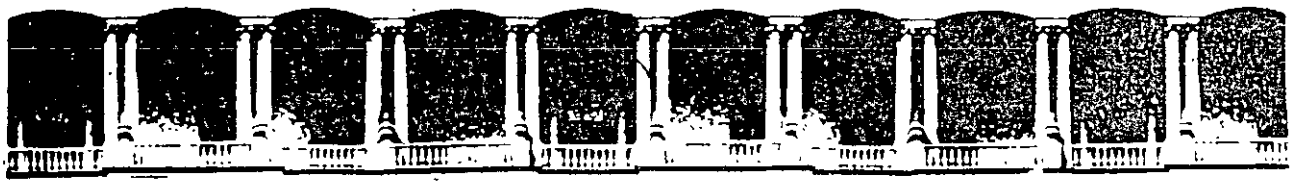


FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

INTRODUCCION A VISUAL BASIC

14 al 25 de abril de 1997

DIRECTORIO DE PROFESORES

TEC. FABIAN OLIVER SUAREZ

ASESOR PARTICULAR
4a. CERRADA DE CAFETAL No. 16
COL. GRANJAS MEXICO
DELEGACION IZTACALCO
C.P. 08400 MEXICO, D.F.
TEL: 650 40 74

'pmc.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

CURSO: *CC021 INTRODUCCION A VISUAL BASIC*
FECHA: *14 al 25 DE ABRIL*

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN. 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
TEC. FABIAN OLIVER SUAREZ				

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 *Excélsior*	
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:

_____

_____

_____

_____

_____

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

CURSO: *CC021 INTRODUCCION A VISUAL BASIC*
FECHA: *14 al 25 DE ABRIL*

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN: 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
TEC. FABIAN OLIVER SUAREZ				

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

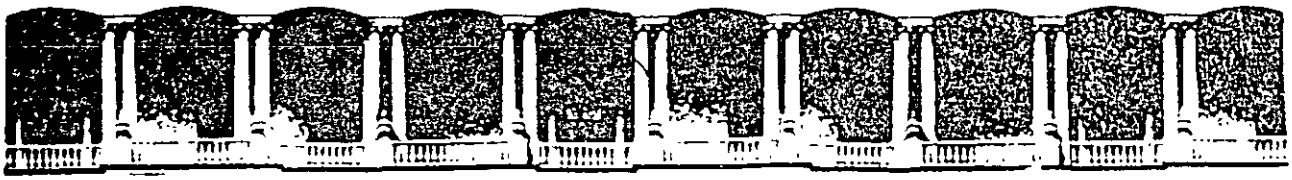
3. ¿Qué cambios sugeriría al curso para mejorarlo?

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 DIDACTICO DEL CURSO

INTRODUCCION A VISUAL BASIC

ABRIL, 1997

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

1. INTRODUCCIÓN A VISUAL BASIC

Introducción

Visual Basic es, sin lugar a dudas, una de las mejores herramientas de programación que se ha creado para las PC's. Puede decirse que es el sueño de cualquier programador, especialmente para los programadores en Windows. Desarrollar una aplicación para ambiente Windows en C era y sigue siendo una tarea muy larga y tediosa. Las interfaces gráficas de usuario (GUI) están teniendo un gran auge y todo parece indicar que el futuro de la programación esta centrado en ellas, sin embargo representan todo un reto para el programador debido a su dificultad para desarrollarse.

Visual Basic cambia este esquema por completo. La idea es simple, si se desea una ventana de cierto tamaño, solo tiene que dibujarse. Si se desea una caja de texto en cierto lugar de la ventana, basta con seleccionar la herramienta adecuada y dibujar la caja de texto en el lugar deseado.

1.1 Acerca de la Programación en Windows

Las Partes de una ventana

La figura siguiente muestra una ventana típica de Windows 3.x. Antes de comenzar a programar, se debe estar seguro de reconocer y estar familiarizado con todos los elementos de una ventana. De hecho, si se trata de un programador que no está familiarizado con Windows, es conveniente que primero se familiarice con el ambiente y después programe en Visual Basic.

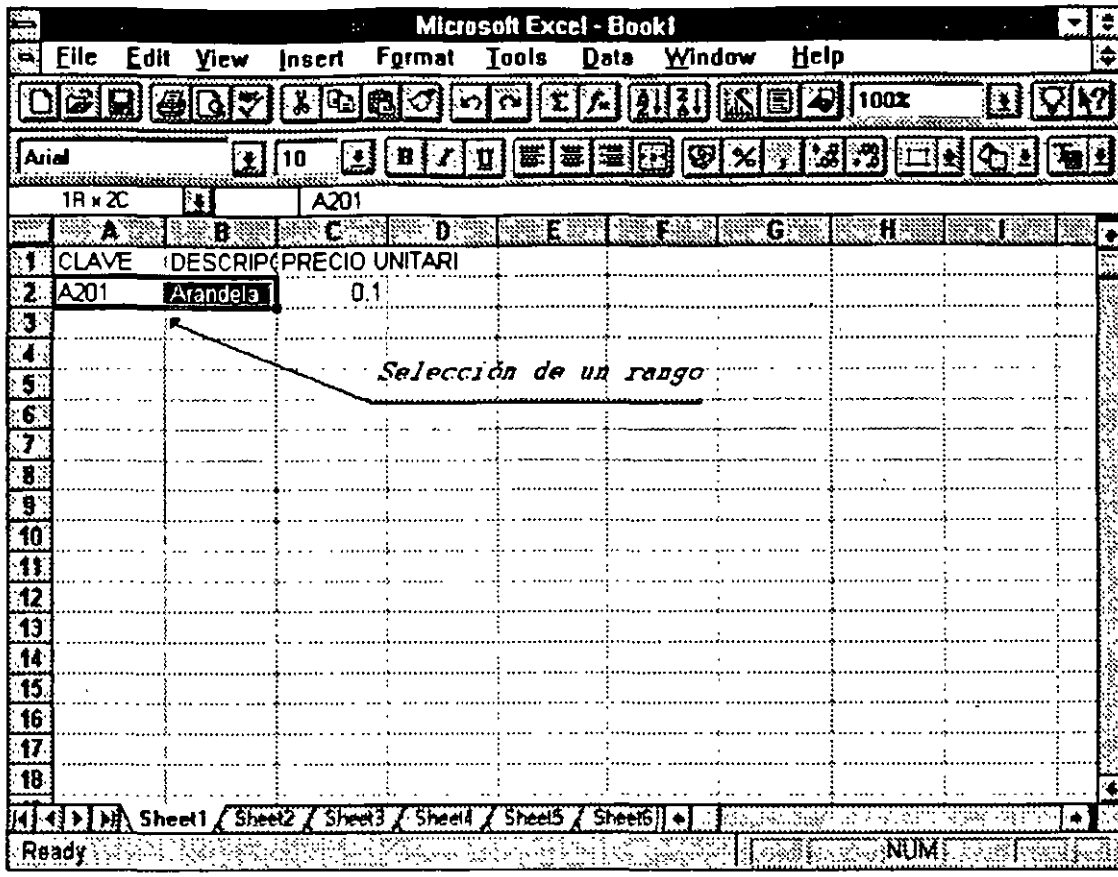


Figura 1.1

Preservando la apariencia de Windows

Como se mencionó, antes de comenzar a programar en Visual Basic se debe estar seguro de conocer la forma en que trabaja Windows. En particular, se debe estar completamente familiarizado con los términos Click y Doble Click, y anticiparse a lo que el usuario espera de una aplicación.

Por ejemplo, el hecho de que en el menú File contenga la opción Exit y esta se encuentre siempre al final, es parte de la interface Windows; y así como esta existen muchas otras características más de Windows que se deben conocer para que puedan ser integradas en las aplicaciones desarrolladas con Visual Basic

Todas estas características hacen muy diferente la programación para Windows de la programación en DOS. Para empezar, los programas DOS están escritos en forma secuencial; esto es, un evento sigue a otro. En un programa DOS, el control sigue la secuencia de las sentencias, más o menos como el programador lo diseño.

Una aplicación bajo Windows presenta típicamente todas las opciones en una pantalla (en la forma de objetos visuales). De esta forma, esto representa una forma completamente nueva de programación - orientada a eventos y orientada a objetos. Es decir, el programador no es completamente responsable del flujo del programa - el usuario es el responsable. El usuario selecciona una de las opciones posibles, y le corresponde al programa responder correctamente.

La forma de trabajar de la programación orientada a eventos requiere que toda la atención del diseño de código recaiga sobre la interface.

1.2 Acerca de la programación en Visual Basic

Bajo Windows el usuario es el que manda - y hasta hace poco el programador era el que pagaba el precio. Ahora con Visual Basic, el programador tiene a su alcance una herramienta para desarrollar aplicaciones para Windows, que le brinda la misma facilidad de manejo que cualquier otra aplicación para este ambiente.

Existen tres pasos principales para generar una aplicación en Visual Basic :

1. Dibujar la(s) ventana(s) deseada(s).

2. Personalizar las propiedades de los botones, cajas de texto, etc.

3. Escribir el código para los eventos asociados

El primer paso -dibujar la ventana de la forma deseada, incluyendo botones y menús- es donde realmente destaca Visual Basic. Anteriormente, la parte más tediosa del desarrollo de una aplicación para Windows, era el diseño de la interface. Añadir o eliminar características resultaba una tarea muy difícil. Bajo Visual Basic, sin embargo, todo este proceso se vuelve algo extremadamente fácil.

El siguiente paso involucra la personalización de las propiedades de todo aquello que se ha dibujado (formas, botones, cajas de texto, cajas de selección, etc.), por ejemplo, se puede cambiar el color de fondo de una forma, o se puede cambiar el tipo de letra para el *caption* de un botón.

Finalmente, se debe escribir el código que responderá a los eventos que se consideran significativos.

1.3 Reconociendo el Ambiente de Visual Basic

Visual Basic está compuesto por varias ventanas que le permiten al programador manipular todos los objetos de su aplicación. Estas ventanas son :

- Forma
- Proyecto
- Propiedades
- Caja de Herramientas
- Barra de Herramientas
- Código

Las ventanas de Forma y Proyecto

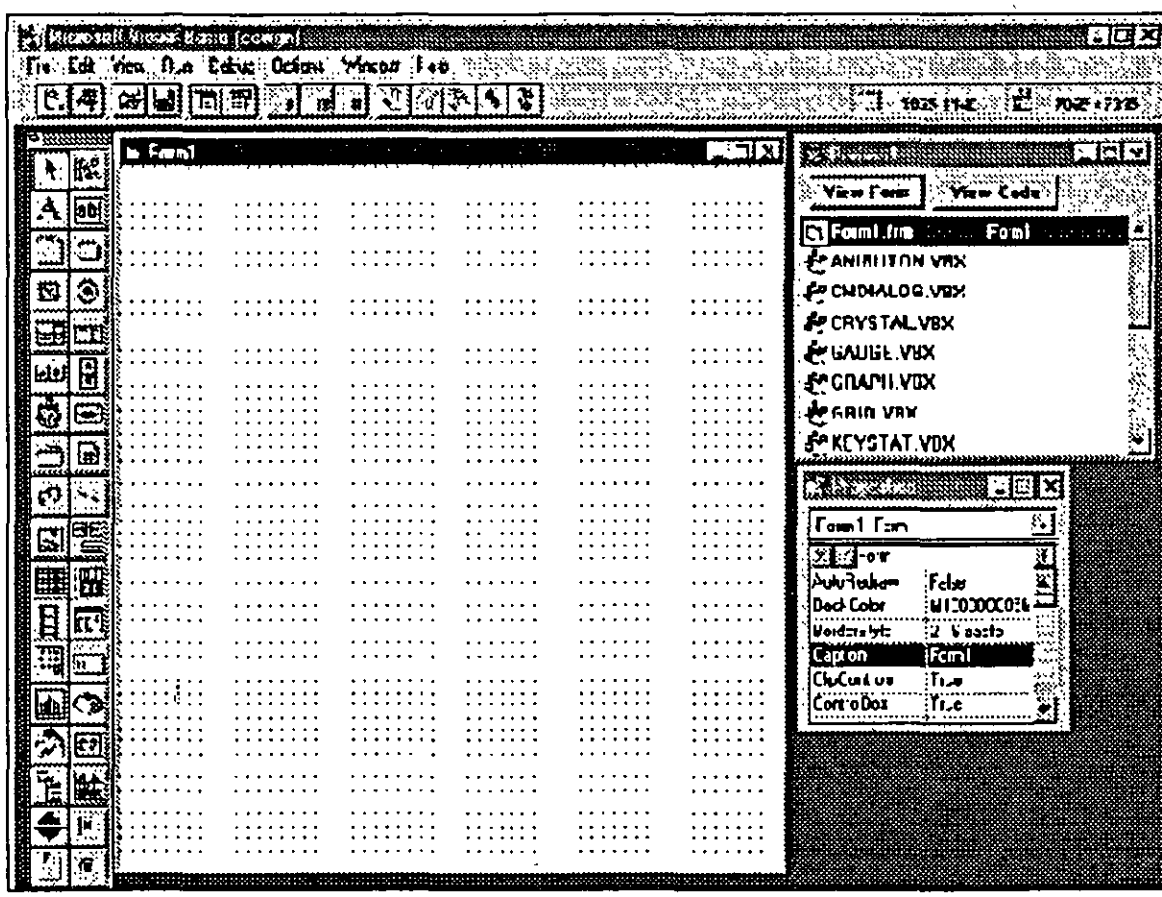


Figura 1.2

Las formas son el corazón de una aplicación gráfica. El usuario interactúa con ellas para realizar alguna tarea.

En las formas, el programador comienza a desarrollar su aplicación y es en ellas en donde se colocan los controles - *command buttons, list boxes, option buttons-* que le dirán al usuario que opciones tiene.

Las formas son las ventanas que se diseñan en Visual Basic. Las aplicaciones usualmente tienen cuando menos una forma (pero no es técnicamente necesario).

La ventana de Proyecto es una lista que Visual Basic utiliza para llevar un registro de las formas que se útilizan en su aplicación. Se tendrán tantos archivos .FRM dentro de la ventana de Proyecto como formas existan en su aplicación. Además, usted puede tener otros archivos - .BAS y .VBX.

Las ventanas de Propiedades y de la caja de herramientas

Como su nombre lo indica, la caja de herramientas (*Toolbox*) es donde se pueden obtener los elementos básicos para construir cualquier aplicación basada en ventanas utilizando Visual Basic.

Existen dos formas para colocar controles dentro de una forma, haciendo doble click sobre el control en la caja de herramientas o haciendo un click y arrastrando el control. En cualquier caso se obtiene el mismo resultado. Cuando se hace doble click sobre un control, este aparece a la mitad de la forma y usted solo tiene que arrastrarlo para dejarlo en el lugar deseado.

Cada control que se coloca en una forma tiene un conjunto de propiedades asociadas que pueden ser modificadas para modificar su apariencia.

¿Que son las propiedades y como se establecen?

Las propiedades para un control indican las características que este tendrá al momento de ejecutarse la aplicación y se establecen utilizando la ventana de propiedades como la que se muestra a continuación :

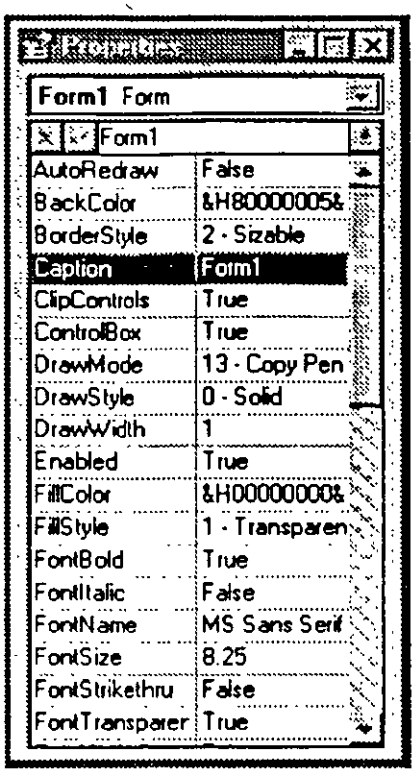


Figura 1.3

Para establecer una propiedad

1. Hacer click sobre el control al cual se le quiere cambiar las propiedades, (p.e. un *command button*) y presionar la tecla F4 para llamar a la ventana de propiedades.

