *La Jornada*	
Folleto anual	
Folleto del curso	
Gaceta UNAM	
Revistas técnicas	
Otro medio (Indique cuál)	

3. ¿Qué cambios sugeriría al curso para mejorarlo?

_____

_____

_____

_____

_____

_____

4. ¿Recomendaría el curso a otra(s) persona(s) ?

SI ☐ NO ☐

5.¿Qué cursos sugiere que imparta la División de Educación Continua?

_____

_____

_____

_____

_____

_____

6. Otras sugerencias:

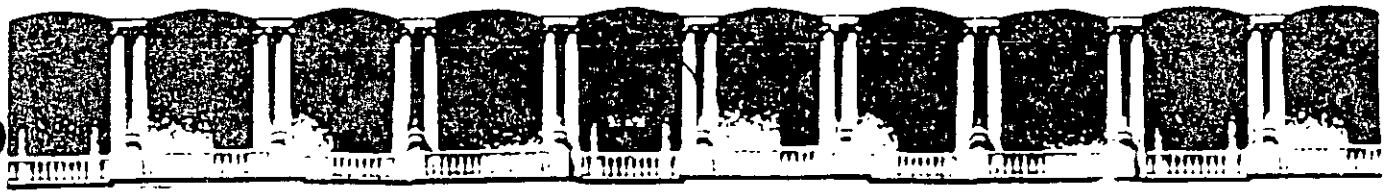
_____

_____

_____

_____

_____



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

Material Didáctico del Curso

Visual Basic 6.0

Básico

Oscar D. Martín del Campo Cárdenas

INDICE

Laboratorios

TEMA I

Introducción al desarrollo de aplicaciones utilizando Visual Basic

TEMA I Introducción al desarrollo de aplicaciones utilizando Visual Basic

Objetivos:

- Al final del capítulo el alumno será capaz de identificar los elementos del Ambiente de desarrollo de Microsoft Visual Basic.
- Sabrá explicar las diferencias entre los términos "tiempo de diseño" y "tiempo de ejecución".
- Podrá explicar el concepto de "programación orientada a eventos".
- Describirá el propósito del archivo de proyecto.
- Listará los tipos de archivo que pueden ser incluidos en un proyecto.

¿Qué es Visual Basic?

Microsoft Visual Basic es la más rápida y fácil forma de crear poderosas y atractivas aplicaciones que explotan la interfaz gráfica de Windows.

Visual Basic ofrece una excelente plataforma para desarrollar aplicaciones de manera rápida. Utilizando Visual Basic se pueden crear soluciones a las necesidades de negocios, desde los más simples hasta los complejos. Algunos de los beneficios que esta plataforma de desarrollo incluye son:

Rápida edición, verificación y depuración: Incluye un ambiente de desarrollo que incluye excelentes herramientas de depuración o entonación de las aplicaciones.

Está basado en el lenguaje BASIC: Visual Basic es un ambiente de desarrollo basado en el lenguaje BASIC, por lo que el tiempo requerido par aprender a programar en él es mínimo ya que BASIC es muy simple.

Es una herramienta de desarrollo de alta productividad: Ya que es tan rápido y fácil crear aplicaciones en Visual Basic esto permite ahorrar tiempo de desarrollo. Se puede rápidamente crear una interfaz o prototipo para la aplicación y escribir el código correspondiente que responda a los eventos o acciones que deben realizar cada uno de los elementos de la interfaz.

Utiliza un lenguaje de programación común a las aplicaciones Microsoft Office: Eventualmente, todas las aplicaciones Office soportan Visual Basic for Applications, que es el mismo lenguaje utilizado en Visual Basic. Con un lenguaje compartido se puede compartir y reutilizar código escrito en diferentes aplicaciones o entre aplicaciones.

Soporta Programación OLE: Se pueden utilizar herramientas u objetos creados en otras aplicaciones. A través de Object Linking and Embedded que es una característica de las aplicaciones Windows.

Soporta múltiples plataformas: Utilizando visual Basic se pueden escribir aplicaciones a 16 bits para Windows 3.x o aplicaciones a 32 bits para Windows 95, 98, Windows NT y Windows 2000.
Visual Basic se puede adquirir en diferentes ediciones y las diferencias entre éstas depende de las herramientas que contienen.

Edición Estándar:

- Ambiente de desarrollo de Visual Basic
- Ejemplos
- Asistente de Instalación.
- Kit de Instalación
- Iconos
- Archivos de ayuda

Edición Profesional:

- Controles adicionales y ayuda
- Metafiles y archivos bitmap
- Compilador de ayuda para Microsoft Windows
- Crystal Reports
- Libros de ayuda en línea
- Referencia en línea de Bibliotecas API de Windows a 32 bits y su declaración en Visual Basic
- Archivos necesarios para crear controles personalizados
- Tanto la versión estándar como la profesional incluyen el Data Control, herramienta de acceso a bases de datos, sin embargo sólo con la versión profesional tenemos acceso a bases de datos que cumplen con el estándar ODBC (Open DataBase Connectivity)
- Data Access Objects (DAO) Objetos de acceso a Bases de Datos
- Otras herramientas (Editor de imágenes, Compilador de recursos, etc.)

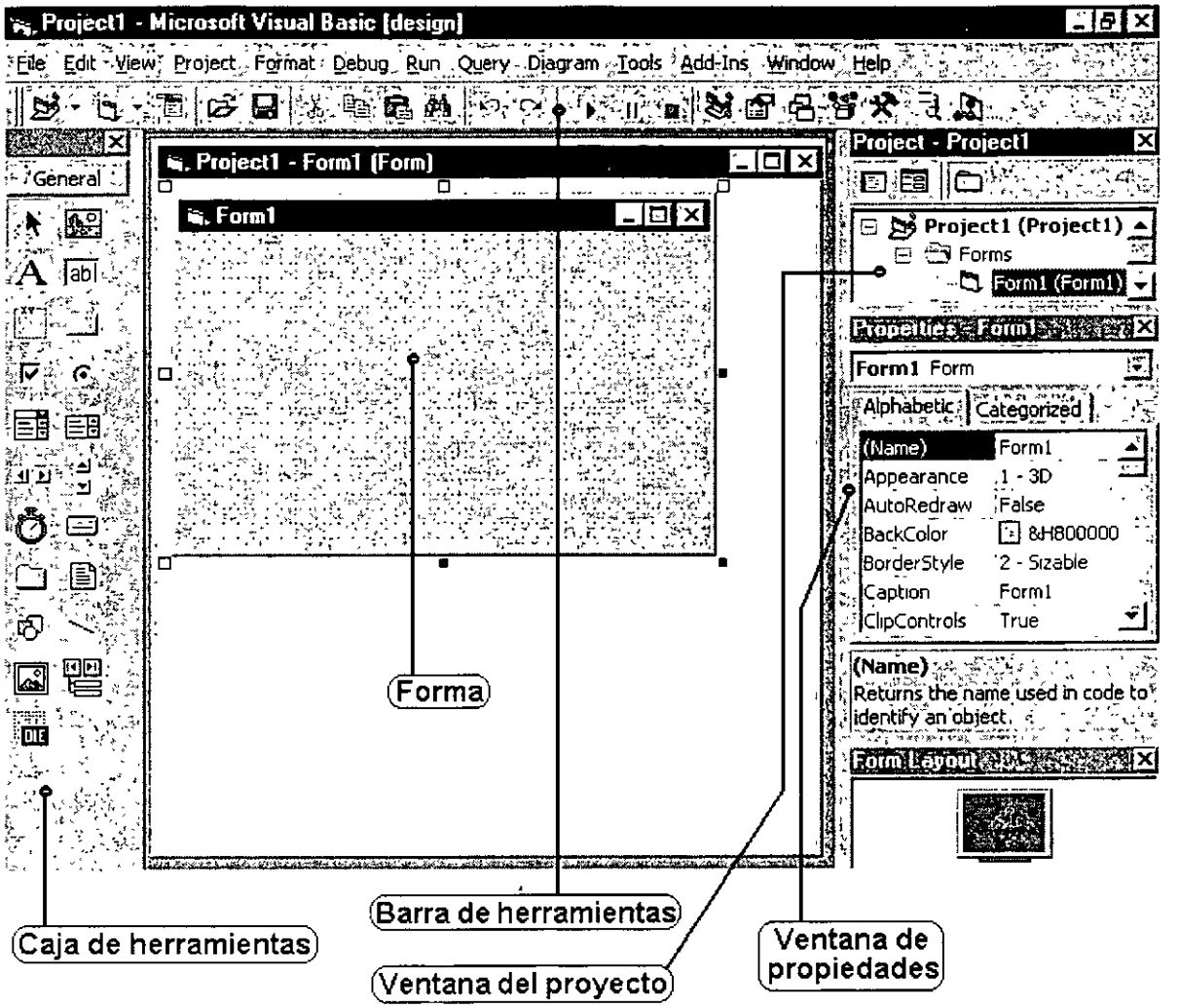
Edición Enterprise:

Esta versión incluye todo lo de la versión profesional más:

- Código Fuente de los controles
- Herramientas de acceso remoto a bases de datos

Y algunas herramientas más.

El ambiente de desarrollo de visual Basic



El ambiente de desarrollo de Visual Basic consiste de los siguientes elementos:

Caja de herramientas: Contiene todos los objetos y controles que pueden ser parte de la aplicación que se está construyendo. Se pueden agregar controles a la caja de herramientas utilizando el comando *Components* del menú *Project*.

Forma: La forma sirve como la ventana que se puede personalizar para la creación de la interfaz de la aplicación.

Ventana de proyecto: La ventana de proyecto lista los archivos de código así como los de recursos que forman el proyecto actual. Un proyecto, es la colección de archivos que se utilizan para crear una aplicación.

Ventana de propiedades: La ventana de propiedades lista las propiedades que se dan o se pueden dar al objeto que se encuentra seleccionado. Una propiedad describe un valor de un objeto, por ejemplo, su tamaño, su color, el título que despliega, etc.

Barra de herramientas: Incluye los comandos más comúnmente ejecutados, de tal forma que podemos ejecutarlos desde aquí o bien desde el menú en donde están incluidos.

Términos Visual Basic

Tiempo de diseño

Es definido como el tiempo en el cuál se está construyendo la aplicación en el ambiente de Visual Basic. Algunas propiedades de los objetos pueden ser modificadas únicamente en tiempo de diseño a través de la ventana de propiedades.

Tiempo de ejecución

Se le define al tiempo en el cuál se ejecuta la aplicación y en la cuál se interactúa con la interfaz como lo hará el usuario final; alguna propiedades de los objetos pueden modificarse en tiempo de ejecución a través del código asociado a ellos.

Formas

Las formas sirven como la ventana que se modificará para presentar la interfaz de la aplicación, es el elemento básico para el desarrollo de toda aplicación; a una forma se le puede agregar cualquier control, gráfico o todo aquello que se requiera para que la aplicación luzca como se requiere. Una aplicación puede estar basada en una sola forma o bien puede ser una combinación de ellas.

Controles

Son representaciones gráficas de objetos como botones de comando, cajas de texto, cajas de listas, etc., todo aquello que el usuario manipula para proveer de información a la aplicación.

Objetos

Término general utilizado para hacer referencia a todas las formas y controles que crean un programa. Un objeto, es un elemento de programación, una abstracción que concebido permite realizar aplicaciones a través de darle valor a sus propiedades y de indicarle cuál de sus métodos ejecutar.

Propiedades

Valores que se le asocian a un objeto como tamaño, color, etc.

Métodos

Acciones que un objeto puede realizar o ha realizado.

Eventos

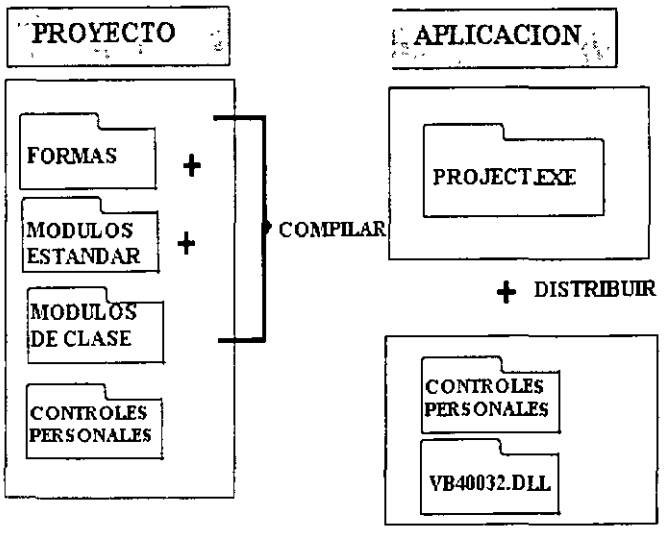
Son las acciones reconocidas por la formas o los controles. Los eventos ocurren cuando el usuario actúa sobre un objeto de un programa; por ejemplo: dando clic sobre algún control, escribiendo o moviendo el ratón.

Programación orientada a eventos

Cuando un programa es orientado a eventos, se escribe código que se ejecuta en respuesta a un evento invocado por el usuario, esto difiere de la programación "procedural" en que ésta inicia en una línea de código indicando la ruta que debe seguir el programa para continuar ejecutándose, llamando procedimientos conforme se vayan requiriendo; mientras que la programación orientada a eventos esta basada esencialmente en una interfaz gráfica, la cual es manipulada por el usuario a través del ratón, y de los controles asociados a dicha interfaz y el código escrito responde a estos eventos generados por el usuario.

Proyectos y archivos ejecutables en Visual Basic

Un archivo de proyecto consta de una lista de archivos que representan formas, módulos de clases y controles personales utilizados durante el desarrollo de la aplicación y las opciones de configuración del ambiente de desarrollo en el momento de realizar la aplicación. Estos archivos son "cargados" cuando un archivo *Project.VBP* es abierto, ya que esta información se almacena en él, y en donde *Project* es el nombre escogido para el proyecto a desarrollar.



Los archivos de proyecto no contienen físicamente a los demás archivos mencionados, sólo contiene la lista de ellos, por ello un archivo de forma o de controles puede ser empleado en

múltiples proyectos. El archivo *Project.VBP* es un archivo de texto y puede ser visto desde cualquier editor de texto.

Un proyecto está conformado por los siguientes archivos:

Tipo de archivo	Extensión	Descripción
Archivos de formas	.FRM .FRX	Este archivo contiene la forma y todos los objetos en la forma más el código asociado a los eventos generados sobre la forma.
Módulos Estándar de Visual Basic	.BAS	Estos módulos contienen procedimientos del tipo **Sub** y **Function** que pueden ser llamados por cualquier forma u objeto dentro de la forma. Estos módulos son opcionales.
Controles personales	.OCX .VBX	Estos controles son proporcionados por Microsoft en adición a los controles estándar o bien son controles desarrollados por terceros, éstos son agregados al proyecto a través de la opción *Components* del menú *Project* y aparecerán en la caja de herramientas del proyecto.
Módulos de Clase Visual Basic	.CLS	Estos módulos contienen la definición de una clase, sus métodos y propiedades. Los módulos de clase son opcionales.
Archivos Fuente	.RES	Estos archivos contienen información binaria utilizada por la aplicación. Estos archivos son comúnmente usados cuando se crean programas desde múltiples lenguajes.

Construyendo un archivo ejecutable (.EXE)

Una vez concluido el proyecto, utilizando el comando Make EXE File del menú de File se crea el archivo .EXE para distribuir la aplicación. El archivo ejecutable sólo contiene las formas y los módulos de código asociados al proyecto. Para poder ejecutar dicho archivo se deben proporcionar otros archivos como son los archivos de controles personales y la biblioteca de ligado dinámico correspondiente a las funciones principales de las aplicaciones.

Pasos para crear una aplicación Visual Basic

1- Crear la interfaz de usuario.

Se crea la interfaz de usuario dibujando sobre las formas los controles y objetos. Para facilitar la lectura del código se recomienda identificar (nombrar) a los objetos de una forma estándar; más adelante dentro de estas notas se definirán dichos estándares.

2- Fijar las propiedades de los objetos de la interfaz.

Después de agregar los objetos a la forma, se le deben determinar sus propiedades; aquellas que así lo requieran ya que se debe recordar que éstas también pueden ser dadas en tiempo de ejecución.

3- Escribir el código para los eventos.

Después de ser determinadas las propiedades de inicialización de los objetos contenidos en la forma, se agrega el código correspondiente a la respuesta de los eventos que se generen sobre ellos. Los eventos se "disparan" cuando una acción sucede sobre los controles u objetos; por ejemplo: un clic sobre un botón, o un doble clic, el movimiento del ratón, etc..

4- Guardar el proyecto.

Cuando se crea un proyecto se debe asegurar el darle un nombre desde la opción *Save Project As* del menú de File, además de recordar guardar constantemente el proyecto para que se almacenen los correspondientes cambios que se realicen al mismo. Al guardar el proyecto se pedirá que se guarden todos los archivos de formas y de módulos que se hayan creado para dicho proyecto; es recomendable crear una carpeta especialmente para cada proyecto, de tal forma que se pueda percibir inmediatamente qué archivos corresponden a qué proyecto.

5- Hacer pruebas y depurar la aplicación.

Conforme se agrega código a la aplicación se puede probar utilizando la herramienta *Run* de la barra de herramientas, para ver el comportamiento de la aplicación. En caso de que no sea el adecuado, Visual Basic cuenta también con la herramienta *Debug* que permite localizar los errores de código generados por problemas lógicos, ya que los errores sintácticos se deben resolver desde el tiempo de escribir el correspondiente código

6- Hacer el archivo ejecutable.

Después de que se terminó el proyecto se puede crear un archivo ejecutable de la aplicación con el comando *Make File EXE* del menú *File*, para hacer a la aplicación independiente del ambiente de Visual Basic.

7- Crear la aplicación de instalación.

Ya que un archivo ejecutable de alguna aplicación creada bajo Visual Basic no debe depender del ambiente de éste. se deben incluir en los archivos de distribución los correspondientes archivos de controles personales .OCX y la biblioteca de Visual Basic vbxxx.dll. Al crear con el *SetupWizard* la aplicación de instalación, estos archivos son incluidos e instalados en las rutas adecuadas para que la aplicación se ejecute sin problemas.

Laboratorio 1 Introducción a Visual Basic

TEMA II

Creando un programa en Visual Basic

TEMA II Creando un programa en Visual Basic

Objetivos:

- Al final del capítulo el alumno será capaz de crear una aplicación simple en Visual Basic.
- Definirá y proporcionará ejemplos de los siguientes términos: objeto, propiedad, método y evento.
- Usará el *Object Browser* para ver las propiedades y métodos asociados a un control.
- Describirá algunas de las propiedades y métodos de las formas.
- Definirá propiedades para el botón de comando, la caja de texto y las etiquetas.
- Podrá utilizar el enunciado *With...End With* para definir valores de múltiples propiedades para un mismo control.
- Demostrará el uso de los eventos asignando código a un control para responder al evento de clic.

Fundamentos de Visual Basic

Objetos

En Visual Basic un objeto es cualquier cosa que se puede controlar.

Cada forma dentro de una aplicación: Cada aplicación inicia con una forma. Se pueden manipular las propiedades y los eventos asociados a la forma, se le pueden agregar controles y otros objetos a la forma.

Controles en una forma: Los controles proveen de una interfaz para que el usuario pueda trabajar con la aplicación. Los controles más comunes son: las cajas de texto, las etiquetas, los botones de comando, los botones de opción, los *frames* o cuadros, las cajas de lista, y las cajas combo.

Clases de Objetos

Una *clase* es la definición formal de un objeto. La clase actúa como plantilla para cada una de las instancias de un objeto creado. La clase define las propiedades del objeto y los métodos que controlan el comportamiento del objeto. Las herramientas en la caja de herramientas representan a las clases de los controles. Cada vez que se agrega un control se crea una instancia de dicho control, es decir de la clase que representa dicho control.

Controlando los Objetos a través de sus propiedades y sus métodos

Para entender la relación entre los objetos, las propiedades y los métodos, pongamos el siguiente ejemplo de una clase: La clase polígono, es una plantilla para objetos que representan figuras geométricas de N número de lados, de allí el nombre de polígono; de hecho el número de lados es una propiedad para un objeto polígono, por ejemplo un triángulo tiene 3 lados, un cuadrado 4, un octágono 8, etc.. Para ser representado en un programa, un objeto polígono puede tener un color de contorno y un valor que determine si será rellena la figura o no; estos valores son también propiedades del objeto. En cuanto a sus métodos, un polígono se puede dibujar, rotar, transladar, obtener su área, su perímetro, etc., los métodos son las acciones que puede realizar el objeto.

Convenciones para nombrar a los objetos

Nombrando a los objetos en tiempo de diseño

Todo objeto que es creado en Visual Basic tiene un nombre por omisión, por ejemplo Form1, Command2, Text3, etc. Para hacer más sencilla la lectura del código, se puede cambiar la propiedad Name de los objetos en tiempo de diseño, dándoles nombres basados en el tipo de objeto que representan.

El nombre es una propiedad que sólo se debe fijar en tiempo de diseño, ya que en tiempo de ejecución se tiene que hacer referencia al objeto a través de su nombre.

La convención de nombres propone un prefijo al nombre de los objetos que indique qué tipo de objeto es.

Objeto	Prefijo	Ejemplo
Forma(form)	frm	frmPrincipal
Caja de verificación(check box)	chk	chkSoloLectura
Caja Combo(combo box)	cbo	cboProductos
Botón de Comando(Command button)	cmd	cmdSalir
Datos(Data)	dat	datCategorias
Caja de lista de diractorio(Directory list box)	dir	dirOrigen
Caja de lista de archivos(file list box)	fil	filDestino
Caja de listado de unidades(Drive list box)	drv	drvUnidad
Cuadro(Frame)	fra	fraOpciones
Red(Grid)	grd	grdPrecios
Barra de desplazamiento horizontal(horizontal Scroll bar)	hsb	hsbVolumen
Imagen(Image)	img	imgIcono
Etiqueta(Label)	lbl	lblPrecio
Linea(line)	lin	linVertical
Caja de lista(list box)	lst	lstCodigos
Menú	mnu	mnuArchivo

Botón de Opción(Option button)	opt	optSoltero
Fotografía(Picture)	pic	picEspacio
Caja de texto (Text box)	txt	txtNombre
Timer(Reloj)	tmr	tmrAlarma
Barra de desplazamiento vertical(Vertical scroll bar)	vsb	vsbRango

Propiedades

Las propiedades como Text, Caption, BorderStyle, etc., cambian la apariencia y el comportamiento de los objetos.

Dando valor a las propiedades en tiempo de diseño

Para dar valor a las propiedades en tiempo de diseño es necesario utilizar la ventana de propiedades. Una propiedad que obtiene su valor en tiempo de diseño es considerada un valor inicial para el objeto y este se reflejará cuando se ejecute la aplicación.

Algunas de las propiedades más comúnmente fijadas en tiempo de diseño para todos los objetos son:

BackStyle	Estilo de fondo	**Name**	Nombre del control
BorderStyle	Estilo de borde	**Visible**	Determina si se verá el control
Caption	Título		
Enabled	Habilitar		
FillColor	Color de relleno		

Dando valor a las propiedades en tiempo de ejecución a través de código

Para poder determinar el valor de una propiedad en tiempo de ejecucuón, se debe escribir código que cumpla con la siguiente sintáxis:

Objeto.Propiedad = Expresión

Por ejemplo:

Para hacer que una caja de texto txtMensaje, despiegue la cadena "Hola!!!", la propiedad Text se debe iniciar así:

txtMensaje.Text = "Hola!!!"

Obteniendo el valor de una propiedad en tiempo de ejecución.

En algunas ocaciones es necesario conocer el valor que almacena una propiedad de un objeto en particular, para realizar operaciones o determinar si se realizan ciertas tareas

dentro de la aplicación. Para obtener el valor de la propiedad del objeto, pon la referencia al objeto y la porpiedad del lado derecho de una expresión como la siguiente:
Variable = Objeto.Propiedad

Por ejemplo, para saber el valor de la caja de texto txtNombre que está almacenado en la propiedad Text, se obtiene ésta a través de una variable llamada strUsuario.

StrUsuario = txtNombre.Text

El siguiente ejemplo multiplica el valor contenido en la caja de texto *txtSubtotal* por un porcentaje de impuesto y lo almacena en otra caja de texto llamada *txtTotal*:

txtTotal.Text = txtSubtotal.Text * .082

Métodos

Métodos como **Move** y **Setfocus** causan que el objeto realice una acción o tarea. Existen diferentes formas de llamar a los métodos de los objetos. La sintáxis depende de dos factores:
1. ¿El método espera argumentos?
2. ¿El método regresa algún valor?, de ser así ¿Se puede utilizar ese valor?

En general la sintáxis que los métodos usan es la siguiente:

Objeto.Método [(arg1, arg2, ...)]

Opcionalmente se colocan paréntesis para distinguir los argumentos que se le están pasando y también para almacenar el resultado que arroje un método. Una regla general, es utilizar los paréntesis cuando el método se encuentre del lado derecho de una expresión antecedido por el signo de igual.

El método **SetFocus**, es común a muchos objetos en Visual Basic y es un método que no tiene argumentos. El siguiente código hará que la caja de texto txtNombre sea el control activo, es decir, en donde se encuentre el cursor dentro de la forma.

TxtNombre.SetFocus

El método **Move** hace un cambio de posición de un objeto. El método Move recibe hasta 4 argumentos, left, top, width, heigth. (punto inicial izquierdo, punto inicial superior, ancho y alto). La siguiente instrucción coloca la forma frmInicio en la posicion 0.0.

frm.Inicio.Move 0,0

Instrucción *With...End With*

Cuando se dan valores a las propiedades y se llaman a los métodos para los objetos, se necesitan escribir varias líneas de código para cada Propiedad de un mismo Objeto. Para ello se puede utilizar la instruccón **With ..End With** de tal forma que se forman bloques de código para determinar los valores de las propiedades de un mismo Objeto.
Utlizando la instruccón **With ..End With** el código se hace más sencillo de escribir y de leer y las instrucciones se ejecutan más rápidamente que si estuvieran escritas una por línea de código.

La sintáxis es:

With Objeto

    [Instrucciones]

End With

El nombre del objeto es puesto en la misma línea que el **With**; las líneas que continúan y hasta antes del **End With** son las propiedades con su respectivo valor y no requieren del identidicador del objeto.

Ejemplo:

Propiedades para el objeto txtNombre

```
txtNombre.Font.Bold =  True
txtNombre.Font.Size = 24
txtNombre.Texr = "Hola!!!!"
```

```
With txtNombre
        .Font.Bold = True
        .Font.Size = 24
        .Text= "Hola!!!"
End With
```

Utilizando el **With..End With** se esciben 2 líneas más de código, pero Visual Basic lo ejecuta más rápido ya que evalúa al objeto en una sóla instrucción.

Eventos

Un evento es una acción reconocida por una forma o un control. Los eventos ocurren como resultado de una acción realizada por el usuario o por el código del programa o bien ellos pueden ser invocados por el sistema. Y se puede escribir código para que se ejecute cuando un evento ocurra. Cualquier acción puede ser asociada a un evento en Visual Basic, dando así el control sobre cómo debe responder la aplicación a cada uno de ellos.

Los siguientes son ejemplo de eventos en Visual Basic:

Activate	DragDrop	KeyUp	MouseMove
Change	GotFocus	Load	MouseUp
Click	KeyDown	LostFocus	
DblClick	KeyPress	MouseDown	

Cada objeto posee un conjunto de eventos que puede reconocer, es decir, que los eventos listados no se aplican a todos los objetos. Por ejemplo: una forma reconoce los eventos Load, Unload y Activate.

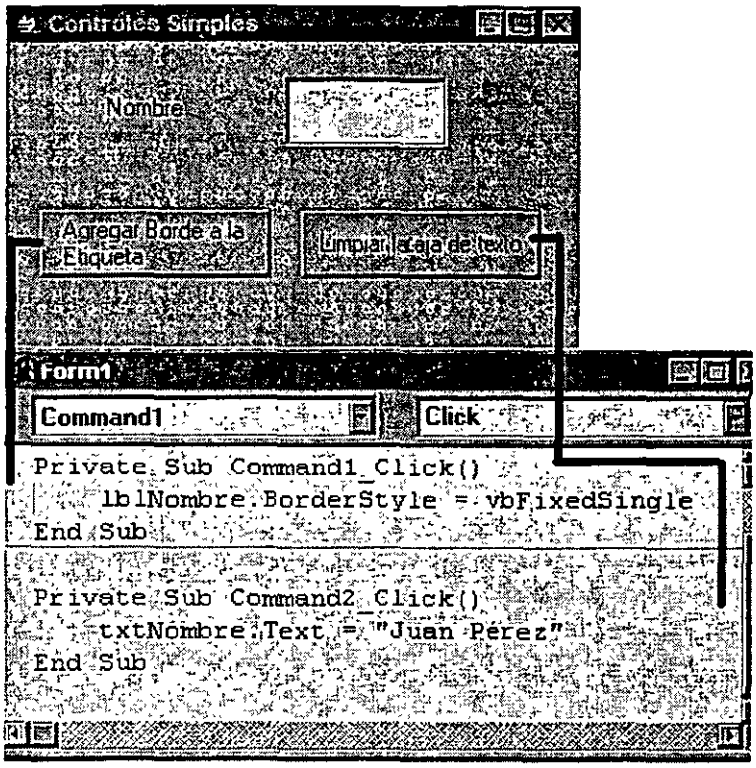
Programación Orientada a Eventos

Programar orientado a eventos significa que el programa debe responder a eventos causados por el teclado, el ratón y el sistema operativo.

Respondiendo a Eventos en Visual Basic
En el siguiente ejemplo, se escribe código para responder al evento Click para cada botón de comando de la forma *Controles Simples*. Cuando el usuario da un clic en el botón *Agregar Borde a la Etiqueta*, un borde es puesto alrededor de la etiqueta *Nombre*.

```
Sub cmdBorde_Click()
        lblNombre.BorderStyle = vbFixedSingle
End Sub
```

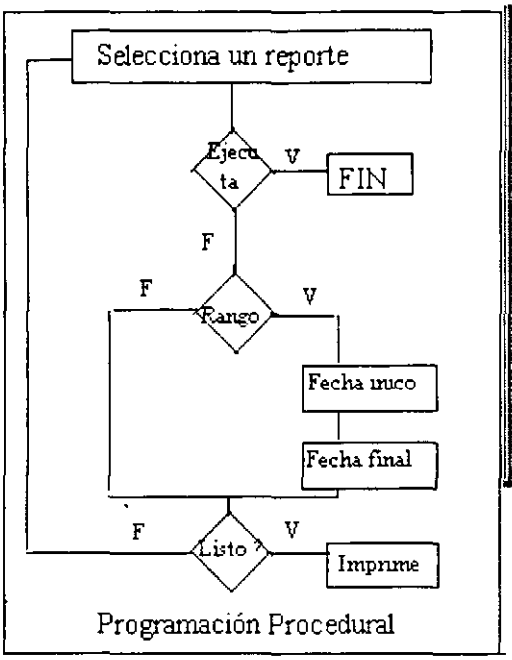
El evento Click para el botón de comando cmdBorde está asociado al procedimiento llamado cmdBorde_Click. El nombre del procedimiento del evento se define con el nombre del control seguido por un guión bajo (subguión) y el evento al que debe responder.



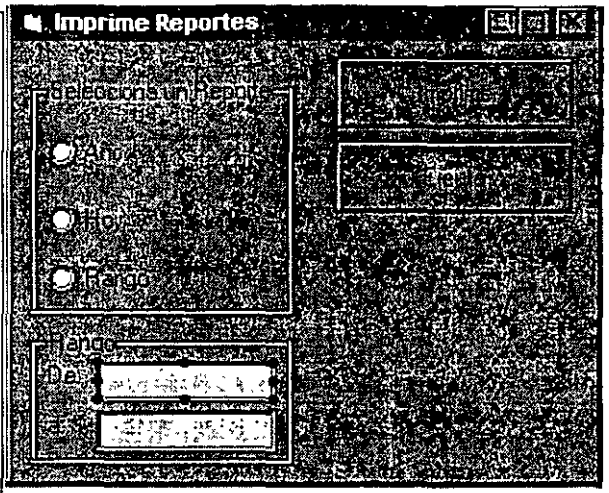
Si se da doble clic sobre un control en tiempo de diseño, Visual Basic genera un procedimiento vacío asociado al evento Click de dicho control.
En el caso del código asociado al evento clic del botón Command2, pondrá como valor por omisión en la caja de texto el nombre "Juan Pérez".

Programación Procedural vs. ProgramaciónOrientada a Eventos

En un programa procedural, el programador controla en forma lineal el camino y preguntando al usuario información en una secuencia específica. En el ejemplo, el programa pregunta por un tipo de reporte. Cuando el tipo de reporte es dado, el programa determina si se require de un rango de fechas; de ser asi, el usuario dará las 2 fechas y así sucesivamete. Un programa procedural típicamente es un largo programa que llama a las subrutinas conforme sea necesario.
En un programa orientado a eventos, se habilita el hecho de que éste responderá al usuario. El usuario controla el flujo del programa dependiendo de las elecciones y la información que proporciona. Un programa orientado a eventos en Visual Basic está creado por muchos y pequeños bloques de código (procedimientos) que realizan una tarea específica. Muchos de estos procedimientos son procedimientos de eventos que se invocan cuando el usuario hace algo.
Los programas orientados a eventos son más sencillos de escribir y de darles mantenimiento que un programa escrito en forma procedural.



Programación Procedural



En la programación orientada a eventos, se tiene una forma a través de la cual el usuario da información en cualquier orden Las acciones del usuario invocan pequeños bloques de código

Catálogo de objetos (Object Browser)

Durante el desarrollo del código se puede hacer referencia al Object Browser para ver la lista de todos los objetos disponibles dentro del proyecto, o bien de otras bibliotecas.

El object Browser cumple con varias funciones:

a) Brinda una forma rápida de navegar dentro del código del proyecto
b) Muestra todos los objetos contenidos en un proyecto, así como su código, procedimientos, propiedades, etc.
c) Da la oportunidad de incluir plantillas de otros módulos dentro del proyecto de manera que sea más sencillo en generar nuevo código; sobre todo cuando haecmos referencia a funciones preestablecidas en Visual Basic, es decir, que nosotros no escribimos, de esta manera también podemos evitar errores a la hora de escribirlos.
d) Nos permite un rápido acceso a la ayuda.

Para mostrar el Object Browser, elige la opción Object Browser del menú View o presiona F2.

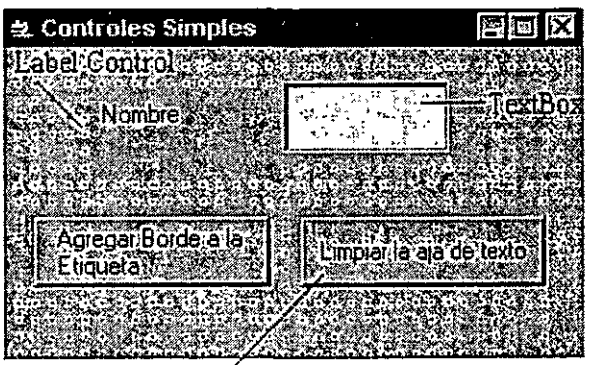
Formas y controles básicos

Las formas son unos de los principales objetos para la construcción de una aplicación Visual Basic. Las usuarios interactúan con los controles dentro de una forma para obtener los resultados que desean; la forma como todo lo que se manipula en Visual Basic es un objeto; por lo tanto tiene propiedades, métodos y eventos asociados a las principales

acciones que pueden sucederle a una forma en una aplicación dependiendo de la secuencia que el usuario le dé a ésta.

Propiedades	Métodos	Eventos
ActiveControl	Hide	Activate
ActiveForm	Move	Click
BackColor	Print	DblClick
BorderStyle	PrintForm	Deactivate
Caption	Refresh	DragDrop
ControlBox	SetFocus	DragOver
Enabled	Show	GotFocus
Heigth		Load
Left		LostFocus
MaxButton		MouseDown
MinButton		MouseMove
Name		MouseUp
Top		Unload
Width		
WindowState		
Moveable (true,false)		

Controles Básicos



Con los controles el usuario puede operar y obtener resultados de la aplicación.Para agregar controles a la aplicación se selecionan de la Caja de Herramientas (ToolBox).

Una forma sencilla de agregar un control a una forma es dando doble clic sobre el control en la caja de herramientas; esto hará que el control aparezca en el centro de la forma con un tamaño estándar; una vez ahí se puede mover a la posición deseada.

Los controles más comunes y simples son:
Labels (Etiquetas)
Text Boxes (cajas de Texto)
Command Buttons (Botones de comando)

Etiquetas

Las etiquetas son la forma más común de mostrar texto que no cambia. Un ejemplo de una etiqueta es incluir el título debajo de un gráfico, o títulos en general, regularmente se utilizan para incluir instrucciones dentro de las formas.

Se puede cambiar la propiedad Caption del control Label para mostrar el estado de un programa. Esto es especialmente útil cuando se está haciendo una depuración o afinación de la aplicación.

Las propiedades más frecuentemente usadas para las etiquetas son:

Alignment	Alineación del texto a desplegar
Caption	Texto a deplegar
Name	Nombre del control
AutoSize	Se hace grande o pequeña dependiendo del texto que se escriba
Font	Tipo de letra del texto

Cajas de texto

Un control Caja de texto (TextBox) es utilizado para obtener información del usuario o para mostrar información generada por la aplicación. Hay que tomar en cuenta que la información que se muestra en una caja de texto puede ser cambiada por el usuario, a diferencia de las etiquetas. A través de cajas de texto y del control **Data** se puede desplegar información contenida en una base de datos.