2. Desplazarse sobre la barra de Scroll vertical hasta encontrar la propiedad indicada, y seleccionarla.

3. Colocar el valor para la propiedad en la caja de Settings.

4. Hacer click en la caja de verificación a la izquierda de la caja Settings.

La barra de herramientas (*Toolbar*) brinda un acceso rápido a los comandos o funciones más comunes. Estas funciones (como guardar proyectos e iniciar una aplicación durante la fase de diseño) se encuentran también disponibles en los menús de Visual Basic.

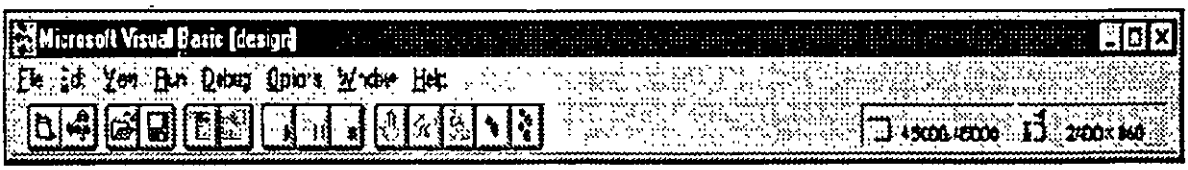


Figura 1.4

Nueve de los elementos más relevantes de la barra de herramientas son :

Toolbar	Menú
New Form	Del menú File elegir New Form
New Module	Del menú File elegir New Module
Open Project	Del menú File elegir Open Project
Save Project	Del menú File elegir Save Project
Menu Design window	Del menú Window elegir Menu Design
Properties window	Del menú Window elegir Properties
Start	Del menú Run elegir Run
Break	Del menú Run elegir Break
End	Del menú Run elegir End

Coordenadas de Posición y Tamaño

Visual Basic despliega las coordenadas de la esquina superior izquierda de cada control de forma relativa a la esquina superior izquierda interior de la forma donde se encuentra dicho control. También despliega el ancho y alto de cada control. Se pueden establecer los valores para estas propiedades colocando la forma y los controles en el lugar aproximado en donde se desean. La unidad en que se miden estas distancias es el *twip*. Una pulgada equivale a 1440 twips.

¿Que es la ventana de código?

Las ventanas de código despliegan el código implementado en una aplicación. Al inicio, las ventanas de código solo contienen las plantillas para procedimientos y funciones, y se puede ir agregando código conforme se desarrolla la aplicación. Existen varias formas para abrir una ventana de código. La forma más fácil es hacer doble click sobre un control en el modo de diseño. Por ejemplo, para ver la ventana de código con la plantilla para el evento de Click de un botón de comando, solo se tiene que hacer doble click sobre él y aparecerá una ventana como la siguiente :

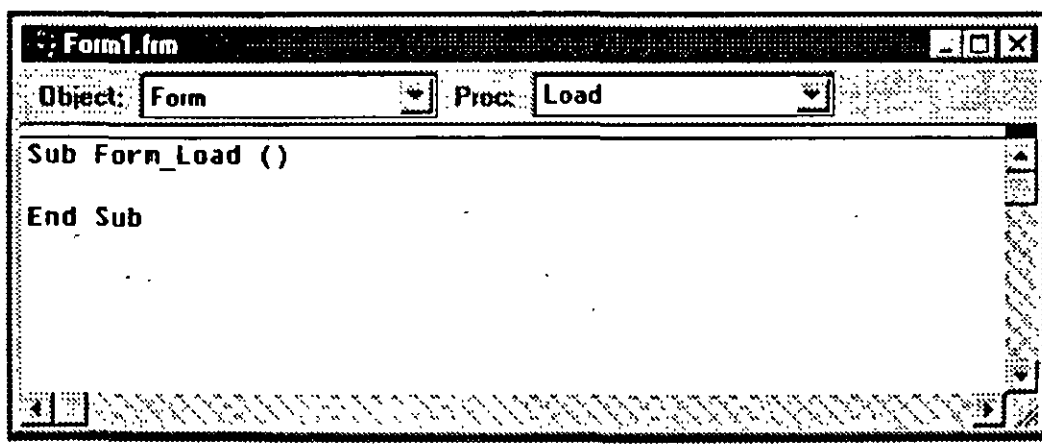


Figura 1.5

1.4 Los proyectos de Visual Basic

Para crear una aplicación en Visual Basic, se trabaja con proyectos. Un proyecto es la colección de archivos que se utilizan para construir una aplicación.

Conforme se va desarrollando la aplicación, se utiliza un proyecto para administrar todos los diferentes archivos que se crean. Un proyecto puede contener cualquiera de los siguientes tipos de archivos :

- Un archivo por cada forma (.FRM).
- Un archivo por cada módulo (.BAS).
- Un archivo para cada control personalizado (.VBX).
- Un archivo de proyecto en donde se tiene toda la información concerniente al proyecto (.MAK)

El archivo de proyecto no contiene formas o módulos; solo contiene una lista de todos los archivos asociados con un proyecto particular, así como información de las opciones del ambiente establecidas.

Visual Basic organiza las tareas por proyectos. Una aplicación puede tener varias formas asociadas con ella y se pueden agrupar todas en un solo proyecto lo que permite una fácil manipulación. Visual Basic permite sólo un proyecto abierto a la vez, y cada proyecto puede tener tres partes diferentes.

Cuando se inicia Visual Basic, automáticamente se crea un proyecto llamado Project1, sin embargo es posible renombrarlo utilizando el comando Save Project As... del menú File. También es posible abrir un proyecto existente utilizando el comando Open Project... del mismo menú.

La ventana de proyecto es útil cuando se tienen múltiples formas o módulos.

También puede notarse en la ventana de proyecto que Visual Basic asigna el nombre para la forma por default como Form1.Frm; la extensión .Frm es la correspondiente para las formas, tal y como la extensión .Bas es la correspondiente para los archivos Basic. Una vez seleccionando la forma, se puede brincar entre la forma y la ventana de código asociado a ésta, utilizando los botones de la ventana de proyecto View Code o View Form.

Los proyectos son grabados con la extensión .Mak (con las formas y módulos asociados, almacenados separadamente). Inicialmente, cuando se crea un nuevo proyecto, incluyendo nuevas formas y módulos, Visual Basic no crea de forma automática los archivos correspondiente en disco. Esto permite grabar sólo los archivos que se desean, aunque Visual Basic preguntará si se desean grabar los archivos al momento de cerrar el programa (en caso de no haberlos grabado antes).

2. FUNDAMENTOS DE PROGRAMACION

2.1 Estructura de una Aplicación Visual Basic

Visual Basic es una lenguaje de programación completo que soporta las construcciones de programación estructurada más frecuentes en otros lenguajes modernos de programación.

Una aplicación puede contener formas (.FRM) y módulos de código (.BAS), así como controles personalizados (.VBX). La forma contiene los elementos visuales incluyendo todos los controles en la forma y el código Basic asociado con ella.

Como funciona una aplicación orientada a eventos

Un evento es una acción reconocida por una forma o un control. Las aplicaciones orientadas a eventos ejecutan código Basic en respuesta a un evento. Cada forma y control en Visual Basic tiene un conjunto predefinido de eventos. Si alguno de estos eventos ocurre, Visual Basic invoca el código en el procedimiento del evento asociado.

Aún cuando los objetos en Visual Basic reconocen de forma automática un conjunto predefinido de eventos, el programador determina si responde y la forma en que lo hace para un evento particular. Cuando se desea que un control responda a un evento, se escribe el código llamado *procedimiento del evento.*

Muchos objetos reconocen el mismo evento, sin embargo dos objetos diferentes pueden ejecutar dos procedimientos diferentes cuando el evento ocurre. Por ejemplo, un evento Click ocurre cuando el usuario hace click sobre algún objeto. Si el usuario hace click sobre una forma, el procedimiento para el evento Form_Click se ejecuta; si el usuario hace click sobre un botón llamado Command1, el procedimiento para el evento Command1_Click se ejecuta.

Esto es lo que sucede con una aplicación típica orientada a eventos:

1. La aplicación comienza y automáticamente carga y despliega la forma de inicio.

2. Una forma o control reciben un evento. El evento puede ser ocasionado por el usuario (por ejemplo el oprimir una tecla), por el sistema (por ejemplo un evento timer), o indirectamente por el código (por ejemplo un evento Load cuando se carga una forma).

3. Si existe algún procedimiento asociado con el evento ocurrido, éste se ejecuta.

4. La aplicación espera por el siguiente evento.

Nota: Muchos eventos ocurren en conjunción con otros eventos. Por ejemplo, cuando el evento DblClick ocurre, los eventos MouseDown, MouseUp y Click ocurren también.

La programación orientada a eventos vs. la programación tradicional

En una aplicación tradicional o "procedural", es la misma aplicación la que controla el flujo del código a ejecutar. La ejecución comienza con la primera línea y sigue un camino predeterminado a través de la aplicación, sólo se llama a los procedimiento cuando sea necesario y según se haya establecido por el programador.

En los programas orientados a eventos, una acción del usuario o un evento del sistema ejecutan un procedimiento de evento. Esto es, el orden en el que se ejecuta el código de la aplicación depende de el orden en el que los eventos ocurren, que a su vez depende de lo que el usuario decida hacer. Esta es la esencia de las interfaces gráficas de usuario y de la programación orientada a eventos: El usuario se encuentra a cargo del programa, y el código sólo responde.

Debido a que no se puede predecir lo que el usuario hará, el código de la aplicación debe hacer algunas consideraciones acerca de "el estado del mundo" cuando se ejecuta. Esto quiere decir que se debe validar de la forma más exhaustiva, la información que entrará al programa. Por ejemplo, si se requiere tener texto en una caja de texto antes de presionar un botón, sería conveniente deshabilitar el botón hasta estar seguros que ha ocurrido el evento Change para la caja de texto.

Código que se ejecuta al iniciar la aplicación

Por default, la primer forma que se crea en la aplicación se toma como la forma de inicio (*startup form*). Cuando la aplicación comienza a ejecutarse, esta forma se despliega (de esta forma el código que se ejecuta primero es el que se encuentra en el procedimiento Form_Load). Si se requiere que otra forma se despliegue al momento de iniciar la aplicación, se debe cambiar la forma *startup*.

Para cambiar la forma *startup:*

1. Del menú Options, seleccionar Project.
   Se desplegará la caja de diálogo de Project Options.

2. Seleccionar la opción Start Up Form.

3. En la lista de la opción Start Up Form, seleccionar la forma que será la nueva forma Startup.

El término de una aplicación

Una aplicación orientada a eventos deja de ejecutarse cuando todas las formas se cierran y no se está ejecutando código alguno. Si existe alguna forma oculta cuando la última forma visible se cierra, la aplicación aparentemente se habrá detenido, sin embargo la forma oculta seguirá abierta y el código asociado con ésta seguirá ejecutándose.

La forma más simple de evitar este problema es utilizando la sentencia **End**, la cual detiene la ejecución del código de toda la aplicación y cierra cualquier archivo que pudiera estar abierto. Por ejemplo, se puede tener un botón llamado cmdQuit que permita salir de un programa. El procedimiento para el evento Click podría ser así de simple :

```
Sub cmdQuit_Click ( )
   End
End Sub
```

La sentencia **End** finaliza una aplicación inmediatamente; ningún código después de la sentencia **End** es ejecutado, no ocurre ningún evento posterior.

Módulos

Las aplicaciones sencillas pueden tener sólo una forma, y todo el código de la aplicación puede residir en el módulo de la forma. Conforme la aplicación va creciendo y se va haciendo más sofisticada, se van agregando formas. Eventualmente se puede encontrar que un fragmento de código debe ser ejecutado en varias formas. No se puede invocar un procedimiento de una forma a otra y no es recomendable duplicar el código en ambas formas. Se debe entonces crear un módulo separado que contenga el procedimiento en común para ambas formas, para poder invocarlo desde cada uno de los módulos correspondientes a cada forma. De esta forma se pueden crear bibliotecas de funciones que pueden inclusive utilizarse en varias aplicaciones.

Cada módulo de forma y código puede contener :

Declaraciones. Se puede establecer constantes, tipos, variables, y declaraciones de procedimientos DLL (Dynamic Link Library) al nivel del módulo de forma o del módulo de código. Sin embargo, no puede colocarse código ejecutable (sólo declaraciones) al nivel módulo.

Procedimientos de Eventos (*Event procedures*). Estos son los procedimientos **Sub** que son ejecutados en respuesta a un evento del usuario o del sistema. Los *event procedures* solo pueden ocurrir en los módulos de las formas.

Procedimientos Generales. Estos son procedimientos que no están directamente asociados con un evento. Los procedimientos generales en un módulo de una forma son locales para ese módulo; no pueden ser invocados desde otro módulo. Todos los procedimientos que se encuentran en un módulo de código son generales, y pueden ser invocados desde cualquier código o forma de la aplicación. Los procedimientos generales pueden ser procedimientos **Sub** (los que no regresan un valor) o pueden ser procedimientos **Function** (los cuales regresan un valor).

Event Procedures

Cuando un objeto en Visual Basic reconoce que un evento ha ocurrido, automáticamente invoca al *event procedure* con el nombre que corresponde al evento. Debido a que el nombre establece una asociación entre el objeto y el código, los *event procedures* se encuentran vinculados con las formas y los controles.

Un *event procedure* para un control combina el nombre del control (especificado en la propiedad Name), un tilde abajo (_), y el nombre del evento. Por ejemplo, si se tiene un botón llamado MyButton y se quiere invocar a un *event procedure* cuando se hace click sobre él, se utiliza el procedimiento MyButton_Click.

Un *event procedure* para una forma combina la palabra Form, un tilde abajo, y el nombre del evento. Si se quiere llamar a un *event procedure* cuando se hace click sobre una forma, se utiliza el procedimiento Form_Click. (Como los controles, las formas tienen nombres únicos, pero éstos no se utilizan en los nombres de los *event procedures*.)

Para aclarar más estos conceptos, en la siguiente figura se encuentra una sintesis de los procedimientos.

En Visual Basic se tienen Formas y Módulos :

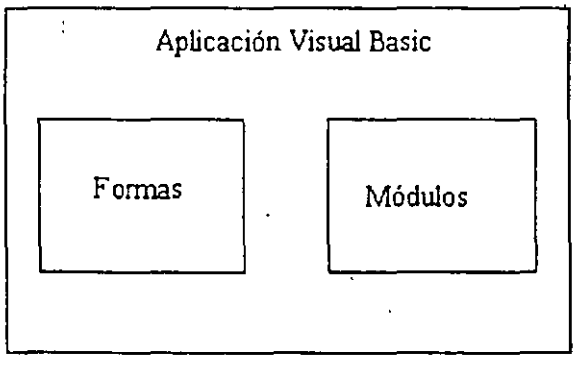


Figura 2.1

En los módulos se pueden encontrar declaraciones, procedimientos Sub y Function, todos globales.

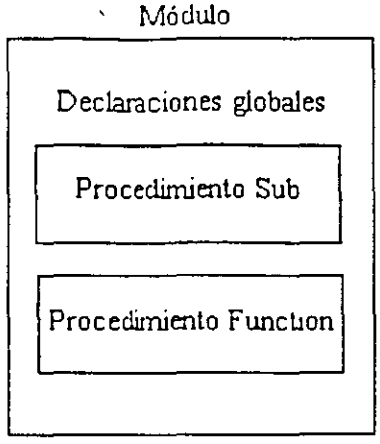


Figura 2.2

En la forma podemos encontrar los procedimientos asociados con los eventos de los objetos que se encuentran dentro de la forma llamados *event procedures*, por ejemplo **Sub** Command1_Click. También en la forma existe una sección llamada *general* en la que se pueden encontrar declaraciones globales de procedimientos, variables, constantes y tipos para los objetos contenidos en esa forma únicamente.

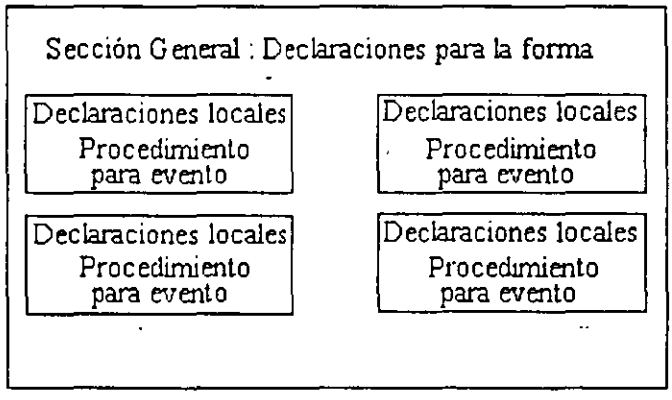


Figura 2.3

2.2 Administración del proyecto

Cuando se inicia Visual Basic, se crea automáticamente el proyecto1. Para renombrar el proyecto seleccione del menú File la opción Save Project As.. Para trabajar sobre un proyecto que ya existe, se puede seleccionar la opción Open Project... del menú File (figura 2.4). Para llevar un rastreo del proyecto actual, Visual Basic mantiene la ventana del proyecto (Project Window); el contenido de la ventana de proyectos para una aplicación se muestran en la figura 2.5.

La ventana de proyecto es muy útil cuando se tienen múltiples formas o módulos de código. Aquí como solo se cuenta con una forma, esta ventana no se ha utilizado.

También se puede notar que el nombre de la forma por default en la ventana de proyectos es Form1.Frm. La extensión .Frm es normal para formas así como la extensión .Bas es normal para archivos de Basic. Cuando se selecciona Form1 en la ventana de proyectos, se puede ir de un lado a otro en la forma y ver la ventana de código (que mantiene todo el código asociado a los objetos en el proyecto) al seleccionar los botones en la ventana de proyectos: View Form o View Code.

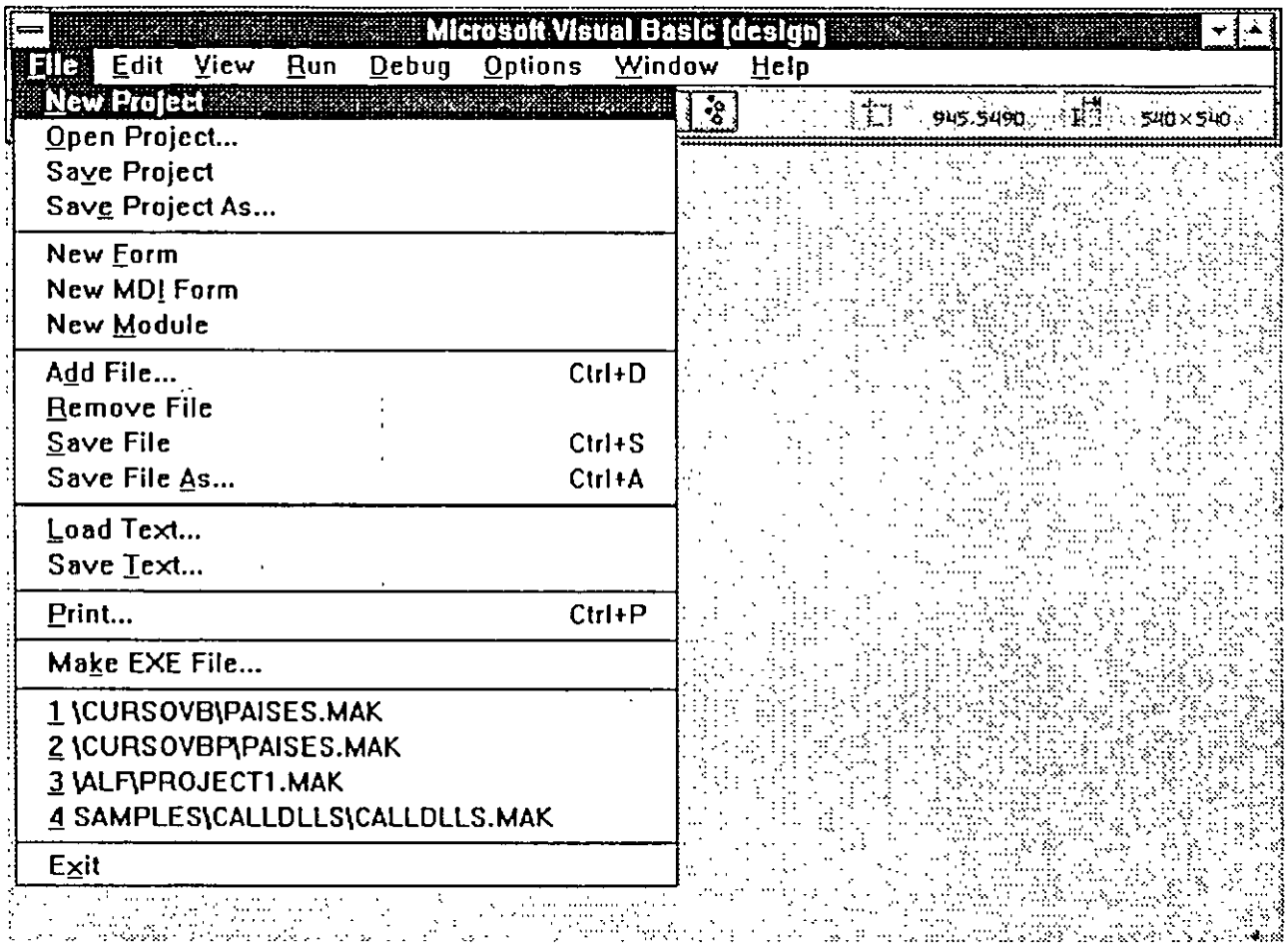


fig. 2.4

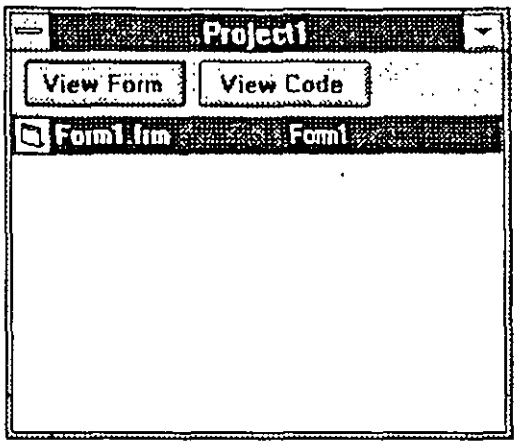


fig. 2.5

Los proyectos en si se guardan como archivos .Mak. Inicialmente cuando se crea un nuevo proyecto incluyendo nuevas formas y módulos, Visual Basic no crea automáticamente los archivos en disco. Los archivos se crean en disco hasta que se salvan. Ahora que se ha creado el proyecto, se necesita asegurar que se guarde en disco para que nada se pierda.

Guardando el trabajo en disco

Existen cinco tipos diferentes de Save en el menú de File de Visual Basic: Save Project, Save Project As.., Save File, Save File As.. y Save Text... La opción de Save Project guarda todos los archivos relacionados con el proyecto en disco. Sin embargo, para salvar un proyecto se debe de dar un nombre, esto se puede hacer con la opción de Save Project As.. que muestra una caja de diálogo. Con esta opción se puede seleccionar un nombre para el proyecto actual y guardar todos los archivos asociados. Una vez que se ha guardado por primera vez la opción Save Project guardará también los cambios sobre ese mismo archivo.

Además puede utilizarse la opción de Save File para guardar un módulo o forma. Nuevamente se le debe de dar un nombre la primera vez con la opción File Save As.. Este menú abre una caja de diálogo donde se debe de especificar el nombre del archivo. Almacenar archivos de esta manera los hace inaccesibles para un usuario fuera de Visual Basic ya que este los almacena con su propio formato binario por default.

Guardando el proyecto

Para salvar el proyecto actual, guarde el archivo Form1.Frm en la ventana de proyectos asignándole un nombre. Es mejor utilizar nombres nuevos para estos archivos, debido a que estos nombres son los default de Visual Basic y pueden ser fácilmente sobrescritos por proyectos posteriores. Después utilice Save Project As.. en el menú de File para salvar todo el proyecto nuevamente seleccionando otro nombre diferente a Project1.Mak. La próxima vez que se inicia Visual Basic, se puede volver a cargar este proyecto utilizando la opción de Open Project... en el menú de File. Visual Basic incluye una lista de los

proyectos utilizados recientemente al final del menú File. Cuando se desea abrir alguno de estos proyectos solamente se selecciona el proyecto deseado.

2.3 Procedimientos y Funciones

Los procedimientos pueden ser del tipo **Sub** o del tipo **Function**. Los procedimientos **Sub** no regresan un valor, así que un llamado a un procedimiento **Sub** es una sentencia completa.

Los procedimientos **Function** regresan un valor, así que el llamado a un procedimiento **Function** es parte de una expresión.

Nota: Los event procedures son siempre del tipo **Sub**, nunca del tipo **Function**.

Procedimientos Sub

La sintaxis para un procedimiento **Sub** es:

Sub *nombreprocedimiento* (*lista_argumentos*)
  sentencias
End Sub

lista_argumentos es una lista de nombres de argumento, separados por comas si existiera más de uno. Cada argumento se parece a la declaración de una variable y funciona como variable dentro del procedimiento. La sintaxis para el argumento es:

[**ByVal**] *nombre_variable* [()][**As** *tipo*]

Si no se especifica el *tipo*, el argumento toma el tipo **Variant**. El tipo **Variant** es muy flexible, muy a menudo no se tendrá que especificar el tipo del argumento. Sin embargo, el *tipo* puede ser cualquiera de los tipos fundamentales (**Integer**, **Long**, **Single**, **Double**, **Currency**, o **String**), un tipo definido por el usuario, o un tipo objeto. Los paréntesis después del *nombre_variable* indican que el argumento es un arreglo.

Cada vez que el procedimiento es llamado, las sentencias entre Sub y End Sub son ejecutadas. Visual Basic substituye cada referencia en un elemento de la lista de argumentos con el correspondiente argumento. Cuando Visual Basic llama a los procedimientos para eventos, reemplaza un valor para cada argumento (si el procedimiento tiene argumentos). Cuando se llaman procedimientos generales, el programador es el que tiene que reemplazar cada argumento con un valor. Por ejemplo, suponiendo que se tiene un procedimiento general definido como MultiBeep de la siguiente forma :

```
Sub MultiBeep (NBeeps)
Dim I
   For I=1 To NBeeps
```

```
        Beep
    Next I
End Sub
```

La siguiente sentencia llama al procedimiento MultiBeep con el argumento 3:

```
MultiBeep 3
```

El procedimiento substituye 3 por NBeeps (el nombre que aparece en la lista de argumentos). El procedimiento entonces ejecuta tres veces la función Beep que emite un pitido.

Normalmente, las llamadas a los procedimientos **Sub** de Visual Basic no requieren los paréntesis alrededor de la lista de argumentos. Si se llama a un procedimiento **Sub** con la sentencia opcional **Call**, sin embargo, se pueden utilizar los paréntesis :

```
Call MultiBeep (3)
```

Procedimientos Function

La sintaxis para un procedimiento **Function** es :

Function *nombre_procedimiento* (*lista_argumentos*) [**As** *tipo*]
  sentencias
End Function

Los argumentos para una función trabajan exactamente de la misma forma que los argumentos para procedimientos del tipo **Sub**. Además de la palabra **Function**, existen tres diferencias entre un procedimiento **Sub** y un procedimiento **Function**:

Siempre se utilizan los paréntesis cuando se llama a un procedimiento **Function** (no se puede utilizar la sentencia **Call** en un procedimiento **Function**).

Los procedimientos **Function** tienen tipos de datos, de la misma forma que las variables. Esto determina el tipo de valor que devolverán. En ausencia de la cláusula **As**, la función toma por default el tipo de dato **Variant**.

El valor de la función se regresa asignándolo a la misma función. Cuando el procedimiento Function regresa un valor, este valor es utilizado entonces cómo parte de una expresión más grande.

Por ejemplo, se tiene una función que calcula la hipotenusa de un triángulo rectángulo:

```
Function Hypotenuse (A, B)
```

```
    Hypotenuse = Sqr(A ^ 2 + B ^ 2)
End Function
```

Los procedimientos Function se llaman de la misma forma en que se llaman las funciones que vienen incluidas con Visual Basic :

```
Label1.Caption = Hypotenuse (Val(Text1.Text), Val(Text2.Text))

X=Hypotenuse(Width, Height)
```

2.4 Llamadas por Referencia y por Valor

Algunos procedimientos o funciones necesitan tomar parámetros al momento de ejecutarse, de esta forma (pasando parámetros) podemos establecer la comunicación entre varias rutinas o procedimientos.

La comunicación entre procedimientos es importante debido a que podemos establecer parámetros globales de control, llamadas *variables globales* o *constantes globales*, las cuales se declaran en la sección [general] [declarations] de un módulo .BAS y utilizarlas dentro de los eventos de los controles en varias formas.

Llamadas por referencia

Las llamadas por referencia y por valor se refieren a que, al momento de llamar a un procedimiento o función desde otra sección del programa, ese procedimiento o función puede necesitar parámetros que el programa que lo llame le tiene que entregar; por ejemplo

```
Sub Form_Load()
        Call Lineas (x1,y1,x2,y2)
End Sub
```

Este procedimiento llama a la función Lineas y le envía los parámetros (x1,y1,x2, y2) para que Lineas haga algo internamente con ellos. A esta forma de enviar los parámetros a una función se le llama por referencia, ya que se están enviando variables, en este caso x1,y1,x2,y2.

Cuando se declara la función se especifica si los parámetros que reciba serán por referencia o por valor. Por ejemplo, para la función anterior, su declaración podría ser:

```
Function Lineas (x1 As integer, y1 As Integer, x2 As Integer, y2 As Integer) As Integer
...
...
End Function
```

Llamadas por valor

Por otra parte, cuando se quieren enviar parámetros a una función o procedimiento, pero en lugar de variables se desean enviar valores constantes, se utiliza el llamado paso de parámetros por valor, esto es, llamar a la función desde cierta sección del programa y mandarle como parámetros valores numéricos constantes, por ejemplo:

```
Sub Form_Load()
        Call Lineas (1,1,25,53)
End Sub
```

De esta forma, se esta llamando a la función Lineas y se le envían los parámetros (1,1,25,53), que son valores constantes.

Al momento de declarar a las funciones se le especifica que recibirá parámetros por valor de la siguiente forma:

```
Function Lineas (ByVal x1 As integer, ByVal y1 As Integer, ByVal x2 As Integer,
                 ByVal y2 As Integer) As Integer
...
...
End Function
```

Con la palabra reservada ByVal, se le esta especificando a la función que los parámetros que reciba tendrán que ser forzosamente valores numéricos, en lugar de variables.

2.5 El Código en los Procedimientos

Comentarios, números y sentencias

Para que un programa sea entendible no sólo para el programador, sino también para otras personas, es necesario que este bien documentado. Visual Basic permite utilizar comentarios que no afectan en lo absoluto el desempeño del programa y que le permiten al programador clarificar detalles acerca de sus procedimientos.