Las propiedades, métodos y eventos asociados a las cajas de texto más comúnmente utilizadas son:

Propiedades	Métodos	Eventos
Alignment	Refresh	Change
BackColor	SetFocus	DragDrop
BorderStyle		DragOver
Enabled		GotFocus
Font		KeyDown
ForeColor		KeyPress
MaxLength		KeyUp
MultiLine		LostFocus
Name		
PasswordChar		
ScrollBars		
Text		
Visible		
locked		

Botones de comando

El botón de comando (Command Button) ejecuta una acción cuando el usuario da un clic sobre él. El más común de los eventos asociados a este control es el evento Click.

Algunas de sus propiedades, métodos y eventos son:

Propiedades	Métodos	Eventos
Cancel	SetFocus	Click
Caption		MouseDown
Default		MouseMove
Font		MouseUp
Name		
Visible		

Forma de Acceso a los controles dentro de una forma

Para hacer una aplicación que pueda ser usada por cualquiera, se deben respetar las formas de acceso a los controles y funciones de la aplicación de cualquier programa Windows. Por ejemplo, el usuario debe poder desplazarse entre los controles a través de la tecla TAB, es decir cambiar el punto de inserción de posición; a esta acción se le conoce como tomar el foco. Si un control tiene el foco, significa que el punto de inserción está sobre él. O bién tambíen se puede obtener el foco dando ALT y una tecla de acceso, como en el caso de los menúes Windows.

Designando el orden de tabulación de los controles

El orden de tabulación, es la secuencia que sigue el apuntador del mouse al dar un tabulador. En Visual Basic, se fija un orden por omisión y depende del orden en que se fueron agregando los controles dentro de la forma. Para cambiar éste orden se debe utilizar la propiedad **TabIndex** de los controles.

La propiedad **TabIndex** debe ser asignada en tiempo de diseño. Al definir esta propiedad para un control, automáticamente se redefine para los otros controles. El valor inicial siempre es 0.

Designando teclas de acceso para un control

Una tecla de acceso es designada al escribir un ampersand (&), antes de la letra de la tecla de acceso en la propiedad Caption del control. Por ejemplo, para hacer de la letra "N" una tecla de acceso para el control que tiene como propiedad Caption "Nombre", dicha propiedad puede ser modificada por "&Nombre"; de esta forma, en tiempo de ejecución el título o Caption aparecerá con la letra N y un subguión bajo ella. Así cuando el usuario dé

la combinación de teclas ALT y la letra designada, el cursor se quedará en el control asignado.

Algunos controles como la Cajas de texto no tienen propiedad Caption para acceder a ellos a través de una combinación de teclas; por ello, se puede hacer lo siguiente, aprovechando que las etiquetas no tienen propiedad TabStop, es decir, el cursor no puede detenerse en ellas:

1. Coloca una etiqueta en la forma, del lado izquierdo al control *Caja de texto* que deseamos acceder.
2. Asigna la propiedad *Caption* de la etiqueta con la tecla de acceso que deseas para el control *Caja de texto*.
3. Asigna la propiedad *TabIndex* en un valor inmediatamente inferior al del control caja de texto.

Así, cuando el usuario dé la combinación de teclas ALT y la tecla designada, y ya que las etiquetas no pueden tomar el foco, éste se quedará en el control inmediato, en este caso en la caja de texto siguiente.

TEMA III

Trabajando con Formas

TEMA III Trabajando con Formas

Objetivos:

Al final del tema el alumno será capaz de:

- Utilizar las herramientas de edición dentro de la ventana de código para escribir de manera organizada y poseer un código bien documentado.
- Distinguir entre las instrucciones Load/Unload y Show/Hide
- Asignar la forma de inicio de una aplicación.

Trabajando con la Ventana de Código

Código es un término general para hacer referencia a cualquier instrucción que se escribe para una aplicación dentro de Visual Basic, por ejemplo, procedimientos de evento, procedimientos generales.

El código en Visual Basic es escrito en la Ventana de Código. El editor de texto es un editor ASCII con colores para diferenciar entre palabras reservadas y el código que escribimos nosotros.

Módulos de Código Visual Basic

Existen 3 tipos de módulos de código: de forma, estándar y clases. Cada uno de los cuales puede contener instrucciones en general.

Módulos de forma

Cada forma dentro de una aplicación tiene asociado un módulo de forma. Dichos módulos son guardados con una extensión *FRM* y contienen:

a) Los valores asignados a los controles dentro de la forma
b) La declaración de las variables que actúan dentro de la forma
c) Procedimientos de evento asociados a los controles, así como procedimientos generales para toda la forma.

La descripción gráfica de una forma y de los controles es almacenada en formato binario en un archivo con extensión *FRX*.

Algunas aplicaciones sólo contienen módulos de forma; sin embargo, entre más grande es la aplicación se vuelve más compleja, ya que determinado código puede ser común para múltiples formas dentro de la misma aplicación. Este código llamado procedimiento general, es almacenado en un módulo de

forma y es difícil de darle mantenimiento y de ser ejecutado por otros proyectos que realicen los mismos procedimientos. Esto se soluciona haciendo módulos estándar.

Módulos estándar

Los módulos estándar pueden contener código que sea común a muchas formas dentro de una aplicación. Este código es público por omisión, lo cual significa que es muy sencillo de compartir con otros módulos de código, como código de formas. Los módulos estándar pueden contener:

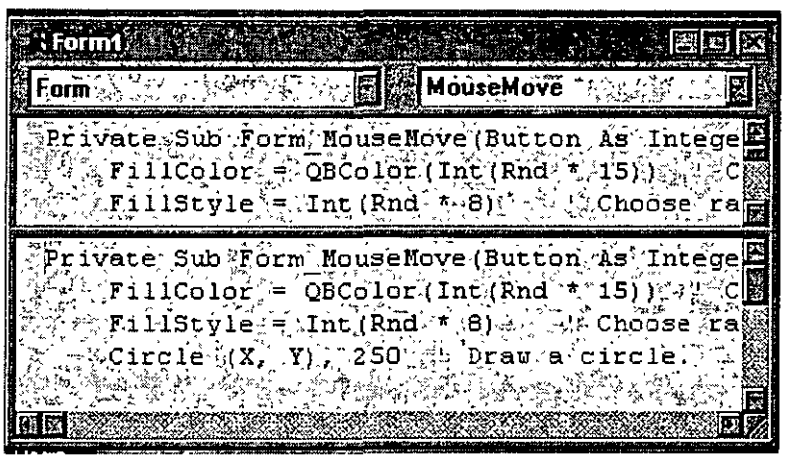
Procedimientos, declaraciones y tipos de variables.
Los módulos estándar no pueden contener procedimientos de eventos ya que no contienen objetos.

Módulos de clase

El tercer tipo de módulo es el módulo de clase. Un módulo de clase puede contener:

Definición de clases (propiedades y métodos)

La ventana de código



La ventana de código es utilizada para escribir, mostrar y editar código de Visual Basic. Se puede abrir la ventana de código para cada módulo dentro de la aplicación; de esta forma se podrá ver código de módulos estándar, de módulos de clases y copiar y pegar líneas entre ellos.

La ventana de código incluye:

La caja de objetos (Object Box) Despliega el nombre del objeto seleccionado. Dando un clic en la flecha que aparece del lado derecho de la caja de objeto se podrá ver la lista de todos los objetos asociados a la forma en la que actualmente estamos trabajando.

La caja de procedimientos (Procedure Box) Lista todos los eventos reconocidos por Visual Basic para la forma o cada uno de los controles desplegados en la caja de objetos. Cuando se selecciona un evento, el procedimiento asociado a ese evento es listado en la ventana de código.

Nota: Todos los procedimientos en un módulo son desplegados en la misma ventana de código.

La barra de separación (Split Bar) Está localizada en la parte baja de la barra de título en la parte superior de la barra de desplazamiento vertical. Dando clic con el ratón y desplazando éste hacia abajo, la ventana de código se podrá dividir en 2 contenedores o paneles, cada uno de los cuales son independientes. La información que aparece en la caja de objeto y en la caja de procedimiento es válida para el panel que se encuentra activo. Para regresar a un sólo panel hay que recorrer la barra de separación a su posición original.

Vista total (Full Module View) Por omisión aparece un procedimiento a la vez en la ventana de código. Se puede cambiar esta forma seleccionando el modo de Vista total, de tal manera que todos los procedimientos para un módulo aparezcan en la ventana de código, con un a línea de separación entre cáda uno de ellos.

Para cambiar al modo Vista Total
1. Desde el menú *Tools*, elige *Options*
2. Selecciona la pestaña de *Editor*
3. Activa la opción *Default to Full Module View*

Trabajando dentro del código

El código crece conforme crece la aplicación, por lo que desplazarse dentro de él se vuelve más complejo. Visual Basic incluye varias herramientas que permiten una mejor búsqueda o navegación dentro del código.

Find el comando Find del menú Edit puede buscar por un texto específico . Para buscar se puede seleccionar que se haga dentro de la sección actual, dentro de un determinado procedimiento o de un módulo o dentro de todo el proyecto.

Doble clic en el objeto Dando doble clic sobre un objeto seleccionado dentro de la forma permite ver el procedimiento de evento asociado con ese objeto.

Clic con el botón derecho del ratón Dando clic con el botón derecho del ratón en una forma o control permite elegir la opción de View Code de un menú colgante.

Ventana de código Utilizando las cajas de Objeto y procedimiento de la ventana de código para mostrar el procedimiento deseado.

Object Browser El Object Browser muestra los procedimientos disponibles para cualquier objeto activo o seleccionado.

Editando Código

Par hacer más fácil de entender y de escribir código Visual Basic recomienda lo siguiente:

Dejar sangrías para identificar las diferentes partes del código, como los ciclos iterativos, las instrucciones condicionales, etc. Para ello se recomienda usar tabuladores o el comando Indent del menú Edit, de la misma forma para regresar a la línea anterior al tabulador se puede utilizar la combinación de teclas SHIFT-TAB o bien el comando Outdent del menú Edit.

Ejemplo:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer,_
        X As Single, Y As Single)
   FillColor = QBColor(Int(Rnd * 15))              ' Choose random FillColor.
   FillStyle = Int(Rnd * 8)                        ' Choose random FillStyle.
   Circle (X, Y), 250                              ' Draw a circle.
End Sub
```

Utilizar el carácter de continuación de línea Para evitar hacer líneas demasiado larga y que no se puedan ver de manera fácil dentro de la ventana de código, se recomienda truncar las línea o continuarla en una línea extra debajo de ésta. Para identificar se trata de una línea de continuación o cuando aún no se concluye la escritura de una instrucción se utiliza el carácter de continuación de línea que es un subguión o guión bajo (_)

Ejemplo:

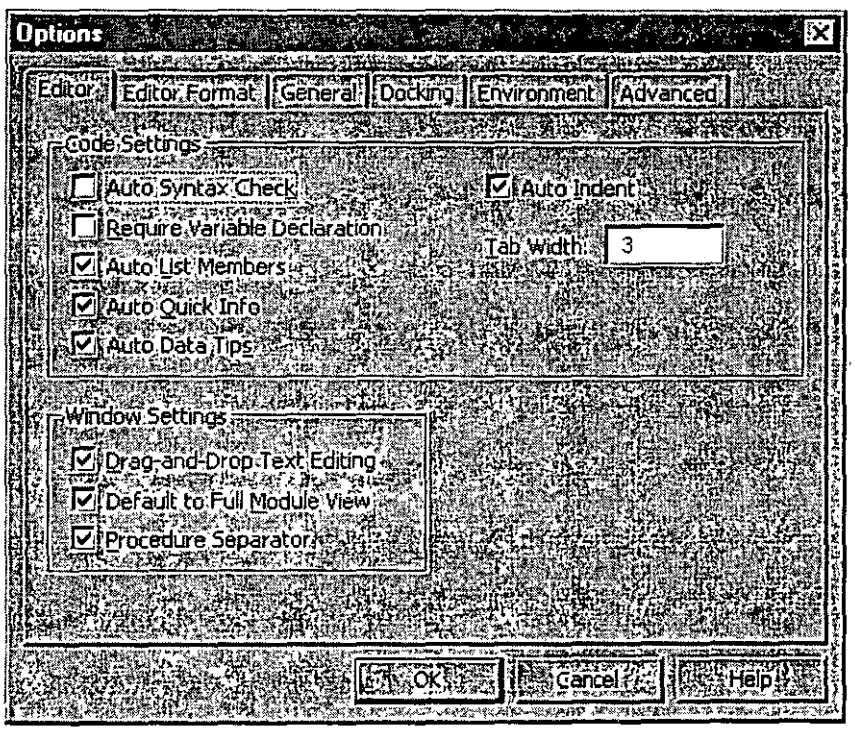
```
MsgBox prompt:= "La contraseña es inválida!", _
        buttons:=49, _
        title:= " Validación", -
        helpfile:= "Validación.hlp", _
        context= 3
```

Comentar las líneas de código Se puede agregar documentación a la aplicación a través de colocar comentarios en las partes de código que realicen alguna función específica, de esta manera será más fácil entender que realizar cada procedimiento o función que se tenga dentro de la aplicación, para colocar comentarios basta con iniciar éste con una comilla simple (') y a partir de allí todo será tomado como comentario, hasta el final de la línea, de hecho Visual Basic pone los comentarios de otro color , por omisión es de color verde, así como utiliza otros colores para identificar palabras reservadas, etc.

Ejemplo:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer,_
        X As Single, Y As Single)
   FillColor = QBColor(Int(Rnd * 15))          ' Choose random FillColor.
   FillStyle = Int(Rnd * 8)                    ' Choose random FillStyle.
   Circle (X, Y), 250                          ' Draw a circle.
End Sub
```

Opciones de ambiente de desarrollo.



Utilizando el comando Options del menú Tools, se pueden cambiar algunos atributos del ambiente de desarrollo de Visual Basic, los cambios realizados sobre la pestaña de **Editor** permiten configurar las ventanas de código y de proyecto:

Auto Syntax Check: Permite que Visual Basic revise línea a línea de código la sintaxis de la misma, desde el momento en que éstas son escritas, evitando así errores en tiempo de compilación.

Requiere Variable Declaration: Determina cuando la declaración de una variable debe ser explícita, es decir, debe ser declarada antes de utilizarse, dentro de un módulo. Seleccionando está opción la instrucción Option Explicít es colocada en la zona de declaraciones generales para cualquier módulo.

Auto List Members: Despliega una caja que despliega información que por lógica podría completar la instrucción en el punto de inserción actual

Auto Quick Info: Despliega información acerca de funciones y sus parámetros.

Auto Data Tips: Despliega el valor de la variable sobre la que se encuentra el cursor.

Auto Indent: Se utiliza está opción par hacer una sangría con respecto a la primera línea de código dentro de un procedimiento.

Tab Width: Fija el ancho del tabulador; el rango es de 1 a 32 espacios; el default es 4 espacios.

Drag-and-Drop Text Editing: Permite que se arrastren elementos entre el código actual y la ventana de código y las otras ventanas.

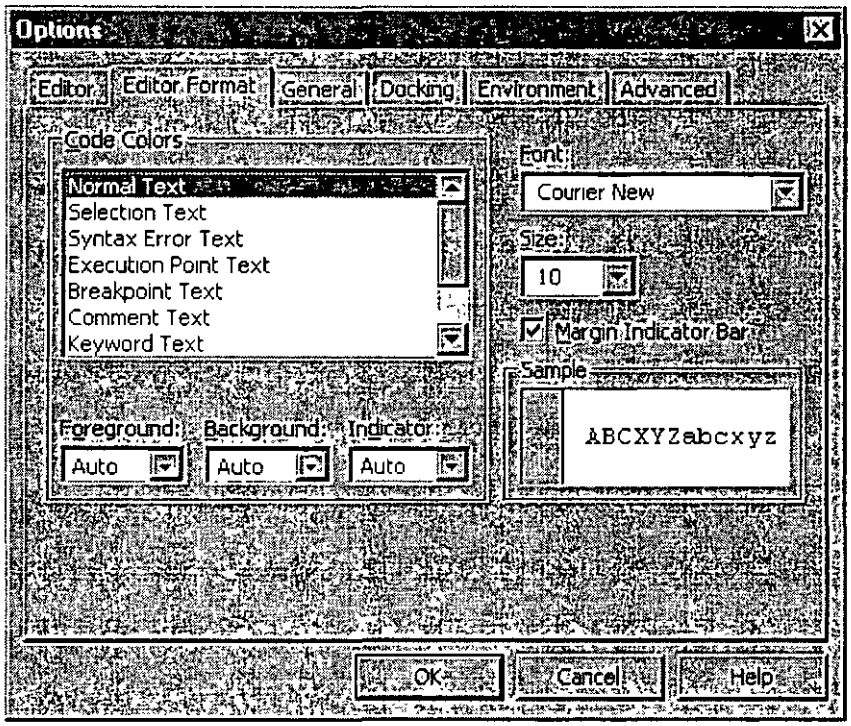
Default to Full Module View : Define el modo en el que se trabajan los códigos de los procedimientos en la ventana de código, ya sea en forma individual o completa.

Procedure Separator : Permite indicar si se desea que aparezca una barra de separación al final de cada procedimiento en la ventana de código. Solamente se permite si el parámetro **Full Module View** está activado.

Nota: Los cambios que se hagan desde ésta caja de diálogo son permanentes, es decir que se tomarán estas opciones para las subsecuentes veces que se inicie Visual Basic.

Opciones del editor

Utilizando la pestaña de **Editor Format** de el comando Options del menú Tools, se puede especificar la apariencia del código que escribirás. Esto incluye un alista de tipos de texto que puedes utilizar, y en las cuales le puedes dar atributos a éstos, una vez que se modifican las opciones del editor serán válidas para todos los proyectos.



Font: Indica el tipo de letra que se utilizará para el código.

Size: Tamaño del tipo de letra.

Code colors: Indica el color que se utilizará par mostrar el código de forma normal y el color para cuando el color este seleccionado.

Sample: Muestra como se vera el texto con el tipo, tamaño y color de letra.

Interactuando con el usuario.

Para interactuar con el usuario se puede: enviar mensajes con la función MsgBox, o bien obtener información de él a través de la función InputBox.

Una de las formas más sencillas de proporcionarle información al usuario es a través de ventanas de mensaje para lo cual utilizamos as función MsgBox y por otro lado para pedirle información lo podemos hacer también a través de una ventana de petición con la función InputBox.

Función MsgBox

Las cajas de mensaje ofrecen una forma rápida y sencilla de enviarle informació al usuario y pedirle que responda ciertas preguntas , o que tome decisiones para determinar la secuencia del programa. Se puee utilizar la función MsgBox para enviar diferentes tipos de mensajes y a través de los cuales le proporciona al usuario diferentes botones para responder a éste.

Sintaxis: **MsgBox** (*prompt* [,*buttons*][,*title*][,*helfile,context*])

Los argumentos que se indiquen a la función determinan la apariencia de la caja de mensaje.

Nota: Todo argumento que se indica dentro de [], es opcional, mientras que si no esta dentro de ellos es necesario.

Argumentos:

Prompt El texto del mensaje para el usuario

Buttons Determina el número, el tipo de botones y el icono que aparecerá en el mensaje. Las opciones de botones y de icono pueden ser dados a través de constantes predefinidas o bien de valores:

Constante	Valor	Descripción
vbOKOnly	0	Despliega un botón de OK
vbOKCancel	1	Despliega un botón de OK y el de Cancel
vbAbortRetryIgnore	2	Despliega los botones Abort, Retry e Ignore
vbYesNoCancel	3	Despliega los botones de Yes,No y Cancel
vbYesNo	4	Despliega los botones de Yes y No
vbRetryCancel	5	Despliega los botones Retry, y Cancel.
VbCritical	16	Despliega el icono de Mensaje Crítico
VbQuestion	32	Despliega un icono de interrogación
VbExclamation	48	Despliega un icono de advertencia
VbInformation	64	Despliega un icono de información
vbDefaultButton1	0	El primer botón está seleccionado por omisión
vbDefaultButton2	256	El segundo botón está seleccionado por omisión
vbDefaultButton3	512	El tercer botón está seleccionado por omisión
vbAplicationModal	0	Cuando una caja de mensaje es modal a la aplicación quiere decir que el usuario debe responder a lo que le solicita ésta para poder continuar con la aplicación. La ventana permanece siempre visible en primer plano, aunque se puede trabajar en otras aplicaciones sin ver esta caja.
VbSystemModal	4096	Cuando es modal al sistema, el usuario tendrá siempre la caja de mensaje, aún dentro de otra aplicación hasta que responda a la caja de mensaje.

El primer grupo de valores (0 - 5) indica el tipo de botones que contendrá la caja de mensaje, el segundo (16, 32, 48, 64) describe el icono que aparecerá, el tercer grupo especifica qué botón estará seleccionado por omisión y el cuarto determina la modalidad de la caja de mensaje; estos valores se pueden sumar para indicar el argumento **buttons** que solicita la función **MsgBox**.

Title el texto que aparecerá en la barra de título de la ventana de mensaje.

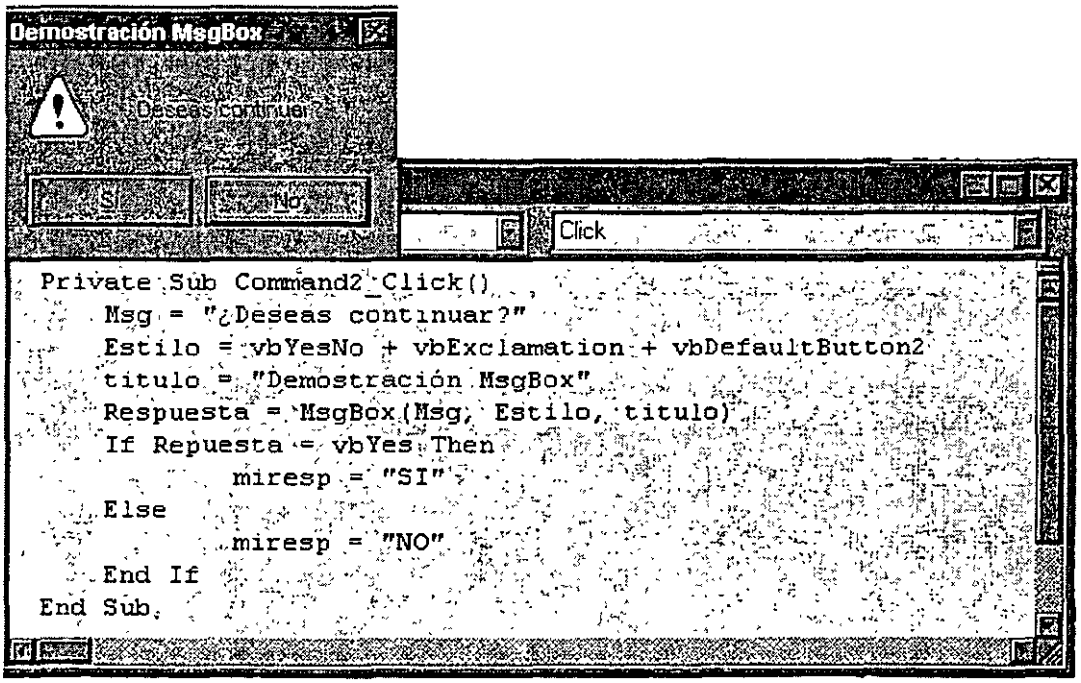
La función **MsgBox** regresa un valor dependiendo del botón que haya elegido el usuario y se puede utilizar ese valor para determinar la secuencia del programa. Cabe aclarar que estos valores se utilizan a través de las constantes asociadas a ellos.

Botón	Valor	
OK	vbOK	(1)
Cancel	vbCancel	(2)
Abort	vbAbort	(3)
Retry	VbRetry	(4)
Ignore	VbIgnore	(5)
Yes	vbYes	(6)
No	vbNo	(7)

el siguiente ejemplo muestra una caja de mensaje y realiza una tarea si se elige el botón de Yes y otra si se elige No.

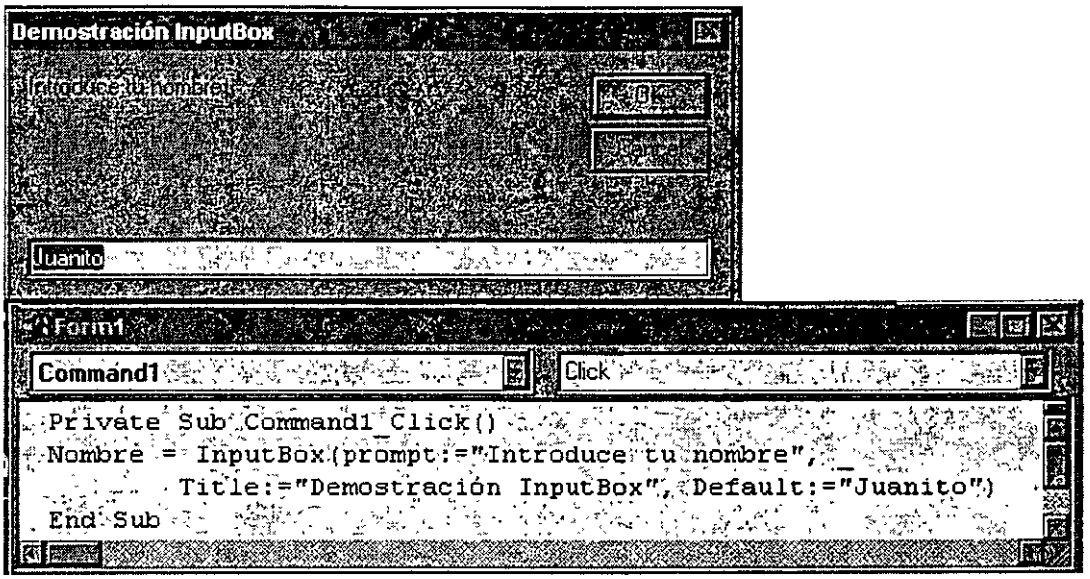
Ejemplo:

```
Msg = "¿Deseas continuar?"
Estilo = vbYesNo + vbExclamation + vbDefaultButton2
Título = "Demostración de caja de texto"
Respuesta = Msgbox (msg, Estilo, Título)
if Respuesta = vbYes then
        Mirespusta = "SI"
else
        Mirepuesta = "NO"
end if
```



Función InputBox

Regresa una cadena dada por el usuario.



Sintaxis **InputBox** (*prompt [,title][,default][,xPos][,yPos][,helpFile, context]*)

Trabajando con instrucciones de código

Visual Basic cuenta con una serie de constantes predefinidas que pueden ser utilizadas en cualquier parte del código a través de sus nombres o bien de los valores que representan, además de que es más sencillo e ilustrativo utilizarlas a través de sus nombres, por ejemplo:

```
frmValidación.WindowState = vbMaximized

MsgBox "Contraseña inválida", _
        vbOKCancel + vbExclamation
```

Otra ventaja de utilizar los nombres de la contantes, es que los valores de éstas pueden cambiar de versión en versión de Visual Basic pero sus nombres no.

Pro ejemplo para la propiedad WindowState de las formas los valores que puede tomar se dan a través de las siguientes constantes o de sus valores:

Constante	Valor	Descripción
vbNormal	0	Normal
vbMinimized	1	Minimizada
vbMaximized	2	Maximizada

se puede dar el valor de las siguientes formas:

frmValidación.WindowState= 2
o bien
frmValidación.WindowState= vbMaximized

Combinando Constantes de Visual Basic

Cada constante es un valor y se puede combinar con una o más constantes sumando sus valores.
Un ejemplo concreto es el argumento ***buttons*** de la función **MsgBox** , que está determinado por la suma de 3 constantes, la que representa el número y tipo de botones, el icono a desplegar y qué botón será el seleccionado por omisión.

Para crear una caja de mensaje que despliegue 2 botones el de OK y el de Cancel y un icono de exclamación:

```
MsgBox "Contraseña inválida", _
        buttons:= vbOKCancel + vbExclamation
```

Enunciando argumentos

Los argumentos son requeridos por métodos, funciones y procedimientos y pueden ser indicados en dos formas: a través de su posición o bien por su nombre.

Posición de los argumentos, cuando se especifican argumentos por posición, se indican de acuerdo el orden que marca la sintaxis del método o función, separando a éstos por comas (,).

Ejemplo:

```
MsgBox "Contraseña inválida", vbOKCancel + _
                vbExclamation, "Validación", "Validacion.hlp", 3
```

Argumentos por nombre: Utilizando el nombre del argumento, estos pueden ser proporcionados en el orden que se desee. Con ésta opción se tiene la ventaja de saber que representa cada valor enviado, así como que se pueden omitir argumentos.

Ejemplo:

```
MsgBox prompt:= "Contraseña inválida", _
buttons:= vbOKCancel + vbExclamation, _
title:= "Validación", _
helpfile:= "Validacion.hlp", _
context:= 3
```

Manejando formas

Desplegando una forma

La forma es el elemento principal de una aplicación Visual Basic, por omisión al crear cualquier aplicación se crea una forma que al ejecutar la aplicación se muestra, esto no es tan transparente cuando en nuestra aplicación existe más de una forma, cuando éste es caso se debe especificar cuando se muestra una forma y cuando se cierra u oculta.

Existen varias formas de hacer referencia a una forma, para poder hacer referencia a ella o a cualquier objeto que contenga, ésta debe estar "cargada" en memoria.

El método Show

El método **Show** despliega una forma, haciendo que se cargue en memoria y haciéndola visible:

Sintaxis Objeto.**Show** style

Ejemplo:

```
frmValidacion.Show
```

Si la forma ya estaba cargada en memoria el método **Show** la hará visible.

La instrucción Load

La instrucción Load carga una forma o un control en memoria, aunque no lo hace visible.

Sintaxis **Load** Objeto

Ejemplo: **Load** frmForm1

La instrucción Load no es necesaria para las formas aún cuando queramos que estén en memoria y no se desplieguen, ya que cualquier referencia a ellas a través de su nombre o de algún control dentro de ellas la cargará en memoria.

El evento Load

El evento Load ocurre cuando la forma es cargada en memoria. Esto sucede cuando se utiliza la instrucción Load o cuando se invoca a la forma con el método Show (siempre y cuando la forma no esté aun en memoria).

El evento Load se utiliza para inicializar la forma, es decir, los valores por omisión para sus controles o para dar valores iniciales a las variables que se utilizarán dentro de ella.

Formas de tipo Modal o Modeless

Una forma Modal no permite que el usuario interactúe con otras formas dentro de la aplicación al mismo tiempo. La mayoría de las caja de mensaje o diálogo en Visual Basic son del tipo Modal. Cuando una de ellas de despliega, la aplicación emite un beep si el usuario trata de pasar a otra forma.

Una forma del tipo Modeless permite al usuario saltar de una forma a otra. Ejemplos de este tipo de formas son las ventanas del ambiente de desarrollo de visual Basic, las ventanas de propiedades, de código, de proyecto.

El argumento *style* del método show determina el tipo de la forma. Si el *style* es **vbModal**, será una forma Modal, si es **vbModeless** será una forma.del tipo Modeless.

Ejemplo:

```
frmForm1.Show vbModal
frmForm1.Show vbModeless
```

Ocultando una forma

Mientras la aplicación se ejecuta, se tiene que quitar una forma de la pantalla para poder desplegar otra. Existen dos métodos para hacer esto .El método **Hide** o la instrucción **Unload**.

El método Hide

El método **Hide** de las formas permite ocultarlas, sin descargarlas de memoria.

Sintaxis: Object.**hide**

Ejemplo: frmForm1.**Hide**

Cuando una forma se oculta con este método se vuelve inaccesible para el usuario, pero el valor de los controles se mantiene.

Si una forma no está cargada en memoria y se invoca su método Hide, el método la carga en memoria pero no es visible.

Instrucción Unload

La instrucción Unload descarga de memoria una forma o un control, libera los componentes de despliegue de la forma, aunque el código asociado a ésta continúan en memoria.

Libera los valores de las formas y los controles, y les devuelve sus valores iniciales.
Invoca al evento Unload.
Termina la ejecución de la aplicación si la forma que se está descargando o invocando a el método es la única forma de la aplicación.

Sintaxis **Unload** Objeto

Ejemplo: **Unload** frmForm1

Descargar una forma con la instrucción Unload es necesario en algunas ocasiones para utilizar esa memoria en otras formas, o bien cuando se desea reiniciar los valores de la forma.

Nota: Una aplicación no termina hasta que todas las formas son descargadas de memoria. Para hacer esto de manera directa se utiliza la instrucción End.

Para hacer referencia a la forma que llama al procedimiento no es necesario utilizar su nombre, basta con utilizar la palabra reservada Me, de esta forma si se va a ejecutar la instrucción Unload dentro de la forma frmForm1, es lo mismo si se escribe la instrucción:

Unload frmForm1 o **Unload Me**

Evento Unload

El evento **Unload** ocurre cuando:

La forma es descargada utilizando la instrucción **Unload**.
La forma se cierra utilizando el comando **Close** del la barra de titulo de la aplicación.

El evento **Unload** no ocurre si se quita de memoria una forma con la instrucción **End**.

Utiliza el evento Uload para verificar que al terminar las formas realicen ciertas acciones como validar los valores proporcionados por el usuario o para preguntar por si se guardan los resultados, etc.

Si en tu aplicación tiene muchas formas y las utilizas todas, es más fácil mostrarlas y ocultarlas (**Show, Hide**) que cargarlas y descargarlas (**Load, UnLoad**).

Fijando el procedimiento o la forma de arranque

Por omisión Visual Basic inicia la aplicación con la primera forma que se creó en el proyecto, sin embargo se puede especificar que inicie desde otra, de acuerdo a lo que se quiera.

Para especificar la forma de inicio:

1. Del menú **Project**, elige la última opción **"tu proyecto" Properties ....**
2. Selecciona la primera pestaña (**General**)
3. Selecciona la forma que desees de la lista **Startup Object**

Para especificar un procedimiento de inicio:

1. Escribe un procedimiento general Sub llamado Main y que se debe guardar como un módulo estándar.
2. Del menú **Project**, elige la última opción **"tu proyecto" Properties ....**
3. Selecciona la primera pestaña (**General**)
4. Selecciona **Sub Main** de la lista **Startup Object**

En este procedimiento se puede inicializar el llamado de todas la formas, cargarlas en memoria y desplegar la que sea la inicial.

Ejemplo:

```
Sub Main()

        frmInicio.Show
        frmPrimera.Load
        frmInicio.hide
        frmPrimera.Show
End Sub
```

Terminando una aplicación

Mientras el usuario pude cerrar una forma utilizando el comando Close de la aplicación o de la barra de título, se debe escribir un programa que controle cómo el usuario termina o cierra la aplicación. El programa debe ser capaz de determinar si la información proporcionada por el usuario es correcta o pedir que regrese a darla nuevamente.

Instrucción End

Se puede terminar la ejecución de la aplicación descargando la última forma de la aplicación o bien utilizando la instrucción End. Esta termina la ejecución del programa y descarga todas las formas de memoria.

Ejemplo: End

Evento Unload

Utiliza la instrucción **Unload** para cerrar una forma

Ejemplo:

```
Sub cmdAceptar_ClicK()
        Unload Me
End Sub
```

Entonces escribe un procedimiento asociado al evento Unload que permita hacer una verificación de la información obtenida por la aplicación y de ser correcta, que ejecute la instrucción End.

Ejemplo:

```
Sub Form1_Unload()
        [Código de verificación]
        End
End Sub
```

El evento **Unload** es invocado cuando se cierra una forma con el comando **Unload**.

Permitiendo al usuario cancelar al momento de cerrar la aplicación.

Para evitar que se termine la aplicación se le permite al usuario que decida si de verdad quiere cerrar la aplicación.

Para ello el evento Unload cuenta con un argumento llamado Cancel, cuando este argumento tiene el valor True, entonces no se descarga la forma, mientras que si el valor para ésta variable es False, la forma se cierra.

Ejemplo:

```
Sub Form_Unload(Cancel As Integer)
        If MsgBox("Deseas terminar?", vbOKCAncel) =vbCancel then
                Cancel = True
        End if
End Sub
```

Tema IV

Variables y Procedimientos

Tema IV Variables y Procedimientos

Objetivos:

Al final del tema el participante será capaz de:

Explicar el rol de los tipos de datos al declarar variables.
Declarar variables locales y globales utilizando las instrucciones Dim y Public.
Utilizar variables públicas para pasar datos entre múltiples formas.
Describir la diferencia entre una variable y una constante.
Diferenciar entre procedimientos Sub y Function.
Crear un procedimiento Function que acepte argumentos y regrese un valor.
Describir cómo los módulos estándar difieren de los módulos a nivel de forma
Agregar un módulo estándar a un proyecto para almacenar procedimientos generales y variables.

Introducción al uso de variables

¿ Qué es un variable ?

Es una localidad de almacenamiento que contiene información que puede ser modificada durante la ejecución de un programa. Típicamente las variables contienen valores que se utilizan en más de una ocasión durante la ejecución del programa. Algunos de los usos que se le pueden dar a una variable son:

a) Hacer la función de un contador que almacene las veces que un bloque de código o un procedimiento se ejecuta.
b) Un lugar para almacenar de manera temporal la propiedad de un objeto que tu programa utiliza para regresarlo después de que los cambios fueron hechos por el usuario.
c) Para almacenar el valor que devuelve un método o función.
d) Para almacenar nombres de directorios o de archivos.