Para establecer un comentario solo se tiene que anteponer un apóstrofe (') al texto. Por ejemplo :

```
' Este es un comentario que comienza en el extremo izquierdo
Text1.Text = "Hola"          ' Este comentario comienza después de una
sentencia.
```

En Visual Basic al igual que en cualquier otro lenguaje de programación, se utilizan muchos números. En la mayoría de los casos se utilizan números en base diez, sin

embargo ocasionalmente se puede requerir manejar un número en forma hexadecimal (bese 16) o en forma octal (base 8). Visual Basic representa los números hexadecimales con la notación &H y los números octales con &O.

Las sentencias de Visual Basic se encuentran normalmente en una línea, y no existe un terminador de sentencia. Sin embargo, se pueden colocar más de una sentencia en una sola línea si se utiliza el separador "dos puntos" (:), por ejemplo :

```
Text1.Text = "Hola" : Red =255 : Text1.BackColor = Red
```

Convenciones para nombres en Visual Basic

Mientras se escribe un programa en Visual Basic, se declaran y nombran muchos elementos (procedimientos Sub y Function, variables y constantes, etc.). Los nombres de los procedimientos, variables y constantes que se declaran en los programas de Visual Basic deben seguir las mismas reglas que se utilizan para nombrar las formas y los controles:

- Deben comenzar con una letra.
- Deben contener únicamente letras, números y el caracter de tilde abajo (_), los espacios y los caracteres de puntuación no se permiten.
- No deben ser mayores a 40 caracteres.
- Los nombres de los elementos que se declaran en el programa no pueden ser palabras reservadas.

Una *palabra reservada* es una palabra que Visual Basic utiliza como parte de su lenguaje. Esto incluye a las sentencias predefinidas (como **If** y **Loop**), funciones (como **Len** y **Abs**), métodos (como **Show** y **Move**), y operadores (como **Or** o **Mod**).

Las formas y controles si pueden llevar el nombre de una palabra reservada. Por ejemplo, se puede tener un control llamado Loop. Sin embargo, la forma de hacer referencia a estos controles varia un poco. En lugar de poner :

```
Loop.Visible = True
```

Se debe poner :

```
[Loop].Visible = True
```

De esta forma Visual Basic sabrá que se trata de un control con el nombre de una palabra reservada.

Estableciendo y rehabilitando las propiedades

Probablemente la sentencia más común en Visual Basic es la *sentencia de asignación*, la cual asigna un valor a una variable o a una referencia de propiedad copiando los datos de un lugar a otro. La sintaxis utiliza el signo igual (=):

destino = origen

La sentencia le indica a la aplicación, "Copia la información que se encuentra en el origen al destino." El destino debe ser una variable o la referencia a una propiedad. El origen puede ser cualquiera de las expresiones válidas en Visual Basic y puede involucrar cálculos.

Las sentencias de asignación en Visual Basic generalmente se utilizan para una de tres cosas :

- Establecer el valor de una propiedad.
- Rehabilitar el valor de una propiedad.
- Almacenar o recuperar datos en una variable.

Para establecer el valor de una propiedad en tiempo de ejecución, se debe poner la referencia a la propiedad (*objeto.propiedad*) en el lado izquierdo de la asignación. Por ejemplo :

```
Text1.Text = "Su nombre aquí"
Text1.BackColor = 0
```

Cuando se trata de una propiedad de una forma, se puede omitir el nombre de la forma :

```
Sub Form_Click ()
' Estableciendo la propiedad Caption de la forma
   Caption = " Ha hecho click sobre la forma "
End Sub
```

La propiedad Text es una cadena de caracteres que especifica el contenido de una caja de texto, una caja de lista, o una caja combo. Se pueden asignar números, texto o una combinación de ambos en las propiedades de texto.

Cuando se desea cambiar las propiedades de una forma distinta a la forma en la que se encuentra en ese momento el programa, se debe especificar el nombre de la forma :

```
Form2.Caption = "Ha cambiado la propiedad Caption de la forma2"
```

Si se desea cambiar las propiedades de un control desde una forma diferente a la forma que lo contiene, se debe mencionar, además del nombre del control, el nombre de la forma

donde se encuentra dicho control. La forma de especificar una forma distinta es la siguiente :

```
Form2!Text1.Text = "Este es el texto de una caja en la forma 2"
Form3!Button1.Caption = "Salir"    ' Esta linea cambia la propiedad Caption de un
Botón
                                   ' que se encuentra en la forma 3
```

Utilizando el valor de un control

Todos los controles tienen una propiedad que se puede utilizar para almacenar o recuperar valores sólo con hacer referencia al control, sin utilizar el nombre de la propiedad. Esta propiedad es llamada el *valor* del control y es usualmente la propiedad más utilizada de cada tipo de control. La siguiente tabla lista la propiedad de cada control, que se considera como el valor de dicho control.

Control	Propiedad
Check box	Value
Combo box	Text
Command button	Value
Data	Action
Directory list box	Path
File list box	Drive
Frame	Caption
Grid	Text
Horizontal scroll bar	Value
Image	Picture
Label	Caption
Line	Visible
List box	Text
Menu	Enabled
Option button	Value
Picture box	Picture
Shape	Shape
Text box	Text
Timer	Enabled
Vertical scroll bar	Value

En cualquier momento en que se desee hacer referencia a una propiedad de un control, y esta propiedad sea el valor para dicho control, bastará con que mencione el nombre del control y asigne el valor para dicha propiedad. Por ejemplo :

```
Text1 = "Este texto es asignado a la propiedad Text del control Text1"
```

En el siguiente ejemplo, a la propiedad Caption del control Label1 se le asigna el valor de la propiedad FileName del control File1 siempre que el usuario haga click sobre el control file list box :

```
Sub File1_Click ()
   Label1 = File1
End Sub
```

Debido a que el *valor* del control Label es el Caption y el *valor* para el control File list box es FileName (ver la tabla anterior) , no se requiere especificar dichas propiedades para realizar la asignación anterior.

2.6 Control de la Ejecución

Las sentencias que controlan las decisiones y los ciclos en Visual Basic son llamadas estructuras de control. Las estructuras de control de Visual Basic son similares alas estructuras que se encuentran en C y Pascal. Las estructuras de control más comúnmente utilizadas en Visual Basic son las siguientes:

- Bloques **If_Then**
- Bloques **If_Then_Else**
- Sentencias **Select Case**
- Ciclos **Do**
- Ciclos **For**

Los primeros tres elementos en la lista son estructuras de decisión. Se utilizan para definir grupos de sentencias que pueden ser o no ejecutadas, dependiendo de las condiciones del programa al momento de ejecutarse. Los últimos dos elementos son estructuras cíclicas. Se utilizan para definir grupos de sentencias que Visual Basic ejecutará repetidamente.

Estructuras de Decisión

Los procedimientos de Visual Basic pueden probar condiciones y después, dependiendo del resultado de la prueba, ejecutar diferentes operaciones. Las estructuras de decisión que soporta Visual Basic incluyen:

- **If...Then**
- **If...Then...Else**
- **Select Case**

If...Then

El bloque **If...Then** se utiliza para ejecutar una o más sentencias de forma condicionada. Se puede utilizar la sintaxis en una línea o la que incluye un bloque de líneas:

If *condición* **Then** *sentencias*

If *condición* **Then**
  sentencias
End If

La *condición* es usualmente una comparación, pero puede ser cualquier expresión que evalúe un valor numérico. Visual Basic interpreta este valor como **True** (verdadero) o **False** (falso); un valor de cero es considerado **False**, y cualquier otro valor diferente de cero es considerado **True**. Si la *condición* es **True**, Visual Basic ejecutará todas las sentencias que siguen a la palabra **Then**. Por ejemplo:

```
If anyDate < Now Then anyDate = Now

If anyDate < Now Then
   anyDate = Now
End If
```

Hay que notar que en el primer caso no se utiliza la sentencia **End If**. Si se desean ejecutar más de una línea de código cuando la condición es verdadera, se debe utilizar la instrucción **If** como se muestra en el segundo caso.

If...Then...Else

Se utiliza el bloque **If...Then...Else** para definir varios bloques de sentencias de los cuales solo uno se ejecutará:

If *condición1* **Then**
  [*bloque de sentencias* 1]
[**ElseIf** *condición* 2 **Then**
  [*bloque de sentencias* 2]]...
[**Else**
  [*bloque de sentencias N*]]
End If

Visual Basic primero prueba la *condición* 1. Si es falsa, Visual Basic procede a probar la *condición* 2 y así sucesivamente, hasta encontrar una condición verdadera. Cuando se encuentra una condición verdadera, Visual Basic ejecuta el *bloque de sentencias* correspondiente y después ejecuta el código siguiente a **End If**. Como una opción, se puede incluir un *bloque de sentencias* **Else**, el cual Visual Basic ejecutará si no se encontró alguna condición verdadera.

If...Then es realmente un caso especial del bloque **If...Then...Else**. Hay que notar que se puede tener cualquier número de cláusulas **ElseIf** o ninguna. La cláusula **Else** se puede incluir se tengan o no cláusulas **ElseIf**.

Select Case

Visual Basic brinda la estructura **Select Case** como una alternativa para el **If..Then...ElseIf** para seleccionar un *bloque de sentencias* que será ejecutado de entre otros muchos bloques. Una sentencia **Select Case** brinda una capacidad similar a la sentencia **If...Then...Else**, pero hace que el código sea más eficiente y entendible.

La estructura **Select Case** trabaja con una *expresión* que es evaluada una vez, al inicio de la estructura. Visual Basic entonces compara el resultado de esta expresión con los valores para cada **Case** en la estructura. Si alguno concuerda, se ejecuta el bloque de sentencias asociado con dicho **Case**:

Select Case *expresión a probar*
  [**Case** *lista de expresiones* 1
    [*bloque de sentencias* 1]]
  [**Case** *lista de expresiones 2*
    [*bloque de sentencias2*]]...
  [**Case Else**
    [*bloque de sentencias N*]]
End Select

Cada *lista de expresiones* es una lista de uno o más valores. Si existe más de un valor en la lista, estos se separan por comas. Cada *bloque de sentencias* contiene cero o más sentencias. Si más de un **Case** concuerda con la expresión probada, solo el *bloque de sentencias* asociado con el primer **Case** concordante es ejecutado. Visual Basic ejecuta las sentencias en la cláusula **Case Else** (la cual es opcional) si ninguno de los valores de las *listas de expresiones* concuerda con la *expresión probada*.

La estructura **Select Case** evalúa una expresión al inicio y solo lo hace una vez a diferencia de la estructura **If...Then...ElseIf** que evalúa una expresión diferente en cada sentencia **ElseIf**. Se puede reemplazar una estructura **If...Then...ElseIf** con una estructura **Select Case** sólo si cada sentencia **Else If** evalúa la misma expresión.

Estructuras de Ciclo

Las estructuras de ciclo permiten ejecutar una o más líneas de código repetidamente. Las estructuras de ciclo que Visual Basic soporta incluyen :

- **Do...Loop**
- **For...Next**

Do...Loop

El ciclo **Do** se utiliza para ejecutar un bloque de sentencias un número indefinido de veces. Existen algunas variaciones de la sentencia **Do...Loop**, pero cada una evalúa una

condición numérica para determinar si se continuará con la ejecución. Como con el **If...Then**, la condición debe ser un valor o expresión que puede ser evaluada como **False** (Cero) o como **True** (diferente de cero).

En la siguiente forma **Do...Loop**, las sentencias son ejecutadas mientras la condición sea verdadera (**True**):

Do While *condición*
    sentencias
Loop

Cuando Visual Basic ejecuta este ciclo **Do**, primero se verifica la *condición*. Si la *condición* es falsa, se brinca todo el bloque de *sentencias*. Si es verdadera, Visual Basic ejecuta todas las *sentencias* y vuelve a la sentencia **Do While** en donde vuelve a verificar la *condición*.

Consecuentemente, el ciclo puede ser ejecutado cualquier número de veces, mientras la *condición* siga siendo diferente de cero o verdadera. Las *sentencias* nunca se ejecutan si la *condición* es inicialmente falsa.

Otra variación de la sentencia **Do...Loop** ejecuta las *sentencias* primero y verifica la *condición* después de cada ejecución. Esta variación garantiza que al menos una vez se ejecutarán las sentencias.

Do
    sentencias
Loop While *condición*

Otras dos variantes son análogas a la anterior, excepto que en esos ciclos la condición debe ser falsa.

El ciclo se repite cero o más veces	**El ciclo se repite al menos una vez**
Do Until *condición* *sentencias* **Loop**	**Do** *sentencias* **Loop Until** *condición*

Hay que notar que **Do Until** condición es exactamente lo mismo que **Do While Not** *condición*.

For...Next

Los ciclos **Do** trabajan bien cuando no se sabe cuantas veces se requiere que se ejecute una sentencia en un ciclo. Cuando se sabe que una sentencia se debe ejecutar un número especifico de veces, resulta más eficiente utilizar el ciclo **For**. A diferencia del ciclo **Do**, el

ciclo **For** utiliza una variable contador que incrementa o decrementa su valor durante la repetición del ciclo. La sintaxis es :

For *contador* = *inicio* **To** *fin* [**Step** *incremento*]
  sentencias
Next [*contador*]

Los argumentos *contador*, *inicio*, *fin* e *incremento* son todos numéricos.

Nota: El argumento *incremento* puede ser positivo o negativo dependiendo de si se trata de un incremento o un decremento respectivamente. Si la cláusula **Step** no aparece, el incremento será de 1.

Cuando se ejecuta el ciclo **For**, Visual Basic:

1. Asigna el valor de inicio al contador.

2. Verifica que el *contador* sea mayor que *fin*. Si es así, Visual Basic se sale del ciclo. (si el *incremento* es negativo, Visual Basic verifica si el *contador* es menor que *fin*).

3. Ejecuta las *sentencias*.

4. Incrementa el *contador* en 1 - o lo decrementa según se haya especificado.

5. Repite del paso 2 al 4.

Estructuras de control anidadas

Como se podrá ver, se pueden colocar estructuras de control dentro de otras estructuras de control. (como un bloque **If** dentro de un ciclo **For**). Cuando se encuentra una estructura de control dentro de otra se dice que se encuentran anidadas.

En Visual Basic, las estructuras de control pueden anidarse a cualquier nivel, es decir, se pueden tener cuantas estructuras se deseen adentro de otra. Para hacer más entendible un programa, se suele indentar cada estructura de control con su cuerpo.

Por ejemplo, este procedimiento imprime todos los Fonts que son comunes tanto para la impresora como para la pantalla :

```
Sub Form_Click ()
Dim  SFont, PFont
   For SFont = 0 To Screen.FontCount - 1
         For PFont = 0 To Printer.FontCount - 1
               If Screen.Fonts(SFont) = Printer.Fonts(PFonts)
                  Print Screen.Fonts(SFont)
```

```
            End If
        Next PFont
    Next SFont
End Sub
```

Hay que notar que el primer **Next** cierra el ciclo **For** más interno (el que se encuentra más a la derecha), el último **Next** cierra el ciclo **For** más externo.

La salida de una estructura de control

La sentencia **Exit** permite salir directamente de un ciclo **For**, de un ciclo **Do**, de un procedimiento **Sub**, o de un procedimiento **Function**. Sintácticamente, la sentencia **Exit** es simple :
Exit For puede aparecer tantas veces como sea necesario dentro de un ciclo **For**, y **Exit Do** puede aparecer tantas veces sea necesario dentro de un ciclo **Do**:

For *contador* = *inicio* **To** *fin* [**Step** *incremento*]
  [*Bloque de sentencias*]
  [**Exit For**]
  [*Bloque de sentencias*]
Next [*contador* [,*contador*] [,...]]

Do [{**While** | **Until**} *condición*]
  [*Bloque de sentencias*]
  [**Exit Do**]
  [*Bloque de sentencias*]
Loop

La sentencia **Exit Do** trabaja con todas las versiones de sintaxis del ciclo **Do**.

Exit For y **Exit Do** son útiles porque algunas veces es apropiado salirse de un ciclo inmediatamente, sin ejecutar futuras iteraciones o sentencias dentro del ciclo.

La sintaxis de **Exit Sub** y **Exit Function** es similar a la de **Exit For** y **Exit Do**. **Exit Sub** puede aparecer las veces que sea necesario, en cualquier lugar del cuerpo de un procedimiento **Sub**. **Exit Function** puede aparecer las veces que sea necesario, en cualquier lugar del cuerpo de un procedimiento **Function**.

Exit Sub y **Exit Function** son útiles cuando el procedimiento ha hecho todo lo que necesita hacer y puede regresar inmediatamente.

2.5 Eventos de Control

Un evento es una acción que se ejecuta cuando el usuario realiza un click, un doble click, selecciona o deselecciona un objeto, entre otros. Para cada control existen distintos eventos que se pueden ejecutar y de ellos, algunos son mas importantes que otros debido a la frecuencia con que se utilizan o a la facilidad que le pueden brindar al usuario. A lo largo de este manual se mencionan varios controles y sus eventos mas importantes ejemplificados con aplicaciones prácticas; sin embargo, a continuación se muestran los principales eventos para la mayoría de los controles:

Click

Este evento se ejecuta cuando el usuario presiona el botón principal del mouse cuando el apuntador de éste se encuentra sobre el objeto. Aplica sobre los controles Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, cuadro, grid, etiqueta, caja de lista, menú, control OLE, botón de opción, caja de imagen y caja de texto.

DragDrop

A este evento se le llama también "arrastrar y soltar" y sucede cuando el usuario arrastra y suelta un control sobre otro. Se utiliza en los controles Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, cuadro, grid, etiqueta, caja de lista, menú, control OLE, botón de opción, caja de imagen, barras de desplazamiento, imagen y caja de texto.

DragOver

Este evento ocurre cuando se esta ejecutando el evento DragDrop y se especifica cierta acción cuando el control se desplaza por cierta área determinada de la pantalla. Los controles sobre los que aplica son: Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, cuadro, grid, etiqueta, caja de lista, menú, control OLE, botón de opción, caja de imagen, barras de desplazamiento, imagen y caja de texto.

GotFocus

Este evento sucede cuando un control obtiene el foco, es decir, es el control activo. Los controles sobre los que aplica son: Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, cuadro, grid, etiqueta, caja de lista, menú, control OLE, botón de opción, caja de imagen, barras de desplazamiento, imagen y caja de texto.

KeyPress

Este evento se ejecuta cuando el usuario presiona una tecla sobre el objeto. Los controles sobre los que aplica son: Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, grid, etiqueta, caja de lista, control OLE, botón de opción, barras de desplazamiento y caja de texto.

LostFocus

Este evento se realiza cuando un control tiene el foco y lo pierde (lo contrario de GotFocus). Los controles sobre los que aplica son: Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, grid, etiqueta, caja de lista, control OLE, botón de opción, barras de desplazamiento y caja de texto.

MouseDown

Este evento se genera cuando se presiona el botón principal del ratón. No es igual al click, porque este evento actúa cuando se detecta que se presionó el botón, no cuando se presionó y soltó. Los controles sobre los que aplica son: Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, marco, grid, etiqueta, caja de lista, control OLE, botón de opción, imagen, barras de desplazamiento y caja de texto.

MouseUp

Este evento aplica cuando se libera el botón principal del mouse, es decir, cuando se suelta el botón y éste se levanta. Los controles sobre los que aplica son: Form, check box, combo box, botón de comandos, caja de lista de directorios, caja de lista de archivos, marco, grid, etiqueta, caja de lista, control OLE, botón de opción, imagen, barras de desplazamiento y caja de texto.

2.8 Depuración y Ejecución

Conforme se escribe un programa en Visual Basic, se pueden tener errores en la sintaxis, esto es, se puede escribir algo como lo siguente:

Circle (Scalewidth/2, SlaceHeight/2)

Cuando se intenta pasar a la siguiente línea, Visual Basic pone en la pantalla una caja de advertencia indicando que se algo más se está esperando. En este caso se necesita indicar el radio del círculo (por lo menos). De esta manera Visual Basic capta errores a la hora del diseño. Por el otro lado se pueden terminar con errores de Run-Time que son imposibles de evitar en el momento de diseño, tales como fuera de memoria o erroes de disco lleno. Una vez que se aprende esto, es posible anticiparse a estos erroes, atraparlos y manejarlos.

Los tipos de errores que se discuten a continuación, errores lógicos en el programa, son más difíciles de encontrar. Un error puede estar enterrado en una larga cadena de sentencias complejas. Afortunadamente Visual Basic brinda algunas herramientas de Debugging para localizar y hasta corregir errores.

Utilización de Cajas de Texto en Debuging

Las cajas de texto pueden ser excelentes herramientas de debug, de hecho, son utilizadas para imprimir resultados inmediatos en los programas. Para utilizarlas simplemente hay que agregar unas cajas de texto extras a la aplicación e imprimir valores cruciales conforme va corriendo el programa. Por ejemplo, se desea ver una variable que cuenta los cada vez que se oprime una tecla, también se podría checar si se están leyendo las coordenas del ratón correctamente cada vez que se entra a un evento de MouseDown( ). Las cajas de texto temporales brindan una ventana de lo que está pasando atrás de la escena del programa. Esta es la escencia principal del debugging.

Para propósitos de Debugging se ordenarán alfabéticamente 10 nombres como los siguientes:

John, Tim, Edward, Samuel, Frank, Tood, George, Ralph, Leonard, Thomas

Se puede hacer el siguiente código que ordene los nombres

```
Sub Form_Click ( )
        Static Names(10) As String

        Names(1) = "John"
        Names(2) = "Tim"
        Names(3) = "Edward"
        Names(4) = "Samuel"
        Names(5) = "Frank"
        Names(6) = "Tood"
        Names(7) = "George"
        Names(8) = "Ralph"
        Names(9) = "Leonard"
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
                If Names( i ) > Names( j ) Then
                        Temp$ = Names( i )
                        Names( j ) = Names( j )
                        Names( j ) = Tmp$
                End If
```

```
                    Names( j ) = Tmp$
                End If
            Next j
Next i
```

Observe que está utilizando el operador lógico > para comparar las cadenas; esto es perfectamente legal en Visual Basic y le permite determinar el orden alfabético de estas cadenas, finalmente se imprime el resultado, nombre por nombre, de la siguiente manera:

```
Sub Form_Click ( )
        Static Names(10) As String

        Names(1) = "John"
        :
        :
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
            If Names( i ) > Names( j ) Then
                Temp$ = Names( i )
                Names( j ) = Names( j )
                Names( j ) = Tmp$
            End If
        Next j
Next i

For k = 1  to 10
        Print Names(k)
Next k
End Sub
```

Desafortunadamente el resultado de este programa al correrlo no es satisfactorio:

```
John


Tim


Todd
```

Este resultado se ve un poco incompleto, es tiempo de un debug. De hecho un debugging puede ser iniciado sin detener el programa. Para hacer esto, seleccione la opción de View

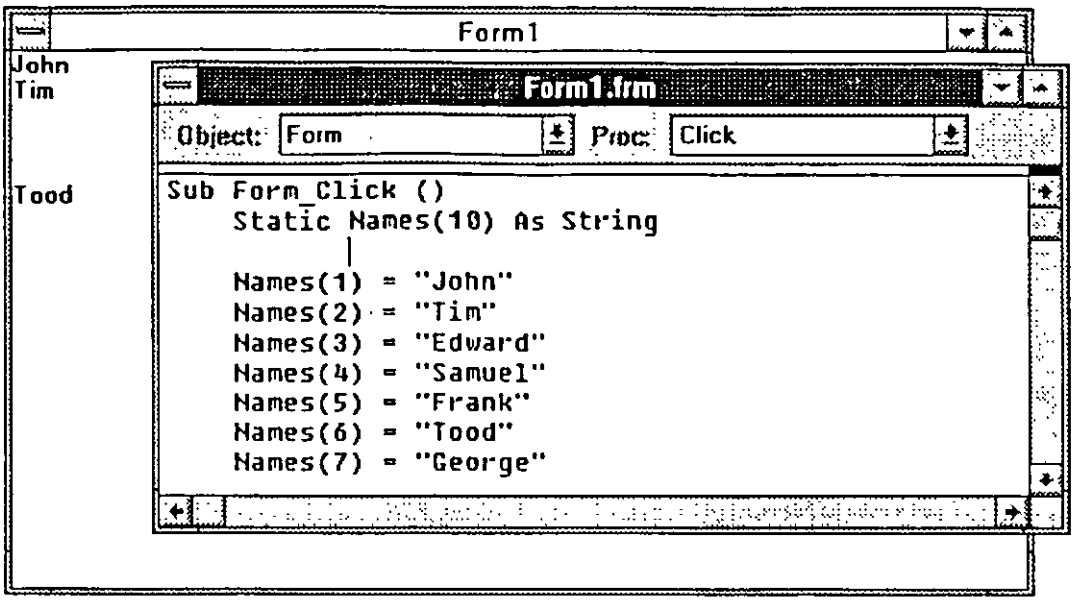


fig. 2.6

Visual Basic da la posibilidad de revisar el código mientras este está corriendo. Cuando se hace esto se pueden identificar errores inmediatamente con sólo leer el código. En particular cuando se cambian elementos dentro de un arreglo, estos se cargan temporalmente en una variable llamada Temp$; cuando se cargan nuevamente en el arreglo se utiliza una variable que está deletreada incorrectamente como Tmp$:

```
Sub Form_Click ( )
        Static Names(10) As String

        Names(1) = "John"
        :
        :
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
                If Names( i ) > Names( j ) Then
                        Temp$ = Names( i )
                        Names( j ) = Names( j )
                        Names( j ) = Tmp$
                End If
        Next j
Next i

For k = 1  to 10
        Print Names(k)
Next k
End Sub
```

```
        Print Names(k)
Next k
End Sub
```

Los errores de escritura en variables probablemente sea el error lógico más común de Visual Basic. Visual Basic no se queja de estos errores porque asume que se está declarando una nueva variable, tmp$, y la inicializa con valor "".

Este problema puede ser arreglado sin necesidad de terminar el programa. Solamente hay que abrir el menú de Run de Visual Basic, donde resaltan tres opciones: Break, End, y Restart. Seleccione Break para detener el programa temporalmente. Ahora puede cambiar código y continuar con el programa En la figura 2.7 se cambia Tmp$ por Temp$

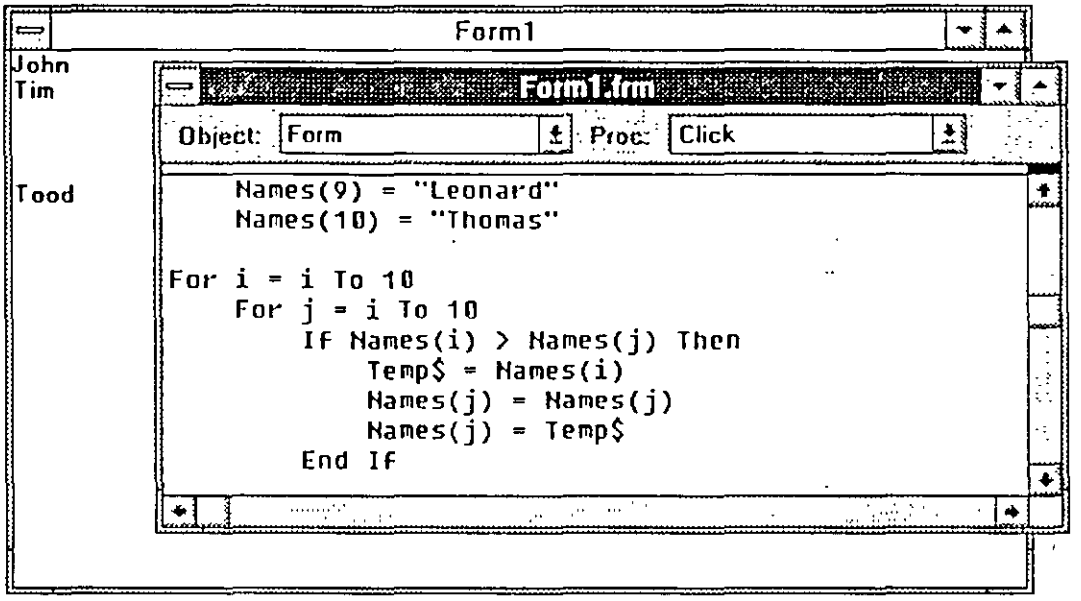


fig. 2.7

Ahora el programa aparece de la siguiente manera:

```
Sub Form_Click ( )
        Static Names(10) As String

        Names(1) = "John"
        :
        :
        Names(10) = "Thomas"

For i = i  to 10
        For j = i to 10
```

```
                If Names( i ) > Names( j ) Then
                        Temp$ = Names( i )
                        Names( j ) = Names( j )
                        Names( j ) = Temp$
                End If
        Next j
Next i

For k = 1  to 10
        Print Names(k)
Next k
End Sub
```

Para que continúe el programa corriendo se puede seleccionar la opción de Continue en el menú de Run. Después se puede dar un click a la forma para ver si hubo algún cambio, La lista aparece como la que se muestra a continuación:

```
John
Tim
Tim
Tim
Tim
Tood
Tood
Tood
Tood
Tood
```

A pesar de que hubo un cambio el resultado no es aún correcto. El problema obvio en el programa es que las entradas en el arreglo de Names( ) están siendo llenados incorrectamente. Para checar lo que está sucediendo se deben de observar los elementos en el arreglo conforme se van llenando. Esto se puede hacer estableciendo un Breakpoint. Esto detiene la ejecución en cierta parte del programa. Para establecer un Breakpoint se debe de ir a la línea donde se detendrá la ejecución y oprimir F9 o seleccionar Toggle Breakpoint en el menú de Debug. La sentencia seleccionada aparece en Bold para indicar que en ese lugar se estableció un breakpoint, como se muestra en la figura 2.8.