Los datos almacenados en una variable pueden ser originados de múltiples formas; a través de una expresión, una entrada dada por el usuario, un objeto, la propiedad de un objeto o un valor regresado por un método o función.

Las variables pueden contener diferentes tipos de datos, como texto, números o valores booleanos (Verdadero o Falso). Las variables pueden ser definidas y ser utilizadas en diferentes parte del programa.

Tipos de datos en Visual Basic

Tipo de datos	Tamaño de almacenamiento	Rango
Byte	1 byte	0 a 255
Boolean	2 bytes	True o False
Integer	2 bytes	-32, 768 a 32, 767
Long(long integer)	4 bytes	-2,147,483,648 a 2,147,483,647
Single(single precisión floating point)	4 bytes	-3.4028223E38 a 1.401298 E45 negativos y de 1.401298E-45 a 3.402823E38 para positivos.
Double (double precisión floating point)	8 bytes	-1.79769313486232 E308 a -4.9406564541247E-324 para negativos de 4.9406564541247E-324 a 1.79769313486232 E308 para positivos.
Currency	8 bytes	
Date	8 bytes	January1,100,a December 31, 9999
Objetc	4 bytes	Cualquier referencia a objetos
String	10 bytes	
Variant	16 bytes	Cualquier valor

Tipos de datos y Memoria

Las variables pueden contener un número diferente de tipos de datos, como lo son texto, valores enteros, booleanos, u objetos, por omisión. Si se le indica a Visual Basic que utilice un tipo de dato específico cuando se declaran las variables, se puede optimizar la velocidad de ejecución del código y el tamaño de éste.

Por omisión, las variables utilizan el tipo de dato llamado Variant. Los beneficios de utilizar este tipo de dato es que puede almacenar cualquier tipo de dato. Sin embargo para tener esta flexibilidad, el tipo Variant requiere de una gran cantidad de espacio de almacenamiento (memoria).

Si se selecciona el tipo de dato apropiado para las variable, se puede hacer el código más eficiente. Por ejemplo, asumiendo que se utilizará una variable para llevar el conteo del número de veces que un bloque de código se ejecuta. Existen varios tipos que pueden utilizarse para este propósito; pero el más eficiente es el Integer. Comparando la memoria utilizada por un Integer y la que utiliza un Variant:

Tipo de dato	Memoria utilizada
Variant	mínimo 16 bytes
Integer	2 bytes.

Si esa diferencia se multiplica por el número de variables utilizadas en un programa, la memoria que se puede ahorrar puede ser muy sustancial.

Además, almacenar datos en variables con los tipos de datos adecuados nos puede evitar procesos de conversión para poder manipularlos.

Dando nombres a las variables

Las reglas para dar nombre a las variables son las siguientes:

a) El nombre debe iniciar con un carácter alfabético.
b) No puede contener puntos, ni tampoco cualquiera de los siguientes caracteres (%,&, !, #, @, o $).
c) Su nombre debe ser único en su ámbito.
d) No debe ser de más de 255 caracteres.

Convención de Nombres

Al igual que para los controles u objetos, existe una convención para dar nombre a las variables. El prefijo de un variable representa el tipo de dato que almacena.

Tipo de dato	Prefijo
Byte	b
Boolean	f(de "flag")
Integer	i
Long	l
Single	s
Double	dbl
Currency	c
Date	dt
String	str
Variant	v

Además del prefijo de tipo de dato se puede agregar una abreviación que indique el ámbito de la variable.

Ámbito	Prefijo
Global	g
Módulo/Forma	m
Local	l

Option Explicit

Por omisión Visual Basic no requiere que las variables que se utilizan sean declaradas, pero esto puede generar errores.

La instrucción **Option Explicit** fuerza a Visual Basic a verificar que cada variable que se utiliza en el programa se haya declarado previamente.

Está instrucción debe se colocada en la sección de declaraciones generales de los módulos; para hacer este proceso más sencillo, se puede hacer de la siguiente manera:

1. Del menú **Tools**, elige **Options**
2. Seleciona la pestaña **Editor**
3. Activa la opción **Require Variable Declaration**
4. Elige **OK**.

Esta opción aplicará para los módulos que se generen del momento que la elegiste en adelante, si existen módulos previamente creados, se les debe especificar manualmente en la sección *General Declaration* de cada módulo.

Es recomendable utilizar siempre la instrucción **Option Explicit** para prevenir errores del programa al definir una variable para múltiples tareas y en diferentes ámbitos.

Sección General Declaration

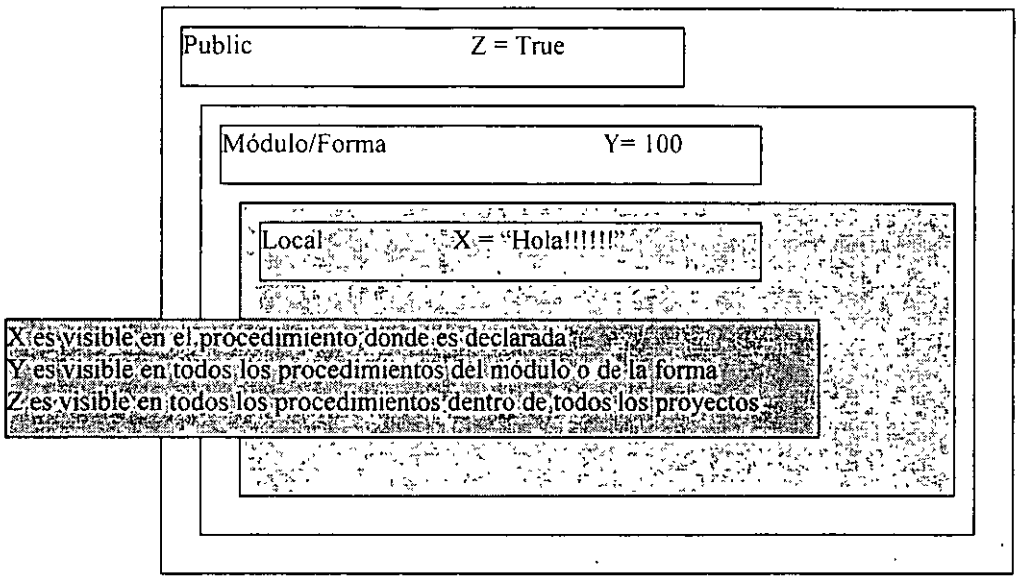
La sección *General Declaration* o de declaraciones generales, se encuentra en el inicio de la ventana de código antes de cualquier procedimiento contenido en el módulo. La sección *General Declaration* contiene declaraciones. las cuales son código no ejecutable que puede llamar a procedimientos externos, constante o variables y definen propiedades. Esta sección tiene una lista de opciones que aplican a todos los procedimientos contenidos en el módulo.

De manera general esta sección es utilizada para declarar variables globales y públicas.

Declarando variables

Para declarar una variable se deben considerar entre otras cosas; su ámbito.

Ámbito o Scope



El ámbito define la visibilidad de una variable, procedimiento u objeto.
Existen tres ámbitos en Visual Basic:

Ámbito	Visibilidad
Local	Un variable definida en un procedimiento es visible sólo en el procedimiento donde es declarada.
Módulo/Forma	Una variable definida en un módulo o en una forma es visible para todos los procedimientos dentro del módulo o de la forma donde ésta fue declarada.
Public	Una variable declarada en un módulo o forma que puede ser visible dentro de todos los procedimientos de todos los proyectos.

Declarando variables locales

Una variable local es declarada dentro de un procedimiento y se inicializa con 0 o con Null cada vez que el procedimiento es llamado. El valor de una variable local no está disponible en otro procedimiento dentro de la aplicación.

Las variables locales son declaradas dentro de un procedimiento utilizando la instrucción **Dim/Private**:

Sintaxis **[Dim/Private]** nombre_variable **[As** *type***]**

Ejemplo:

```
Sub cmdCommand1_Click()
        Dim strTemName As String
        StrTempName = frmForm1.txtName.Tex
        MsgBox "Bienvenido," & strTempName & "!"
End Sub
```

Declaración implícita y explícita de variables

Las variables locales puedes ser declaradas simplemente asignándoles un valor a un nombre de variable. Esto se conoce como declaración implícita. Las variables declaradas de esta manera son siempre del tipo **Variant** y su ámbito es local.

Sintaxis *varname = value*

Ejemplo: vTemp = 100

El permitir utilizar declaración implícita, puede acarrear problemas en el programa que resultan difíciles de detectar; éstos errores regularmente son errores al llamar una variable con un nombre diferente al que se utilizó cuando se declaró.

Por ejemplo, al no tener habilitada la instrucción **Option Explicit**, si se hace una declaración explícita de la siguiente forma:

```
Dim iTemp As Integer
```

y después nos equivocamos al invocarla y lo hacemos así:

```
iTmp=5
```

Se habrá hecho la declaración implícita de una variable diferente y esto nos puede causar problemas que no se pueden detectar tan fácilmente.

Que por el contrario si se habilita la opción **Option Explicit**, Visual Basic al encontrar una expresión como :

iTmp=5

y ver que no existe una declaración explícita de la variable iTmp, la sustituye por aquella que es más parecida en su identificador y automáticamente corrige nuestros errores de escritura cambiando el nombre de la variable al correcto.

La instrucción Static

El tiempo de vida describe por cuanto tiempo el valor de una variable durará en memoria mientras se ejecuta la aplicación.

Las variables locales existen sólo mientras el procedimiento en el cual fueron declaradas está en ejecución. Por omisión cuando un procedimiento termina de ejecutarse, los valores de sus variables no se conservan más en memoria, esto es para dejar libre el espacio para otras variables que lo necesiten. La siguiente ocasión que el procedimiento es ejecutado, todas sus variables son inicializadas en (0) o null.

Sin embargo, Visual Basic puede conservar el valor de una variable local utilizando el calificador **Static** al momento de su declaración, en el lugar de **Dim**.

Sintaxis **Static** varname [**As** tipo]

Esto hará que el valor de una variable se conserve aún entre diferentes llamadas al procedimiento en donde fue declarada.

Ejemplo: En el siguiente procedimiento la variable iCount se declaró como estática, esto hará que cada que se ejecute el procedimiento se le sume un 1 al valor almacenado en iCount.

```
Sub cmdCommand1_Click
        Static iCount As Integer
        iCount = iCount + 1
        MsgBox "El valor de iCount es ahora " & iCount
End Sub
```

Nota: Se puede obtener el mismo resultado declarando al variable iCount como una variable a nivel de Modulo/Forma. Sin embargo, ésta pudiera ser utilizada por otros procedimientos alterando su valor y produciendo resultados no deseados en la aplicación.. Por ello una forma de asegurar que sólo puede ser utilizada por el procedimiento donde se declaró y que mantenga su valor entre llamadas al procedimiento durante la ejecución de la aplicación es utilizar el calificador **Static**.

Declarando Variables Publicas y a nivel de Módulo/Forma

Las variables declaradas a nivel de Módulo/Forma pueden ser públicas o privadas **[Public/Private]**. Estas variables son declaradas en la sección **General Declarations** de la forma o del módulo estándar o del módulo de clase.

Sintaxis **[Public| Private|Dim]** varname **[As** tipo]

A diferencia de las variables locales que son siempre privadas, las variables a nivel de módulo/forma pueden ser privadas al módulo al módulo en el cual fueron declaradas o bien públicas:

Ámbito	Visibilidad	Sintaxis
Private	Privada al módulo en el cual fue declarada	**Dim** varname [**As** tipo] **Private** varname [**As** tipo]
Public	Disponible para todos los módulos en todos los proyectos.	**Public** varname [**As** tipo]

Las variables a nivel de forma son declaradas en la sección de General Declarations, pero no es posible asignarles valor ahí. Éste debe ser dado dentro de un procedimiento.

Las variable a nivel de módulo/forma son inicalizadas con (0) o null cuando el programa inicia. Los valores se mantienen durante la duración del programa.

Nota: Las variables públicas en un módulo de forma, deben referenciarse con su nombre completo si es que se llaman dentro de otra forma o módulo estándar (por ejemplo Form1.X)

Ejemplo: Se declara una variable a nivel de módulo llamada iCount como entera y que es utilizada por 4 procedimientos.

Procedimiento	Tarea
frmMain_Load	Inicializa el valor de iCount en cero
cmdEnterRecords_Click	Suma 1 a iCount cada vez que el procedimiento de ejecuta.
CmdDeleteRecords_Click	Resta un 1 a iCount cada vez que se ejecuta.

CmdFinisehed_Click	Utiliza iCount para mostrar el número de registros insertados.

```
Private iCount  As Integer

Sub frmMain_Load()
        iCount=0
End Sub

Sub cmdEnterRecords_Click()
        iCount = iCount+1
        [Código para insertar un registro a una base de datos]
End Sub

Sub CmdDeleteRecords_Click()
        iCount = iCount-1
        [código para borrar un registro de una base de datos]
Ens Sub


Sub CmdFinisehed_Click()
        MsgBox iCount & Registros ingresados a la base de datos.".
End Sub
```

Constantes

Las constantes son similares a las variables en cuanto a la forma de llamarlas y en espacio y tipo de datos que almacena en memoria. También una constante puede ser local, a nivel de módulo, es decir, también tiene un ámbito definido. Sin embargo la diferencia con respecto a las variables, es que las constantes contienen valores que no cambian durante la ejecución del programa.

Sintaxis **[Publi|Private] Const** constname [**As** tipo] = expresión

Ejemplo:
```
Const strPASSWORD As String = "Secreto"
Public Const PI As Double = 3.14159
```

Se recomienda dar nombre a las constantes que tengan todas las letras mayúsculas, esto distinguirá las constantes de las variables y a recordarnos que las constantes no pueden cambiar su valor dentro del programa.

Conversión de tipos de datos

Para hacer más eficiente la ejecución de nuestros programas, se ha determinado la necesidad de declarar las variables de acuerdo al tipo de datos que almacenarán, en ocasiones será necesario hacer conversión de tipos de datos entre variables dependiendo de los procesos que se vayan a calcular o bien de lo que se necesite mostrar.

Visual Basic realiza algunas conversiones de tipo de manera automática. En ocasiones, si una cadena del tipo "1234" es almacenada en una variable declarada del tipo Integer, Visual Basic hace una conversión automática de cadena a entero. En otro ejemplo, cuando una cadena del tipo "100" es sumada a un valor numérico utilizando la expresión "100" + 10, el resultado será un valor numérico de 110.

Existen algunos otros casos en los que no sabremos cual será el resultado entre operar expresiones de distinto tipo. Por ejemplo, si 2 cadenas se suman, Visual Basic no sumará su valor numérico en caso de tenerlo, sino que concatenará las cadenas ("100" + "10" = "10010").

Se pueden evitar problemas en tu código haciendo explícitamente la conversión de valores antes de ser utilizados. También , cuando los valores son convertidos de manera explícita nuestros programas se hacen más rápidos ya que no dejamos en Visual Basic la tarea de analizar los tipos de datos y ver si es posible una conversión.

Ejemplo: El siguiente código realiza la conversión de pies y pulgadas a metros de acuerdo a los siguiente:

Acepta 2 cadenas
Convierte las cadenas a valores numéricos
Convierte los valores numéricos de pies y pulgadas a metros.
Convierte el resultado a una cadena
Muestra la cadena

El código de la conversión aparece en el procedimiento de evento cmdCalculate_Click . Las contantes MILLIMETERS_PER_INCH e INCHES_PER_FOOT se declaran en la sección General Declarations de la forma.

```
Const MILLINETERS_PER_INCH = 24.5
Const INCHES_PER_FOT =12

Private Sub cmdCalculate_Click()
Dim feet As Double, inches As Double
Dim millimeters As Double, meters As Double

' Primero se extraen los valores de las cajas de texto.
```

‘ Las cajas de texto siempre devuelven un valor del tipo string, pero nosotros necesitamos un ‘ Double, para ello la función CDbl() es utilizada para realizar esta conversión.

```
feet = CDbl(txtFeet.Text)
inches = CDbl(txtInches.Text)

‘ ahora se hará la conversión de pies y pulgadas a milímetros.

Millimeters = (inches * MILLIMETERS_PER_INCH) + _
              (feet* MILLIMETERS_PER_INCH * INCHES_PER_FOOT)


‘ Convierte milímetros a metros

meters = millimeters /1000

‘ Muestra el resultado en una etiqueta , a través de su propiedad Caption
‘ ya que ésta es una cadena se hace una conversión previa del valor Double.

LblMeterResult.Caption = Format(CStr(meters), “#.##”)
```

Funciones de Conversión

Las funciones de conversión (o funciones “C”) convierten un tipo de dato a otro tipo de dato.

Las funciones de conversión con las que cuenta Visual Basic son:

Tipo al que convierten	Función
Booleano	CBool
Byte	CByte
Currency	CCur
Date	CDate
Double	CDbl
Integer	CInt
Long	CLng
Single	CSng
String	CStr
Variant	CVar

Es importante que para hacer la conversión de un tipo a otro estos se puedan realizar, es decir, no podemos convertir una cadena del tipo “ABC” a un entero, pero sí una cadena “1020”; para asegurarnos de que se pueden realizar las conversiones podemos hacer uso de la funciones **Is**; por ejemplo, **IsNumeric** para determinar si alguna variable o control

almacena o tiene valores numéricos o bien la función **IsDate** para saber si determinada cadena tiene un formato de fecha, etc.

Elementos de Región o Localidad

Permitir escribir fechas y valores monetarios en el programa, puede hacer que las funciones de conversión que trabajan con cadenas obtengan resultados no esperados.

Por ejemplo, el siguiente código no servirá en ningún otro sitio sino en aquel donde el símbolo de peso ($) sea el símbolo de moneda.

Ejemplo:
```
Money = "$1.22"
NewMoney = CCur(Money)
```

Como se puede ver, en el ejemplo, el punto decimal es el separador de cifras en esta región. Sin embargo, el código de abajo trabajará correctamente, no importando cuál sea el separador de cifras de la localidad o región del usuario.

Ejemplo:
```
Money = 1.22
NewMoney = CCur(Money)
```

Trabajando con Procedimientos

Hasta el momento se ha hablado de procedimientos de evento, los cuales son invocados automáticamente en respuesta a una acción del usuario o del sistema; es decir, de un evento. Existen otros tipos de procedimientos en visual Basic, los llamados Procedimientos Generales. Un procedimiento general es un bloque de código que debe ser explícitamente llamado desde otro procedimiento. Los procedimientos generales pueden dividir una aplicación compleja en varias unidades manejables.

Los procedimientos generales pueden ser almacenados en un módulo de forma o bien en un módulo estándar. Los módulos estándar pueden almacenar procedimientos generales, pero no procedimientos de evento.

Los procedimientos pueden ser llamados desde cualquier otro procedimiento de cualquier otro módulo, mientras que los procedimientos privados solo pueden ser llamados desde cualquier procedimiento dentro del mismo módulo.

Para llamar a un procedimiento desde otro módulo, se debe hacer referencia a él a través de su nombre y del nombre del módulo en el cual se encuentra definido, por ejemplo: Form1.cmdOK_Click.

Trabajando con procedimientos Generales

Creando un procedimiento general

Los procedimientos generales pueden ser del tipo Sub, Function, o Property. Los procedimientos generales pueden ser creados utilizando el comando *Add Procedure...* del menú Tools o escribiendo directamente el código en la ventana de código.

Para crear un procedimiento utilizando la caja de diálogo *Add Procedure*

1. Abre la ventana de código del módulo para el cual deseas escribir el procedimiento general.

2. Del menú **Tools**, elige la opción ***Add Procedure***.

3. En la caja de texto **Name,** escribe el nombre que tendrá el procedimiento. (No se permiten espacios en blanco en los nombres de los procedimientos).

4. En el grupo de opciones **Type**, selecciona **Sub** para crear un procedimiento, **Function** para crear una función, **Property** para crear un procedimiento de propiedad o **Event** para el de un evento.

   Recuerda que la diferencia entre un procedimiento o **Sub** y una función o **Function**, es que ésta última es un procedimiento que devuelve un valor.

5. En el grupo de opciones de **Scope** elige el ámbito de trabajo que deseas para tu procedimiento, por omisión es **Public**, pero puedes elegir que sea privado con la opción **Private**.

6. Elige OK.

En este momento Visual Basic habrá generado la plantilla para tu procedimiento de tal forma que tu puedes empezar a escribir el código correspondiente.

Para crear una aplicación desde la ventana de código

Si no se inserta el procedimiento a través de esta opción, se puede hacer escribiéndolo al final del código del módulo.

Escribiendo *Sub Nombre_del_Sub* al inicio del procedimiento y *End Sub* al final
o bien *Function Nombre_de_la_función* al inicio de la función y *End Function* al final.

Llamando a un Procedimiento General

Se tiene que hacer un llamado explícitamente a un procedimiento para que éste se ejecute, de lo contrario nunca se ejecutará. Típicamente se hace el llamado a un procedimiento

general Sub o Function dentro de un procedimiento de evento o bien desde otro procedimiento general.

Ejemplo:

```
Sub cmdCommand1_Click()
        PrintReport   'Ejecuta el procedimiento PrintReport
End Sub
```

Insertando un Módulo Estándar

Como se mencionó anteriormente, un procedimiento general pude estar almacenado en módulos estándar. El código de un módulo estándar es Public por omisión, de tal forma que los procedimientos generales almacenados en ellos pueden ser referenciados fácilmente desde cualquier procedimiento de evento y procedimientos generales de cualquier otro módulo o proyecto. Almacenar código compartido en módulos estándar es una forma de hacer más sencilla la organización de los proyectos.

Para crear un módulo estándar

1. Desde el menú **Project**, elige **Add Module**.
2. Al momento de guardar el proyecto pedirá un nombre para este nuevo archivo, al que le asignará la extensión .Bas por omisión.

Procedimientos Sub

Comúnmente se escribe un procedimiento Sub para realizar una tarea. Después de que el procedimiento Sub completa su trabajo (ejecuta su código), regresa el flujo de las acciones a la siguiente línea de donde fue llamado.

Son unidades de código Los procedimientos Sub. son llamados unidades de código porque tiene distintos puntos de inicio y de fin y porque el cuerpo de su código usualmente realiza una sola acción.

Están denotados por las instrucciones Sub y End Sub Todos los procedimientos **Sub** inician con la instrucción **Sub** y terminan con **End Sub.** El nombre del procedimiento siempre es seguido de paréntesis, aunque no tenga argumentos.

Sintaxis

```
Sub SubNombre()

        [Bloque de instrucciones]
End Sub
```

Ejemplo

```
Sub ObtenDatos()
        [...]
End Sub
```

Para llamar al Procedimiento ObtenDatos, sólo se escribe su nombre dentro de cualquier parte del código en dónde se desee ejecutar a dicho procedimiento.

Ejemplo

```
Sub cmdCommand1_Click()
        ObtenDatos
End Sub
```

Pueden recibir argumentos. Esto se cubrirá más adelante dentro del tema

No devuelven valor. Un procedimiento Sub no regresa ningún valor al ser llamado, sin embargo puede modificar información al término de su ejecución si es que se llama a través de argumentos o bien modifica valores de variables públicas o bien a nivel de módulo/forma.

Ejemplo:

```
Public fValido As Boolean

Public Sub Validacion()
        If condición = True then
                fValido = True
        Else
                Fvalido = False
        End If
End Sub
```

Procedimientos de tipo Function

Se escriben procedimientos de tipo **Function** o Funciones cuando necesitas verificar por un error o cuando se desea realizar un cálculo y por lo tanto se espera un valor de resultado.

La mayor diferencia entre un procedimiento **Function** y un **Sub**, es que los **Function** regresan un valor después de ser ejecutados mientras que los **Sub** no.

El valor que devuelve una función por omisión es un tipo de dato **Variant**. Sin embargo se puede especificar el tipo de dato que deseamos que devuelva a través del calificador **As**.

Sintaxis

```
Function Nombre_Función () As tipo

        [ bloque de instrucciones ]
        Nombre_Función = Valor de regreso

End Function
```

El uso más común para las funciones es el de realizar cálculos.

Nota: Para ignorar el valor que regresa una función, al llamarla no se deben poner los paréntesis encerrando a los argumentos, si es que recibe alguno.

Las funciones también se utilizan para verificar errores, como en el siguiente ejemplo donde se muestra una función que abre un archivo. Si es posible abrir el archivo, se devuelve un True, de otra manera la función devuelve un False.

Ejemplo:

```
Function OpenFile() As Boolean
        [Intenta abrir un archivo]
        if [operación exitosa]
                OpenFile = True
        else
                OpenFile = False
        End if
End Function
```

La función OpenFile es llamada a través de otro procedimiento

Ejemplo:

```
If OpenFile() = True then
        MsgBox "Exitoso"
Else
        MsgBox "Fallido"
End If
```

Pasando argumentos a los procedimientos

Cualquier procedimiento Sub o Function puede aceptar argumentos, los argumentos son la información que se le pasa a los procedimientos cuando éstos son llamados. Los argumentos están determinados por los parámetros especificados cuando los procedimientos (Sub) o las Funciones (Function) son definidos.

Cuando un procedimiento es definido, se debe poner cada parámetro con su respectivo tipo de dato dentro de los paréntesis.

Sintaxis para Sub

```
Sub Nombre_procedimiento (Param1 As tipo, Param2 As tipo,...)
        [bloque de instrucciones]
End Sub
```

Sintaxis para Function

```
Function Nombre_Función (Param1 As tipo, Param2 As tipo,....) As tipo
        [Bloque de instruciones]
End Function
```

Cuando el procedimiento es llamado, se debe pasar el mismo número de argumentos en el mismo orden en el que aparecen en la definición del procedimiento.

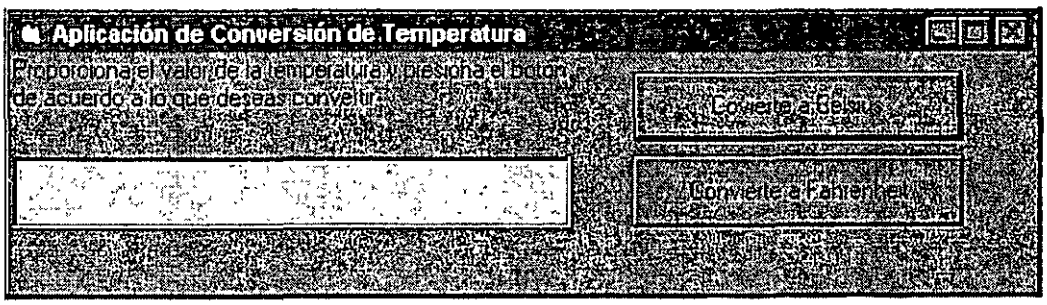
Nota: La enumeración de argumentos también es válida para los procedimientos, es decir, que si no recordamos el orden en el que los argumentos están definidos, pero recordamos todos, los podemos nombrar y definir en el orden deseado.

En el siguiente ejemplo, la función acepta un parámetro llamado Temperatura. La temperatura puede ser convertida de grados Fahrenheit a Celsius o de Celsius a Fahrenheit y el nuevo valor es devuelto por la función.

Ejemplo:

```
Function FtoC(Tempetatura As Single) As Single
        ' Convierte Fahrenheit a Celsius
        FtoC = (Temperatura -32) * ( 5/9)
End Function

Function CtoF(Temperatura As Single) As Single
        ' Convierte Celsius a Fahrenheit
        CtoF = Temperatura* (9/5) + 32
End Function
```



Las funciones son llamadas dentro de los procedimientos asociados al evento Click de los botones de comando. El contenido de la caja de texto es pasada a la función adecuada para que su valor sea convertido y regresado a la caja de texto.

```
Private Sub cmdFtoC_Click()
        Text1.Text =FtoC(CSng(Text1.Text))
End Sub

Private Sub cmdCtoF_Click()
        Text1.Text =CtoF(CSng(Text1.Text))
End Sub
```

Tema V

Utilizando la herramienta Debugger

Tema V Utilizando la herramienta Debugger

Objetivos:

Al final del tema el participante será capaz de depurar aplicaciones a través de:

- Interrumpir la ejecución de un programa utilizando las instrucciones *Breakpoint* (Punto de rompimiento) y *Watch* (visualización de variables).
- Ejecutar la aplicación seleccionando partes del código.
- Llevar la secuencia de ejecución de un programa utilizando la caja de diálogo *Call Stack.*
- Monitorear los valores de las variables a través del panel *Watch*
- Probar con datos y procedimientos desde el panel *Immediate.*

Tipos de errores

Los errores de programación generalmente son de tres categorías: de lenguaje, de ejecución y de lógica.

Errores de sintaxis Los errores de lenguaje o de sintaxis son resultado de la incorrecta construcción del código o de haber escrito una instrucción de forma indebida o escrito mal una palabra reservada, por omisión de un signo de puntuación o por utilizar de forma incorrecta alguna estructura de control de flujo. Visual Basic, de forma opcional detecta estos errores en cuanto se escribe la línea y se pasa a la siguiente o bien, inmediatamente antes de que se ejecute el código notificando el problema.

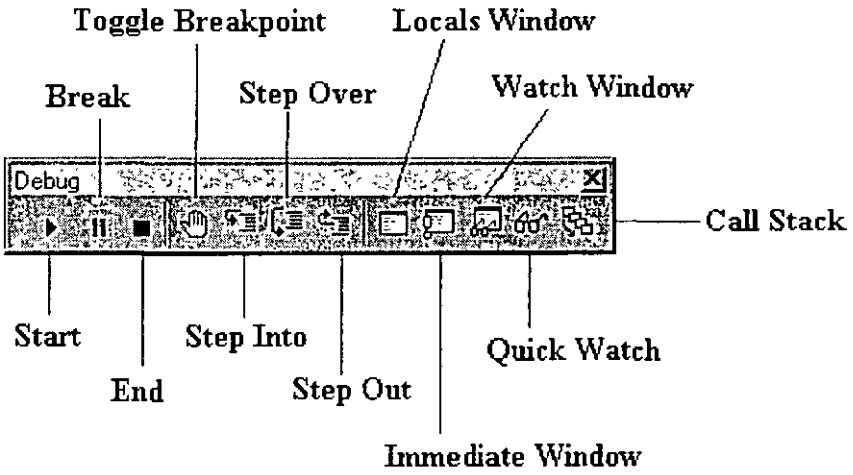
Errores en tiempo de ejecución Estos errores se detectan cuando una instrucción intenta realizar alguna operación imposible, por ejemplo una referencia a un objeto que no existe en el ámbito actual. Para esos tipos de errores se debe escribir código para prevenirlos.

Errores de lógica Estos se presentan cuando la aplicación no realiza la tarea deseada, el código se ejecuta pero los resultados no son los esperados. Para ello se requiere probar el código y verificar qué es lo que muestran los resultados. Los errores de lógica se perciben hasta que la aplicación se ejecuta.

Verificación de errores por Visual Basic

Los dos primeros tipos de errores, de sintaxis y de ejecución, pueden ser detectados por Visual Basic cuando el código está siendo escrito o cuando se está tratando de ejecutar la aplicación. Visual Basic enviará un mensaje para determinar qué se debe hacer para corregir ese tipo de errores. El tercer tipo de errores, los de lógica, son más difíciles de detectar puesto que el programa se ejecuta sin marcar ningún error. Para corregir este tipo de errores existen varias herramientas que permiten monitorear en cualquier parte del código y observar su desempeño.

Depurando errores de lógica en Visual Basic



Para poder corregir los diferentes errores que ocurren en un programa, primero se deben conocer las herramientas que Visual Basic nos brinda para corregirlos; estas herramientas las encontramos en la barra de herramientas.

Las principales son:

Botón	Función
Toggle Breakpoint	Crea o remueve un *breakpoint* en la línea actual. Un *breakpoint* es un lugar en el código en donde Visual Basic interrumpe la ejecución del programa.
Quick Watch	Muestra el valor actual de una expresión que se agrega en el panel *Watch*.
Call Stack	Lista las llamadas a procedimientos activas.
Step Into	Ejecuta la siguiente línea de código; si en ésta se hace llamada a un procedimiento, entonces saltará a ejecutar cada línea del procedimiento.
Step Over	Ejecuta la siguiente línea de código; si en ésta se hace llamada a un procedimiento, este se ejecuta como una unidad y se pasa a la siguiente línea en el procedimiento actual.
Step Out	Ejecuta las líneas restantes del procedimiento en donde se encuentra.
Locals Window	Despliega la ventana **Locales**
Immediate Window	Despliega la ventana de depuración **Inmediata**
Watch Window	Despliega la ventana de depuración **Watch**

Quick Watch Despliega la caja de diálogo **Quick Watch** con el valor actual de la expresión seleccionada.

Modo de Rompimiento (Break Mode)

El modo de rompimiento interrumpe la ejecución del código, dándonos una vista de la condición o estado del programa hasta ese momento. Cuando se está en *Break Mode*, se pueden observar los valores de las variables, las expresiones y las propiedades, se puede realizar un recorrido paso a paso por las líneas del código viendo la lógica del programa a través del flujo de éste.

Cuando Visual Basic está en Modo de rompimiento, lo siguiente ocurre:

La ejecución se interrumpe en el momento en que Visual Basic encuentra una condición que la hace pasar a *break mode*, la ejecución del código es detenida.

Las variables y los valores de las propiedades se conservan En modo de rompimiento los valores se conservan de tal forma que se puede:

a) Ver el valor de las variables, propiedades y expresiones.
b) Cambiar el valor de las variables y propiedades.

Entrando en modo de rompimiento

Visual Basic entra en modo de rompimiento por cualquiera de las siguientes razones:

Se ejecutan los comandos *Step Over* o *Step Into* ya que con ellos se permite ejecutar línea a línea el código.

Se encuentra un Breakpoint o punto de rompimiento Un breakpoint es una marca en el código que le dice a Visual Basic que suspenda la ejecución. Los breakpoints son insertados cuando se sospecha que existe algún problema. Los breakpoints son temporales y no se almacenan con el código.

Para insertar un breakpoint

1. Coloca el punto de inserción en cualquier línea del procedimiento, en donde quieres que se interrumpa la ejecución del programa
2. Utiliza cualquiera de los siguiente métodos
   a) Del menú *Debug*, elige *Toggle Breakpoint*
   b) Presiona la tecla F9
   c) Elige el botón de *Toggle Breakpoint* de la barra de herramientas
   d) Da clic en el botón derecho del mouse y selecciona *Toggle breakpoint* del menú colgante.
   e) Da clic en el margen izquierdo del código.

El breakpoint es agregado y la línea cambia de color de acuerdo a la configuración que se tiene en el editor.

Para borrar o eliminar un breakpoint se puede utilizar cualquiera de los procedimientos para insertarlo. O bien desde el menú ***Debug*** elegir la opción ***Clear All Breakpoints***.

Instrucciones de interrupción (Stop) Coloca la instrucción *Stop* en cualquier parte del código. Cuando visual Basic encuentra esta línea, entra en modo Break. La diferencia entre la instrucción *Stop* y un breakpoint es que la instrucción continúa siendo parte del código a menos que se retire manualmente.

Expresiones de monitoreo (Watch Expressions) Dependiendo de cómo esté definida una expresión Watch o de monitoreo, puede causar que el programa entre en modo Break.

Interacción con el teclado Para entrar en modo Break durante la ejecución del programa se pueden pulsar las teclas:

CTRL+BREAK

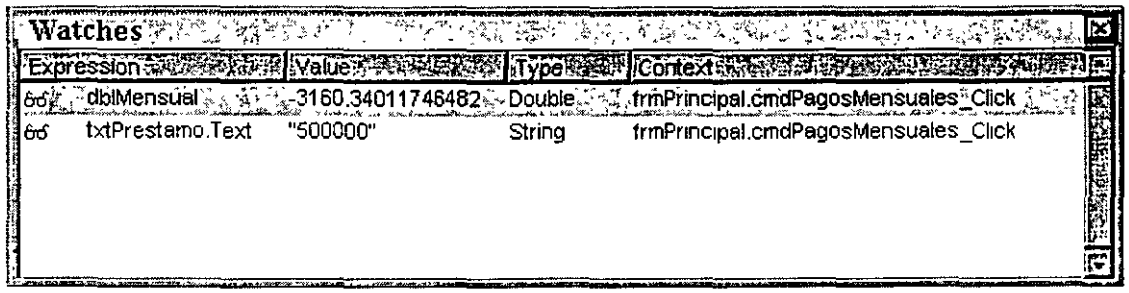
Errores en tiempo de ejecución Cuando se genera un error en tiempo de ejecución Visual Basic entra en modo break.

Las ventanas depuración (debugg)

A través de las ventanas de *debugg* se pueden observar los valores de expresiones y de variables mientras se va paso a paso dentro del código del programa. Dentro de ella se pueden modificar los valores de variables y propiedades para analizar como se comporta el programa con diferentes valores y recordando que se encuentra el programa en modo Break.

Se tienen dos ventanas para realizar este tipo de monitoreo: la ventana **Watch** y la **Inmediata**, estas ventanas siempre están presentes cuando se ejecuta una aplicación dentro del ambiente de desarrollo de Visual Basic.

La ventana Watch



La ventana **Watch** muestra la expresiones Watch, es decir, las expresiones de monitoreo que se decidió insertar para ver el comportamiento del código.
En esta ventana se muestra una columna que indica el Contexto, es decir el procedimiento, la forma o el módulo en el cual es válida la expresión. Esta ventana mostrará el valor de la expresión de monitoreo sólo si se está en el contexto en la cual fue declarada, de otra forma indicará a través de un mensaje que la expresión está fuera de contexto.