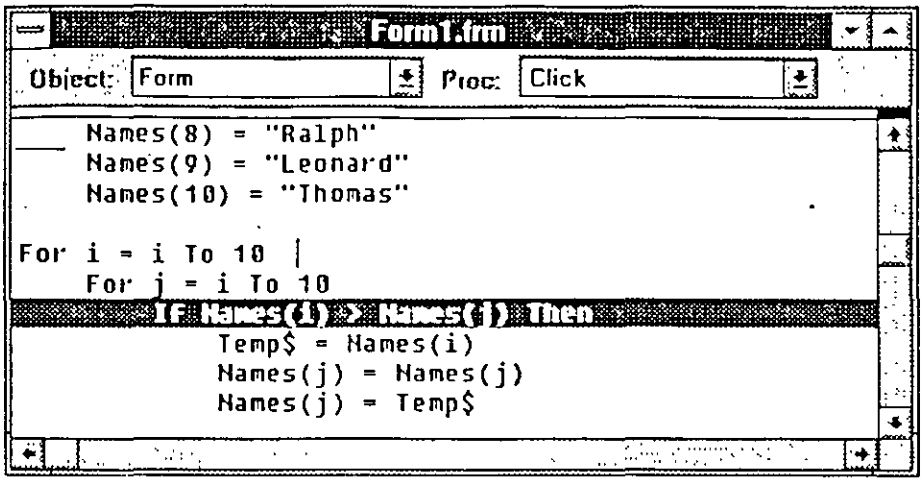


fig. 2.8

Después se corre el programa seleccionando Start en el menu de Run. La ejecución del programa continúa hasta que llega al breakpoint. Esta línea se encuentra con bordes como se muestra en la figura 2.9

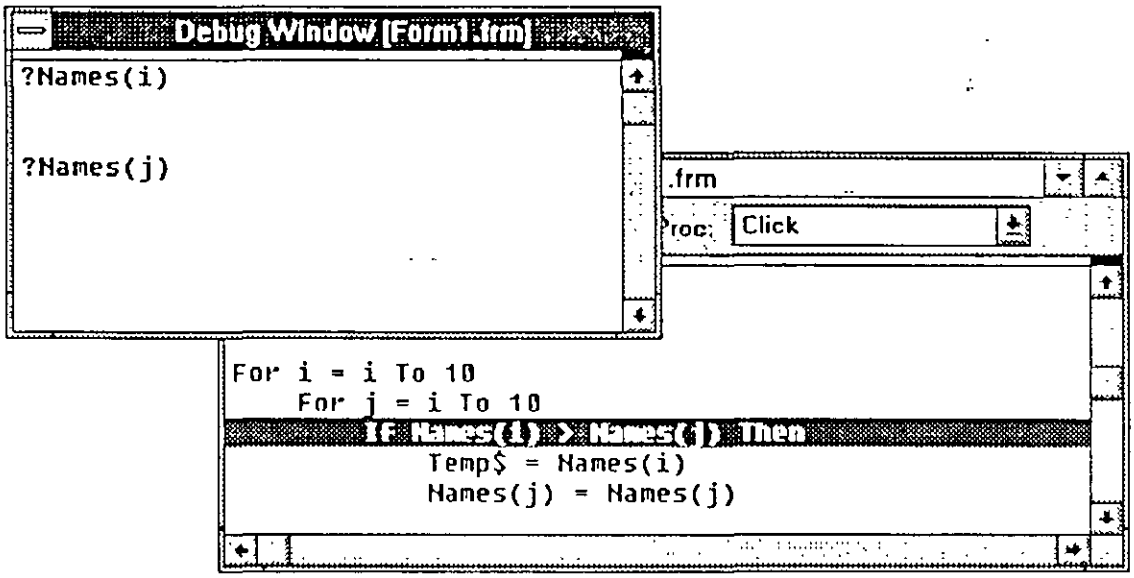


fig. 2.9

Los valores de Names(i) y Names (j) en la ventana de debug de Visual Basic. Esta permite checar los valores mientras se está en un estado interrumpido. Para checar los valores de las variables solamente se necesita poner el signo de interrogación antes de la variable que se desea conocer el valor como se muestra en la figura 2.9.

3 TIPOS DE DATOS EN VISUAL BASIC

3.1 Constantes y Variables

Las variables son utilizadas para almacenar valores mientras se ejecuta el código Visual Basic. Dentro de un procedimiento, las variables se declaran utilizando la sentencia **Dim** y el nombre de la variable :

Dim *nombre_variable* [**As** *tipo*]

Los nombres de las variables siguen las mismas reglas que para todos los objetos creados y a los cuales se les asigna un nombre. El nombre de la variable :

- Debe comenzar con una letra.
- Debe contener solo letras, números y el caracter de tilde abajo (_); los caracteres de puntuación y los espacios no se permiten.
- No debe exceder los 40 caracteres.
- No puede ser una palabra reservada.

La cláusula opcional **As** *tipo* en la sentencia **Dim** permite definir el tipo de dato de la variable que se está declarando. Si se omite, Visual Basic declarà la variable como el tipo de dato por default: **Variant**.

Declaración implícita

No se requiere declaran una variable antes de utilizarla. Por ejemplo, se puede tener una función como esta :

```
Function SafeSqr(num)
   TempVal = Abs(num)
   SafeSqr = Sqr(TempVal)
End Function
```

Aquí, no se requirió declaran la variable TempVal antes de utilizarla en la función. Visual Basic crea automáticamente una variable con ese nombre, así que puede ser usada como si se hubiera declarado de forma explícita.

Aunque es conveniente, la declaración implícita puede ocasionar errores como este:

```
Function SafeSqr(num)
   TempVal = Abs(num)
   SafeSqr = Sqr(TemVal)
End Function
```

En apariencia, se ve igual a la función anterior, sin embargo presenta un error en la variable TemVal que no será detectado por el compilador. Esta función regresará siempre el valor de cero ya que Visual Basic declara ambas variables de forma implícita y sólo a una de ellas se le está asignando un valor. Cuando Visual Basic encuentra un nuevo nombre no sabe si el programador desea que se genere otra variable o si se equivocó al escribir y se trata de una variable ya declarada.

Declaración Explícita

Para eliminar el problema de renombrar variables, se puede estipular que Visual Basic siempre genere un mensaje de error siempre que encuentre un nombre que previamente no ha sido declarado explícitamente como una variable. Para hacer esto, se debe colocar esta sentencia en la sección Declarations de un módulo o de una forma :

```
Option Explicit
```

Debido a que la sentencia Option Explicit ayuda a encontrar esta clase de errores, quizá sea conveniente utilizarla en todo el código. Estableciendo una opción de medio ambiente, se puede hacer que Visual Basic añada la sentencia Option Explicit en cada forma o módulo que se cree.

Para insertar automáticamente la sentencia Option Explicit en todas las nuevas formas y módulos, se deben seguir las siguientes instrucciones:

1. Del menú Options seleccionar Environment.

2. En el cuadro de diálogo de Environment, seleccionar la opción Require Variable Declaration.

3. En la caja Settings teclear yes.

Alcance y tiempo de vida de las variables

Cuando se declara una variable dentro de un procedimiento, sólo el código dentro de ese procedimiento puede accesar o cambiar el valor de la variable; ésta tiene un alcance (scope) que es local para ese procedimiento. En algunas ocasiones, sin embargo, se requiere usar una variable con un alcance más amplio, como por ejemplo, una variable cuyo valor está disponible para todos los procedimientos dentro de una misma forma, o para todos los procedimientos en la aplicación. Visual Basic permite especificar el alcance de una variable cuando se declara.

Dependiendo de como se declara, una variable puede tener tres diferentes alcances:

Scope	Declaración
Local	Dim, Static, o ReDim (dentro de un procedimiento)
Module	Dim (en la sección Declarations de una forma o un módulo)
Global	Global (en la sección Declarations de un módulo)

Variables locales

Una variable local es reconocida sólo en el procedimiento en el cual aparece. Son útiles para realizar cualquier tipo de cálculo temporal. Una docena de procedimientos pueden tener una variable llamada Temp, pero como cada una es local, cada procedimiento tiene su propia variable. Un procedimiento puede alterar su variable local Temp sin alterar las variables Temp en otros procedimientos.

Un procedimiento puede también llamarse así mismo (una técnica llamada recursión), y cada invocación del procedimiento obtiene su propia copia de las variables locales en ese procedimiento. Las variables locales son declaradas dentro de los procedimientos con la sentencia **Dim**. Las variables locales declaradas con **Dim** sólo existirán mientras el procedimiento se ejecute. Las variables locales declaradas con **Static**, existirán mientras la aplicación esté corriendo.

Variables a nivel módulo

Las variables a nivel módulo comparten información con todos los procedimientos de una forma o módulo. Estas variables se encuentran disponibles para todos los procedimientos en ese módulo, pero no para el código en otra forma o módulo. Las variables a nivel módulo se crean declarándolas con la sentencia **Dim** en la sección Declarations de un módulo. Las variables a nivel módulo existirán mientras la aplicación esté corriendo.

Variables globales

Una variable global tiene el alcance más amplio de todas. Los valores en las variables globales están disponibles para cualquier procedimiento en cualquier forma o módulo de la aplicación. Como las variables a nivel módulo, las variables globales son declaradas en la sección Declarations de un módulo. Sin embargo, se pueden declarar variables globales con la sentencia **Global** en lugar de la sentencia **Dim**.

Se pueden declarar las variables globales en cualquier módulo. Para propósitos de organizar el código, sin embargo, es buena idea mantener todas las variables globales en el menor número posible de módulos, de tal forma que sean fáciles de encontrar. Esto también ayuda a eliminar la declaración de la misma variable global en dos módulos diferentes, que a veces causa un error cuando se trata de compilar el código.

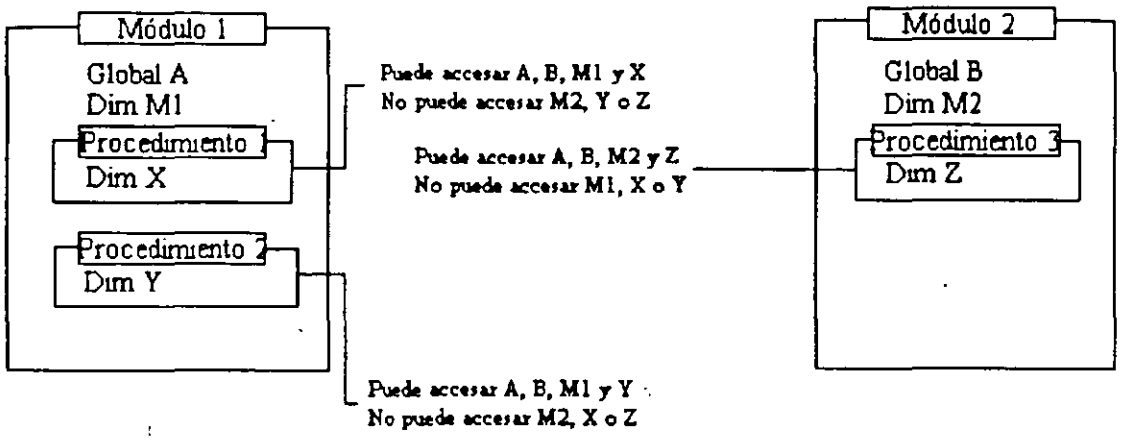


Figura 3.1

Variables estáticas

Además del alcance, las variables también tienen un tiempo de vida. El valor en una variable a nivel módulo y global es preservado mientras la aplicación se esté ejecutando. Sin embargo, las variables locales declaradas con Dim sólo existen mientras el procedimiento en el que fueron declaradas se encuentran ejecutándose. Normalmente, cuando un procedimiento termina su ejecución, el valor de las variables locales no es preservado y la memoria utilizada por estas variables es liberada. La siguiente vez que el procedimiento se ejecuta, todas sus variables locales se reinicializan.

Se puede hacer que Visual Basic preserve el valor de una variable local haciendo de ésta una variables estática. Se utiliza la sentencia **Static** para declarar una o más variables dentro de una procedimiento, exactamente como se hace con la sentencia **Dim**:

```
Static Depth
```

Para hacer que todas las variables locales en un procedimiento sean estáticas, se coloca la palabra **Static** al inicio del encabezado del procedimiento. Por ejemplo:

```
Static Function RunningTotal (num)
```

Esto hace que todas las variables locales en este procedimiento sean estáticas, sin importar si fueron declaradas con **Static** o **Dim** o declaradas implícitamente. Se puede colocar **Static** antes de cualquier encabezado de procedimientos **Sub** o **Function**, incluyendo *event procedures* y aquellos que también han sido declarados como **Privated**.

3.2 Tipos Fundamentales de Datos

Cuando se declara una variable, también se tiene que indicar un tipo de dato para ella. Todas las variables tienen un tipo de dato que determina que tipo de datos se pueden almacenar. Por default, si no se le indica un tipo de dato, la variable toma el tipo de dato **Variant**.

El tipo de dato Variant

El tipo de dato **Variant** puede almacenar muchos tipos de datos. Como un control Text Box en una forma, una variable **Variant** es igualmente capaz de almacenar números, cadenas de texto, o fechas y horas. No se requiere hacer conversiones entre estos tipos de datos cuando se asignan a una variable **Variant**; Visual Basic automáticamente realiza cualquier conversión necesaria. Por ejemplo:

```
Dim SomeValue
SomeValue = "17"
SomeValue = SomeValue - 15
' SomeValue ahora contiene el valor numérico 2
SomeValue = "U" & SomeValue
' SomeValue ahora contiene la cadena "U2"
```

El valor Null

El tipo de dato **Variant** puede contener otro valor especial: **Null**. **Null** es comúnmente utilizado en aplicaciones de bases de datos para indicar datos desconocidos o incompletos. Debido a que **Null** es utilizado en bases de datos, presenta algunas características únicas:

Las expresiones que involucran **Null** siempre resultan **Null**. Esto es, el valor **Null** se "propaga" a través de las expresiones; si cualquier parte de la expresión se valúa como **Null**, la expresión entera se valúa como **Null**.

Pasar un **Null**, una variable **Variant** conteniendo **Null**, o una expresión que se evalúa en **Null** como argumento de la mayoría de las funciones ocasiona que la función regrese **Null**.

Los valores **Null** se propagan a través de las funciones intrínsecas que regresan tipos de datos **Variant**.

Otros tipos fundamentales de datos

El tipo de dato **Variant** maneja todos los tipos fundamentales de datos y realiza las conversiones entre ellos de manera automática. Si se quiere crear código conciso y rápido, sin embargo, se deben utilizar otros tipos de datos más apropiados. Por ejemplo, si una variable contendrá siempre valores enteros pequeños, se pueden utilizar pocos bytes declarando la variable como **Integer** en lugar de **Variant**.

Existen siete tipos fundamentales de datos en Visual Basic, incluyendo **Variant**, como se muestra en la siguiente tabla:

Tipo de Dato	Descripción	Caracter de Declaración	Rango
Integer	entero de 2 bytes	%	-32,768 a 32,767
Long	entero de 4 bytes	&	-2,147,483,648 a 2,147,483,647
Single	número de punto flotante de 4 bytes	!	-3.402823E38 a -1.401298E-45 (valores negativos) 1.401298E-45 a 3.402823E38 (valores positivos)
Double	número de punto flotante de 8 bytes	#	-1.79769313486232D308 a -4.94065645841247D-324 (valores negativos) 4.94065645841247D-324 a 1.79769313486232D308 (valores positivos)
Currency	número con punto decimal fijo de 8 bytes	@	-922337203685477.5808 a 922337203685477.5807
String	cadena de caracteres	$	0 a aprox. 65,500 caracteres
Variant	Fecha/hora, punto flotante, o string	(ninguno)	valores de fecha: enero 1, 0000 a diciembre 31, 9999; valores numéricos: el mismo rango que Double; valores para string: el mismo rango que String

Tipos Numéricos

Si se sabe que se una variable siempre almacenará números enteros (como 12), ésta se puede declarar como **Integer** o **Long**. Las operaciones son más rápidas con los enteros, y consumen menos memoria que el tipo **Variant**. Son especialmente útiles como contadores en los ciclos **For**.

Si la variable contiene una fracción, se debe declarar como una variable **Single**, **Double**, o **Currency**. El tipo de dato **Currency** soporta hasta cuatro dígitos en la parte derecha del punto decimal y hasta 15 a la izquierda; se trata de un tipo de dato rápido y preciso para cálculos monetarios. Los números de punto flotante (**Single** o **Double**) tienen rangos mucho mayores que **Currency**, pero los errores de redondeo son menores.

Tipos de datos para String

Si se tienen una variable que siempre contendrá una cadena de caracteres y nunca un valor numérico, se puede declarar como tipo String:

```
Dim S As String
```

Se puede asignar cadenas de caracteres a esta variable y manipularlas utilizando funciones de cadena como :

```
S="Database"
S=Left(S,4)
```

Por default, una variable string o argumento es de longitud variable; la cadena puede crecer o reducirse de acuerdo con los datos que se le asignan. Sin embargo, se pueden declarar cadenas con una longitud fija. Para especificar una longitud fija en una variable **String** se sigue la sintaxis :

```
String * tamaño
```

Por ejemplo :

```
Dim EmpName As String * 50
```

Si se asigna una cadena menor a 50 caracteres a esta variable, Visual Basic añadirá los espacios necesarios para acompletar los 50 lugares. Si se le asigna una cadena más larga, se trunca dicha cadena hasta los 50 caracteres y los caracteres excedentes se pierden.

3.3 Arreglos

Los arreglos permiten hacer referencia a una serie de variables bajo el mismo nombre y utilizando un índice. Esto ayuda a crear código más pequeño y simple en muchas situaciones. Los arreglos tienen un límite superior y otro inferior, y los elementos de estos arreglos tienen continuidad dentro de estos limites. Debido a que Visual Basic reserva el espacio en memoria para cada índice del arreglo, se tiene que procurar no hacer los arreglos más extensos de lo necesario.

Todos los elementos en el arreglo son del mismo tipo. Por supuesto, cuando el tipo de dato es **Variant**, cada elemento del arreglo puede contener diferentes tipos de datos. Un arreglo puede ser declarado como cualquiera de los tipos fundamentales de datos, incluyendo los tipos definidos por el usuario y variables objetivas.

Existen tres formas de declarar un arreglo de dimensión fija, dependiendo del alcance que se desea para el arreglo :

- Creando un arreglo global, utilizando la sentencia **Global** en la sección Declarations de un módulo.
- Creando un arreglo a nivel de módulo, utilizando la sentencia **Dim** en la sección Declarations de un módulo o forma.
- Creando un arreglo local, utilizando la sentencia **Static** en un procedimiento.

Existen algunas reglas adicionales cuando se trata de arreglos dinámicos (un arreglo cuyo tamaño puede variar durante la ejecución).

Cuando se declara un arreglo, entre paréntesis y después del nombre, debe ir el límite superior. Por ejemplo, esta declaración de arreglos puede ir en la sección Declarations de un módulo:

```
Dim Counters(14) As Integer
Dim Sums(20) As Double
```

Para crear un arreglo global, simplemente se utiliza **Global** en lugar de **Dim**:

```
Global Counters(14) As Integer
Global Sums(20) As Double
```

La misma declaración dentro de un procedimiento se hace con **Static**:

```
Static Counters(14) As Integer
Static Sums(20) As Double
```

El primer arreglo consta de 15 posiciones que van de la 0 a la 14, y el segundo arreglo consta de 21 posiciones de la 0 a la 20. Por default el limite inferior es 0 pero se puede especificar cualquier limite.

Para especificar otro limite explicitamente, se utiliza una sintaxis como la siguiente:

```
Dim Counters (1 To 15) As Integer
Dim Sums(100 To 120) As Double
```

En la anterior declaración, los índices del primer arreglo van de 1 a 15 y del segundo de 100 a 120.

Arreglos multidimensionales

Con Visual Basic, se pueden declarar arreglos hasta de 60 dimensiones. Por ejemplo, la siguiente sentencia declara un arreglo bidimensional de 10 por 10 dentro de un procedimiento :

```
Static MatrixA(9, 9) As Double
```

Aún con dos dimensiones, se pueden expresar los límites del arreglo de forma explícita :

```
Static MatrixA(1 To 10, 1 To 10) As Double
```

Arreglos dinámicos

Algunas veces no se conoce exactamente que tan grande será un arreglo. En esos casos se quisiera tener la capacidad de cambiar el tamaño del arreglo cuantas veces sea necesario.

Un arreglo dinámico puede ser redimensionado en cualquier momento. Los arreglos dinámicos constituyen una de las ventajas más flexibles de Visual Basic, y ayudan a administrar mejor la memoria. Por ejemplo, se puede utilizar un arreglo grande por un tiempo corto y liberar memoria al sistema reduciendo el tamaño del arreglo cuando ya no se requiera tan grande.

Una alternativa es declarar un arreglo de la mayor dimensión posible e ignorar aquellas posiciones que no se necesitan. Sin embargo, este método, muy utilizado, puede ocasionar que el ambiente operativo tenga poca memoria para trabajar.

Para crear un arreglo dinámico :

1. Declarar el arreglo con la sentencia **Global** (si se desea un arreglo de alcance global) o con la sentencia **Dim** a nivel de módulo (si se quiere que el arreglo se encuentre a nivel módulo), o una sentencia **Static** o **Dim** en un procedimiento (si se quiere que el arreglo sea local). El arreglo dinámico se declara dejando los paréntesis vacíos. Por ejemplo:

   ```
   Dim DynArray()
   ```

2. Redimensionar el arreglo con el número de elementos deseado utilizando la sentencia ReDim. Por ejemplo :

   ```
   ReDim DynArray(x+1)
   ```

La sentencia **ReDim** puede aparecer sólo en un procedimiento. A diferencia de las sentencias **Dim** y **Static**, **ReDim** es una sentencia ejecutable - hace que la aplicación ejecute una acción mientras está corriendo.

La sentencia **ReDim** soporta la misma sintaxis descrita anteriormente para los arreglos fijos. Cada **ReDim** puede cambiar el número de elementos en un arreglo así como los límites superiores e inferiores de cada dimensión. Sin embargo, el número de dimensiones en el arreglo no puede cambiarse.

Por ejemplo, el arreglo dinámico Matrix1 es creado declarándolo a nivel módulo :

```
Dim Matrix1() As Integer
```

Un procedimiento reserva entonces el espacio para el arreglo:

```
Sub CalcValuesNow ( )
   .
   .
   .
   ReDim Matrix1( 19, 29)
End Sub
```

La sentencia **ReDim** genera un arreglo de 20 por 30 enteros (un total de 600 elementos). Alternativamente, los límites del arreglo dinámico pueden establecerse utilizando variables:

```
ReDim Matrix1( X, Y)
```

3.4 Conversiones

Visual Basic incluye una serie de funciones que permiten convertir un tipo de dato en otro distinto. Por ejemplo, una cadena a un número. También pueden realizarse conversiones entre tipos semejantes como un número de punto flotante a un entero.

A continuación se presenta una tabla con las posibles conversiones y algunos ejemplos de como se utilizan dichas funciones.

Nota: Si se desea una revisión completa de la sintaxis de cada función, se puede buscar dentro de la ayuda en linea en la sección de funciones.

Acción	Funciones / Sentencias
Valor ANSI a string	Chr, Chr$
Fecha a un número serial	DateSerial, DateValue
Números decimales a otros	Hex, Hex$, Oct, Oct$
Números a string	Format, Format$,Str, Str$
Un tipo numérico a otro	CCur, CDbl, CInt, CLng, CSng, CStr, CVar, CVDate, Fix, Int
Numero serial a fecha	Day, Month, Weekday, Year
Número serial a hora	Hour, Minute, Second
Una cadena a un valor ASCII	Asc
Cadena a número	Val
Tiempo a número serial	TimeSerial, TimeValue

Función Val

La función **Val** regresa el valor numérico de una cadena de caracteres. La sintaxis de la función **Val** es :

Val (*cadena de caracteres*)

El argumento cadena de caracteres es una secuencia de caracteres que puede ser interpretada como un valor numérico. La función **Val** detiene la lectura de la cadena al encontrar el primer caracter que no reconoce como parte de un número. La función **Val** también cuenta los espacios en blanco, tabuladores y cambios de línea dentro del argumento. Por ejemplo, la siguiente línea regresa el valor de 1615198 :

Val (" 1615 198th Street N.E.")

Los símbolos que a menudo son reconocidos como parte de un valor numérico, tales como el signo de pesos y las comas, no son reconocidos por la función **Val** como números. **Val** reconoce los prefijos &O y &H para notaciones octal y hexadecimal respectivamente.

La función **Val** siempre devuelve un valor **Double**. Si el resultado de la variable **Val** es asignado a una variable, el valor **Double** regresado por la función **Val** forza al tipo de dato de la variable.

Por ejemplo, se puede utilizar la función **Val** para determinar si el primer caracter de una cadena es un número:

```
Sub Form_Click ( )
   Dim Msg
   Dim Number As Double
   Number = Val(InputBox$("Ingrese un número"))
   Msg= "El número que ingreso es :" & Number
   MsgBox Msg
End Sub
```

Función Str y Str$

La función Str regresa la representación en forma de cadena de caracteres correspondiente a un valor numérico. Se utiliza para convertir un simple valor numérico en una cadena.

Cuando los números se convierten a texto, se reserva siempre un espacio para el signo del número. Si el número es positivo, la cadena que regresa **Str[$]** contiene un espacio al inicio y el signo más está implícito.

Su sintaxis es :

Str[**$**] (*número*)

El siguiente ejemplo utiliza **Str** para regresar la representación en forma de cadena del valor contenido en dos variables. Hay que notar que debido a que los números son positivos, un espacio para el signo más implícito, precede a la primer cadena de caracteres.

```
Sub Form_Click ( )
   Dim Dice1, Dice2, Msg
   Randomize                  ' Inicializa el generador de números aleatorios.
   Dice1 = Int (6 * Rnd +1)   ' Genera el primer valor.
   Dice2 = Int (6 * Rnd +1)   ' Genera el segundo valor.
   Msg = "You rolled a" & Str(Dice1)
   Msg = Msg + "and a" & Str(Dice2)
   Msg = Msg & "for a total of"
   Msg = Msg & Str(Dice1 + Dice2) & "."
   MsgBox Msg
End Sub
```

Funciones CCur, CDbl, CInt, CLng, CSng, CStr y CVar

Estas funciones realizan una conversión explícita de un tipo de dato a otro. Su sintaxis es la siguiente :

CCur(*expresión*)
CDbl(expresión)
CInt(*expresión*)
CLng(*expresión*)
CSng(*expresión*)
CStr(*expresión*)
CVar(*expresión*)

El argumento expresión puede ser cualquier expresión de tipo caracter o numérica válida.

Estas funciones convierten cualquier expresión válida sin importar el tipo de que se trate, a el tipo especificado en la siguiente tabla :

Función	Convierte a
CCur	Currency
CDbl	Double
CInt	Integer
CLong	Long
CSng	Single
CStr	String
CVar	Variant

Las funciones de conversión numérica **CCur**, **CDbl**, **CInt**, **CLong**, y **CSng** controlan explícitamente el tipo de dato de una expresión numérica. Por ejemplo, se puede usar **CCur** para forzar una operación con tipo **Currency** en casos en que los enteros, los números de doble precisión o los simples pueden ocurrir.

Ejemplo

CCur convierte un valor calculado como doble al tipo **Currency** con cuatro decimales de precisión, desplegando el resultado en formato de moneda convencional con dos dígitos decimales.

```
Sub Form_Click ( )
   Dim Amount, Msg, Percent, SalesTax
   Percent = 8.2
   Msg = "¿Cual es el monto para el cálculo de intereses?"
   Amount = InputBox(Msg)                          ' Entrada del usuario
   SalesTax = CCur(Amount * (Percent * 0.1))       ' Cálculo del impuesto
   Msg = "El impuesto de " & Format(Amount, "Currency")
   Msg = Msg & " is " & Format(SalesTax, "Currency")
   MsgBox  Msg
End Sub
```

4. CREACION Y USO DE CONTROLES

4.1 Controles de Visual Basic

Existen dos tipos de objetos en Visual Basic: Las formas y los controles. Los controles son todos aquellos objetos gráficos que se diseñan y se colocan en una forma, tales como *list boxes*, *buttons*, *labels* y *timers*. En otras palabras, un control es cualquier objeto que el usuario puede manipular y que no es una ventana. Las ventanas que se diseñan y crean son formas; un control es utilizado para entradas/salidas con el usuario (cajas y botones son un ejemplo). Juntos, formas y controles son llamados objetos en Visual Basic, debido a que ambos son tratados como objetos gráficos. Los objetos tienen propiedades asociadas con ellos, que es lo mismo que decir que un objeto tiene datos asociados a él.

Si se está familiarizado con la programación orientada a eventos, entonces se debe saber que los objetos no sólo tienen asociados datos, sino también procedimientos que pueden ser utilizados, por ejemplo para mover un botón en la ventana. En C++, estos procedimientos conectados a los objetos se llaman funciones miembro, en Pascal y Visual Basic se les llama métodos.

La Caja de Herramientas

Aquí en Visual Basic, se tienen que dibujar los controles que se desean dentro de la forma. La forma de realizar esto es muy similar al proceso de dibujo en un programa como PaintBrush, en donde se selecciona la herramienta para dibujar y se realiza un trazo con ella. En el caso de Visual Basic, se selecciona una herramienta de un control en particular y se procede a dibujar en control sobre la forma. Para seleccionar la herramienta deseada solo se tiene que hacer click en uno de los botones de la caja de herramientas. Ver siguiente figura.

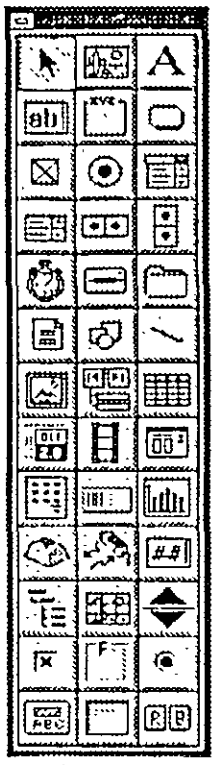


Figura 4.1

La caja de herramientas juega un papel muy importante en el diseño de la aplicación, ya que ésta permitirá añadir a las formas todos los controles que sean necesarios.

En los siguientes apartados se verá con más detalle algunos de los controles más importantes, utilizados en la mayoría de las aplicaciones para ambiente Windows.

4.2 Botones

El control *Command Button* permite tanto al usuario como al programador, darle instrucciones a la aplicación para que ésta siga con su ejecución o se detenga. Los botones son, tal vez, los controles más utilizados. Los botones para Aceptar, Cancelar, Salir, son un ingrediente que no puede faltar en una aplicación gráfica. Un botón puede servir también como un interruptor con el que se activa cierto proceso, por ejemplo, se puede tener un procedimiento para impresión de facturas, pero este no comenzará hasta que el usuario haya tecleado los datos correspondientes y al final "oprima el botón de Imprimir".

Uno de los eventos más utilizados del *Command Button* es el de Click. Para ver más claramente como funciona el *Command Button* se puede desarrollar el siguiente ejemplo.

Primero se tiene que generar una nueva forma (en caso de que Visual Basic ya este abierto) o trabajar sobre la forma por default que aparece al iniciar Visual Basic.