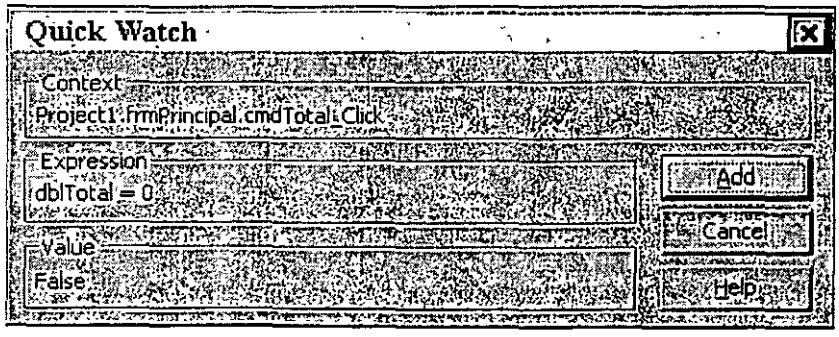
Agregando una expresión de monitoreo (Watch expression)

Existen dos formas de agregar una expresión de monitoreo al código del programa.

Para agregar una expresión de monitoreo desde la caja de diálogo Quick Watch

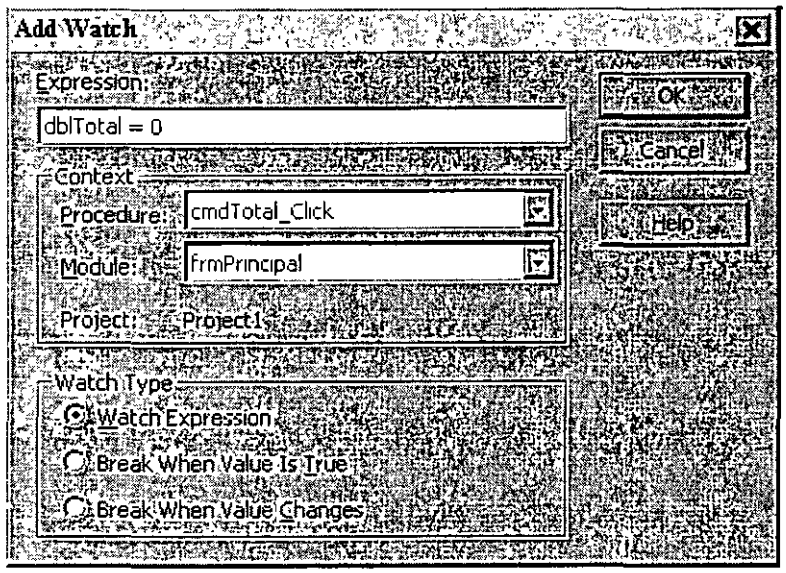
1. Desde la ventana de código que aparece en el modo break, selecciona la expresión que se desea monitorear.
2. Para ello se pueden utilizar cualquiera de los siguientes métodos:
   a) Desde el menú *Debug* elegir ***Quick Watch***
   b) Elegir el botón ***Quick Watch*** de la barra de herramientas ***Debug***

La ventana ***Quick Watch*** muestra el valor de la expresión seleccionada sin agregarla a la ventana ***Watch***; para hacerlo se da clic en *Add*.



Para agregar una expresión de monitoreo con el comando Add Watch

1. Desde el menú *Debug*, elegir ***Add Watch***
2. En la caja de texto *Expression*, se escribe la expresión a evaluar.
3. Para elegir el ámbito de la expresión a verificar, selecciona el procedimiento o módulo en la caja de contexto.
4. Para determinar cómo deseas que Visual Basic responda a la expresión, selecciona el tipo adecuado en la opción *Watch Type*.
5. Elige *OK*.



Tipos de monitoreo (Watch Types)

Existen 3 tipos de monitoreo en el ambiente de depuración de Visual Basic.

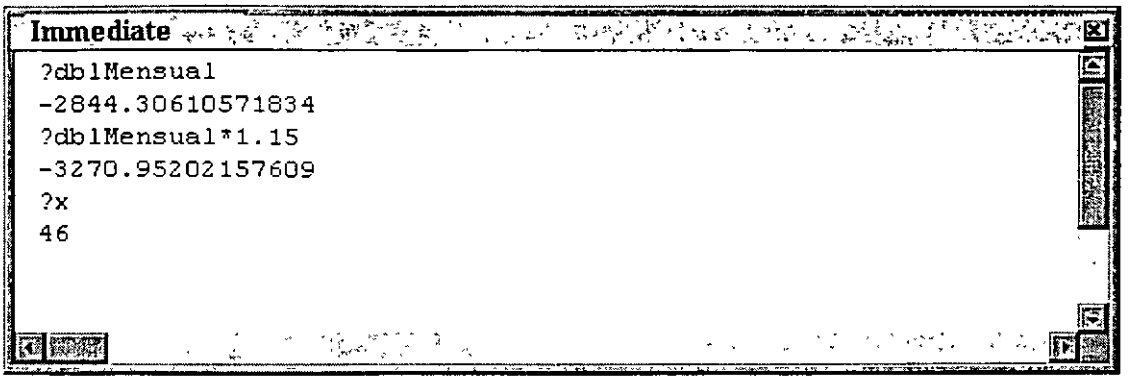
Watch Expression Una expresión de monitoreo, es una expresión cuyo valor será mostrado en la ventana Watch cuando el programa entre en modo Break.

Se puede dar una expresión de monitoreo desde el menú *Debug* ya sea escogiendo la opción *Add Watch* o *Edit Watch*. Esto puede ser hecho en modo de diseño o bien en modo de rompimiento(break). Otra tipo de expresión de monitoreo está dada por la opción *Quick Watch*, el valor de una expresión es mostrada en una caja de diálogo.

Break When Value is True Este tipo de monitoreo causará que Visual Basic entre en modo Break cuando el valor de la expresión se haga verdadera durante la ejecución del programa.

Break When Value Changes Cuando este tipo de monitoreo se elige, en cualquier caso en el que la variable o propiedad cambie con respecto a su valor inicial durante la ejecución del programa, se entrará en modo Break.

La ventana Inmediata (Immediate)



La ventana inmediata (immediate) muestra información que resulta de instrucciones de depuración que se han insertado en el código escrito directamente desde esta ventana.

Por ejemplo, si estamos depurando o experimentando con el código y se está probando en particular un procedimiento, evaluando expresiones o asignando nuevos valores a las variables o propiedades, se puede hacer eso directamente en la ventana inmediata.

Se pueden evaluar expresiones para saber sus valores preguntando por ellas directamente a través del comando Print o el signo "?" de la siguiente manera:

Print *variable*
    10
?x+y
    30
?text1.text
    John Smith

Recuerda que el ámbito de la ventana inmediata se limita al procedimiento actual, esto incluye a las variables locales, a las variables de módulo /forma. Cualquier variable fuera de éste ámbito no es desplegada.

Utilizando Debug.Print

Conforme el código se ejecuta se puede seguir la historia de los valores para verificar o corregir errores de lógica.

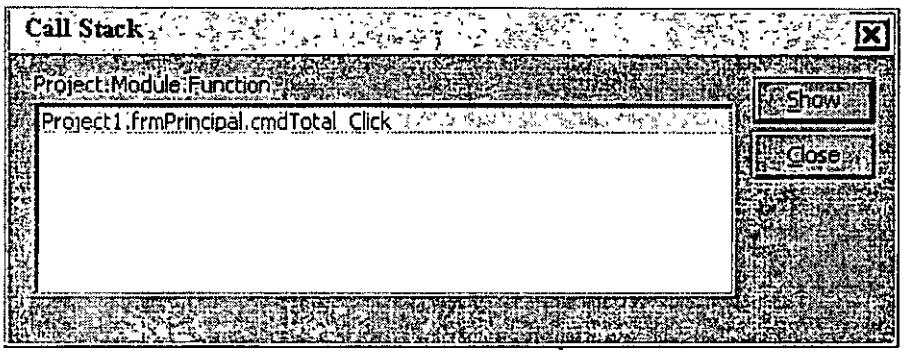
Para ello se puede utilizar el método Print del objeto Debug dentro del código para enviar la salida a la ventana inmediata. Esta técnica permite almacenar la historia de los valores de una variable o propiedad en la ventana inmediata; cuando la ejecución del código se interrumpa o bien concluya se pueden observar los valores enviados.

Ejemplo: La siguiente instrucción imprime el valor de la variable sSalario al panel inmediato cada vez que la instrucción es ejecutada.

```
Debug.Print "Salario =" & sSalario
```

Esta técnica trabaja mejor en un lugar en dónde en particular se sabe que el valor cambiará, por ejemplo dentro de un ciclo iterativo.

Siguiendo el flujo del programa a través de la caja de diálogo de llamadas a procedimientos (Call Stack)



La caja de diálogo Call Stack (llamadas a procedimientos) es una lista de procedimientos que marcan el flujo del código a través de múltiples procedimientos a los que se hacer referencia durante la ejecución del código. Utilizando esta herramienta se puede verificar si el código ha seguido la secuencia adecuada en la ejecución de los procedimientos.

Por ejemplo, un procedimiento llama a un segundo procedimiento, el cual a su vez llama a un tercero; todo lo anterior al primero debió haber sido completado, por lo que al tener procedimientos anidados se puede complicar el seguir el flujo, la caja de llamadas a procedimientos nos puede mostrar ese flujo.

Para mostrar la caja de llamadas a procedimientos

1. En la ventana *Locals*, elige el botón *Calls* que está en la esquina superior derecha de la caja de procedimientos.

3. Desde la barra de herramientas en el botón *Call*

TEMA VI:

Control en la ejecución de un Programa

Funciones comunes de Visual Basic

Visual Basic proporciona funciones predefinidas que permiten realizar muchas tareas de programación comunes. Estas funciones se pueden clasificar en las siguientes categorías:

- Conversión
- Fechas y hora
- Directorios y archivos
- Errores
- Finanzas
- Entrada y salida
- Matemáticas
- Manejo de cadenas

En la siguiente sección se tratan algunas tareas comunes que se llevan a cabo a través del uso de funciones.

Referencia: Para ver una relación de las funciones de Visual Basic por categoría, buscar en el 'Help' por "Keywords".

Funciones de Fechas y Hora

Las fechas y la hora son almacenadas internamente como números. Esto permite realizar cálculos con una fecha, como por ejemplo adicionar varios días a la fecha de hoy para generar una nueva fecha en el futruro:

Ejemplo

```
Dim dtNuevaFecha As Date
DtNuevaFecha = Date + 5
```

El valor 34800.75 representa una fecha y hora determinada: 11 de abril de 1995 a las 6:00 P.M. La parte del número que se encuentra a la izquierda del punto decimal representa una fecha entre el 1 de enero del año 100 y el 31 de diciembre del año 9999, inclusive. El valor a la derecha de el punto decimal representa una hora entre las 0:00:00 y las 23:59:59, inclusive. El mediodía se representa con 0.5.

Visual Basic proporciona funciones que regresan la fecha y la hora actual, así como para generar valores tipo fecha en base a una cadena u otra expresión.

Acción	Funciones
Obtener la fecha u hora actual	Date, Now, Time
Obtener la hora, minuto o segundo actual	Hour, Minute, Second
Obtener el día, mes, año o día de la semana actual	Day, Month, Year, Weekday
Regresar una fecha	DateSerial, DateValue
Regresar una hora	TimeSerial, TimeValue
Realizar cálculos con fechas	DateAdd, DateDiff, DatePart

Sintáxis

DateSerial (año, mes, día)
Year (fecha)
Month (fecha)
Now

El código siguiente calcula el último día del mes actual. Esto se realiza generando el número serial para el primer día del mes siguiente y substrayendo un día.

Ejemplo

```
Dim dtUltimoDiaMes As Date
DtUltimoDiaMes = DateSerial (Year (Now) , Month (Now) + 1, 1) - 1
```

Sintáxis

DateDiff (*intervalo*, *fecha1*, *fecha2* [, *primerdia_semana* [,*primerdia_anio*]])

La siguiente instrucción calcula el número de días entre semana (lun-vie) existentes entre el primer día del año actual y la fecha dada por el usuario en la caja txtDate:

Ejemplo

```
iNumDiasHabiles = DateDiff("w", DateSerial (Year(Now), 1, 1), _
        DateSerial (TxtDate.Text))
```

La función Format

La función Format regresa un número formateado en una cadena de acuerdo a la forma especificada:

Sintáxis **Format** (*expresión* [, *formato* [, *primerdia_semana* [, *primerdia_anio*]]])

El formato de argumento puede ser uno definido por el usuario, como "$#,###.00".

Referencia Para ver una relación completa de los formatos, buscar en el 'Help' por "User-Defined Date/Time Formats", "User-Defined Numeric Formats", y "User-Defined String Formats".

Se tienen también los formatos con nombre, los cuales toman en cuenta la notación internacional. Los formatos para fecha y hora en Visual Basic son los siguientes:

Nombre de Formato	Descripción
General Date	Despliega fecha y/u hora. Para números reales, despliega una fecha y hora (por ejemplo. 4/3/93 05:34 P.M.); si no tiene parte fraccional, despliega solamente una fecha (por ejemplo 4/3/93); si no tiene parte entera, despliega solamente la hora (por ejemplo, 05:34 P.M.). El despliegue de la fecha está determinado por la configuración del sistema.
Long Date	Despliega una fecha acorde al formato de fecha largo del sistema.
Medium Date	Despliega una fecha acorde al formato de fecha mediano, apropiado para la versión del lenguaje de Visual Basic.
Short Date	Despliega una fecha acorde al formato de fecha corto del sistema.
Long Time	Despliega una hora usando el formato de hora largo del sistema: incluye horas, minutos y segundos.
Medium Time	Despliega la hora en formato de 12-horas usando horas, minutos y segundos y la especificación AM/PM.
Short Time	Despliega una hora usando el formato de 24-horas (por ejemplo, 17:45)

Manejo de Cadenas

Las funciones para el manejo de cadenas se utilizan para obtener información de una cadena, extraer parte de la misma, o desplegar información en formato en particular.

Acción	Función
Comparar dos cadenas.	StrComp
Convertir a minúsculas o mayúsculas.	Format, Lcase, UCase
Crear una cadena con caracteres repetidos.	Space, String
Encontrar la longitud de una cadena.	Len
Formatear una cadena.	Format
Justificar una cadena.	Lset, RSet
Manejar cadenas.	InStr, Left, Ltrim, Mid, Right, Rtrim, Trim
Convertir cadenas.	StrConv

La función **Len** regresa el número de caracteres de una cadena. La función **Left** regresa un número específico de caracteres a la izquierda de una cadena.

Sintáxis **Len** (cadena)
Left (cadena, longitud)

El siguiente código elimina la extensión de archivo (los últimos cuatro caracteres) de un nombre de archivo.

Ejemplo

```
strFileName = Left (strFileName, Len (strFileName) - 4)
```

El siguiente ejemplo extrae el primer y último nombre de una cadena (por ejemplo, "Claudia" y "García" de la cadena "Claudia García".)

Ejemplo

```
Private Sub cmdReverse_Click ( )
    Dim strFirst As String
    Dim strLast As String
    Dim iSpace As Integer
    Dim iLastSpace As Integer
    Dim strName As String
    StrName = txtName.TEXT

    'se encuentra el último espacio en el nombre completo
    iSpace = 0
```

```
    Do
        ILastSpace = iSpace
        ISpace = InStr (iLastSpace + 1, strName, " ")
    Loop While iSpace <> 0

    'se separa el último nombre y todo lo que esté antes de él
    strFirst = Left ( strName, iLastSpace - 1)
    strLast = Right ( strName, Len (strName) - iLastSpace)

    'se invierte el nombre
    lblReversed.Caption = strLast & ", " & strFirst
End Sub
```

Referencia Para más información sobre funciones de Visual Basic para el manejo de cadenas, ver "String Manipulation" en el 'Help'.

Instrucciones Condicionales

Las instrucciones condicionales permiten evaluar condiciones y dependiendo de su resultado, realizar diferentes operaciones. Se tienen dos tipos de instrucciones condicionales en Visual Basic:

Instrucción **If ... Then ... Else**
Instrucción **Select Case**

Las dos instrucciones condicionales hacen básicamente lo mismo: ejecutar un bloque de instrucciones dependiendo del resultado que se obtenga al evaluar una condición específica. Sin embargo, las dos instrucciones varían un poco para utilizarse en distintos propósitos.

Instrucción If... Then... Else

La instrucción **If** es usada para evaluar si una condición es verdadera o falsa. Por ejemplo, se puede utilizar una instrucción **If** para probar si una contraseña del usuario es válida, comparando con la contraseña correcta:

Ejemplo

```
If UserPassword = "Orion" Then
   [Se le permite al usuario entrar al sistema]
Else
   [Se despliega un mensaje indicando que la contraseña es inválida]
End If
```

Este ejemplo sencillo compara la variable UserPassword con una cadena. Si las dos son iguales, la expresión es verdadera y entonces el código que le permite al usuario entrar al sistema es invocado. De otra manera, el usuario verá un mensaje indicándole que la contraseña es incorrecta.

El principio esencial de la instrucción **If... Then... Else** es: "Si (if) esto es verdadero, entonces (then) haz esto . . . " El resto de la sintáxis permite tener variaciones de la estructura general.

Sintáxis

```
If condición Then
   [instrucciones]
[Elseif condición-n Then
   [instrucciones elseif] | . . .
[Else
   [instrucciones else] |
End If
```

Note que si más de una de las condiciones en una instrucción **If** es verdadera, solamente la instrucción después de la primera condición verdadera es ejecutada.

Operadores Existen seis operadores que pueden usarse para definir la condición del If:

Operador	Significado
=	Igual
<>	Diferente
<	Menor que
<=	Menor o igual a
>	Mayor que
>=	Mayor o igual a

Ejemplo

```
If Salario < 5000 Then
    sTasa = 0.10
ElseIf Salario < 10000 Then
    sTasa = 0.20
Else
    sTasa = 0.50
End If
```

Note que las comparaciones clasifican primero los números más pequeños. Un salario de 2,000 es menor que 5,000 y también menor que 10,000.

Instrucción Select Case

Una estructura **Select Case** trabaja con una expresión que se evalúa una vez. El resultado de la expresión se compara con múltiples valores. Esta misma estructura puede ser escrita usando la instrucción **If**, pero el **Select Case** es más fácil de leer.

Ejemplo

```
Select Case UserName
      Case "Administrador"
            [Se le otorgan al usuario privilegios de administrador]
      Case "Usuario"
            [Se le otorgan al usuario privilegios a nivel usuario]
      Case Else
            [Se le otorgan al usuario privilegios de visitante]
End Select
```

Este ejemplo examina el contenido de la variable UserName. Se compara el nombre del usuario para determinar si se le asignan privilegios de administrador, usuario o visitante.

En muchas ocasiones se desea tomar una acción basados en el valor de una expresión. Tales problemas pueden ser resueltos con anidamientos de instrucciones **If**, pero usualmente resulta más sencillo hacerlo con instrucciones **Select Case**.

Sintáxis

```
Select Case  expresión
 [Case  lista-de-expresiones-n
     [instrucciones-n] ] . . .
 [Case  Else
     [instrucciones-else] ]
End Select
```

Una intstrucción **Select Case** realiza las acciones asociadas a una situación en particular identificada por cada caso (**Case**) específico.

En el siguiente ejemplo, la instrucción Select Case examina el valor devuelto por la función MsgBox y realiza una tarea basada en el botón presionado por el usuario.

Ejemplo

```
iBotonOpri  = _
    MsgBox (prompt := "Salvar cambios antes de salir ?", _
    buttons := vbYesNo)

Select Case  iBotonOpri
   Case vbYes
        SaveChanges
   Case vbNo
        Unload me
End Select
```

Instrucciones Iterativas

Las iteraciones ejecutan un grupo de instrucciones más de una vez dependiendo de la condición en particular que se plantea. Las dos estructuras iterativas que se tienen son: **Do** y **For...Next**.

Instrucción Do

La estructura iterativa **Do** ejecuta un bloque de instrucciones un indefinido número de veces mientras una condición exista. Se utiliza esta estructura cuando no se conoce el número de veces que el bloque de instrucciones se ejecutará, pero se puede definir una condición que determine cuándo parar.

Se tienen dos tipos de estructuras **Do**: una que verifica la condición antes de ejecutar el bloque de instrucciones, y otra que verifica la condición después de ejecutar el bloque. La versión que realiza la verificación antes de ejecutar el bloque se muestra en la siguiente sintáxis:

Sintáxis 1

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

En el siguiente ejemplo, una estructura **Do** se utiliza para encontrar todos los archivos de un directorio que concuerden con una especificación de archivo dada por el usuario (por ejemplo, "c:\windows*.txt).

El ejemplo utiliza la función **Dir**, la cual regresa el primer nombre de archivo que concuerda con un 'pathname' (especificación de archivo) dado. Para obtener los siguientes nombres de archivo que concuerden con la especificación, se llama nuevamente a la función **Dir** sin argumentos. Cuando ya no existen más nombres de archivo que concuerden, **Dir** regresa una cadena de longitud cero. La estructura **Do** provoca una condición verdadera al no existir más archivos y regresar la función **Dir** un valor cero.

Ejemplo

```
Msg = "Dar una especificación de archivo." 'Creando el mensaje
E_archivo = InputBox (Msg)                 'Leyendo la especificación
Nombre = Dir (E_archivo)                   'Primer nombre de archivo
Do Until Len (Nombre) = 0
    MsgBox Nombre      'Despliega el nombre concordante
    Nombre = Dir       'Encuentra la siguiente concordancia
Loop
MsgBox "Ya no hay más archivos que concuerden."
```

El segundo tipo de estructura **Do** verifica la condición después de ejecutar el bloque de instrucciones:

Sintáxis 2

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

Nota Las instrucciones dentro de la estructura **Do** se ejecutan al menos una vez; la condición determina si se reejecutan nuevamente.

Ejemplo

```
Do
    StrPassword = InputBox ("Escriba su contraseña")
Loop While strPassword <> "secret"
```

Instrucción For...Next

La estructura iterativa **For...Next** ejecuta un bloque de instrucciones un número específico de veces. En la sintáxis que se muestra enseguida, se puede notar que el valor del contador puede incrementarse o decrementarse dependiendo de que el incremento sea positivo o negativo.

Sintáxis

```
For contador = inicial To final [Step incremento]
    [instrucciones]
    [Exit For]
    [instrucciones]
Next [contador]
```

El siguiente ejemplo demuestra el uso de la estructura For...Next. Se genera una tabla de multiplicación de 10 x 10 y, usando el método Print, despliega la tabla:

El primer ciclo For...Next genera una nueva linea en la tabla. El segundo ciclo For...Next se encuentra anidado dentro del primero y genera cada columna dentro de un renglón.

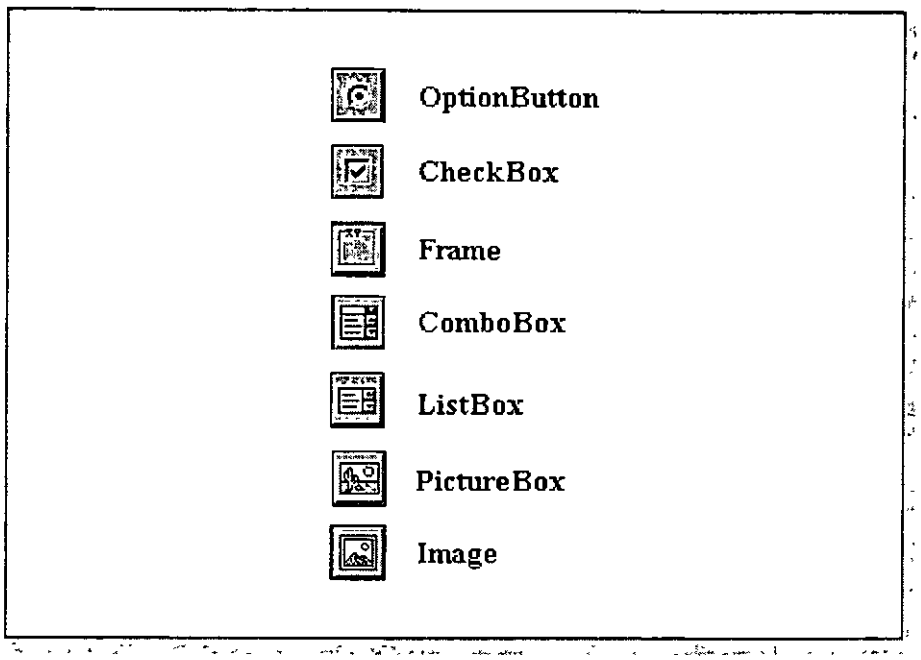
Ejemplo

```
Dim iRen As Integer, iCol As Integer
Dim strTabla As String
For iRen = 1 To 10              ' Ciclo exterior
    For iCol = 1 To 10          ' Ciclo interior
        'Concatenación del producto iRen * iCol en strTabla
        strTabla = strTabla & " " & Format (iRen * iCol, "@@@")
    Next iCol
    'Agregando un 'linefeed' y 'carriage return' al fin de la linea
    strTabla = strTabla & Chr (13) & Chr (10)
Next iRen
Me.Font = "Courier New"    'Font monoespaciado.
Me.Print strTabla          'Despliega strTabla en una forma.
```

TEMA VII

Controles

Controles Estándar



Los controles estándar mostrados arriba representan a los controles más comúnmente utilizados. Estos son:

OptionButton (Botón de Opción)

Un **OptionButton** es un control que puede ser activado o desactivado por el usuario. Estos controles son usualmente acompañados con otros controles **OptionButton** para formar un grupo del cual el usuario selecciona solamente uno.

CheckBox (Caja de Activado)

Un control **CheckBox** es también activado o desactivado por el usuario. Difiere de un control **OptionButton** en que éste es usado para presentar una opción individual o varias opciones diferentes, activando cualesquiera de ellas.

Frame (Marco)

Un control **Frame** es utilizado para agrupar visual o funcionalmente a varios controles que tienen una relación entre sí. Un uso muy común del control **Frame** es agrupar a varios controles **OptionButtons** en una forma.

ListBox (Caja de Lista)

Un control **ListBox** despliega una lista de elementos de los cuales el usuario selecciona uno o más.

ComboBox

Un control **ComboBox** combina las características de un control TextBox y un control ListBox. Los usuarios pueden escribir información en la parte de TextBox o seleccionar un elemento de la parte de ListBox del control. Un control **ComboBox** puede utilizarse también para crear un control ListBox drop-down.

PictureBox

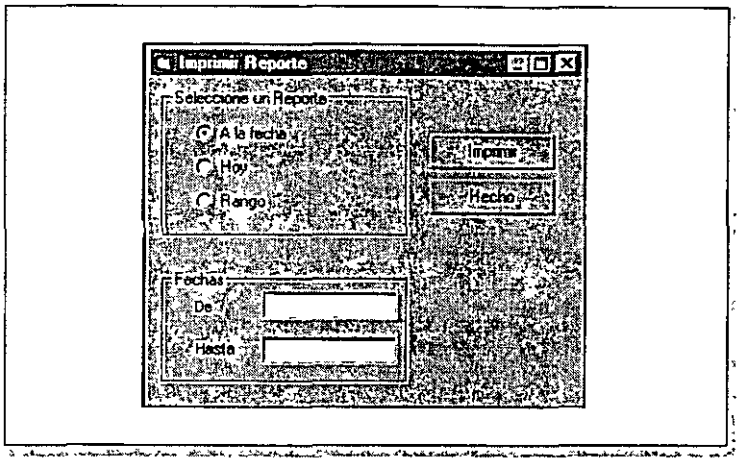
Un control **PictureBox** puede desplegar una gráfica o texto. Puede utilizarse también para crear gráficas animadas.

Image (Imagen)

El control **Image** es usado también para desplegar gráficas, pero utiliza menos recursos del sistema que un control **PictureBox**. También soporta menos funciones que un control **PictureBox**.

La siguiente sección muestra tareas frecuentes en las que se utilizan estos controles.

Uso de Botones de Opción con un Marco



Se utiliza el control OptionButton cuando se desea que el usuario seleccione solamente una opción de un grupo de opciones disponibles. Cuando se seleciona un botón de opción, todos los demás botones de opción dentro del grupo se deseleccionan. Un grupo de botones de opción pueden incluirse dentro de:

- Una forma
- Un control Frame (Marco)
- Un control PictureBox

Se puede utilizar el control Frame o PictureBox como un control "contenedor". El contenedor no solamente permite distinguir visualmente el grupo de botones de opción, también hace que los botones de opción funcionen independientemente de otros grupos en la forma. Esto significa que se pueden utilizar varios grupos de botones de opción en la misma forma.

Para crear un grupo de botones de opción

1.- Dibujar la Forma, el control Frame, o control PictureBox que contendrá los botones de opción.
2.- Dibujar los botones de opción dentro del control contenedor.

Nota Es importante dibujar primero el contenedor. Se dificulta agregar botones de opción al grupo después de haber sido dibujados.

Nota No se puede usar el método de doble-click para colocar un control **OptionButton** dentro de un control **Frame** o **PictureBox**. Al dar doble-click al control OptionButton en la Caja de herramientas (Toolbar) se colocará el control en el centro de la forma, en lugar de dentro del contenedor.

Arreglos de Controles

Una vez que los botones de opción son colocados dentro del contenedor, se puede entonces asignar, si se desea, un evento por cada botón de opción. Otro método, sin embargo, permite compartir el mismo evento entre varios controles del mismo tipo. Para utilizar este método, se deberá crear un arreglo de controles.

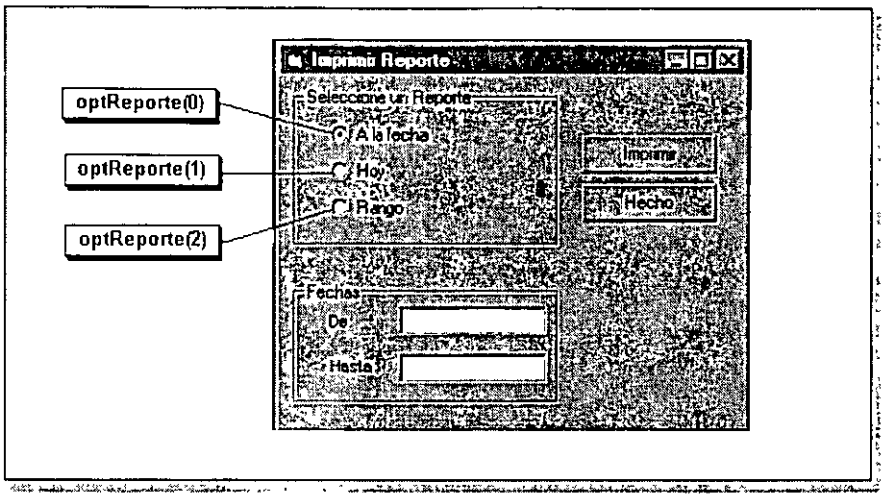
Un arreglo de controles es un grupo de controles que comparten el mismo:

- Tipo de objeto
- Nombre de control
- Evento

Código fácil de escribir y mantener Los arreglos de controles pueden hacer que su código sea más facil de escribir y mantener porque se puede escribir un solo evento que funcione para todos los controles del arreglo de controles. Por ejemplo, la forma la figura de arriba contiene un arreglo de controles consistenete de tres botones de opción. Cada control OptionButton tiene el mismo nombre, optReporte. Cuando cualquier botón de opción en el arreglo de controles es seleccionado, el mismo evento es llamado.

Código más eficiente Los arreglos de controles logran que el código sea más eficiente y mejoran el funcionamiento de la aplicación debido a que un arreglo de controles usa menos recursos del sistema que controles individuales.

Creación e Indización de un Arreglo de Controles



Para crear un arreglo de controles

1.- Dibujar el primer control y definir las propiedades iniciales que serán compartidas por todos los controles en el arreglo de controles.

2.- Definir la propiedad **Name** para el control.

3.- Hacer uno de dos:

- Seleccionar el primer control y copiarlo. Cuando se pegue el control regresar sobre la forma. Visual Basic preguntará si se desea un arreglo de controles
- Dibujar el siguiente control. Definir la propiedad **Name** a el nombre seleccionado para el primer control. Visual Basic preguntará si se desea un arreglo de controles.

4.- Continuar con el paso 3 hasta que todos los controles hayan sido agregados a la forma.

Indizando Miembros en el Arreglo de Controles

Cada control en el arreglo de controles tiene un valor único almacenado en su propiedad **Index**. El primer control creado tiene un valor de índice 0. Conforme un control es agregado, el valor del indice es incrementado en 1. Para referirse a un específico miembro dentro del arreglo de controles, se puede dar este formato:

Ejemplo optReporte (1)

Creación de un Evento para un Arreglo de Controles

```
Private Sub optReporte_Click (Index As Integer)
   Select Case Index
      Case 0,1
         fraDates.Enabled = False
         txtTo Enabled = False
         txtFrom Enabled = False
         lblTo.Enabled = False
         lblFrom Enabled = False
      Case 2
         fraDates.Enabled = True
         txtTo.Enabled = True
         txtFrom Enabled = True
         lblTo.Enabled = True
         lblFrom.Enabled = True
   End Select
   MsgBox optReporte (Index).Caption " fue seleccionado."
End Sub
```

Un evento para los controles en un arreglo de controles tiene un parámetro que dice cuál miembro llamó al evento:

Sintáxis **Sub** ControlArrayObjectName_**Click** (**Index As Integer**)

End Sub

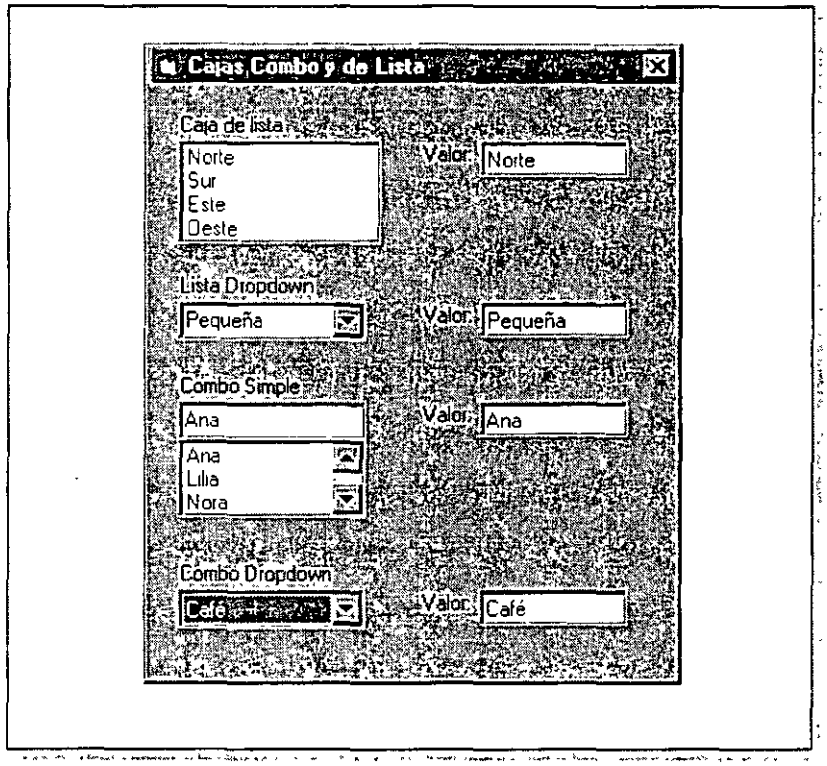
El parámetro Index es un entero que identifica únicamente al control que llamó al evento. Cuando se necesita referir a ese control, el parámetro Index puede utilizarse de varias maneras. Por ejemplo, se puede invocar una propiedad del control que invocó al evento:

Ejemplo `MsgBox optReporte (Index).Caption " fue seleccionado."`

El evento en la figura de arriba es asignado a los botones de opción en el diálogo "Imprimir Reporte". En éste, la propiedad **Enabled** de txtTo, txtFrom, frmDates, lblTo, y lblFrom se define como **True** o **False** dependiendo del botón de opción seleccionado. El usuario puede dar fechas en esas cajas de texto solamente si la opción Rango es seleccionada (Index = 2). Cuando las opciones "A la fecha" u "Hoy" son seleccionadas (Index = 0 ó 1), las cajas de texto del rango de fecha son deshabilitadas.

Tip Definiendo una variable igual al control que fue seleccionado permite referirse de manera más fácil a ese control más adelante dentro del procedimiento.

Controles ComboBox y ListBox



Los controles **ComboBox** y **ListBox** se utilizan para pedirle al usuario decisiones. Son cuatro las variaciones que estos controles tienen disponibles, cada uno con ligeras diferencias funcionales:

Control **ListBox**

Control **ComboBox** --- El tipo de control **ComboBox** es determinado por la propiedad **Style**:

- 0 - Combo 'Dropdown'
- 1 - Combo 'Simple'
- 2 - Lista 'Dropdown'

La principal diferencia entre una caja tipo-combo (combo simple y combo drop-down) y una caja tipo-lista (caja de lista y caja drop down) es que una caja combo permite escribir para dar una nueva selección, más que limitarse a únicamente los elementos que proporciona el programa.

Nota Al escribirle a la caja combo una nueva selección, el nuevo elemento no se agrega automáticamente a la lista. Se debe escribir código para ello.

El uso de una caja estilo drop-down minimiza la cantidad de espacio requerido para desplegarla inicialmente en la forma.

Trabajando con Listas

Más adelante se tiene algunas propiedades y métodos de los controles **ComboBox** y **ListBox**.

Construyendo una lista

Una lista es creada en tiempo de diseño definiendo la propiedad **List**. Para hacerlo en tiempo de ejecución, usar el método **AddItem**.

Sintáxis objeto.AddItem elemento, índice

Definiendo o Regresando la Selección Actual

Para definir u obtener qué elemento en la lista es seleccionado, usar la propiedad **ListIndex**. **ListIndex** regresa o define el índice del elemento, el cual es un número entre 0 (primer elemento) y el número total de elementos en la lista menos 1 (**ListCount** – 1). Si no se selecciona ningún elemento, **ListIndex** regresa –1.

La propiedad **NewIndex** regresa el índice del último elemento agregado a la lista. Esto es útil si se desea hacerle algo al elemento agregado,por ejemplo, definirlo como el elemento seleccionado de la lista:

Ejemplo
```
Combo1.AddItem "Oeste"
Combo.ListIndex = Combo1.NewIndex
```

Para obtener la cadena del elemento seleccionado en una caja de lista, aplicar la propiedad **ListIndex** a la propiedad **List**:

Ejemplo strSeleccion = lstMiLista.List (lstMiLista.ListIndex)

Para obtener la cadena de una caja combo, usar la propiedad **Text** del control:
Ejemplo strSeleccion = cmbMiLista.Text

Eliminando elementos de una Lista

Utilizar el método **RemoveItem** para eliminar un elemento de una lista.

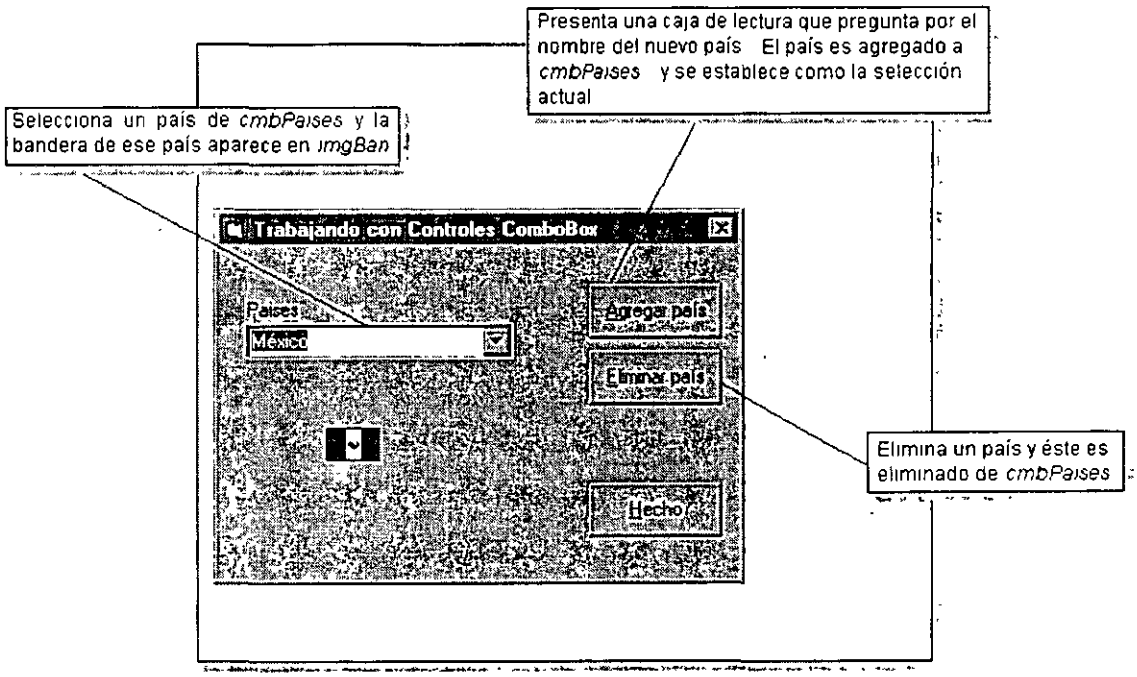
Sintáxis objeto.RemoveItem índice

Ordenando la Lista

Definir la propiedad Sorted como True para hacer que la lista aparezca en orden alfanumérico cuando sea desplegada en la forma. False despliega los elementos en el orden en que fueron agregados a la lista.

Referencia Para ver más información sobre propiedades y métodos de los controles, consultar "ComboBox" y "ListBox" en el 'Help'.

Ejemplo con ComboBox



El siguiente ejemplo muestra el uso del control ComboBox. El programa despliega la bandera de un país seleccionado de la lista. Se permite también agregar o eliminar paises de la lista.

Construyendo la lista inicial

La lista inicial es creada durante el evento Form_Load usando el método AddItem.

Ejemplo

```
Private Sub Form_Load ( )
    'Construye la lista inicial
    cmbPaises.AddItem "Canadá"
    cmbPaises.AddItem "Estados Unidos"
    cmbPaises.AddItem "México"
    cmbPaises.AddItem "Chile"
    cmbPaises.AddItem "Brasil"
    imfBan.Picture = imgNA.Picture .
End Sub
```

Cambio de bandera cuando un país es seleccionado

Cuando el usuario hace click sobre el nombre de un país en la caja combo, el nombre del país es pasado a una función llamada FindFile usando la propiedad Text. Esta función regresa el nombre de archivo de la imagen de bandera para el país seleccionado. Si el nombre de archivo no está disponible, aparecen la imagen de una bandera blanca y un mensaje. Si la imagen está disponible para el país seleccionado, ésta es cargada en imgFlag.

Ejemplo

```
Private Sub cmbPaises_Click ( )
    Dim NombreArchivo As String
    NombreArchivo = FindFile (cmbPaises.Text)
    If NombreArchivo < > "Not Available" Then
        ImgFlag.Picture = _
        LoadPicture ("C:\Program Files\" & _
            "Microsoft Visual Basic\Icons\Flags\" & _
            NombreArchivo)
        LblMessage.Visible = False
    Else
        ImgFlag.Picture = imgNA.Picture
        LblMessage.Visible = True
    End If
End Sub
```

Agregando paises a la lista

Cuando el botón 'Agregar Elemento' es seleccionado, aparece una caja que le solicita al usuario el nombre del nuevo país, strNuevoElem. Se agrega el nuevo país a la lista usando el método **AddItem** y entonces se define al nuevo elemento como la selección actual definiendo a su vez a la propiedad **ListIndex** con el valor que regresa **NewIndex**.

Ejemplo

```
Private Sub cmdAgregarElemento_Click ( )
   Dim strNuevoElem As String
   StrNuevoElem =InputBox (prompt:="Especifique Nombre del país")
   CmbPaises.AddItem  strNuevoElem
   CmbPaises.ListIndex = cmbPaises.NewIndex
End Sub
```

Nota La propiedad **Sorted** de la caja de lista debe quedar definida como **True** en tiempo de diseño para que la lista aparezca ordenada después de agregar un país.

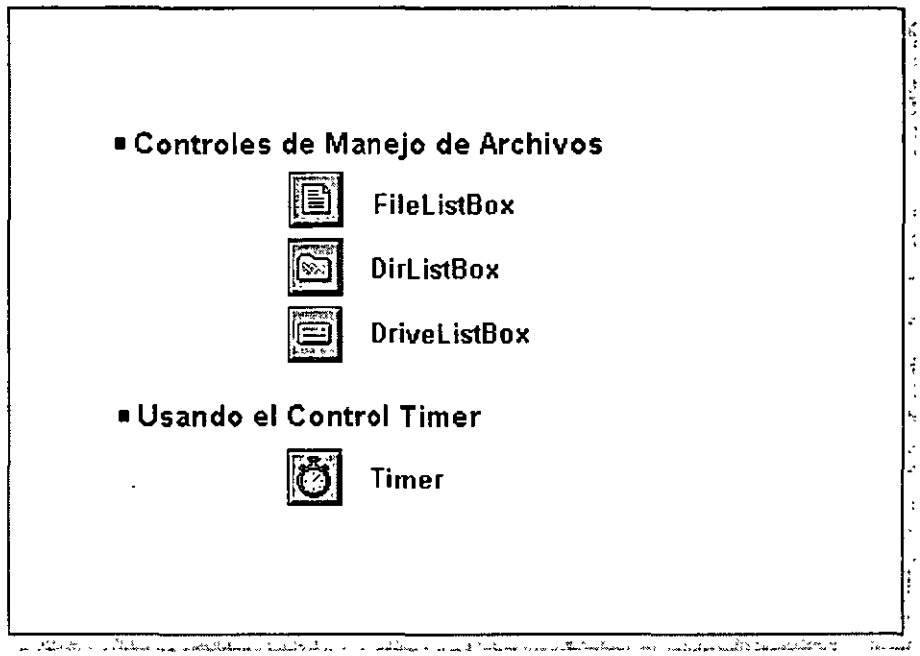
Eliminando paises de la lista

Cuando el botón 'Eliminar elemento' es seleccionado, el elemento actualmente selecionado es eliminado de la lista usando el método **RemoveItem**. La selección está determinada por el índice que regresa **ListIndex**.

Ejemplo

```
Private Sub cmdEliminarElemento_Click ( )
    CmbPaises.RemoveItem cmbPaises.ListIndex
    ImgFlag.Picture = imgNA.Picture
    LblMessage.Visible = False
End Sub
```

Controles Estándar Avanzados



Los controles estándar restantes son:

FileListBox

En tiempo de ejecución, un control **FileListBox** localiza y lista los archivos en el directorio especificado por la propiedad **Path**. Se utiliza este control para desplegar una lista de archivos seleccionados por tipo tipo de archivo.

DirListBox

Un control **DirListBox** despliega directorios y rutas en tiempo de ejecución. Se utiliza este control para desplegar una lista jerárquica de directorios.

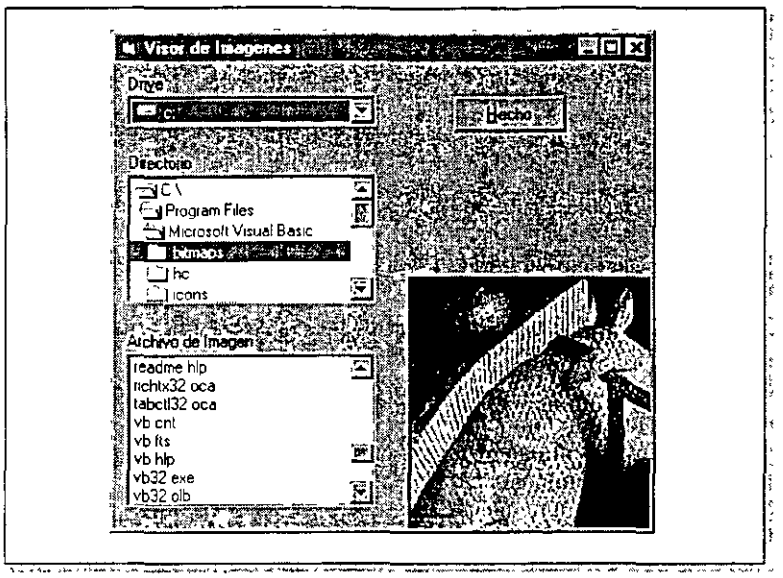
DriveListBox

Un control **DriveListBox** habilita al usuario para que seleccione un dispositivo (drive) de disco válido en tiempo de ejecución. Se utiliza para desplegar una lista de todos los dispositivos válidos en el sistema en el que trabaja el usuario.

Timer

Un control **Timer** puede ejecutar código en intervalos regulares causando un evento Timer.

Controles para Manipular Archivos



Visual Basic proporciona los controles **DriveListBox**, **DirListBox**, y **FileListBox** para poder implementar en las aplicaciones facilidades para la manipulación de archivos. Sin embargo, para hacer que los controles trabajen simultáneamente, se deberá escribir código que cambie ciertas propiedades que definan cómo el usuario interactuará con los controles.

Arriba se tiene un ejemplo de una aplicación que utiliza los tres controles. La aplicación 'Visor de Iconos' permite al usuario cambiar de dispositivos de disco (drives) y directorios y entonces desplegar el archivo .BMP o .ICO seleccionado en el control **FileListBox** sobre un control **Image**.

Cuando el usuario selecciona un 'drive' en el control **DriveListBox**, drvDrive, se define la propiedad de la caja de lista de directorios **Path** con la propiedad **Drive** de la caja de lista de drives:

Ejemplo

```
Sub drvDrive_Change ( )
    dirDirectorio.Path = drvDrive.Drive
End Sub
```

Cuando el usuario selecciona un directorio en la caja de lista de directorios, dirDirectorio, se define la propiedad **Path** de la caja de lista de archivos con la propiedad **Path** de la caja de lista de directorios:

Ejemplo

```
Sub dirDirectorio_Change ( )
    FilListaArch.Path = dirDirectorio.Path
End Sub
```

Finalmente, cuando el usuario selecciona un archivo en la caja de lista de archivos, filListaArch, carga la imagen en el control de imágenes:

Ejemplo

```
Sub filListaArch_Click ( )
    Imagen1.Picture = LoadPicture (dirDirectorio.Path _
    & "\" & filListaArch.FileName)
End Sub
```

El Control Timer

El control **Timer** se utiliza cuando se desea realizar una tarea en base al paso del tiempo. El control **Timer** trabaja lanzando un evento Timer que ocurre en un intervalo que se especifica.

Algunas propiedades importantes del control **Timer** son:

Propiedad	Descripción
Interval	La propiedad **Interval** es medida en milisegundos, y es representada por un valor desde 1 hasta 65535. Por ejemplo, un valor de 10000 milisegundos es igual a 10 segundos. El máximo, 65535 milisegundos, es equivalente a sólo un poco más de 1 minuto.
Enabled	La propiedad **Enabled** determina si el conmtrol **Timer** invocará al evento Timer en el tiempo especificado por la propiedad **Interval**.

En el siguiente ejemplo, el evento Timer es usado para actualizar la hora desplegada en un programa de un reloj digital.

Las siguientes propiedades del control **Timer** son definidas en tiempo de diseño:

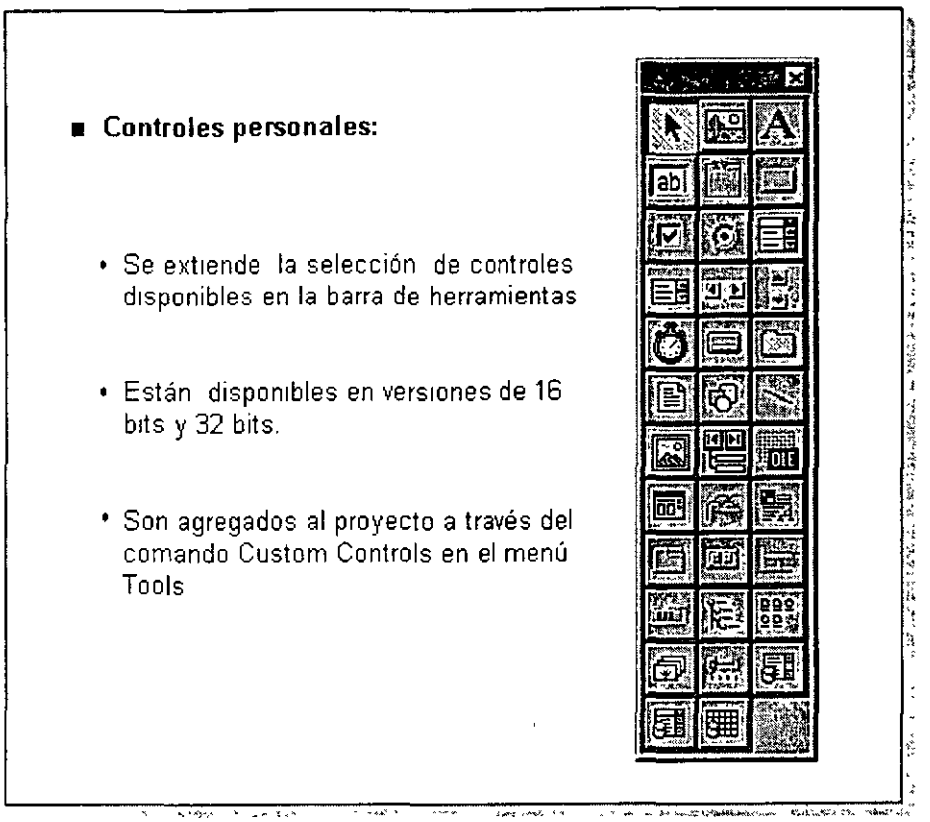
Propiedad	Valor
Enabled	True
Interval	1000 (igual a 1 segundo)

El evento Timer para el control **Timer** define la propiedad **Caption** de una etiqueta, con la hora vigente.

```
Private Sub Timer1_Timer ( )
Label1.Caption = Time   'Actualiza la hora desplegada
End Sub
```

Tip Un control **Timer** usualmente inicia deshabilitado. Este es habilitado por algún evento durante la ejecución del programa y deshabilitado de nuevo cuando el evento Timer es invocado.

Controles Personales



Un control personal es un control que se encuentra separado de los controles estándares que siempre aparecen en la caja de herramientas. Un control personal es un archivo que puede ser agregado a su proyecto. Una vez que un control personal es parte del proyecto, un botón es agregado a la caja de herramientas, por lo tanto se puede colocar el control personal, como una barra de herramientas o un diálogo tabular sobre la forma.

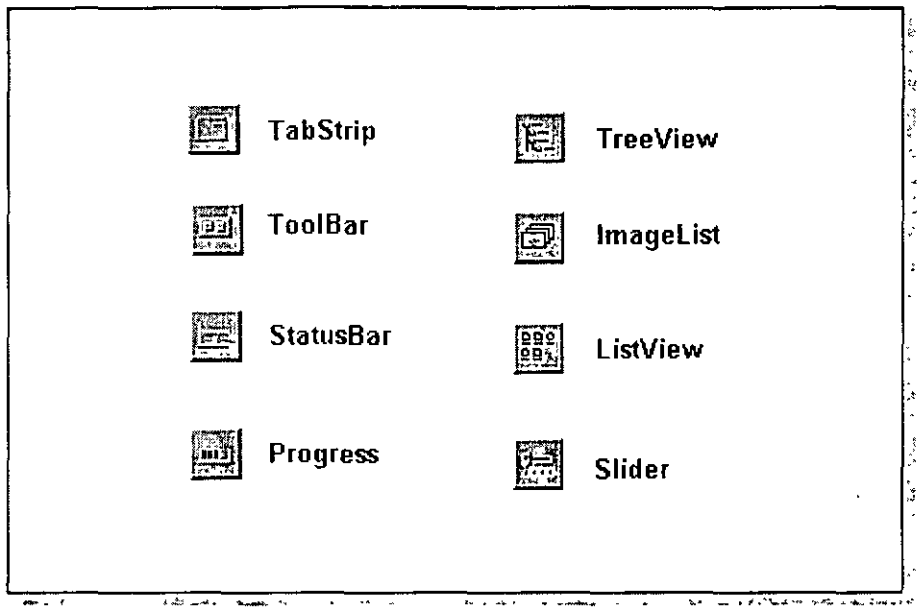
Un control personal tiene una extensión de archivo .VBX o .OCX. Los controles personales con la extensión .OCX toman ventaja de la tecnología OLE y pueden ser escritos como controles a 16-bits o a 32 bits. (Estos controles son referidos también como a controles OLE). Los controles personales con la extensión .VBX usan la vieja tecnología y son encontradas en aplicaciones escritas en versiones anteriores de Visual Basic.

Las reglas para usar cada tipo de control personal son como sigue:

Versión de Visual Basic	.VBX	.OCX (16 bits)	.OCX (32 bits)
Visual Basic 32 bits	No	No	Yes
Visual Basic 16 bits	Yes	Yes	No
Versiones anrteriores de VB	Yes	No	No

Nota Cuando Visual Basic abre un proyecto conteniendo un control .VBX, por default reemplaza el control .VBX con un control .OCX, pero sólo si una versión .OCX del control está disponible.

Controles Comunes de Microsoft Windows



Los controles comunes de Microsoft Windows son una colección de controles personales que permiten implementar la apariencia y funcionalidad de Windows 95 en sus aplicaciones Visual Basic.

Para agregar los Controles Comunes de Microsoft Windows a un proyecto

1.- Del menú Project, seleccione Components.
2.- Seleccione *Microsoft Windows Common Controls 6.0* de la lista de controles disponibles.
3.- Seleccione OK.

Nota Los Controles Comunes de Microsoft Windows son almacenados en el archivo mscomctl.ocx. Si se usan en un proyecto, el archivo mscomctl.ocx deberá incluirse con el archivo .EXE para dejarlo disponible a otros usuarios. Estos controles no están disponibles en una versión de 16 bits.

La colección de controles incluye:

TabStrip

Un 'tab strip' es análogo a los separadores en un cuaderno o a las etiquetas en un archivero. Se usa el control TabStrip para crear diálogos tabulares similares a aquellos que se encuentran a través del ambiente de Windows 95 – por ejemplo, en la caja de diálogo Display en el Panel de Control.

Toolbar

El control **Toolbar** facilita el agregar una barra de herramientas a una forma. Típicamente, los botones en una barra de herramientas corresponden a elementos en un menú de aplicación, proporcionando a los usuarios una manera más directa para tener acceso a un comando de la aplicación.

StatusBar

Una barra de estado (status bar) es un tipo de control desplegado a menudo al pie de una ventana de aplicación. Una barra de estado típicamente despliega información acerca del estado actual de la aplicación. La barra de estado de Microsoft Word, por ejemplo, despliega la página actual y número de sección, el conteo total de páginas, la hora, número de línea y más información.

Progress

Un indicador de progreso le puede permitir al usuario saber que la aplicación está procesando información y en qué punto se encuentra de dicho proceso. Cuando se copian o mueven archivos en Windows 95, un indicador de progreso da inmediatamente una idea acerca de la operación.

TreeView

Un control TreeView despliega una lista jerárquica de nodos, como los encabezados en un documento, las entradas en un índice, o los archivos y directorios en un disco. Un ejemplo de un control TreeView puede encontrarse al lado izquierdo del Explorador de Windows.

ImageList

El control **ImageList** contiene una colección de imágenes que pueden ser usadas por otros controles. El control **Toolbar**, por ejemplo, porta las imágenes de los botones de la barra de herramientas desde un control **ImageList**.

ListView

Una ventana de control **ListView** despliega una colección de elementos como archivos o carpetas. Un control **ListView** tiene cuatro modos de despliegue: Iconos grandes, Iconos pequeños, Lista y Detalles. Un ejemplo de un control **ListView** se encuentra en el lado derecho del Explorador de Windows. Desde el menú View, se pueden ver los diferentes modos para seleccionar el comando correspondiente.

Slider

Un control **Slider** contiene un deslizante y marcas opcionales. El usuario puede mover el deslizante arrastrándolo, dando click a cualquier lado de él, o usando el teclado. Los controles **Slider** son útiles cuando se desea que los usuarios seleccionen un valor discreto. Ejemplos del control **Slider** pueden encontrarse en la caja de diálogo de Propiedades del Mouse en el Control Panel.

Nota Los controles **Toolbar**, **ImageList**, y **Status** se discuten en detalle en el módulo 8.

Control CommonDialog

Se puede establecer un ambiente familiar en sus aplicaciones usando algunas de las cajas de diálogo proporcionadas por el sistema operativo. Aunque se puede contar con su propia caja de diálogo para abrir un archivo, al usuario de la aplicación le podría resultar ya confortable hacerlo con la caja de diálogo Open usada por muchas aplicaciones existentes, como Microsoft Office.

El control CommonDialog trabaja como una interfaz entre la aplicación y comdlg32.dll, la librería de ligado dinámico que genera los diálogos para Windows 95. Usando el control CommonDialog pueden desplegarse los siguientes diálogos:

- Color
- Font
- Help
- Open
- Printer
- Save

Para agregar el control CommonDialog a un proyecto

1.- Del menú Project, seleccione Components. .
2.- Seleccione *Microsoft Common Dialog Control 6.0* de la lista de controles disponibles.
3.- Dar click a OK.

Nota El control **CommonDialog** está contenido en el archivo de controles personales comdlg32.ocx. Si se usa en un proyecto este archivo, deberá incluirse con el archivo .EXE cuando se deje a disposición de otros usuarios.

Usando el control CommonDialog

El control CommonDialog no es visible al usuario cuando la forma está desplegada. El programa, sin embargo, tiene acceso completo al control y puede ser manipulado en un número de maneras.

El control CommonDialog tiene seis métodos y cada uno es usado para desplegar un diálogo diferente:

- ShowColor
- ShowFont
- ShowHelp
- ShowOpen
- ShowPrinter
- ShowSave

Por ejemplo, el siguiente código usa el método ShowOpen para desplegar la caja de diálogo Open:

Ejemplo CommonDialog1.ShowOpen

La caja de diálogo desplegada por uno de los métodos Show no causa un suceso en ese momento. Se deberá escribir código necesario para realizar la función, como abrir o salvar un archivo, o cambiar el font. Cada tipo de caja de diálogo tiene un conjunto de propiedades que se pueden usar para definir el estado inicial de la caja de diálogo o para determinar que es dada por el usuario.

Los siguientes ejemplos usan varias propiedades de las cajas de diálogo de archivo (Open y Save)

Sintáxis objeto.DialogTitle [=título]
objeto.Filter {= descripción1 | filtro1 | descripción2 | filtro2 ...]
objeto.FileName [= pathname]

Elk ejemplo define algunas de las propiedades de la caja de diálogo Open definiendo la propiedad DialogTitle y definiendo la propiedad Filter para desplegar solo esos archivos que tienen la extensión .BMP o .ICO. Se despliega entonces la caja de diálogo Open. Cuando se selecciona el botón Open, se verifica que la propiedad FileName tenga una extensión de archivo .BMP o .ICO y si es así, cargar la imagen en un control de imagen.

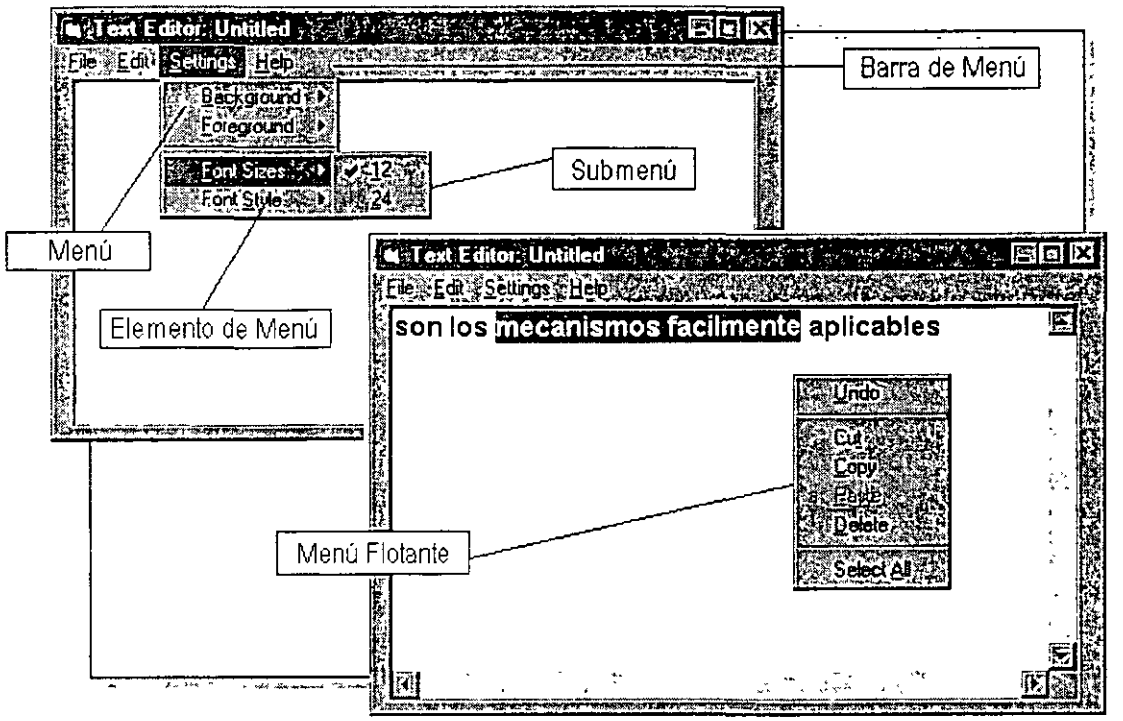
Ejemplo

```
Sub Command1_Click ( )
    CommonDialog1.DialogTitle = "Seleccione un nombre de archivo"
    CommonDialog1.Filter = "Pictures (*.bmp; *.ico) | *.bmp; *.ico"

    CommonDialog1.ShowOpen
    If CommonDialog1..FileName Like "*.bmp" _
        Or CommonDialog1.FileName Like "*.ico" Then
        Image1.Picture = LoadPicture (commonDialog1.FileName)
    Else
        MsgBox prompt:="El archivo seleccionado es incorrecto", _
        Buttons := vbOKOnly + vbExclamation
    End If
End Sub
```

TEMA VIII:

Menús, Barras de estado y Barras de herramientas



Terminología de Menús

Los siguientes términos se refieren a los elementos que normalmente se utilizan cuando se trabaja con menús en una aplicación.

Barra de Menús

La barra de menús siempre aparece debajo de la barra de título de la aplicación y contiene los encabezados de cada menú.

Menú

El menú contiene la lista de comandos que aparecen cuando se selecciona un elemento de una barra de menús. Esta lista incluye el título del menú en la primera posición.

Elemento de menú

Un elemento de menú, también llamado *comando*, se refiere a una de las opciones que se listan en un menú. Acorde a la pauta de diseño de interfaces de usuario estándar, cada menú deberá contener al menos un comando.

Submenú

Un submenú, o menú de cascada, es un menú que se despliega desde otro elemento de menú. El comando desde el cual se presenta el menú de cascada tiene una flecha enseguida que indica que un nuevo menú aparecerá cuando el usuario seleccione ese comando.

Menú flotante (Pop-up menu)

Un menú flotante es un menú latente que sólo aparece cuando se oprime el botón derecho del mouse en la aplicación. Un menú flotante contiene comandos que son comúnmente asociados con el objeto sobre el que se oprime el botón. Por ejemplo, si se oprime el botón derecho sobre un texto seleccionado, el menú flotante podría contener los comandos Cortar, Copiar, Pegar, y Borrar.

Usando el Editor de Menús

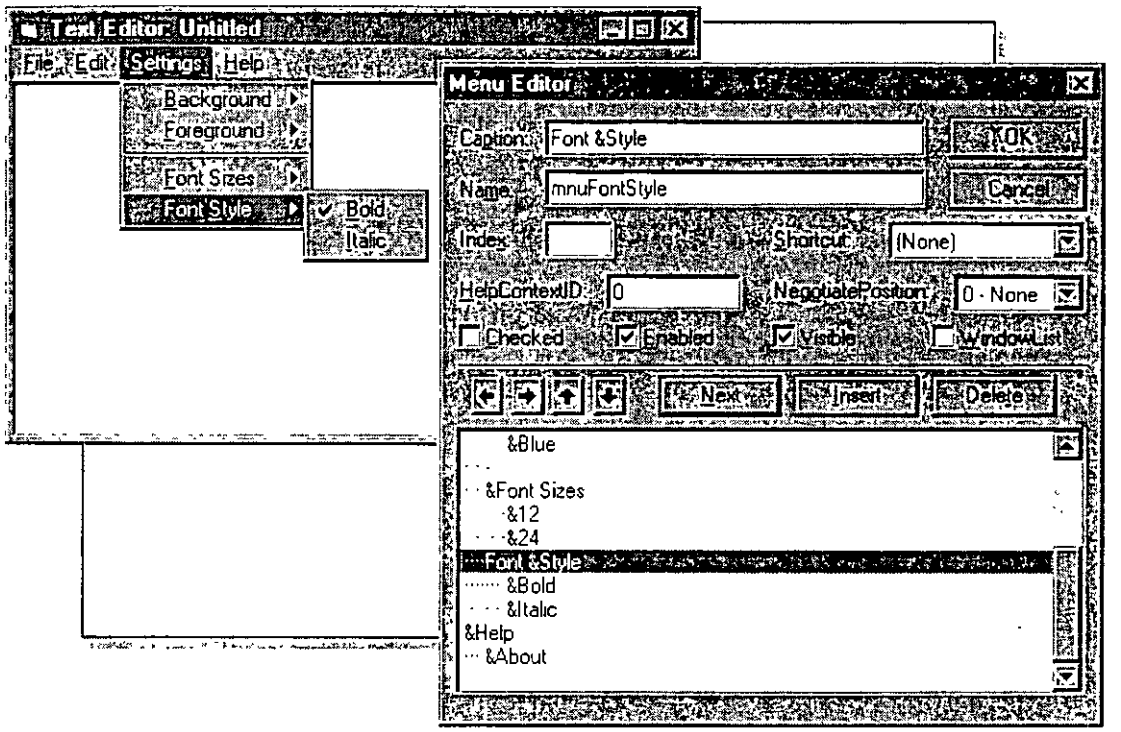
Se puede utilizar el editor de menús para crear un sistema de menús, comandos, submenús, y/o menús flotantes para ayudarle al usuario a manejar un programa de manera fácil e ituitiva.

Con el Editor de Menús, se podrá:

- Crear barras de menús, menús con comandos (elementos de menú), y menús flotantes (pop-up menus).
- Editar menús en el proyecto.
- Asignar propiedades a los elementos de un menú, tales como **Name** (nombre), **Caption** (título), **Checked**, **Enabled**, **Visible**, **NegotiatePosition**, **HelpContextID**, e **Index**.
- Asignar un tópico de Ayuda a un menú.
- Determinar si el menú aparecerá si la aplicación es activada por el parámetro *OLE in-place editing*.

Nota Cada barra de menú, menú, submenú, menú flotante, y elemento de menú se convierte en un control. Entonces se escribirá código que manipule las propiedades de esos controles y responda a sus eventos.

Referencia Para más información acerca de menús y Editor de menús, ver "Menu Design" en el 'Help'.



Agregando Elementos de Menú a una Forma

Los menús, elementos de menú, submenús, y elementos de submenús para una forma son desplegados en el Editor de Menús con un perfil del formato. Los elementos son sangrados para indicar su posición. Se puede dar o quitar sangría a un elemento seleccionado para cambiar su posición dentro de la jerarquía dando click sobre los botones de flecha izquierda o derecha.

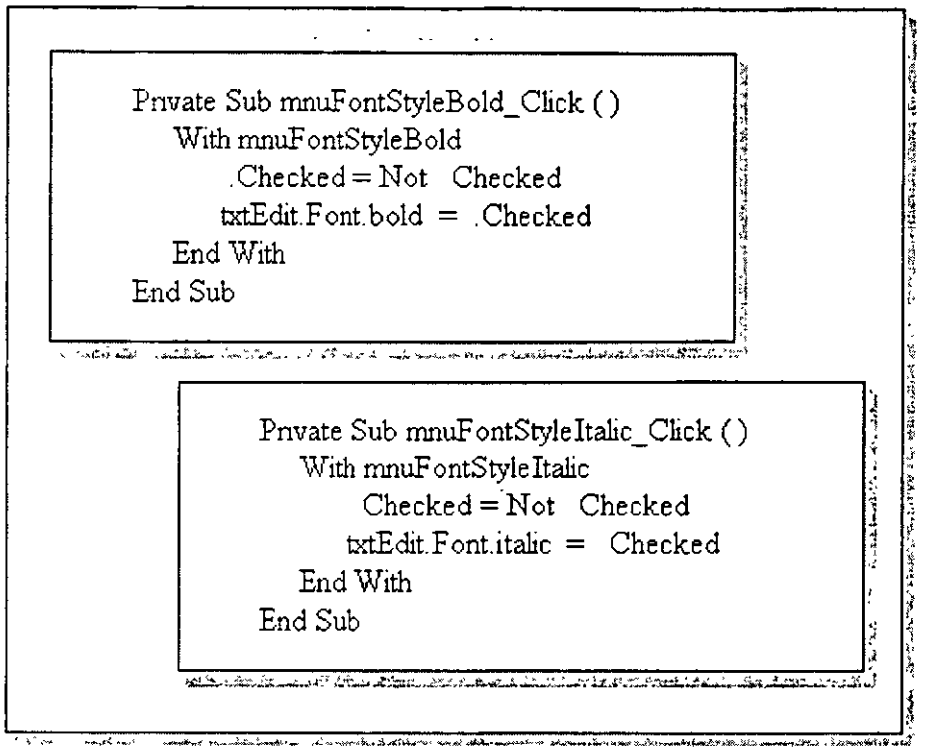
Nota Hasta cuatro niveles de submenús son permitidos, para un total de cinco niveles de menús. Aunque sean permitidos cinco niveles de menús, se recomienda limitar la jerarquía de menús a dos niveles solamente.

Agregando un elemento de un menú a una forma

1.- Activar la forma que contendrá los menús.
2.- Desde el menú *Tools*, seleccionar el comando *Menu Editor*.
También se puede seleccionar el botón *Menu Editor* de la barra de herramientas.
3.- En la caja *Caption*, teclee el nombre del menú o comando.
Se puede crear una línea de separación en el menú especificando aquí un guión (-).
Se puede diseñar también una tecla de acceso dando un ampersand (&) antes de cualquier letra del nombre. Por ejemplo, "&Salvar" o "Salvar &como".

4.- En la caja *Name*, teclee un nombre para el elemento de menú.
Se usará el prefijo *mnu* para el nombre del elemento de menú.
Tip Se puede tener un código más eficiente creando un arreglo de controles para los elementos de menú. Por ejemplo, definir todos los comandos de un menú con el mismo arreglo de controles, los elementos de un submenú con otro arreglo de controles, y así sucesivamente.
5.- Si se desea cambiar la posición jerárquica del elemento de menú, dar click sobre el botón con flecha derecha.
6.- Cambiar cualquiera de las propiedades en la ventana del Editor de Menú.
7.- Para agregar el elemento de menú a la barra de menús, seleccionar el botón *Next*.
8.- Repetir del paso 3 al 7 hasta que se haya completado el menú.
9.- Seleccionar el botón *OK* cuando se termine

Los botones de flecha arriba y abajo se utilizan para cambiar la posición de un elemento de menú dentro del mismo nivel de menú (arriba o abajo en la lista del menú). Se puede agregar un nuevo elemento de menú antes del elemento seleccionado, dando click al botón *Insert*.



```
Private Sub mnuFontStyleBold_Click ( )
    With mnuFontStyleBold
        .Checked = Not  Checked
        txtEdit.Font.bold  =  .Checked
    End With
End Sub
```

```
Private Sub mnuFontStyleItalic_Click ( )
    With mnuFontStyleItalic
        Checked = Not  Checked
        txtEdit.Font.italic  =  Checked
    End With
End Sub
```

Agregando Código a Elementos de Menú

Con los procedimientos de Visual Basic se pueden definir y cambiar las propiedades de las barras de menús, menús, elementos de menús, y submenús así como responder al evento Click del elemento de menú.

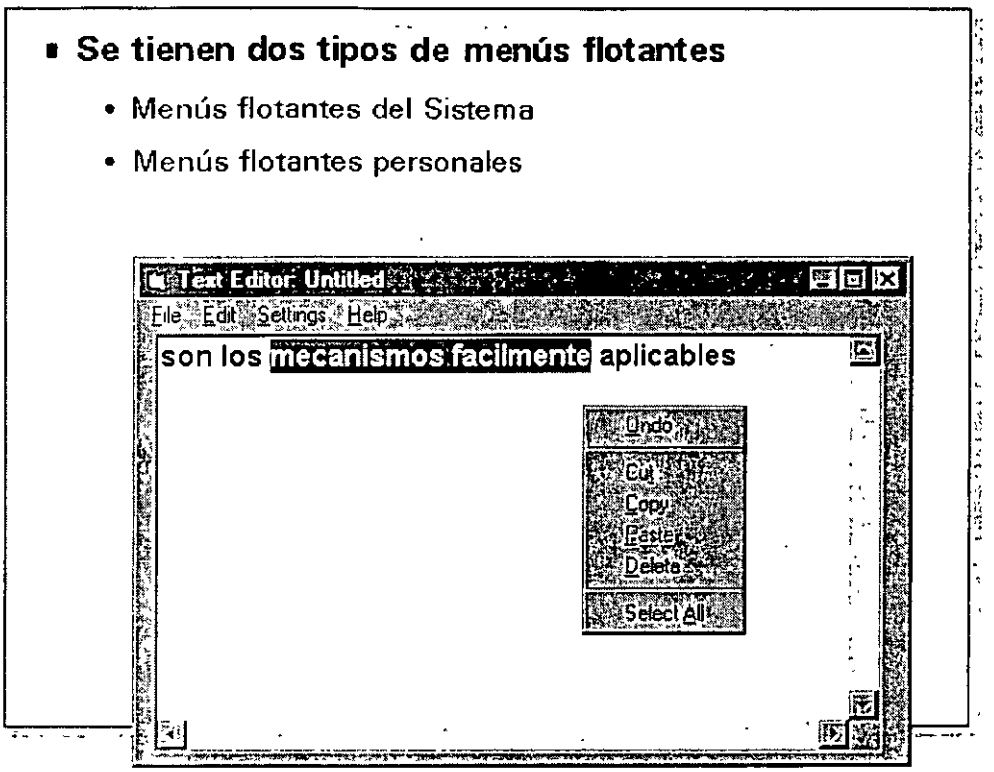
Con código de Visual Basic se puede, por ejemplo:

- Deshabilitar un elemento de menú usando la propiedad **Enabled**.
- Colocar marcas de checado enseguida de los elementos de menú usando la propiedad **Checked**.
- Mostrar u ocultar contenidos dinámicamente usando la propiedad **Visible**.

Tip Solamente se permite una barra de menú por cada forma. Se pueden simular múltiples barras de menús definiendo la propiedad **Visible** para varios menús con los valores **True** o **False**, dependiendo cuándo sea apropiado desplegar el menú.

En el ejemplo de arriba, los comandos Negrilla e Itálica en el menú Estilo de Letra ejecutan los procedimientos **mnuEstiloLetraNegrilla_Click** y **mnuEstiloLetraItalica_Click** cuando los comandos son seleccionados. Estos procedimientos cambian la propiedad **Font.Bold** o la propiedad **Font.Italic** de la caja de texto txtEdit, respectivamente. También colocan una marca de checado enseguida del comando si la propiedad es **True**.

Nota Cuando se use este tipo de código, tener cuidado de definir una inicialización. En el ejemplo de arriba, la definición inicial hace la letra en la caja de texto negrilla.



Menús Flotantes (Pop-up menus)

Un menú flotante es un menú que se basa en el contexto en el que se invoca. Windows 95, Microsoft Office, y Visual Basic utilizan menús flotantes que aparecen según el contexto en el que está el apuntador del mouse. Dando click al botón derecho del mouse se presenta un menú flotante. Los menús flotantes proporcionan los comandos más comunes para usarse en el contexto ya sea del apuntador del mouse o de la selección que se tenga.

En Visual Basic se tienen dos tipos de menús flotantes:

Menús Flotantes del Sistema

Estos menús son proporcionados automáticamente con ciertos controles. Por ejemplo, el menú flotante mostrado en la figura de arriba, es el menú flotante para un control **TextBox.** Este contiene comandos que típicamente se aplican a edición de texto, incluyendo Deshacer, Cortar, Copiar, Pegar, Borrar, y Seleccionar Todo. Todos los comandos en los menús flotantes del sistema son completamente funcionales (no requieren de código adicional). Sin embargo, no hay manera para modificar o deshabilitar los menús flotantes del sistema porque esto es una característica del sistema operativo.

Menús Flotantes Personalizados

Estos menús flotantes son creados de la misma manera que otros menús personalizados, utilizando el Editor de Menús; sin embargo, requieren de código para asociar el menú flotante con el contexto que se desee.

Creando y Desplegando un Menú Flotante Personalizado

Inicialmente, los menús flotantes se crean de la misma forma que en la que se crea cualquier otro menú, usando el Editor de Menús. Una vez que el menú es creado, se define la propiedad **Visible** con el valor **False**.

Para desplegar el menú flotante, se debe:

- Escribir código que responda al evento MouseUp del objeto que servirá como contexto para el menú flotante. En la figura de arriba, el objeto es una forma.
- Llamar al método **PopupMenu** de la forma desde el procedimiento del evento MouseUp para desplegar el menú.

El siguiente código llama a un menú flotante cuando se da click al botón derecho sobre la forma Form1.

```
Private Sub Form_MouseUp (Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    If Button = vbRightButton Then
        Form1.PopUpMenu mnuShortcut
    End If
End Sub
```

En este código, el evento MouseUp de la forma analiza el botón del mouse que se oprimió. Si el usuario utilizó el botón del mouse alterno, el método **PopupMenu** de la forma activa el menú mnuShortcut.

Sintáxis objeto.**PopupMenu** nombre_de_menú. flags. x. y. boldcommand

Por omisión, el menú flotante responderá sólo al botón izquierdo. Si se quiere que responda a otro botón, codificar el método **PopupMenu** como:

Ejemplo Form1.PopupMenu mnuShortcut vbPopupMenuRightButton

Barras de Estado

Anteriormente se trató ya una de las formas más simples de retroalimentar al usuario: Agregando una etiqueta a la forma que contiene instrucciones o información generada por el programa. Se pueden diseñar las formas de tal manera que el usuario sepa qué hacer al proporcionársele estos tipos de indicadores visuales. Los asistentes, una característica en los productos Microsoft Office, son un buen ejemplo de esto.

Algunas veces, sin embargo, no es factible (o necesario) incluir todas las instrucciones y documentación sobre la forma. Se pueden usar también, algunas de las facilidades que Visual Basic tiene para implementar otras formas más complejas para ofrecer retroalimentación al usuario de la aplicación.

El control **StatusBar** se utiliza para desplegar varios tipos de estado de datos. A diferencia de una caja de mensaje, una barra de estado despliega sus datos de una manera discreta, en una sección usualmente localizada al pie de la forma.

La barra de estado puede desplegar y actualizar ciertos tipos de información automáticamente.

Definir la propiedad **Style** para desplegar:

- El estado del teclado para las teclas BLOQ MAYUS, BLOQ NUM, INSERT, y BLOQ DESPL
- La hora.
- La fecha.

Las barras de estado pueden contener texto y/o un mapa de bits. Se pueden desplegar mensajes desde la aplicación en la barra de estado usando la propiedad **Text**

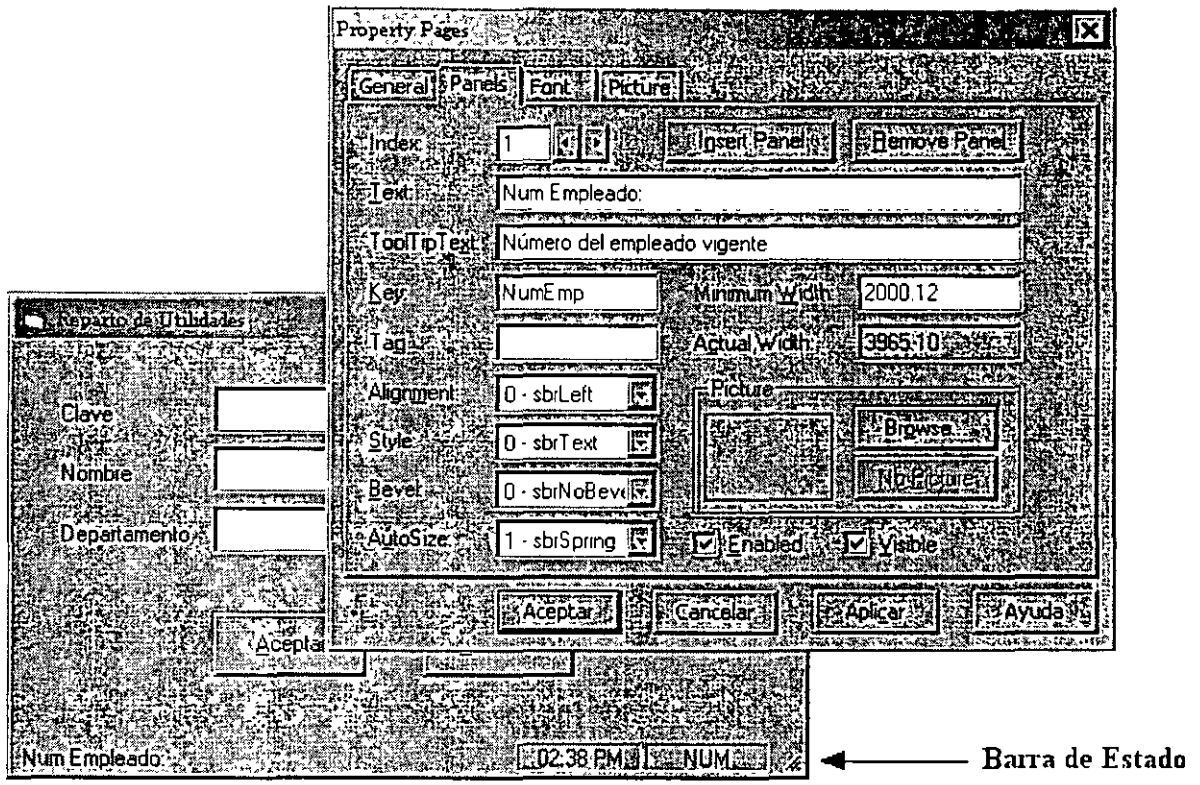
Nota El control **StatusBar** es parte de los Controles Comunes de Microsoft Windows y se encuentra disponible para la versión de 32 bits exclusivamente.

Para agregar un control StatusBar al proyecto

1.- Del menú *Project*, seleccionar *Components ...*
2.- De la lista de controles disponible active *Microsoft Windows Common Controls 5.0* y después seleccione *OK*.

Para desplegar la ventana de propiedades del control StatusBar

1.- Dibujar un control **StatusBar**.
2.- Dar click-derecho a la barra de estado y seleccione Properties del menú flotante.



Definiendo el texto de un panel en tiempo de ejecución

Para cambiar el texto que aparece en un panel de la barra de estado, se define la propiedad **Text**. El siguiente ejemplo despliega el número de empleado en una caja de texto conforme se van procesando:

Ejemplo

```
StatusBar1.Panels ("NumEmp").Text = "Num Empleado: " & _
                txtNum.Text
```

Si la barra de estado contiene solamente un panel (la propiedad **Style** se define como sbrSimple), se puede definir el texto usando la propiedad **SimpleText**:

Ejemplo

```
StatusBar1.SimpleText = "Procesando, por favor espere . . ."
```

Referencia Para más información sobre el control **StatusBar**, ver "Visual Basic Custom Control" en el 'Help'

Barras de Herramientas

Una barra de herramientas típicamente contiene botones que corresponden a elementos de un menú de la aplicación, proporcionándole al usuario una interface gráfica para que acceda a los comandos y funciones más frecuentemente utilizados.

Una barra de herramientas se agrega a la aplicación colocando dos controles sobre la forma: un control **Toolbar** y un control **ImageList**. Estos controles son parte de los Controles personales Comunes de Microsoft Windows.

- El control **Toolbar** contiene una colección de objetos **Button**. El código es asignado al evento Click de los botones. (Un evento Click es compartido por todos los botones). La imagen para cada botón es almacenada en un control **ImageList**.

- El control **ImageList** contiene una colección de objetos ImageList, o imágenes, cada una de las cuales puede ser referida por su índice o tecla. No tiene significado usar solo el control ImageList. Se le utiliza como recipiente central para suministrarle a otros controles que requieren de imágenes.

Nota Los controles **Toolbar** e **ImageList** son parte de los Controles Comunes de Microsoft Windows y están disponibles en versión de 32 bits solamente.

Para agregar los controles Toolbar e ImageList al proyecto

1.- Del menú *Project*, seleccionar *Components ...*
2.- De la lista de controles disponible active *Microsoft Windows Common Controls 5.0* y después seleccione *OK*.

Implementando el Control Toolbar

Para implementar una barra de herramientas en una aplicación, se siguen los siguientes pasos:

Para construir una lista de imágenes

1.- Agregar un control **ImageList** a la forma.
2.- Desplegar las caja de propiedades dando click con el segundo botón sobre el control y seleccionando *Properties* desde el menú flotante.
3.- Seleccionar un tamaño de imagen desde la pestaña *General.*
El tamaño estándar del botón es de 16 x 16 pixeles.

Tip Visual Basic tiene un editor de imágenes que se puede utilizar para crear y editar imágenes. Esta aplicación se encuentra en el CD-ROM de Visual Basic bajo la carpeta \tools\imgedit.

4.- Inserte el número deseado de imágenes en la pestaña *Images.*
5.- Seleccione *OK.*

Para crear una barra de herramientas

1.- Agregar un control **Toolbar** a la forma.
2.- Desplegar la caja de propiedades para el control **Toolbar**.
3.- Sobre la pestaña General, definir la propiedad **ImageList** al control **ImageList**.

Tip Seleccionando la opción AllowCustomize sobre la pestaña General se habilitará una caja de diálogo de edición de barras de herramientas que aparece cuando el usuario da doble click sobre un área vacía de la barra de herramientas.

4.- Crear los botones utilizando la pestaña Buttons.
5.- Asignar las siguientes propiedades a cada botón:

Propiedad	Descripción
Key	Regresa o define una cadena que identifica únicamente un miembro en una colección..
ToolTipText	Regresa o define un ToolTip
Image	Regresa o define un valor que especifica qué objeto ImageList en un control ImageList.utilizar con otro objeto.

6.- Seleccionar *OK.*

Para asignar código al evento ButtonClick

Una vez que la barra de herramientas está en un lugar, se le asigna código al evento ButtonClick del control **Toolbar**.

El evento ButtonClick regresa el parámetro Button, el cual es el botón que llamó al procedimiento.

Ejemplo

```
Sub tlbToolbar1_ButtonClick (ByVal Button As Button)
   Select Case Button.Key
      Case "Nuevo"
         [Se abre un nuevo archivo]
      Case "Abrir"
         [Se abre un archivo existente]
      Case "Guardar"
         [Se salva el archivo]
   End Select
End Sub
```

TEMA IX:

Acceso a datos con el control DATA

El Acceso a Datos en Visual Basic

Data Access (acceso a datos) es el proceso que se refiere a visualizar o manipular información que se origina en una fuente de datos externa, como lo sería Microsoft Access o Microsoft SQL Server. Las facciones de un *Data Access* pueden ser incorporadas a un programa Visual Basic utilizando:

Control Data

Proporciona acceso limitado a bases de datos existentes sin programación.

Data Access Objects (DAO)

Proporciona una completa interfaz de programación que permite un control total de la base de datos.

Librerías ODBC

Habilita al usuario para llamar a la *interfaz de programación de aplicaciones* (API, por sus siglas en inglés) directamente.

Librerías Visual Basic SQL (VBSQL)

Proporciona una liga directa a Microsoft SQL Server.

Este módulo proporciona una introducción a una de las formas más sencillas para tener acceso a datos (data access), con el control **Data**. Los otros métodos para el acceso a datos se consideran tópicos avanzados y no son cubiertos en este manual.

¿Cómo Visual Basic Accede a Bases de Datos?

Se tienen tres categorías de bases de datos a las que se puede tener acceso desde Visual Basic:

Bases de datos Jet

Estas bases de datos son creadas y manipuladas directamente por el 'Jet engine' y proporciona máxima flexibilidad y velocidad. Microsoft Access y Visual Basic utilizan el mismo 'Jet engine' para bases de datos.

Bases de datos ISAM

Las bases de datos ISAM (Método de Acceso Secuencial Indexado) viene en varios formatos, incluyendo Btrieve, DBASE, Microsoft Visual FoxPro, y Paradox, entre otros. Se pueden crear o manipular todos estos formatos en Visual Basic.

Bases de datos compatible con ODBC (Open Database Connectivity)

Estas incluyen bases de datos cliente-servidor que conforman el estándar ODBC, como Microsoft SQL Server y Oracle. A cualquier base de datos que soporte ODBC se puede tener acceso desde Visual Basic.

La Base de datos Northwind

La base de datos Nwind.mdb es una base de datos que se incluye en Microsoft Access.

Una Base de datos está compuesta de Tablas

El modelo de base de datos relacional presenta los datos como una colección de tablas. Una *tabla* es un agrupamiento lógico de información relacionada. Por ejemplo, la base de datos Northwind tiene una tabla que lista todos los empleados y otra tabla que lista todas las órdenes.

Las Tablas pueden tener una Llave Primaria

Cada tabla tendrá una *llave primaria*. La llave primaria es un campo o combinación de campos único para cada renglón en la tabla. Por ejemplo, el campo de la Clave de Empleado es la llave primaria para la tabla de empleados porque ésta identifica a cada empleado en la compañía y dos empleados no pueden tener la misma clave.

Una tabla puede contener también campos que son *llaves foráneas*. Por ejemplo, en la base de datos Northwind, en lugar de duplicar toda la información de los clientes en cada orden, la tabla de órdenes contiene un campo con la Clave de identificación del cliente. El campo

con la Clave de identificación del cliente en la tabla de órdenes es llamado *llave foránea* porque éste relaciona a la llave primaria de una tabla "foránea" (Clientes).

La relación entre Ordenes y Clientes es una relación uno-a-muchos ya que, por cada orden, se tiene sólo un cliente. Sin embargo, un cliente puede tener muchas órdenes.

Componentes de una Tabla

Las tablas se componen de *renglones* y *columnas*. En una base de datos Jet, los renglones son referidos como *registros* y las columnas son referidas como *campos*.

Registros

Un registro en una tabla contiene información acerca de un artículo. Por ejemplo, un registro en la tabla de empleados, tendrá información sobre un empleado en particular. Generalmente, no se necesitarán tener dos registros en una tabla que tengan exactamente los mismos datos.

Campos

Cada campo en una tabla contiene un dato simple de información. Por ejemplo,. La tabla de Empleados tiene campos para Clave de empleado, Nombre, Dirección, Teléfono, y así sucesivamente.

El control Data

El control Data permite fácilmente ligar una forma de Visual Basic a una base de datos. Con el control Data, se puede crear una aplicación que despliegue y actualice datos desde una base de datos con muy poco código.

Usando el control Data

A fin de regresar un conjunto de registros usando el control Data, se deberán definir tres de sus propiedades. Estas propiedades pueden definirse en tiempo de diseño con la ventana Properties o en tiempo de ejecución en el evento Form_Load.

1. **Connect**
   La propiedad **Connect** establece el tipo de base de datos. Esta podría incluir Microsoft Access (el default), Microsoft Visual FoxPro, dBASE, Paradox, o un número de otros tipos de bases de datos.

   Si se está dfiniendo la propiedad **Connect** en tiempo de ejecución, el código podría aparecer como sigue:

   Ejemplo Data1.Connect = "Access"

2. **DatabaseName**
   La propiedad **DatabaseName** determina la base de datos utilizada por el control **Data**.

   Si se conecta a una base de datos Microsoft Access, definir la propiedad **DatabaseName** a el archivo .MDB. Todas las tablas para una base de datos Microsoft Access son contenidas dentro de un solo archivo .MDB. Si se está conectando a una base de datos dBASE, Paradox o Btrieve, definir la propiedad **DatabaseName** al directorio que contiene los archivos de la base de datos. Estos tipos de bases de datos almacenan cada tabla en un archivo separado.

   Para definir la propiedad **DatabaseName** en tiempo de ejecución, usar:

   Ejemplo Data1.DatabaseName = "C:\VB\NWIND.MDB"

3. **RecordSource**
   La propiedad **RecorSouce** puede ser una tabla individual en la base de datos, un 'query' almacenado, o una cadena 'query' que use el lenguaje SQL (que se discute más adelante en este capítulo).

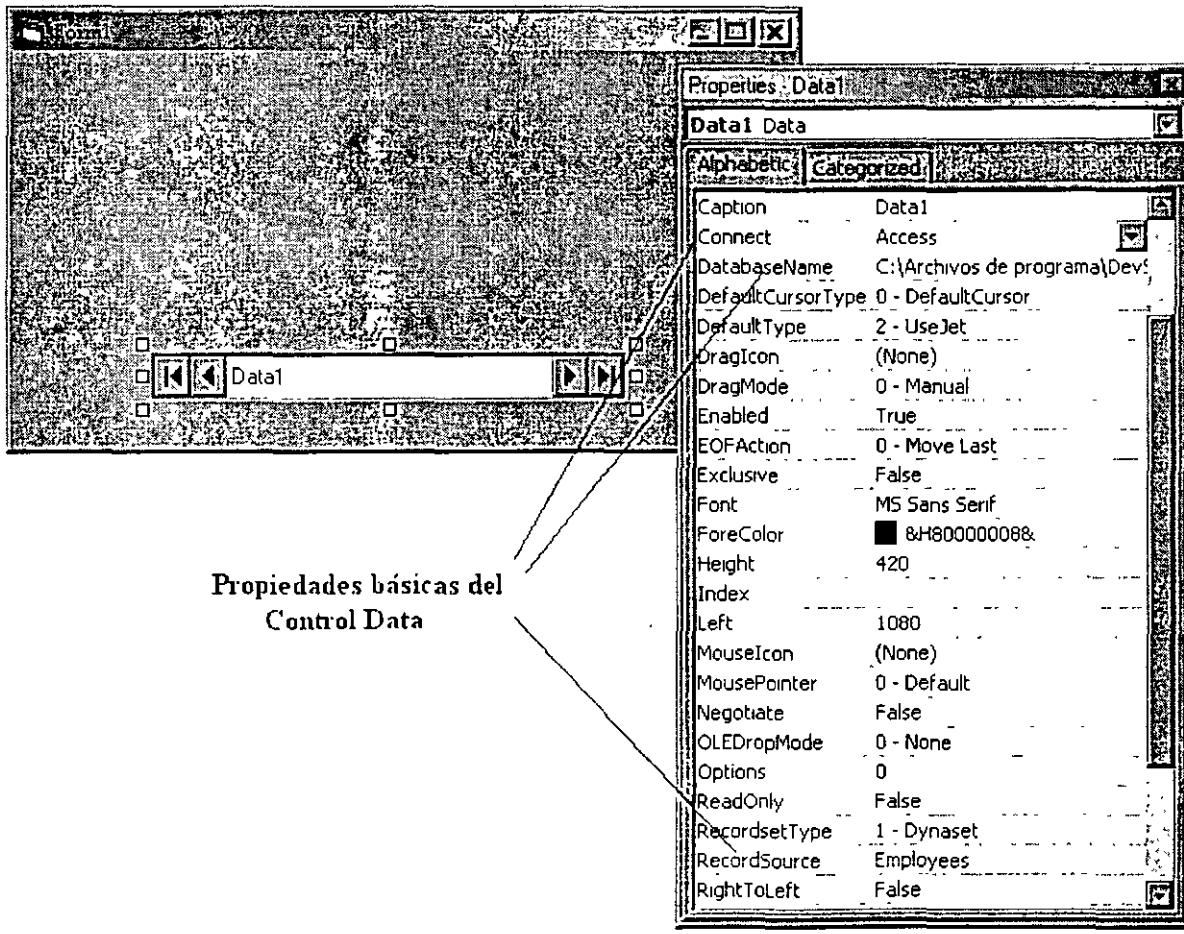
La propiedad **RecordSource** puede definirse en tiempo de ejecución como sigue:

Ejemplo Data1.RecordSource = "Empleados"

Tip Para mejorar la eficiencia, evitar definir la propiedad **RecordSource** con una tabla entera. Definir esta propiedad con una cadena SQL que tome los registros necesarios.

Cuando la propiedad **RecorSource** es cambiada en tiempo de ejecución, se deberá invocar al método **Refresh** del control **Data** para tomar el nuevo conjunto de regsitros.

Ejemplo Data1.Refresh



¿Qué es un Recordset?

Definiendo la propiedad **RecordSource** de un control **Data,** se crea un 'recordset' de registros desde ese registro fuente.

Objeto que contiene un conjunto de registros desde el control Data

El conjunto entero de registros al que se refiere un control Data es llamado el 'recordset'. Después de que una base de datos ha sido abierta, la propiedad **Recordset** del control **Data** contiene el 'recordset'.

El Recordset tiene propiedades y métodos

La propiedad **Recordset** del control **Data** es un objeto. Por lo tanto, tiene propiedades y métodos como cualquier otro objeto.

El 'recordset' tiene propiedades nombradas **BOF** y **EOF** que indican si se está antes del inicio o al final del 'recordset'. Los valores de ambos BOF y EOF son True si no hay registros en el 'recordset'.

El objeto Recordset tiene un método **AddNew** para agregar un registro al 'recordset':

Ejemplo Data1.Recordset.AddNew

Viendo Registros con Controles Ligados

Una vez que las propiedades **Connect**, **DatabaseName**, y **RecordSource** para el control **Data** son definidas, se puede agregar un control para desplegar los datos del 'recordset' en la forma.

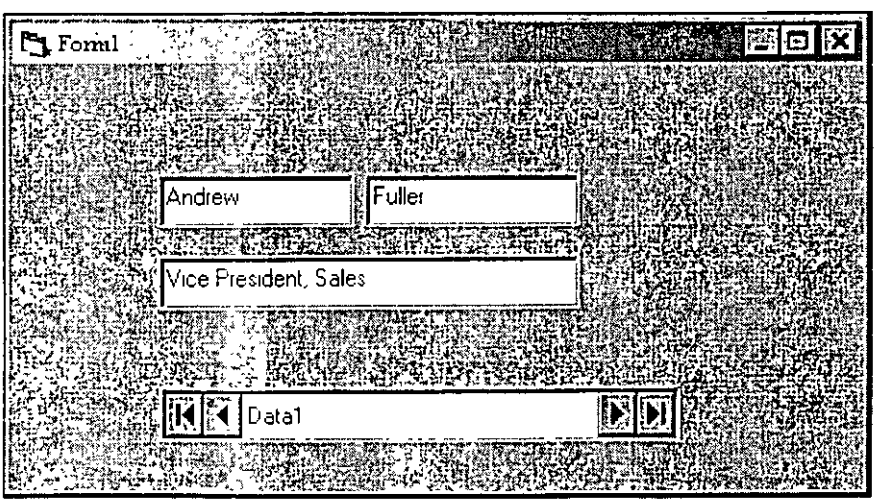
Un control que se asocia al control **Data** es llamado un control ligado. Cuando un control **Data** se mueve de un registro al siguiente, ya sea con código o dando click a las flechas del control **Data**, todos los controles ligados conectados al control **Data** cambian para desplegar los datos de los campos en el registro actual. Además, si el usuario cambia los datos en el control ligado, esos cambios son automáticamente fijados a la base de datos cuando el usuario se mueve a otro registro. Solamente un registro de información puede ser editado y actualizado a la vez. Este es el registro actual.

Para ligar un control a un control **Data**, se deberán definir dos propiedades del control ligado:

1. **DataSource**
   La propiedad **DataSource** especifica el control **Data** a través del cual el control es ligado a la base de datos – por ejemplo, Data1. La propiedad **DataSource** deberá ser definida en tiempo de diseño a través de la ventana de propiedades.

2. **Datafield**
   La propiedad **Datafield** indica cuál campo es desplegado en el control ligado. La propiedad **Datafield** puede definirse en tiempo de diseño o de ejecución. El siguiente es un ejemplo de cómo se define la propiedad con código:

Ejemplo txtLastName.Datafield = "Last Name"



Controles Ligados Disponibles

Los controles ligados pueden pulir y facilitar el uso de los programas de acceso a datos. En lugar de pedirle al usuario que escriba cada valor para un campo, se puede proporcionar una lista de todos los posibles valores de los cuales se puede escoger. O se puede proporcionar una caja de activación (check box) que aparezca activada o desactivada en vez de una caja de edición que despliega "True" o "False". Debido a que el valor del control ligado es automáticamente tomado de y escrito a la base de datos, hay poca o nula programación relacionada.

Hay 10 controles ligados en Visual Basic. Los controles asociables al control **Data** que se encuentran en la caja de herramientas estándar incluye:

- Label
- TextBox
- CheckBox
- ComboBox
- ListBox
- PictureBox
- Image

Los otros controles ligados son agregados al proyecto como controles personales. Los controles *Microsoft Data-Bound List* incluyen:

DBCombo
DBList

El control Microsoft Grid incluye:

- DBGrid

Referencia Para más información sobre controles ligados, ver Capítulo 23, "Using The Data-Bound Controls" en el volumen I de la guía del Programador de Microsoft Visual Basic.

SQL (Structured Query Language)

Anteriormente se vio cómo definir el **RecordSource** para el control **Data** utilizando una tabla completa. Es más eficiente obtener solamente los registros y campos que se necesitan actualmente.

¿ Qué es SQL ?

Una vez que los datos son almacenados en la base de datos, la obtención de éstos se realiza más fácilmente utilizando un lenguaje semejante al Inglés llamado Structured Query Language, o SQL. SQL es el medio más ampliamente aceptado para "conversar" con una base de datos. El usuario envía una solicitud y la base de datos regresa todos los renglones que concuerden con ésta.

SQL está definido por el estándar ANSI, aunque muchas implementaciones de SQL tienen variaciones menores de este estándar. La versión de SQL soportada por Jet está basada generalmente en el ANSI.

¿ Porqué Usar SQL ?

Utilizando SQL, se pueden especificar exactamente cuáles registros se desean obtener y en qué orden. El usuario puede crear una instrucción SQL que obtenga información de múltiples tablas a la vez, u obtener solamente un registro específico.

¿ Cuándo Utilizar SQL ?

Se pueden utilizar instrucciones SQL con el control Data. Se pueden definir en tiempo de diseño o en tiempo de ejecución:

Ejemplo Data1.RecordSource = SELECT [Nombre] FROM Empleados

Referencia Para más información, ver Capítulo 6, "Writing SQL Queries" en el libro *Visual Basic Professional Features Book 2: Data Access Guide.*

Instrucción Select

La instrucción Select selecciona campos específicos de una o más tablas en una base de datos. Para seleccionar todos los campos en la tabla, utilizar el asterisco (*):

Ejemplo SELECT * FROM Empleados

Cláusula Where

Utilizar la cláusula Where para delimitar la selección del registro. Utilizar el símbolo # para indicar valores tipo fecha.

Where básico

Ejemplo
```
SELECT * FROM Empleados WHERE [Apellido] =
"'" & txtApellido.text & "'"
```

Where In

Ejemplo
```
SELECT [Apellido] FROM Empleados
    WHERE Empleados.Estado IN ( 'GTO', 'DF' )
```

Where Between

Ejemplo
```
SELECT [Identif de Pedido] FROM Pedidos WHERE
    ( [Fecha de Pedidi] BETWEEN = #01/10/97# AND #31/10/97#)
```

Where Like

Ejemplo
```
SELECT [Apellido] FROM Empleados WHERE
    [Apellido] LIKE 'A'
```

Cláusula Order By

Usar la cláusula Order By para crear un Recordset en un orden en particular. La opción ASC indica orden ascendente. DESC indica orden descendente. El siguiente ejemplo selecciona todos los campos de la tabla Empleados ordenados por apellido:

Ejemplo SELECT * FROM Empleados ORDER BY [Apellido]

Tip Utilizar una aplicación como Microsoft Access o Microsoft Query para generar las instrucciones SQL. Se puede crear el Query utilizando la interfaz gráfica de usuario y entonces copiar la instrucción SQL generada por la aplicación hacia el programa.

Data Form Designer

El **Data Form Designer** es un ejemplo de aplicación que viene con Visual Basic. Utilizando el **Data Form Designer**, se puede crear una forma que se puede desplegar, agregar, borrar y editar registros de bases de datos. La tarea normalmente tediosa de agregar controles y definir propiedades se reduce a unos cuantos pasos fácilmente.

Características del Data Form Designer:

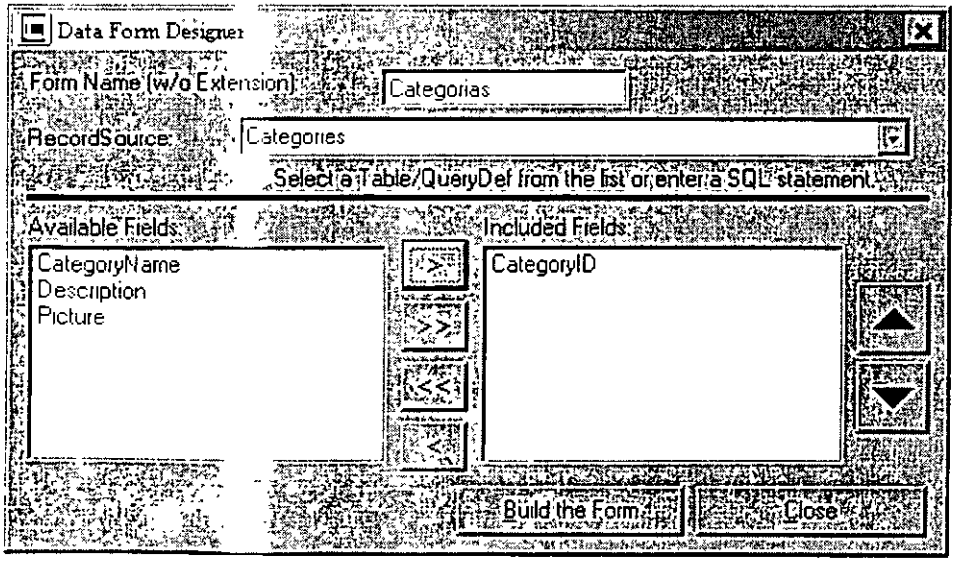
Crea y agrega una nueva forma al proyecto actual.
Agrega un control **Data** y define la propiedad **RecordSource**.
Por cada campo seleccionado en una tabla:

Agrega una etiqueta con el nombre del campo.
Agrega un control ligado. El tipo de control depende del tipo de dato almacenado en el campo:

Tipo de Dato	Control
Cadena, fecha y numérico	TextBox
Booleano	CheckBox
Campos Memorandum	TextBox multilínea
Datos binarios	Contenedor OLE

Agrega botones de comando para realizar varias funciones de acceso a datos, incluyendo Add, Delete, Refresh, Update, y Clear.

Agrega código con comentarios detrás de los botones de comando y los controles **Data**.

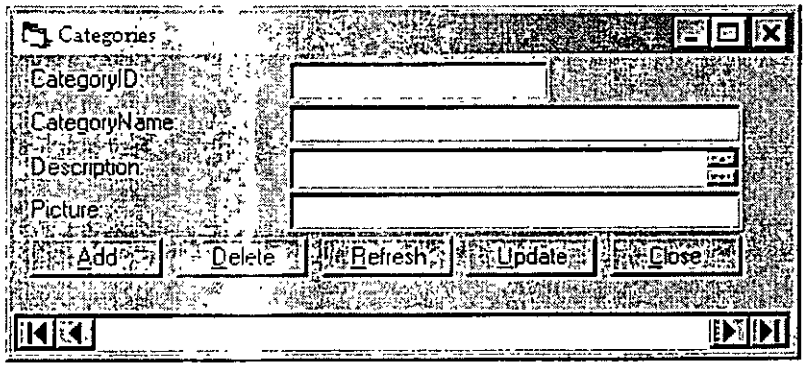


Usando el Data Form Designer

Para crear una forma con el Data Form Designer

1. Desde el menú Add-Ins, elige **Visual Data Manager**
2. Desde el menú File de la nueva ventana elige **Open database -- Microsoft Access...**
3. Selecciona Nwind.mdb del directorio de Visual Basic y da **Open**
4. Del menú **Utility** selecciona **Data Form Designer ...**
5. Da como **Base Form Name**, Categorías (éste será el nombre de la nueva forma)
6. Seleccionar la tabla **Categories** desde la sección **Record Source**.
7. Agregar todos los campos de la tabla **Categories** usando el botón Add All (>>)
8. Seleccionar el botón **Build The Form**
9. Da clic en **Close**
10. Cierra la ventana de **Visual Data Manager**
11. Cambiar la forma de arranque para el proyecto a la forma **frmCategorías** a través del comando Project Properties... del menú Project
12. Ejecutar la aplicación

Mientras que el **Data Form Designer** es rápido y fácil de usar, sólo puede generar formas simples de acceso a datos. Sin embargo, esta forma simple puede servir como marco de trabajo para crear formas de acceso a datos más complejas en las aplicaciones.



Creando y Editando Registros (Opcional)

Utilizando el control **Data** con cajas de texto ligadas (**TextBox**) se abren muchas facilidades, como ver y editar en un programa sin necesidad de programar. Sin embargo, se deberá tener control sobre cómo el programa interactúa con los datos externos usando las propiedades, métodos, y eventos asociados al control **Data** y los controles ligados.

A continuación se tiene una relación de las propiedades y métodos del control **Data** que se discutirán en esta sección:

Propiedad	Descripción
DataBaseName	Regresa/define nombre y localización del origen de los datos.
RecordSource	Regresa/define la tabla subordinada, instrucción SQL, u objeto QueryDef.
Connect	Regresa/define un valor que proporciona información sobre el origen de una base de datos abierta, una base de datos usada en un *pass-through query*, o una tabla conectada.
Recordset.BOF	Regresa un valor que indica si la posición del registro actual está antes del primer registro de un Recordset.
Recordset.EOF	Regresa un valor que indica si la posición del registro actual está después del último registro de un Recordset.
Recordset.Bookmark	Regresa/define una marca que identifica al registro actual en un objeto Recordset o define el registro actual en un recordset a una marca válida.
Recordset.NoMatch	Regresa True si no se encuentran registros cuando se ejecuta un método Find.

Método	Descripción
Refresh	Actualiza los objetos en una colección para reflejar la base de datos actual.
UpdateRecord	Salva el contenido actual de los controles ligados a la base de datos durante el evento Validate sin disparar el evento Validate de nuevo.
UpdateControls	Restaura los contenidos de los controles ligados a su valores originales, como cuando un usuario realiza cambios a los datos y decide finalmente cancelarlos.
Recordset.FinfFirst, FindLast, FindPrevious, FindNext	Localiza el primer, último, siguiente, o previo registro en un objeto Recordset que satisface el criterio especificado y hace a ese registro, el registro actual.
Recordset.Update	Salva los contenidos del buffer de copia al objeto Recordset.
RecordsetDelete	Borra el registro actual.
Recordset.AddNew	Crea un nuevo registro.

Modificando y Actualizando Registros

El control **Data** automáticamente proporciona la habilidad de modificar registros existentes.

Para modificar un registro con el control Data

1.- Moverse al registro que se desea modificar.
2.- Cambiar cualquier cosa de la información desplegada en los controles ligados.
3.- Dar click en cualquier flecha en el control **Data** para moverse al siguiente, previo, primer o último registro.

El control **Data** automáticamente cambia el registro en la base de datos sin necesidad de código adicional. Sin embargo, este método de actualizar un registro podría no ser intuitivo para los usuarios. Por consiguiente, el control **Data** crea un botón *Update* (de actualización). Antes que seleccionar las flechas en el control **Data**, los usuarios podrían entender mejor el significado de un botón de actualización (u otra etiqueta que describa la función del botón, tal como *Aceptar Edición*).

Otra razón para crear código con una función que actualice es que permite que se valide la información y se complete correctamente.

Para indicar a Visual Basic que actualice el registro actual, usar el método UpdateRecord.

Ejemplo Data1.UpdateRecord

El método UpdateControls ignora cualquier cambio realizado a los datos y restaura los datos desplegados en los controles ligados a sus valores originales. Si se desea agregar un botón que ignore los cambios realizados al registro actual, se deberá habilitar el botón *Ignore* cuando los controles ligados sean editados.

Ejemplo Data1.UpdateControls

El siguiente ejemplo es asignado a un botón *Update*. Cuando el usuario seleccione este botón, verifica que el campo CategoryID contenga un entero y envía los cambios a la base de datos con el método UpdateRecord. Si no contiene un entero, con una caja de mensaje se le notifica al usuario que el identificador de categoría es incorrecto. Si el usuario selecciona OK, nada sucede. Si el usuario selecciona Cancel, el método UpdateControls es invocado.

Ejemplo

```
Private Sub cmdUpdate_Click ( )
   Dim txt As String
   Dim iReturn As Integer
   txt = txtField0.Text
   If IsNumeric(txt) And _
      Val (txt) = Int (Val (txt) ) Then
      Data1.UpdateRecord
      Data1.Recordset.Bookmark = _
         Data1.Recordset.LastModified
   Else
      iReturn = _
         MsgBox ( "Category ID deberá ser un entero. Intenta de nuevo?", _
         VbYesNo)
      If iReturn = vbNo Then
         Data1.UpdateControls
      Else
         'Se define el foco en el campo incorrecto
         With txtField0
            .SetFocus
            .SelStart = 0
            .SelLength = Len (.TEXT)
         End With
      End If
   End If
End Sub
```

Agregando Nuevos registros

Otra acción muy común en una forma de dato es agregar nuevos registros. El Data Form Designer crea un botón Add con código.

Nuevos registro se pueden agregar al recordset llamando al método AddNew.

Ejemplo Data1.Recordset.AddNew

El método AddNew limpia los controles ligados y automáticamente da valores de default.

El registro no se agrega actualmente a la base de datos hasta que un método UpdateRecord es explícitamente ejecutado o el usuario usa el control Data para moverse a otro registro.

Cuando el método UpdateRecord se ejecuta, el registro es agregado al final del recordset.

Tip Se puede pulir la interfaz del usuario deshabilitando los botones que no se aplican en algunas situaciones. Cuando el usuario selecciona el botón Add, deshabilita todos los otros botones, excepto los botones Update y Cancel.

Borrando Registros

Para eleiminar un registro del recordset, usar el método Delete.

Ejemplo Data1.Recordset.Delete

Después de borrar un registro los campos continúan sin cambiar. Por tanto es necesario moverse al siguiente registro en el recordset.

Revise la propiedad EOF en caso de que se haya borrado el último registro. Si EOF es True, entonces mover al último registro en el recordset.

Ejemplo

```
Data1.Recordset.Delete
Data1.Recordset.MoveNext
If Data1.Recordset.EOF = True Then
   Data1.Recordset.MoveLast
End If
```

LABORATORIOS

(Introducción a Visual Basic)

Laboratorio 1

Introducción a Visual Basic

Laboratorio 1 Introducción a Visual Basic

Ejercicio 1.1

Utilizando controles

1. Inicia Visual Basic. Ve al botón de Inicio -> Programas -> Visual Basic 6.0 -> Visual Basic 6.0.

2. Da un clic sobre cualquier control de la Caja de Herramientas.

3. Pon el cursor (que ahora tiene una forma de cruz) en alguna parte de la forma y presionando el botón izquierdo del mouse en forma sostenida arrástralo hacia la derecha hasta que alcance el tamaño que deseas.

4. Deja de oprimir el botón izquierdo del mouse.

5. Repite esta operación para diferentes controles de la caja de herramientas.

Obteniendo ayuda acerca de un control

1. Da un clic en un control que esté sobre tu forma.
   Con ello seleccionas dicho control, te darás cuenta que aparecerá rodeado por una línea especial y algunos puntos a través de los cuales puedes cambiar el tamaño del control.

2. Presiona la tecla F1.
   Visual Basic abre el archivo de ayuda en el tema correspondiente al control.

3. Cierra el archivo de ayuda cuando hayas terminado.

Ejecutando una aplicación 

1. Desde el menú Run, escoge la opción Start, o bien desde el botón que aparece en la barra de herramientas.
2. Juega con los controles que aparecen en la forma. Nota que ellos funcionan a pesar de que no se ha escrito nada de código.

3. Cierra la aplicación que está en ejecución para regresar al modo de diseño.
   Existen diferentes formas de cerrar una aplicación que se está ejecutando.

   a) Desde la barra de herramientas escoge el botón de End. 

   b) Desde el menú de Run , escoge la opción End.

d) En la barra de título, da clic en el boton de cerrar 

Eliminando controles de una forma

1. Mueve el cursor del mouse a la esquina superior izquierda de la forma.
2. Presionando el botón izquierdo del mouse arrástralo hacia abajo a la derecha de tal forma que incluyas dentro del rectángulo que estás dibujando todos aquellos controles que desees eliminar.
3. Desde el menú *Edit*, escoge la opción *Delete*, o bien, presiona la tecla *Supr*. Todos los controles que seleccionaste serán eliminados.

Ejercicio 1.2

Creando una aplicación con código.

Agregando una Caja de texto y una Etiqueta a una forma.

1. Agrega un control del tipo TextBox a la forma. 
2. Agrega un control del tipo Label a la forma. 
3. Mueve los controles de forma que queden alineados horizontalmente y la etiqueta quede del lado izquierdo del la caja de texto.

Fijando propiedades en tiempo de diseño

1. Selecciona la etiqueta y presiona la tecla F4 para abrir la ventana de propiedades del control.
2. Cambia la propiedad **Caption** por "Mi Etiqueta".
3. Selecciona la caja de texto y presiona F4 .
4. Cambia la propiedad **Text** por "My Caja de Texto".

Cambiando las propiedades de un control en tiempo de ejecución

1. Da doble clic en la caja de texto. La ventana de código se abre con el siguiente segmento de código.

```
Private Sub Text1_Change()

End Sub
```

2. Agrega la siguiente línea de código como cuerpo del procedimiento.

```
Private Sub Text1_Change()
        Label1.Caption=Text1.Text
End Sub
```

Esta línea de código cambia la propiedad de **Caption** de la etiqueta por lo que se escriba en la caja de texto.

3. Ejecuta la aplicación y ejecuta cualquier cosa dentro de la caja de texto. ¿Qué sucede?
4. Cierra la aplicación y regresa al modo de diseño.

Guardando el proyecto

1. Desde el menú File, escoge Save Project.

Guardar un proyecto incluye guardar todas las formas en el proyecto.

2. Cuando pregunte en dónde se guardará la forma, cambia el directorio destino a \vb6int\labs\lab01, deja el nombre por omisión de Form1, y escoge Save.

3. Cuando pregunte en dónde se guardará el proyecto, cambia el directorio destino a \vb6int\labs\lab01, deja el nombre por omisión de Project1, y escoge Save.

Creando un archivo .EXE

1. Desde el menú File, escoge Make ...
2. Asegúrate que el folder destino sea \vb6int\labs\lab01
3. En la caja de texto de nombre del archivo escribe **Primero** y escoge OK.

Visual Basic compila el proyecto actual y crea un archivo ejecutable que puede ser ejecutado directamente desde el sistema operativo.

4. Sal de Visual Basic.

Ejecutando un .EXE

1. Abre el explorador de Windows.
2. Busca en el directorio \vb6int\labs\lab01 el archivo **Primero.exe** y da doble clic sobre él.
3. Prueba la aplicación introduciendo algo sobre la caja de texto.
4. Cierra la aplicación

Ejercicio 1.3

Usando código desde el archivo de ayuda

Abriendo un proyecto

1. Desde el botón de inicio, inicia Visual Basic
2. Desde el menú File, escoge Open Project.
3. Escoge el directorio \vb6int\labs\lab01 en la caja de diálogo de Open
4. Selecciona Project1 y escoge Open.
5. Si la forma no es visible, selecciona desde la ventana de Project el botón de View Form.

Usando la ayuda de contexto sensitivo

1. Da doble clic en cualquier parte de la forma. La ventana de código aparecerá con las siguientes líneas.

```
Private Sub Form_Load()

End Sub
```

2. En la lista de Procedimientos selecciona MouseMove. La Ventana de código cambia mostrando el siguiente código

```
Private Sub_Form_MoseMove (Button As Integer. Shift As_
        Integer. X As Single. Y As Single)

End Sub
```

3. Escribe FillColor entre las líneas Sub y End Sub.
4. Coloca el cursor sobre cualquier parte de la palabra FillColor y oprime la tecla F1.

El archivo de ayuda de Visual Basic se abre en el tema correspondiente a la propiedad FillColor.

Copiando el código de ejemplo del archivo de ayuda y pegándolo dentro de la aplicación Visual Basic.

1. Escoge la liga de Example que se encuentra en la parte superior de la pantalla de ayuda del tema.

   Se muestra un ejemplo de cómo se utiliza la propiedad FillColor.

2. Selecciona las tres líneas que aparecen entre *Sub* y *End Sub* del ejemplo.
3. Escoge Copy para que estas líneas se guarden en el portapapeles.
5. Cierra el archivo de ayuda

   Con esto debes regresar a la ventana de código, y se debe estar desplegando el procedimiento Form_MouseMove.

1. Desde el menú de Edit, escoge Paste para pegar el código dentro del procedimiento asociado al evento Form_MouseMove, y borra la palabra FillColor que habías escrito anteriormente.

2. El resultado debe ser:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, _
        Y As Single)
        FillColor=QBColor(Int(Rnd * 15))
        FillStyle=Int(Rnd *8)
        Circle(X,Y),250.0
End Sub
```

   Este código dibuja un círculo con un color aleatorio y con un estilo de llenado cuando el usuario mueve el cursor a través de la forma.

3. Guarda y prueba tu aplicación.

Laboratorio 2

Creando una aplicación en Visual Basic

Laboratorio 2 Creando una aplicación en Visual Basic

En está aplicación se creará una forma de validación para la una aplicación que calcula un "Estimado de Pagos de préstamos". Iniciarás un nuevo proyecto, darás nombre a la forma e incluirás en ella los controles adecuados, así como sus propiedades.

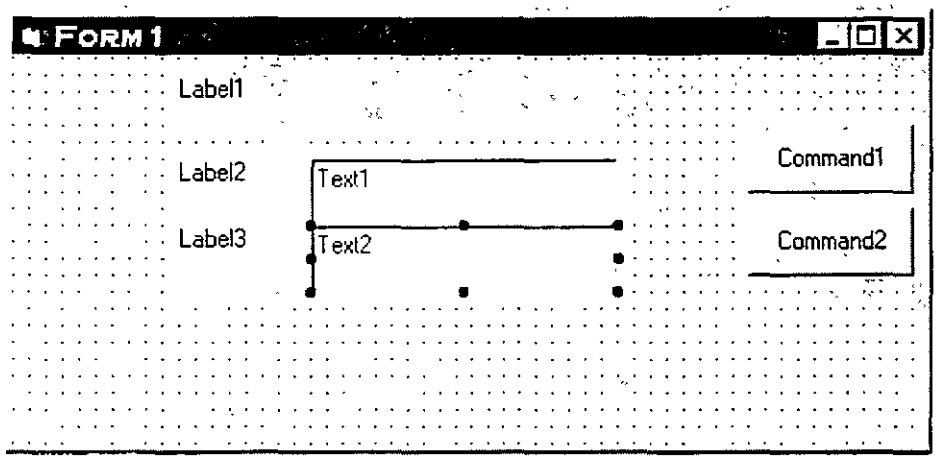
Ejercicio 2.1

Inicia un nuevo proyecto

1. Si Visual Basic no está ejecutándose, ejecútalo desde el botón de Inicio.
2. Crea un nuevo proyecto, elige la opción New Project del menú File.

Agregando controles a la forma por omisión

1. Agrega 3 controles de tipo etiqueta (Lablel) a la forma.
2. Agrega 2 controles del tipo Caja de texto (Text).
3. Agrega 2 controles del tipo Botón de Comando (CommandButton).
4. Muévelos y dales el tamaño para que la forma luzca como la siguiente figura.



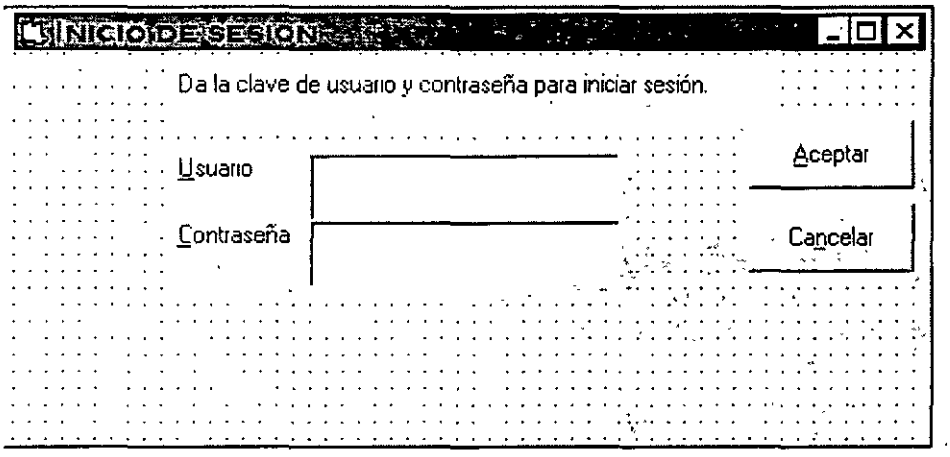
Dando valor a las propiedades de los controles

1. Selecciona el control dentro de la forma
2. En la ventana de Propiedades, selecciona una a una las propiedades que se listan en la tabla asignando los valores indicados.

Nombre actual	Propiedad	Nuevo Valor
Label1	Name	lblInstrucion
	Caption	Da la clave de usuario y contraseña para iniciar sesión.
Label2	Name	lblUsuario
	Caption	&Usuario

Label3	Name	lblContraseña
	Caption	&Contraseña
Text1	Name	txtUsuario
	Text	<blanco>
Text2	Name	txtContraseña
	Text	<blanco>
Command1	Name	cmdAceptar
	Caption	&Aceptar
	Default	True
Command2	Name	cmdCancelar
	Caption	Ca&ncelar
	Cancel	True
Form1	Name	frmValidacion
	Caption	Inicio de sesión
	BorderStyle	1-Fixed Single

Al terminar la forma se verá así



Agregando código al evento Click del botón de comando cmdAceptar

Agregarás una caja de mensaje que muestre los valores proporcionados por el usuario en los campos txtUsuario y txtContraseña.

1. Da doble clic sobre cmdAceptar Automáticamente se abre la ventana de código y se genera el siguiente código.

   ```
   Private Sub cmdAceptar_Click()
   End Sub
   ```

2. Agrega el siguiente código entre las línea Private... y End Sub, para mostrar el contenido de las cajas de texto en una caja de mensaje.

```
MsgBox "Usuario = " & txtUsuario.Text & _
        ", Contraseña = " & txtContraseña.Text
```

3. Ejecuta la aplicación. Escribe un nombre de usuario y una contraseña y da clic en el botón de Aceptar. ¿Qué sucede?
   Escribe un nombre de usuario distinto y otra contraseña y pulsa la tecla ENTER. ¿Qué sucede y por qué?

4. Cierra la aplicación y regresa al modo de diseño.

Agregando código al evento Click del botón de comando cmdCancelar

1. Da doble clic sobre cmdCancelar. Automáticamente se abre la ventana de código y se genera el siguiente código.

```
Private Sub cmdCancelar_Click()

End Sub
```

2. Agrega el siguiente código entre las línea Private... y End Sub, para mostrar el contenido de las cajas de texto en una caja de mensaje.

```
MsgBox "Este es el botón de cancelar"
```

3. Ejecuta la aplicación. Escribe un nombre de usuario y una contraseña y da clic en el botón de Cancelar. ¿Qué sucede?
4. Escribe un nombre de usuario distinto y otra contraseña y pulsa la tecla ESC. ¿Qué sucede y por qué?

5 Cierra la aplicación y regresa al modo de diseño.

Guardando el proyecto

1. Del menú File, elige Save Project
2. Cuando pregunte por guardar la forma, cambia el directorio a \vb6int\labs\lab02, y escribe en el nombre frmValidacion y da clic en Save.
3. Cuando pregunte por guardar el proyecto, cambia el directorio a \vb6int\labs\lab02, y escribe en el nombre PRESTAMO y da clic en Save.

Habilitando el botón de Aceptar

Es bueno crear aplicaciones que prevengan los posibles errores del usuario.
Una forma de hacerlo es deshabilitar los controles para que no se puedan seleccionar en caso de que exista un error. Por ejemplo, en la ventana de Inicio de sesión el botón de

Aceptar no tiene porque habilitarse hasta que no se proporcione un nombre de usuario y una contraseña.

1. Desde la ventana de propiedades, a la propiedad Enabled da el valor de False.
2. Da dos clics en la caja de texto txtUsuario. La ventana de código aparece con el inicio y fin del procedimiento txtUsuario_Change.

3. Agrega las siguientes lineas entre el inico y el fin del procedimiento.

```
If txtUsuario.Text <> "" And txtContraseña.Text <> "" then
    CmdAceptar.Enabled = True
End If
```

4. Agrega el mismo código al evento Change de la caja de texto txtContraseña.
5. Guarda y prueba la aplicación
   Escribe un nombre de usuario, ¿Está habilitado el botón de Aceptar? ¿Por qué?
   Escribe una contraseña, ¿Está habilitado el botón de Aceptar? ¿Por qué?

Laboratorio 3

Trabajando con Formas

Laboratorio 3 Trabajando con Formas

En este laboratorio trabajarás con los archivos que creaste en el laboratorio pasado o bien puedes hacerlo con los archivos que están en el directorio \vb6int\labsol\lab03.

Ejercicio 3.1

Agregando una nueva forma al proyecto

1. Desde el menú Project , escoge la opción Add Form.

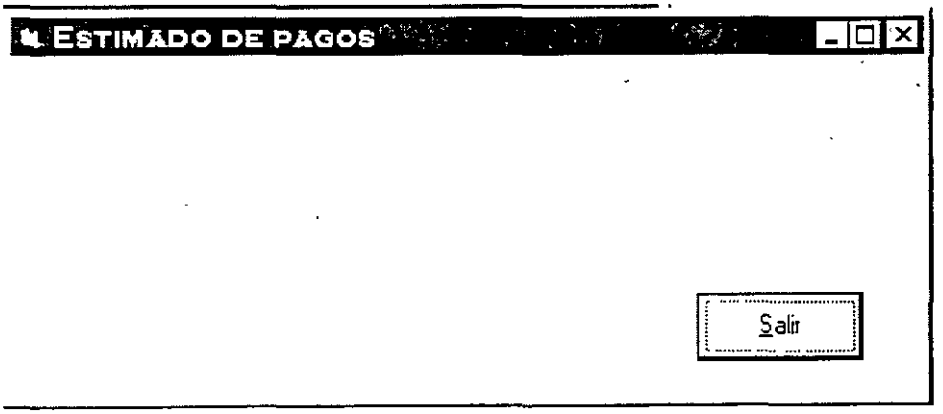
   Una nueva forma es agregada al proyecto y es desplegada en el ambiente de Visual Basic.

2. Da valores a las siguientes propiedades de la forma

Name	frmPrincipal
Caption	Estimado de pagos
BorderStyle	1- Fixed Single
MinButton	True

Agregando controles a frmPrincipal

1. Agrega un botón de comando de manera que la forma quede como se muestra en la figura.



2. Da los siguientes valores a las propiedades del botón

Name	cmdSalir
Caption	&Salir
Cancel	True

Mostrando la forma Validación

1. Da doble clic en cualquier parte de la forma frmPrincipal. La ventana de código aparece con el siguiente segmento de código

   Private Sub Form_load()

   End Sub

2. Dentro de este procedimiento, agrega una línea de código que permita mostrar la forma **Validacion** de manera modal.

Cambiando la forma de inicio

Fija como forma de inicio la nueva forma que se creó dentro del proyecto

1. Desde el menú **Project**, elige la última opción *Project properties ...*
2. Selecciona la pestaña General
3. Desde Startup Object selecciona frmPrincipal.
4. Da clic en OK.

Guarda y prueba la aplicación

1. Guarda el proyecto
2. Ejecuta la aplicación. Escribe un nombre de usuario y una contraseña. Da clic en *Aceptar*. ¿Qué sucede?, ¿Por qué la forma de validación no finaliza?.
3. Termina la aplicación y regresa a modo de diseño.

Ejercicio 3.2

Cerrando formas y finalizando una aplicación

Descargando (Unload) la forma de validación cuando el usuario dé clic en *Aceptar* en frmValidación

1. Agrega el siguiente código al procedimiento asociado al evento Click del botón cmdAceptar de la forma frmValidación para cerrar la forma después de que se muestre la caja de mensaje.

   Unload frmValidación

Cerrando la aplicación cuando el usuario elija el botón de Cancel de la forma frmValidación o bien el botón de Salir de frmPrincipal.

1. Agrega el siguiente código al procedimiento asociado al evento Click del botón cmdCancelar de la forma frmValidación para cerrar la forma después de que se muestre la caja de mensaje.

```
End
```

1. Agrega el siguiente código al procedimiento asociado al evento Click del botón cmdSalir de la forma frmPrincipal para cerrar la forma.

```
Unload frmPrincipal
```

2. Agrega el siguiente código al evento Unload la forma frmPrincipal para terminar la aplicación.

```
End
```

Guarda y prueba la aplicación.

Consultado al usuario antes de que abandone la aplicación

Edita el código asociado al evento Unload de frmPrincipal (Form_Unload) y ahí agrega el código que permita preguntar al usuario si quiere salir de la aplicación o no.

```
Dim iAnswer As Integer
IAnswer= MsgBox ("¿Estás seguro de querer terminar la aplicación.?", vbYesNo)
If iAnswer= vbNo then
        Cancel = True
Else
        End
End if
```

Laboratorio 4

Escribiendo Procedimientos

Laboratorio 4 Escribiendo Procedimientos

Seguirás utilizando los archivos creados en los laboratorios anteriores o bien podrás usar los archivos que están en el directorio \vb6int\labsol\lab04

Ejercicio 4.1

Creando la interfaz de usuario

1. Abre el proyecto Prestamo, creado en los laboratorios anteriores o bien abre el que se encuentra en el directorio \vb6int\labsol\lab04.

Personalizando el ambiente de código

1. Desde el menú de **Tools** elige **Options**.
2. En la pestaña **Editor**, activa **Require Variable Declaration**.
3. En la misma pestaña, activa **Default to Full Module View**.
4. Da clic en **OK**.

Agregando Option Explicit a las formas ya existentes

1. Abre la ventana de código de frmValidación.
2. De la lista de Object, elige General, y la lista de Procedimientos, selecciona Declarations.
   Esto te ubicará en la sección de declaraciones generales General Declarations. Esta sección se encuentra en el inicio del código, antes de cualquier procedimiento de evento o de cualquier procedimiento general.

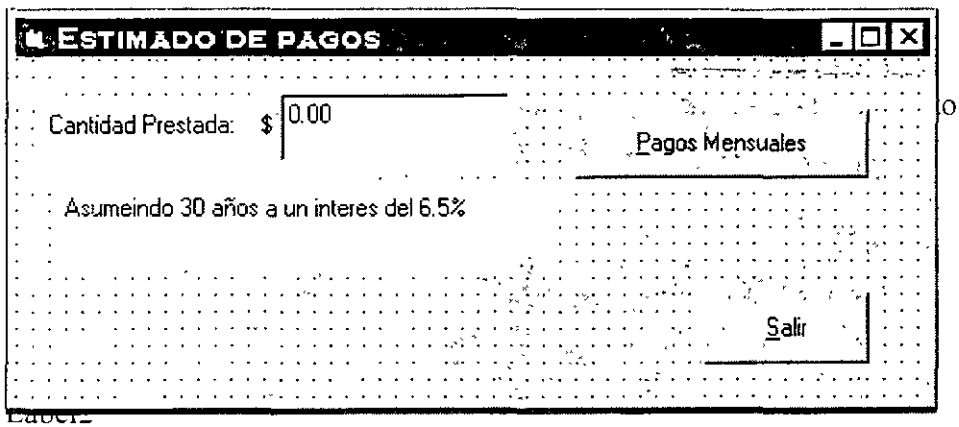
3. Agrega la línea de código

   Option Explicit

Ya que esta forma, así como frmPrincipal fueron creadas antes de que se configurará el ambiente de código, se debe agregar el código para obligar a la declaración de variables al inicio del código correspondiente, por lo tanto repite los pasos 1al 3 para la forma frmPrincipal.

Agregando controles a frmPrincipal

1. Agrega los controles necesarios a la forma frmPrincipal para que luzca como muestra la figura siguiente:



o y proporciona los

Label2

Caption	Asumiendo 30 años a un interés del 6.5%
Name	lblInteres

Text1

Text	0.00
Name	txtPrestamo

Command2

Caption	&Pagos Mensuales
Name	cmdPagosMensuales

Ejercicio 4.2

Escribiendo una función

Creando una función para calcular los Pagos Mensuales

Los pagos mensuales para el préstamo dependen la duración del préstamo, la tasa de interés, y la cantidad prestada.

1. Abre la ventana de código de frmPrincial
2. Desde el menú de **Tools** selecciona la opción **Add Procedure** ...
3. Da por nombre a la función **PagosMensuales**, en el tipo elige **Function** y en el Scope **Public** y da clic en **OK**.

   Una plantilla para la función es agregada a la forma:

   Public Function PagosMensuales()

   End Function

4. Indica que el valor que devuelve la función es del tipo Real Doble, es decir agrega delante del encabezado de la función las palabras reservadas As Double.

   Public Function PagosMensuales()As Double

5. Para crear el cuerpo de la función sigue los siguientes pasos:

   a. Declara una variable del tipo Double para almacenar la tasa de interés mensual, dblInteresM.

   b. Declara una variable del tipo Integer para almacenar el número de pagos, iNPagos.

   c. Declara una variable del tipo Double para almacenar la cantidad prestada, dblCantidadP.

   d. Convierte el texto que se proporciona en la caja de texto txtPrestamo y almacénalo en la variable dblCantidadP.

   e. Ya que en muchos lugares se utilizará la cantidad de meses por año (12), será mejor declararla como una constante al nivel de la forma en General Declarations.
   Const MESES As Integer = 12

   f. Asume que la duración del préstamo es de 30 años, el número de pagos será (30*MESES), almacénalo en iNPagos.

g. Considera que la tasa de interés es del 6.5% o bien del .065, calcula la tasa mensual en dblInteresM como (.065/MESES)

h. Utiliza la función **Pmt** de Visual Basic para calcular el monto de los pagos mensuales y regresa éste como el valor que devuelve la función.

```
PagosMensuales = Pmt( Rate:= dblInteresM, _
        Nper:= iNPagos, PV:= dblCantidadP)
```

Al final la función debe aparecer aproximadamente de la siguiente manera:

```
Public Function PagosMensuales()

    Dim dblInteresM As Double
    Dim iNPagos As Integer
    Dim dblCantidadP As Double

    DblCantidadP = CDbl(txtPrestamo.Text)
    iNPagos = 30 * MESES
    dblInteresM = 0.065/MESES

    PagosMensuales = Pmt( Rate:= dblInteresM, _
        Nper:= iNPagos, PV:= dblCantidadP)

End Function
```

Ejercicio 4.3

Llamando a la función PagosMensuales

Muestra los pagos mensuales

1. Da doble clic en el botón de comando Pagos Mensuales de frmPrincipal.
   Esto abrirá la ventana de código asociada al evento clic del botón

2. En el curpo del procedimiento escribe el código necesario para llamar a la función PagosMensuales y que se muestre el resultado en una caja de mensaje:

   a. Declara una variable del tipo Double para almacenar el resultado que devuelve la función. dblMensual
   b. En la caja de mensaje despliega el contenido de la variable.

El procedimiento se verá de la siguiente forma:

```
Private Sub cmdPagosMensuales_Click()
        Dim dblMensual As Double
        DblMensual = PagosMensuales()
        MsgBox "Tus pagos mensuales serán de: " & dblMensual

End Sub
```

1. Guarda tu proyecto y trabaja con él.

Dando formato a la salida

Cuando se despliega un valor monetario, se debe utilizar el correspondiente a aquel país en el cual se encuentra el usuario de la aplicación, en el caso de México se pondrá el signo de $ antecediendo las cantidades en pesos. Por ello visual Basic cuenta con una función **Format** que permite mostrar el signo de moneda correspondiente.

1. Edita el procedimiento asociado al evento cmdPagosMensuales_Click().
2. Cambia la cadena que se envía en el MsgBox de tal forma que se llame a la función **Format** antes de que se despliegue el valor de los pagos mensuales, esto debe quedar:

   ```
   MsgBox "Tus pagos mensuales serán de: " & Format(dblMensual, "currency")
   ```

   Donde currency es el formato monetario definido desde Windows.

Verificando si se inserta un valor permitido

La función PagosMensuales requiere de un valor numérico proporcionado a través de la caja de texto txtPrestamo, si se llama a la función enviándole un tipo de valor diferente, puede ser causa de un error en tiempo de ejecución. Por lo cual se debe asegurar que el usuario introduzca el valor requerido.

1. Ejecuta la aplicación, escribe un valor no numérico en la caja de texto, txtPrestamo y da clic en el botón Pagos Mensuales.

   El error en tipo de ejecución "Type Mismatch" se produce. Da clic en el botón End de la caja de mensaje del error.
2. Abre la ventana de código de frmPrincipal y ve al procedimiento cmdPagosMensuales_Click()

3. Antes de que se llame a la función PagosMensuales, pregunta si el valor insertado en txtPrestamo es un valor numérico.

   If IsNumeric(txtPrestamo.Text) Then

4. Si el valor es numérico se llama a la función PagosMensuales y se despliega el resultado en una caja de mensaje.
5. Si el valor no es numérico se debe desplegar un mensaje indicando el error y dejando al usuario ubicado en la caja de texto para que intente con otro valor.

El procedimiento debe quedar....

```
Private Sub cmdPagosMensuales_Click()
        Dim dblMensual As Double
        If IsNumeric(txtPrestamo.Text) Then
                DblMensual = PagosMensuales()
                MsgBox "Tus pagos mensuales serán de: " & Format(dblMensual, _
                "currency")
        Else
                MsgBox "La Cantidad Prestada debe ser un valor numérico"
                With txtPrestamo
                        .SetFocus
                        .SelStart = 0
                        .SelLength = Len(.Text)
                End With
        End If
End Sub
```

Ejercicio 4.4

Agregando un archivo de módulo

La función para calcular la suma de todos los pagos ya fue escrita. Se llama TotalPagos y está en un módulo llamado totalpagos.bas que se encuentra en el directorio \vb6int\labsol\lab04

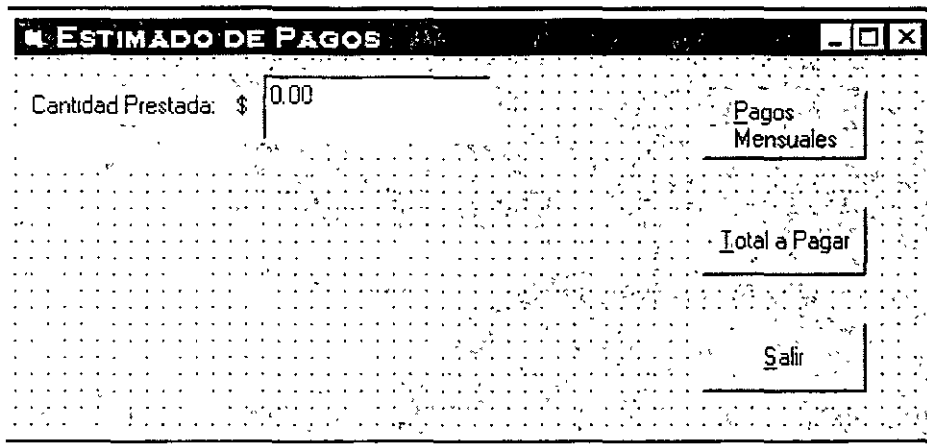
1. De el menu **Project**, elige Add File
2. Selecciona el archivo **Totalpagos.bas** del directorio \vb6int\labsol\lab04 y da clic en el botón Abrir (Open).

   Esto agregará el módulo a tu proyecto. Para ver el procedimiento que realiza el módulo selecciona el módulo de la ventana de proyecto y da clic en el botón de View Code

3. Verifica la función, nota que ésta recibe como parámetro un valor Integer que se refiere al número de años por los cuales se otorga el préstamo y regresa un valor Double.

Para desplegar el monto total de los pagos

1. Agrega un botón de comando a frmPrincipal para desplegar el total de pagos del préstamo.



2. Da como nombre al botón cmdTotal.
3. Para ejecutar el procedimiento del módulo totalpagos genera el código asociado al evento clic del botón cmdTotal.

   a. Declara una variable de tipo Double para que regrese el valor de la función TotalPagos. dblTotal.
   b. Llama a la función. da como parámetro 30. y almacena el resultado en dblTotal
   c. Envía en una caja de mensaje el resultado.

1. Como la función TotalPagos depende del valor del préstamo crea el mismo procedimiento que para la función PagosMensales para verificar si el valor proporcionado por el usuario es numérico.

2. Guarda y prueba la aplicación.

Laboratorio 5

Utilizando la herramienta Debugger de Visual Basic

Laboratorio 5 Utilizando la herramienta Debugger de Visual Basic

Ejercicio 5.1

Verificando lógica del programa con Breakpoints y ejecutándolo paso a paso

1. Abre el proyecto \vb6int\labsol\lab05\debug
2. Haz que se muestre la forma debug.frm

Insertando un punto de rompimiento (Breakpoint)

1. Da doble clic en el botón **Trace** de la forma frmDebug de tal manera que se muestre la ventana de código en el procedimiento cmdLogic_Click.
2. Coloca el cursor al inicio de la línea PicOutput.Cls
   Ya que es la primera línea de código ejecutable del procedimiento.

   Nota: Recuerda que las declaraciones de variables no se consideran enunciados ejecutables.

3. Del menú Debug, elige Toggle Breakpoint o pulsa la tecla F9 o escoge el botón de Toggle Breakpoint en la barra de herramientas **Debug**. 

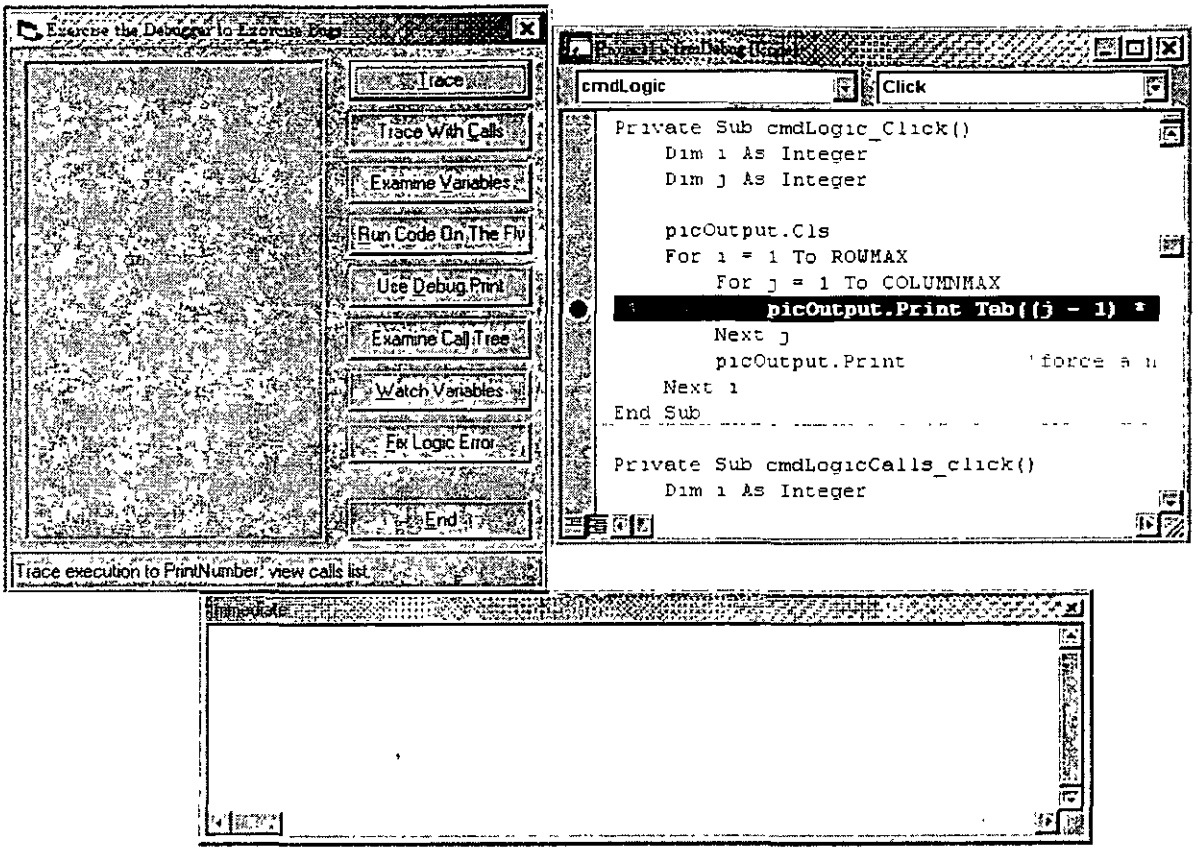
   Por omisión el fondo de la línea cambia su color a rojo indicando que cuando se ejecute el código, Visual Basic hará un alto en este punto y el programa se ejecutará en modo **Break**.

Ejecutando la aplicación paso a paso

1. Da clic en el botón de ejecutar de la barra de herramientas.

   Mueve el ratón por encima de los nueve botones de la aplicación y pon atención a los mensajes que se envían en la barra de estado (status bar).
   Los mensajes de la barra de estado te recuerdan lo que realiza cada botón de la aplicación con respecto a la herramienta Debugger.

2. Coloca las ventanas **Inmediata** (Immediate), de la aplicación y de código para que puedas ver las 3 claramente todo el tiempo.



3. Elige el botón **Trace**

   La ventana de código se activará y la línea que se ejecutará aparecerá iluminada

4. Escoge el botón de **Step Into** de la barra de herramientas **Debug**, para ejecutar paso a paso el procedimiento.

5. Continúa paso a paso en el código de procedimiento hasta que llegues a la instrucción End Sub.

6. Elige el botón **Continue** de la barra de herramientas para continuar la aplicación de manera normal.

7. Elige el botón **End** para terminar con el programa.

Ejercicio 5.2

Verificando lógica del programa, ejecutándolo paso a paso y saltando llamadas a procedimientos

En este ejercicio verás la diferencia entre ejecutar paso a paso un procedimiento o saltarlo. es decir ejecutar una llamada a procedimiento en un solo paso.

1. Da doble clic en el botón **Trace With Calls**
2. Inserta un Breakpoint en la primera línea de código ejecutable dentro del procedimiento CmdLogicCalls_Click.
3. Ejecuta la aplicación.
4. Elige el botón **Trace With Calls**.

   El código hace lo mismo que en el procedimiento del ejercicio anterior, solo que ahora esta dividido en 2 procedimientos, en lugar de hacer la operación en 2 ciclos For anidados

Ejecutando paso a paso un procedimiento 

1. Da 2 clics en el botón **Step Into**
   La línea actual contiene la llamada al procedimiento PrintRow.
2. Da un clic más en **Step Into**
   El debugger salta al procedimiento PrintRow
1. Da 3 clics en **Step Into**
   El debugger salta al procedimiento PrintNumber
2. Elige **Continue** para seguir ejecutando de manera normal la aplicación

Saltando la ejecución de un procedimiento 

1. Elige el botón **Trace With Calls** nuevamente
2. Da 3 Clics sobre el botón **Step Over** 

   Cuando la llamada al procedimiento PrintRow está iluminada, ejecuta entonces un **Step Over** para que el procedimiento se ejecute en un solo paso incluyendo las llamadas a otros procedimientos dentro, evitando así que tengas que ir paso a paso en cada línea del procedimiento y sus procedimientos internos.

3. Continúa dando clics en el botón **Step Over** hasta encontrar la instrucción End Sub.

4. Escoge el botón **End** para terminar el programa.

Ejercicio 5.3

Examinando el contenido de las variables en la ventana Inmediata (Immediate)

En este ejercicio usarás la ventana **Inmediata** para ver los valores almacenados en las variables mientras el programa se está ejecutando. Incluso puedes cambiar los valores de dichas variables desde esta ventana para ver como se comporta el programa con otro valores.

1. Da doble clic en el botón **Examine Variables**.
2. Inserta un Breakpoint en la primera línea de código ejecutable en el procedimiento cmdExamine _Click.
3. Ejecuta la aplicación.
4. Elige el botón **Examine Variables**.
5. Salta paso a paso (**Step Into**) dentro del procedimiento, hasta que la instrucción MsgBox este iluminada.
6. Ve a la ventana **Inmediata**.
7. Escribe ?a y da **ENTER**.

   Visual Basic muestra el número 1, indicando que este es el valor que actualmente almacena la variable a.

8. Usa el mismo método para verificar los valores de las variables b, c, d, e y sum.

Asignado valor a una variable

1. Observa el valor de la variable sum.
2. En la ventana **Inmediata** escribe sum=20 y da **ENTER**.
3. Escribe ?sum y da **ENTER**.

   Comprueba que el valor de sum ha cambiado.

4. Da clic en el botón **End** para terminar el programa.

Ejercicio 5.4

Ejecutando código desde la ventana Inmediata (Immediate)

Borrando puntos de rompimiento (Brakpoints)

- Del menú Debug , elige **Clear All Breakpoints**.

Escribiendo código en la ventana Debug y ejecutándolo

1. Da doble clic en el botón **Run code On The Fly**.
2. Agrega la siguiente instrucción dentro del procedimiento cmdCode_Click.

Stop

3. Ejecuta la aplicación.
4. Elige el botón **Run code On The Fly**.
5. Escribe las siguientes líneas de código dentro de la ventana **Inmediata** para probar qué operaciones aritméticas pueden realizarse en ella.

```
x =  sqr(2.0)
?x
x =x +1
?x
```

6. Escribe las siguientes líneas de código dentro de la ventana **Inmediata** para probar que la asignación de propiedades pueden realizarse en ella.

```
?me.height
me.height =  me.height÷100
?cmdCode.caption
```

7. Escribe las siguientes líneas de código dentro de la ventana **Inmediata** para probar que la ejecución de procedimientos puede realizarse en ella.

```
CmdLogic_Click
PicOutput.cls
End
```

Ejercicio 5.5

Utilizando la ventana Inmediata como ventana de salida de ejecución del programa

En este ejercicio usarás la instrucción Debug.Print, este metodo asociado al objeto Debug nos permite ver en que momento del código los valores de las variable cambian. de esta manera sabremos en que parte de éste se generan errores .

Enviando la salida directamente a la ventana Inmediata

1. Da doble clic en el botón **Use Debug.Print**.
2. Inserta un Breakpoint en la primera línea de código ejecutable
3. Ejecuta la aplicación
4. Elige el botón **Use Debug.Print**
5. Da varios clics sobre el botón **Step Into** hasta encontrar las instrucciones Debug.Print

   Observa que la aplicación envía a la ventana **Inmediata** su salida.

6. Continua línea a línea hasta encontrar la instrucción End Sub y da clic en el botón **Continue** de la barra de herramientas para continuar con la ejecución normal de la aplicación.
7. Finaliza la aplicación.

Ejercicio 5.6

Examinando la lista de llamadas a procedimientos

En este ejercicio seguirás el flujo del programa para poder examinar la lista de llamadas a procedimientos en la ventana **Call Stack**. las lista de llamadas también se conoce como árbol de llamadas a procedimientos.

1. Da doble clic en el botón **Examine Call Tree**
2. Inserta un Breakpoint en la primera línea de código ejecutable del procedimiento cmdCallTree_Click.
3. Ejecuta la aplicación.
4. Da clic en el botón **Examine Call Tree**
5. Da clic 9 veces en el botón **Step Into**. de tal forma que te posiciones en la parte más interna de las llamadas al procedimiento PrintNumber.
6. Da clic en el botón **Call Stack** de la barra de heramientas. 
   Visual Basic mostrará la ventana de llamadas a procedimientos (**Call Stack**), la cual muestra el número y orden de llamadas al procedimiento PrintNumber.

¿Cuántos niveles de profundidad se tienen?

7. Elge **Close** para cerrar la ventana, después permite que se termine la aplicación de manera normal (botón **Continue**).
8. Cierra la aplicación.

Ejercicio 5.7

Monitoreando a las variables con Watch Expressions (Expresiones de monitoreo)

En este ejercicio agregarás una expresion de monitoreo para probar el valor de la variable *wordlength* que se romperá cuando su valor sea igual a 6.

1. Ejecuta la aplicación.
2. Elige el botón **Watch Variables**.

   La aplicación envía un mensaje en su propia área de salida, una palabra por línea. El código asociado a este botón extrae las palabras de una cadena.

3. Cierra la aplicación.

Agregando una expresión de monitoreo

1. Da doble clic en el botón **Watch Variables**
2. En el procedimiento cmdWatch_Click coloca el cursor sobre la variable wordlength, en la tercera línea.
3. De el menú de Tools, elige **Add Watch**
4. En la caja **Expresssion**, escribe wordlength=6
5. En **Watch Type**, elige la opción **Break When Value Is True**.
   Esto le dirá a Visual Basic que pare la ejecución del código cuando wordlength = 6
6. Da clic en **OK**.
   Observe que aparece la ventana de monitoreo **Watch** con la expresión definida en el paso anterior.

Ejecutando la aplicación con la expresión de monitoreo

1. Ejecuta la aplicación

   La ventana de monitoreo muestra todas las expresiones de monitoreo y sus valores. El valor **"Out of context"** indica que la variable aún no se encuentra definida, en el caso de la variable wordlength recuerda que existe en el procedimiento cmdWatch_Clic el cual aún no ha sido llamado.

2. Elige el botón **Watch Variables**.

   La aplicación para en la segunda línea del ciclo While. Observa que cuatro palabras fueron desplegadas en el área de salida de la aplicación, ninguna de las palabras tiene más de 6 caracteres de longitud. También observa que el valor de la expresión de monitoreo en la ventana **Watch** ahora está en True.

3. Finaliza la aplicación.

Laboratorio 6

Controlando el flujo del programa

Laboratorio 6 Controlando el flujo del programa

Ejercicio 6.1

Validando la información de inicio de sesión

1. Continuarás trabajando con tu proyecto del Laboratorio 4.

Agregando una constante a nivel de forma

1. Abre la ventana de código para la forma frmPrincipal y ve a la sección de declaraciones generales "General Declarations".
2. Declara una constante String, **CONTRASEÑA**, que sea igual a **"secreto"**

   Const CONTRASEÑA As String = "secreto"

   Nota: Para esta aplicación, la única contraseña valida es "secreto". En una aplicación ya en servicio, es decir, en uso de una empresa o compañía, cada usuario deberá tener una contraseña diferente y esa información deberá estar almacenada en una base de datos.

Validando la contraseña en la forma frmPrincipal

En el procedimiento asociado al evento Load de frmFrincipal, se debe desplegar la forma de validación hasta que la contraseña válida sea proporcionada.

1. En la ventana de código de frmPrincipal, localiza el procedimiento Form_Load, actualmente este procedimiento muestra la forma frmValidación en forma modal.
2. Declara una variable entera de tipo estática, siIntentos, que será el contador de intentos de validación del usuario.

   Static siIntentos As Integer

3. Escribe una instrucción iterativa que continúe preguntándole al usuario por la contraseña mientras la que escriba en txtContraseña no sea igual a la constante CONTRASEÑA.
4. Escribe en el cuerpo de la instrucción iterativa el código necesario para que haga lo siguiente:
   a. Incremente el contador estático, siIntentos en uno.
   b. Envíe un mensaje diciendo que la contraseña es inválida
   c. Limpie la caja de texto txtContraseña
   d. Muestre la forma frmValidación en forma modal

El procedimiento de evento debe aparecer más o menos así;

```
Private Sub Form_Load()
        Satic siIntentos As Integer
        FrmValidacion.Show vbModal
        Do Until frmValidacion.txtContraseña.text = CONTRASEÑA
                SiIntentos = siIntentos +1
                MsgBox prompt:= "Haz dado una contraseña inválida", _
                        buttons:=vbOKOnly + vbInformation
                frmValidacion.txtContraseña.text = ""
                frmValidacion.Show vbModal
        Loop
End Sub
```

Escondiendo la forma frmValidacion en lugar de descargarla

Para poder utilizar el valor de la caja de texto txtContraseña desde la forma frmValidacion, después de que ha sido proporcionado, se debe ocultar la forma en lugar de descargarla.

1. Abre la ventana de código para frmValidación
2. En el procedimiento de evento asociado al clic del botón de comando cmdAceptar, oculta la forma en lugar de descargarla.

```
FrmValidacion.Hide
```

Guarda y prueba la aplicación

1. Guarda el proyecto en \vb6int\labs\lab06
2. Ejecuta la aplicación, recuerda que la contraseña válida es "secreto", intenta introducir una contraseña inválida y da clic en aceptar. ¿Qué sucede?.
   Intenta con la contraseña válida. ¿Qué sucede?
3. Cierra la aplicación.

Ejercicio 6.2

Limitando los intentos de inicio de sesión.

En este ejercicio se limitará el número de intentos que el usuario tendrá para iniciar una sesión en la aplicación.

Limitando el número de intentos

1. Abre la ventana de código para la forma frmPrincipal en su evento Load.
2. Declara una constante entera que se llame MAX_NUM_INTENTOS cuyo valor sea igual a 3.

```
Const MAX_NUM_INTENTOS As Integer = 3
```

Este será el número de veces que un usuario podrá intentar iniciar una sesión en la aplicación.

3. Dentro de la instrucción iterativa que se escribió en el ejercicio anterior, ahora se debe validar si el número almacenado en nuestra constante estática siIntentos no ha alcanzado el número máximo, que ahora es 3 y que está referenciado por MAX_NUM_INTENTOS.

```
SiINTENTOS = siINTENTOS +1
If siINTENTOS < MAX_NUM_INTENTOS Then
```

4. Si el número de intentos es menor al número máximo entonces se seguirá haciendo lo mismo que ya se tenía, es decir, desplegar el mensaje de contraseña inválida y seguir mostrando la forma.

5. Si el número es mayor o igual al número máximo de intentos, entonces se debe envíar un mensaje en el que se le informe al usuario que ha rebasado el límite de intentos permitido y se debe terminar la aplicación.

```
Else
    Msgbox Prompt:= "Han sido demasiados intentos, Adiós!!!!!", _
        Buttons:= vbOKOnly +vbExclamation
    End
End if
```

6. Guarda y prueba la aplicación

Laboratorio 7

Trabajando con controles

Laboratorio 7

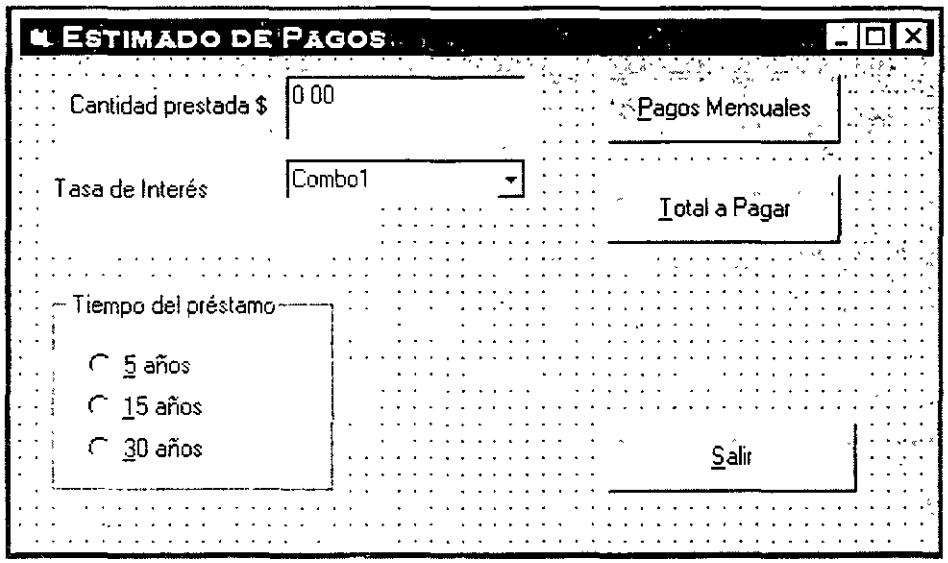
Trabajando con controles

Ejercicio 7.1

Creando la Interfaz de Usuario

En este ejercicio agregarás controles a la forma frmPrincipal para solicitar más información del usuario.

1. Trabajarás con el proyecto del laboratorio anterior
2. Agrega controles a forma frmPrincipal de tal manera que aparezca como a continuación se muestra.



Las propiedades de los controles proporciónalas de acuerdo a la siguiente tabla:

Nota: Para insertar la lista de valores en la propiedad **List** de la caja combo (combo box), en tiempo de diseño da CTRL+ENTER después de cada valor.

Tipo de control	Caption	Propiedad	Valor
Label	Tasa de Interés		
Combo box		Name	cboTasa
		Style	0- Dropdown Combo
		List	4.5, 6.25, 7, 8.325, 9, 10
Frame	Tiempo del préstamo	Name	fraTiempo

Option button	&5 años	Name	optDuracion
		Index	0
Option button	&15 años	Name	optDuracion
		Index	1
Option button	&30 años	Name	optDuracion
		Index	2

Ejercicio 7.2

Implementando la Interfaz de Usuario

En este ejercicio se agregará código para guardar los valores de la opción del tiempo de préstamo, dado por los botones de opción así como se inicializarán los valores de los controles de la forma frmPrincipal.

Implementando un grupo de botones de opción en un arreglo de controles.

1. Abre la ventana de código de la forma frmPrincipal.
2. En la sección de General Declarations, declara una variable Integer, fiDuración. para almacenar el valor seleccionado por el usuario. a través de los botones de opción de tiempo del préstamo.

   Dim fiDuración As Integer

3. Edita el procedimiento del evento click para los option buttons y utilizando la instrucción **Select Case**, determina el valor que almacenará fiDuración. dependiendo del número de años que durará el préstamo, dependiendo del correspondiente botón que elija el usuario. Por ejemplo si la propiedad **Index** es 0. la duración del préstamo es de 5 años.

El código debe ser algo así:

```
Private Sub optDuracion_Click(Index As Integer)
  Select Case Index
    Case 0
      fiDuración = 5
    Case 1
      fiDuración = 15
    Case 2
      fiDuración = 30
  End Select
End Sub
```

Inicializando los controles de la forma a través del procedimiento de evento Form_Load

Para cualquier aplicación se recomienda utilizar el procedimiento Load de las formas para inicializar los valores de los controles. Esto se centraliza en el código y es más fácil de localizar y de cambiar, que si se hiciera en tiempo de diseño a través de la ventana de propiedades.

1. En la ventana de código de frmPrincipal ve al procedimiento Load
2. Agrega el código que haga lo siguiente al final del código que ya tienes escrito.

   a) Inicia la propiedad Text de txtPrestamo con "0.00"
   b) Inicia la propiedad Text de CboTasa con "4.25"
   c) Fija la propiedad Value del botón de opción 5años como True
   d) Llama al procedimiento de evento Click para el botón de opción 5 años ya que este procedimiento determina el valor de una variable a nivel de forma (fiDuracion).

El código final debe ser algo así:

```
txtPrestamo.text = "0.00"
cboTasa.text = "4.5"
optLength(0).Values = True
optLength_Click 0
```

Guarda y prueba la aplicación.

Ejercicio 7.3

Empleando Valores proporcionados por el usuario

Empleando el valor de los controles en la función PagosMensuales

Ahora existen controles a través de los cuales la información para la aplicación "Prestamo" pueden variar, por lo que se tendrá que cambiar los datos "fijos" que se tenían en la función y serán reemplazados por los valores que se obtengan de los controles que manipula el usuario.

1. En la ventana de código de la forma frmPrincipal, ve al código de la función general PagosMensuales.
2. Declara una variable del tipo Double para almacenar la tasa de interés que ahora en lugar de ser de 6.5 %, será determinada por el usuario a través de la caja combo.

```
Dim dblTasa As Double
```

3. Obtén el valor dado a través de la caja combo y almacénalo en esta variable, previamente haz la conversión al tipo de dato y divídelo entre 100 para sacar el porcentaje.

```
dblTasa = CDbl(cboTasa.Text) /100
```

4. Utiliza el valor contenido en la variable fiDuración para determinar la duración del préstamo y sustitúyelo por la constante 30 que aparece el cálculo de iNPagos y la variable DblTasa se debe sustituir por la constante 0.065 del cálculo de dblInteresM.

5. Guarda y prueba la aplicación.

Utilizando los controles en la función TotalPagos

Ahora modifica la función TotalPagos para que trabaje con los valores dados por el usuario.

1. En la ventana de código de frmPrincipal, selecciona el evento asociado a cmdTotal_Click
2. Da como parámetro la variable fiDuración al llamar a la función TotalPagos

```
DblTotal =totalPagos(fiDuración)
```

3. Guarda y prueba la aplicación

Laboratorio 8

Agregando Menúes

Laboratorio 8
Agregando Menúes

Ejercicio 8.1

Agregando un Menú

Un menú, provee de otras opciones para que el usuario pueda realizar acciones sobre la aplicación. En este ejercicio se borrarán los botones de comando de la forma frmPrincipal y serán sustituidos por elementos de un menú, desde donde se ejecutarán las mismas acciones.

Creando un menú en frmPrincipal

1. Con la forma frmPrincipal abierta, inicia el editor de menúes
2. Agrega los siguientes elementos de menú

Caption	Name	Shortcut
&File	mnuFile	
...E&xit	mnuFileExit	
&Préstamo	mnuPrestamo	
...Pagos &Mensuales	mnuPrestamoMensual	CTRL+M
...&Total a Pagar	mnuPrestamoTotal	CTRL+T

Creando nuevos procedimientos Sub

1. Abre la ventana de código de frmPrincipal
2. Inserta 2 nuevos procedimientos **Sub VerMnsual** y **Sub VerTotal**
3. Copia el código del procedimiento cmdPagosMensuales_Click al nuevo procedimiento VerMensual.
4. Copia el código del procedimiento cmdTotal_Click al nuevo procedimiento VerTotal.

Agregando código al el evento click de los elementos del menú

1. Agrega el código necesario al evento click del elemento Exit para que cuando sea invocado descargue la forma (Unload frmPrincipal)
2. Agrega el código necesario al evento click del elemento Pagos mensuales, para que cuando sea invocado llame al procedimiento VerMensuales
3. Agrega el código necesario al evento click del elemento Total a Pagar, para que cuando sea invocado llame al procedimiento VerTotal

Guarda y prueba la aplicación

1. Selecciona cada elemento del menú
2. Prueba las teclas de acceso
3. Prueba con CTRL+M y CTRL+T

Borrando los botones de comando

Ahora que los elementos del menú realizan las acciones que antes se tenían en botones de comando, ahora estos son innecesarios.

1. Borra los botones de comando cmdSalir, cmdPagosMensuales y cmdTotal.
   Con estas acciones el código asociado a ellos se moverá a la sección de **General Declarations** del módulo de forma

2. Borra los procedimientos de evento asociados al los eventos click de cada uno de los botones que fueron borrados.

Ejercicio 8.2
Agregando una barra de estado

Una barra de estado, además de las cajas de mensaje o diálogo, proporciona al usuario un medio a través de la cual obtener retroalimentación. En este ejercicio se eliminarán las cajas de mensajes y se enviarán al usuario a través de una barra de estado.

Agregando los controles Microsoft Windows Common Controls al proyecto

La barra de estado es un control considerado común a las aplicaciones Windows.

1. Del menú Project, elige **Components ...**
2. De la lista disponible de controles, elige **Microsoft Windows Common Controls 6.0**.

Nota: si **Microsoft Windows Common Controls 6.0**, no aparece en la lista de controles disponibles, asegúrate de que la opción *Selected Items Only* no esté seleccionada.

1. Elige **OK**
2. Observa que nuevos controles fueron agregados a la caja de herramientas de Visual Basic.

Creando una barra de estado

1. Agrega un control de barra de estado **StausBar** a la forma frmPrincipal
2. Da en la propiedad Name, stbPrestamo

3. Abre la caja de diálogo de propiedades de la barra de estado eligiendo la propiedad **Custom** de la caja de propiedades o bien, dando botón derecho sobre la barra de estado y eligiendo la opción **Properties**.
4. Elige la pestaña **Panels**
5. Inserta un segundo panel
6. Da las propiedades a los paneles de acuerdo a la lista que aparece en la siguiente tabla.

Panel Index	Propiedad	Valor
1	Key	msg
	Style	0-sbrText
2	Key	time
	Style	5-sbrTime

7. Da valor a la propiedad **Minimun Width** del panel 1 de tal forma que tenga el mayor tamaño posible, sólo dejando espacio para que la hora se despliegue en el segundo panel, el tamaño aproximado es de 5500 (recuerda que estos son twips).
8. Da clic en el botón **Aplicar**, para ver los resultados.
9. Da clic en el botón **OK** cuando los paneles tengan el tamaño que deseas.

Enviando mensajes a través de la barra de estado

Los procedimientos VerMensuales y VerTotal envían actualmente cajas de mensaje al usuario. En este ejercicio se cambiarán dichas caja por la propiedad text de la barra de estado.

1. En el procedimiento VerMensuales

   a) Reemplaza la caja de mensaje por un mensaje en la barra de estado

   ```
   stbPrestamo.Panels("msg").Text = "Inserta un valor numérico, por favor."
   ```

   b) Reemplaza la caja de mensaje que se utiliza para desplegar los resultados del procedimineto VerMensuales, por un mensaje en la barra de estado.

   ```
   stbPrestamo.Panels("msg").Text = "Pagos mensuales de " & _
                                    Format( dblMensual, "currency")
   ```

1. En el procedimiento VerTotal, reemplaza cada caja de mensaje por un texto en la barra de estado.

2. Guarda y prueba la aplicación.

Ejercicio 8.3

Agregando una lista de imágenes (image list) a frmPrincipal

1. Agrega un control imagelist a la forma frmPrincipal
   La lista de imágenes es un control invisible en tiempo de ejecución, por lo cual no hay que preocuparse por la posición que tiene.

2. En la ventana de propiedades, da el nombre de imgPrestamo en la propiedad Name para en la lista de imágenes.

3. Abre la ventana de propiedades Custom para la lista de imágenes.

4. Selecciona la pestaña general y da la opción de tamaño por omisión que es de 16x16

5. En la pestaña de Images inserta 2 imágenes que están en el directorio

   \vb6int\lab08\exit.bmp
   \vb6int\lab08\month.bmp

6. Elige OK

Agregando una barra de herramientas (toolbar) con 2 botones a frmPrincipal

1. Agrega una barra de herramientas (toolbar) en la parte superior de frmPrincipal

2. En la ventana de propiedades de Visual Basic, da el nombre tlbPrestamo, desde la propiedad Name.

3. Abre la ventana de propiedades Custom para la barra de herramientas

4. En la pestaña General, da el valor a la propiedad ImageList, proporcionando el nombre de la lista de imágenes que ya creaste, imgPrestamo.

5. En la pestaña de Buttons, inserta 2 botones con las siguientes propiedades

Button index	Propiedad	Valor
1	Description	Botón de Exit
	key	exit
	ToolTip Text	Exit
	Image	1
2	Description	Botón de Pago Mensual

key	mes
ToolTip Text	Pago Mensual
Image	2

6. Da OK

Respondiendo al evento click del botón de la barra de herramientas

Todos los botones dentro de la barra de herramientas responden al mismo evento click. Por ello se debe determinar exactamente cuál de ellos fue invocado; y para ello se debe hacer una selección a través de la propiedad "key" de la barra de herramientas.

1. Abre la ventana de código para el evento click de la barra de herramientas
   Sub tlbPrestamo_Click()

   End Sub

2. Agrega el código que determina cual botón ha sido seleccionado y que ejecute el procedimiento adecuado.

```
Select Case Button.Key
        Case " mes"
                VerMensuales
        Case "exit
                Unload frmPrincipal
End Select
```

3. Guarda y prueba la aplicación.

Laboratorio 9

Accediendo a bases de datos

Laboratorio 9
Accediendo a bases de datos

Ejercicio 9.1
Creando una forma para la información de Clientes a través del Data Form Designer

Construyendo forma Cliente

1. Abre un nuevo proyecto
2. Desde el menú Add-Ins, elige **Visual Data Manager**
3. Desde el menú File de la nueva ventana elige **Open database -- Microsoft Access...**
4. Selecciona Nwind.mdb del directorio de Visual Basic y da **Open**
5. Del menú **Utility** selecciona **Data Form Designer ...**
6. Da el nombre *Clientes* a la forma en **Form Name**
7. En la lista de RecordSource, elige **Customers**
8. Copia los campos CustomerID y CompanyName de la lista Available Fields a la lista de Included Fields
9. Da clic en **Build The Form**
10. Da clic en **Close**
11. Cierra la ventana de **Visual Data Manager**

Borrando la forma por omisión y fijando la forma de arranque

En esta aplicación no se utilizará la forma Form1 que crea por omisión Visual Basic al generar un nuevo proyecto.

1. En la ventana de proyecto, selecciona Form1
2. Desde el menú Project, elige **Remove Form1**
3. Desde el menú Project, elige **Project Properties ...** En la pestaña **General** ve a la opción **Startup Object** y elige la forma *Clientes* que se generó con **el Data Form Designer**.

Guarda y prueba tu trabajo

1. Guarda el proyecto y la forma en el directorio \vb6int\labs\lab09 seleccionando la opción **Save Project As** del menú File

   Esto es porque el **Data Form Designer** intenta guardar los archivos en los directorios de Visual Basic.

2. Prueba la aplicación, utiliza los botones de adelantar y regresar del control Data, para pasar entre los registros. Cambia algunos valores de los campos y selecciona el botón **Update**. Elige **Close** para terminar la aplicación.

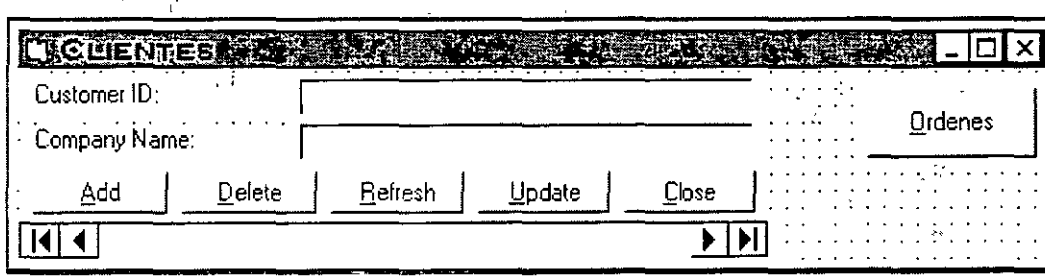
Ejercicio 9.2

Creando una forma para la información de Ordenes a través del Data Form Designer

1. Desde el menú Add-Ins, elige **Visual Data Manager**
2. Desde el menú File de la nueva ventana elige **Open database -- Microsoft Access...**
3. Selecciona Nwind.mdb del directorio de Visual Basic y da **Open**
4. Del menú **Utility** selecciona **Data Form Designer ...**
5. Da el nombre *Ordenes* a la forma en **Form Name**
6. En la lista de RecordSource, elige Orders
7. Copia los campos CustomerID, OrderID, OrderDate y RequiredDate de la lista Available Fileds a la lista de Included Fields
8. Da clic en **Build The Form**
9. Da clic en **Close**
10. Cierra la ventana de **Visual Data Manager**

Mostrando la forma Ordenes desde la forma de Clientes

1. En la forma Clientes agrega un botón de comando cuya propiedad Caption sea Ordenes y su propiedad Name cmdOrdenes.



Nota: Fija la propiedad Align del control Data en 0-None para que puedas darle el tamaño que desees y no tome siempre todo el ancho de la forma.

2. En el evento click del botón cmdOrdenes que se muestre la forma Ordenes en forma modal

```
frmOrdenes.Show vbModal
```

Guarda y prueba la aplicación

1. Guarda la forma órdenes en el directorio \vb6int\labs\lab09 con la opción **Save Form As** del menú File.
2. Ejecuta la aplicación. Elige el botón **Ordenes**, recorre algunos registros de la tabla Ordenes. Cierra la forma y regresa a la forma de Clientes.

Ejercicio 9.3

Creando una instrucción SQL

En este ejercicio harás que exista un vínculo entre las formas de clientes y órdenes, es decir, que para un cliente en particular, al dar clic en el botón de Ordenes, solo se desplieguen las órdenes correspondientes a él en la forma de órdenes, y no todas las órdenes como hasta el momento.

Cambiando la forma Ordenes

1. Abre la ventana de código de la forma frmOrdenes
2. En la sección de General Declarations, declara una variable publica de tipo String que se llame strCustomerID

```
Public strCustomerID as String
```

3. En el procedimiento asociado al evento Load de la forma Ordenes:
   a. Crea una cadena SQL que selecciones todos los campos de la tabla Orders, cuando el CustomerID sea igual a la variable strCustomerID
   b. Da esta cadena como valor de la propiedad RecordSource del control Data
   c. Genera el evento Refresh para el control Data.

```
Dim strSQL as String
StrSQL = "SELECT * FROM Orders " & _
         "WHERE [Customer ID] = ' " & strCustomerID & " ' "
Data1.RecordSource = strSQL
Data1.Refresh
```

Cambiando la forma Clientes

1. En la forma frmClientes
2. Edita el procedimiento asociado al evento clic del botón de comando cmdOrdenes
3. Da valor a la variable publica que se definió en la forma de órdenes frmOrdenes.strCustomerID igual al contenido de la caja de texto de la forma que contiene el valor de Customer ID

```
frmOrdenes.strCustomerID = txtFields(0).Text
```

Guarda y prueba la aplicación

1. Guarda el proyecto
2. Ejecuta la aplicación, recorre los registros eligiendo el botón de órdenes y verifica que efectivamente sólo se muestren las órdenes correspondientes al cliente que actualmente se despliega.
3. Cierra la aplicación.