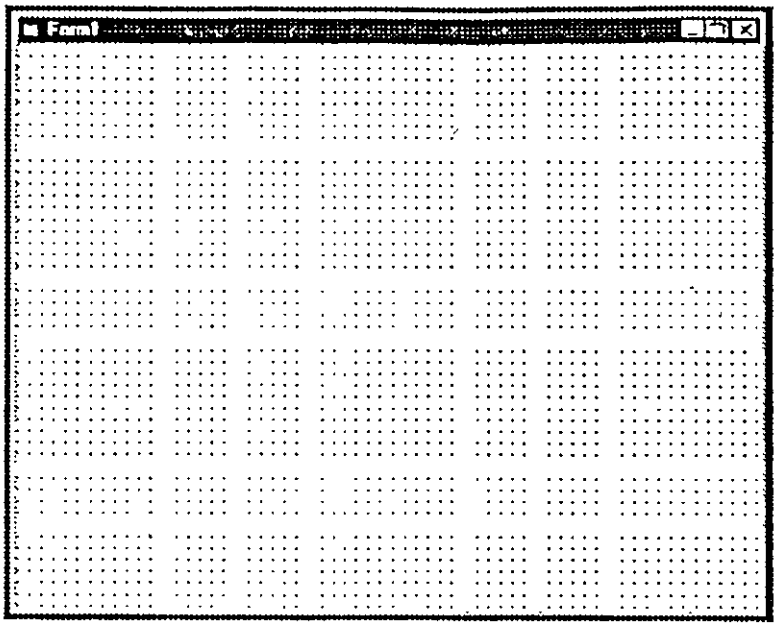


Figura 4.2

Posteriormente se cambiará el título de la ventana :

1. Seleccionar la forma (en este caso Form1) y oprimir la tecla F4 para llamar a la ventana Properties.

2. En la ventana Properties buscar la propiedad Caption.

3. Seleccionar la leyenda que contiene dicha propiedad y escribir el nuevo texto (p.e. Mi Primer Ventana)

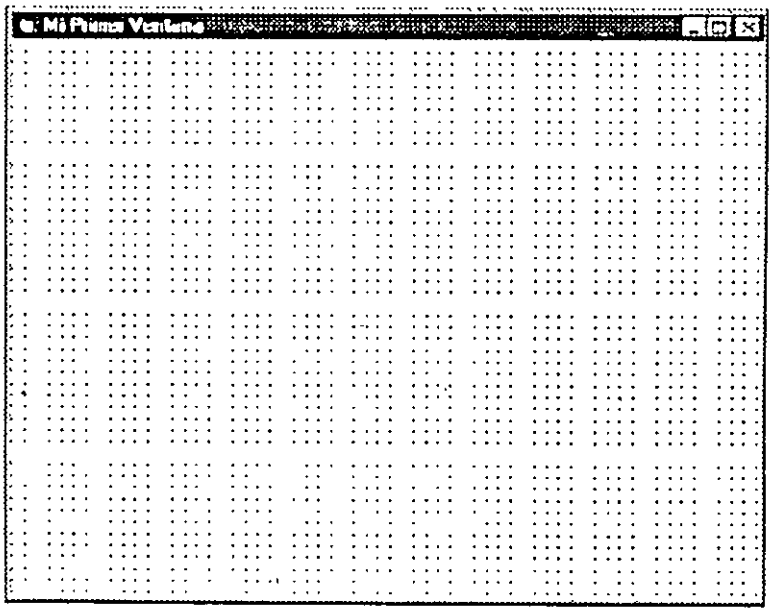


Figura 4.3

Ahora, se tiene que colocar un control *Command Button* :

1. Seleccionar la forma en donde se desea colocar el botón (haciendo click sobre ella).

2. Hacer doble click sobre la herramienta *Command Button* de la caja de herramientas (tercera de arriba a abajo en la segunda columna).
   Aparecerá un botón sobre la forma con la leyenda "Command1".

3. Colocar el botón en el lugar indicado dentro de la forma : Hacer click sobre el control y sin soltar el botón del mouse se arrastra hasta la posición deseada y después liberar el botón.

Para personalizar el botón que se acaba de colocar, se debe recurrir nuevamente a la ventana de propiedades. Por ejemplo, se puede cambiar la leyenda que aparece en él de la siguiente forma:

1. Seleccionar el control.

2. Llamar a la ventana Properties con la tecla F4.

3. Buscar la propiedad Caption y editarla poniendo la leyenda deseada (p.e. Mi Botón)

Hasta ahora, lo único que se ha hecho es dibujar el control y cambiar su leyenda. Anteriormente se mencionó que los paso para desarrollar una aplicación para Windows eran, primero dibujar la interface, en segundo lugar, personalizar los elementos gráficos; y por último, escribir los procedimientos y rutinas asociadas con cada uno de los elementos en dicha interface.

Para escribir una rutina asociada con un control sólo se debe llamar a la ventana de código. (revisar el capítulo 2 "Fundamentos de Programación"). Por lo pronto, realizaremos una pequeña rutina que esté asociada con el botón que se acaba de dibujar.

Para llamar a la ventana de código para escribir un procedimiento asociado con un control, basta con hacer doble click sobre dicho control.

El procedimiento que aparece a continuación es un *event procedure* (los cuales se vierón también en el capítulo 2) que cambia el color del fondo de la forma al momento que se hace click sobre el botón.

```
Sub Command1_Click ( )
   Form1.Caption = "Al hacer click, ha cambiado el caption"
   End
End Sub
```

Para introducir este procedimiento solo se tiene que hacer doble click sobre el botón que está en la forma. Al hacerlo, aparecerá una ventana de código con la plantilla :

```
Sub Command1_Click
End Sub
```

De esta forma Visual Basic hace más fácil la creación de dicho procedimiento.

Para ver el resultado de este ejercicio, se puede ejecutar el programa utilizando el botón de la barra de herramientas :

Figura 4.4

Al hacer click sobre el botón, el caption de la ventana cambiará de forma similar a la siguiente figura :

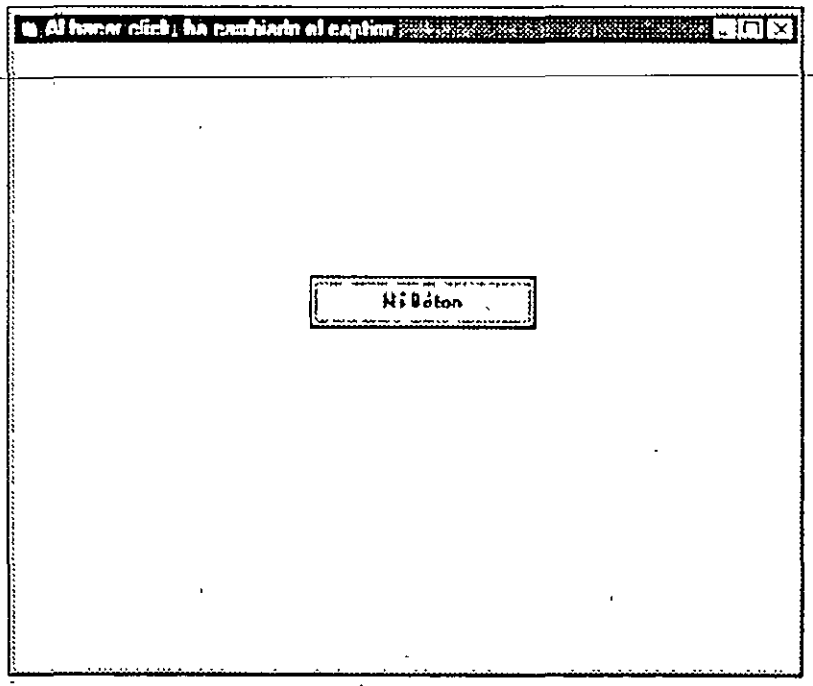


Figura 4.5

Como no se tiene ningún otro procedimiento, el programa se detendrá automáticamente.

A continuación se presentan todas las propiedades, eventos y métodos para el control *Command Button*:

Propiedades

BackColor	FontBold	HelpContextID	TabIndex
Cancel	FontItalic	hWnd	TabStop
Caption	FontName	Index	Tag
Default	FontSize	Left	Top
DragIcon	FontStrikethru	MousePointer	Value
DragMode	FontUnderline	Name	Visible
Enabled	Height	Parent	Widht

Eventos

Click	GotFocus	KeyUp	MouseMove
DragDrop	KeyDown	LostFocus	MouseUp
DragOver	KeyPress	MouseDown	

Métodos

Drag	Refresh	Move	SetFocus
ZOrder			

4.3 Despliegue y Captura de Texto

Las etiquetas y las cajas de texto son utilizadas para desplegar texto en una aplicación Visual Basic. Las etiquetas contiene texto que es sólo de lectura, mientras que el texto contenido en una caja de texto es editable.

Etiquetas

Un control etiqueta (*Label*) despliega texto que el usuario no puede cambiar directamente. Las etiquetas también pueden ser utilizadas para identificar controles, como cajas de texto y barras de enrrollamiento (*scroll bars*), que no tienen la propiedad Caption.

Por default, el *caption* es la única parte visible del control etiqueta. Si se establece la propiedad BorderStyle en 1, sin embargo, la etiqueta aparecerá con un borde - lo que hará que se vea similar a una caja de texto.

Para ajustar el tamaño de una etiqueta de tal forma que el texto que se encuentra en ella no tenga problemas para verse, es necesario modificar los bordes en la etapa de diseño. Sin embargo, ¿que sucede si el texto de la etiqueta cambiará durante la ejecución?. Las etiquetas cuentan con dos propiedades que ayudan a resolver este problema, se trata de las propiedades AutoSize y WordWrap.

La propiedad AutoSize determina si un control debe ser ajustado automáticamente para albergar el texto. Si se establece como **True**, la etiqueta crecerá horizontalmente hasta dar cabida a todo el texto como se ve en la siguiente figura.

Figura 4.6

La propiedad WordWrap ocasiona que la etiqueta crezca verticalmente para albergar el texto, mientras mantiene la misma anchura, como se ve en la figura siguiente.

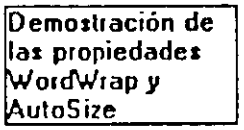


Figura 4.7

Cuando las etiquetas se utilizan como *caption* para algún otro control, éstas pueden también funcionar como llave de acceso a dichos controles. Una llave de acceso no es más que una letra subrayada que permite seleccionar el control (obtener el foco para el control) sin necesidad de utilizar el mouse. Es algo muy similar a las letras subrayadas que tienen las opciones de los menús en cualquier aplicación para Windows.

Para asignar una llave de acceso a un control utilizando una etiqueta, se puede realizar lo siguiente :

1. Dibujar primero la etiqueta; después el control.

2. Utilizar un ampersand (&) antes de la letra que se desea como llave, en la propiedad Caption de la etiqueta.

Cajas de texto

Las cajas de texto son uno de los controles más versátiles con que cuenta Visual Basic y que pueden ser utilizadas para recibir la entrada de un usuario o para desplegar un texto. Debido a que permiten que se edite el texto que en ellas se encuentra, las cajas de texto no deben utilizarse si no se desea que el usuario modifique la información.

La apariencia de un texto es fuertemente influenciada por dos propiedades, MultiLine y ScrollBars, las cuales solo se pueden establecer durante el diseño.

Nota: La propiedad ScrollBars no debe confundirse con los controles de *scroll bars* puesto que son dos cosas muy distintas.

Si se establece la propiedad de MultiLine como **True**, permitirá que la caja de texto despliegue varias líneas durante la ejecución. Una caja de texto multilíneas automáticamente manipula y ajusta el texto de forma horizontal. La propiedad ScrollBars por default se encuentra en 0-None. La forma automática de cambio de línea le evita al usuario tener que insertar un fin de línea al final de cada renglón. Cuando el texto es más largo que lo que puede ser desplegado en una línea, la caja de texto envía el texto a la siguiente línea.

Los finales de línea no se pueden ingresar durante el diseño en las propiedades. Dentro de un procedimiento, se puede crear un fin de línea utilizando un caracter de nueva línea - un retorno de carro seguido de un avance de línea (los caracteres ANSI 13 y 10). Por ejemplo, el siguiente procedimiento coloca dos líneas de texto en una caja de texto multilíneas (Text1) cuando la forma es cargada:

```
Sub Form_Load ( )
   NL = Chr(13) + Chr(10)
   Text1.Text = "Aquí hay dos líneas" & NL & "en una caja de texto"
End Sub
```

Una caja de "Password" es una caja de texto que permite al usuario ingresar caracteres de su clave mientras despliega caracteres, como asteriscos, para ocultar la clave. Visual Basic utiliza dos propiedades para cajas de texto, PasswordChar y MaxLenght, las cuales hacen que la creación de una caja para Password sea muy sencilla.

La propiedad PasswordChar permite especificar qué caracter reemplazará a los caracteres que el usuario teclea. Por ejemplo, si se quiere que aparezcan asteriscos en la caja de password, se tiene que especificar * para la propiedad PasswordChar. Sin importar que caracter teclee el usuario, siempre aparecerá un asterisco en la caja.

La propiedad MaxLenght permite determinar el número máximo de caracteres que se le permitirá teclear al usuario en una caja de texto. Después de que este número es sobrepasado, la caja de texto emitirá un sonido y no aceptará ningún caracter más.

4.4 Controles que Presentan Opciones al Usuario

En la mayoría de las aplicaciones, se puede requerir presentar varias opciones al usuario. Esto se puede realizar utilizando botones de opciones o cajas de verificación, listas con entradas para seleccionar, o barras de enrollamiento que se utilizan para seleccionar un valor dentro de una escala. La siguiente tabla engloba estos controles:

Para realizar :	Se utiliza el control :
Un conjunto de opciones de las cuales el usuario sólo puede escoger una.	Option buttons
Un conjunto de opciones de las cuales el usuario puede elegir una o más.	Chek boxes
Una lista de opciones de las cuales el usuario puede seleccionar.	List box
Una lista con opciones en donde además el usuario puede escribir la opción deseada en lugar de buscarla y escogerla de la lista.	Combo box
Un rango de opciones correspondientes a una escala numérica	Horizontal o Vertical scroll bar

Check Boxes

Una caja de verificación (*check box*) indica si una determinada opción se encuentra activa o no. Se utilizan las *check boxes* en una aplicación para proporcionarle al usuario una opción de verdadero/falso o de Si/No. Debido a que las *check boxes* funcionan de forma independiente una de otra, un usuario puede seleccionar cualquier número de estos controles al mismo tiempo.

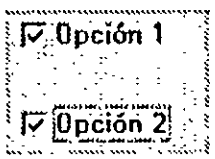


Figura 4.8

En el siguiente ejemplo se muestra el uso de las cajas de verificación. Las opciones que aparecen se utilizan para determinar si el texto se encuentra o no en un *font* regular o en *italic*.

La aplicación tiene una caja de texto y dos *check boxes*, como se ve en la siguiente figura.

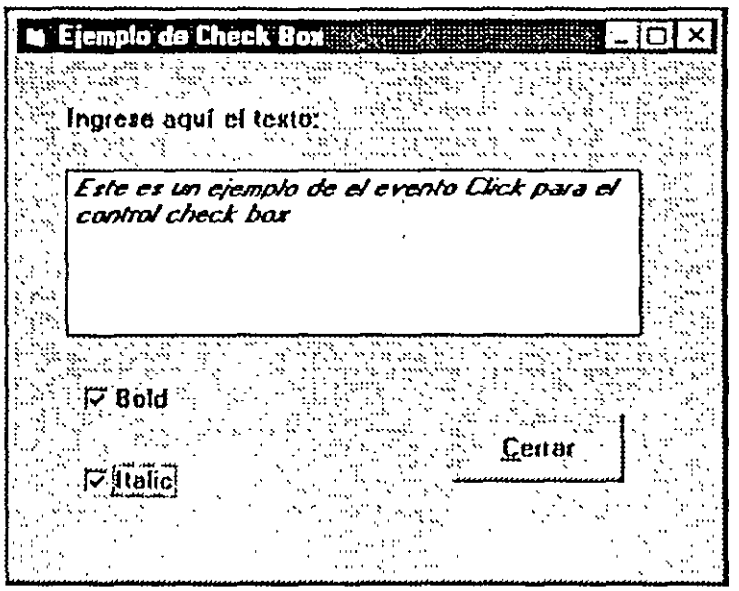


Figura 4.9

La siguiente tabla lista las propiedades y sus valores para los objetos de la aplicación.

Objeto	Propiedad	Valor
Form	Name	frmCheck
	Caption	Ejemplo del uso de Chek boxes
Text box	Name	txtDisplay
	Text	(vacío)
Check box	Name	chkBold
	Caption	&Bold
Check box	Name	chkItalic
	Caption	&Italic

Cuando se activa la caja de Bold o Italic, la propiedad de Value cambia a uno, y si se desactiva, cambia a cero.

El evento Click para el control *Check box* ocurre al momento de hacer click sobre la caja. El procedimiento para el evento Click para el control *Check box* prueba para ver si la caja ha sido seleccionada. Si fue así, el texto es convertido a negritas o itálicas cambiando las propiedades FontBold o FontItalic a **True** :

```
Sub ChkBold_Click ( )
   If ChkBold.Value = 1 Then
         txtDisplay.FontBold = True
   Else
         txtDisplay.FontBold = False
   End If
End Sub
```

```
Sub ChkItalic_Click ( )
   If ChkItalic.Value = 1 Then
         TxtDisplay.FontItalic = True
   Else
         TxtDisplay.FontItalic = False
   End If
End Sub
```

Options Buttons

Los botones de opciones presentan un conjunto de opciones al usuario. Sin embargo, a diferencia del control *Check box*, los botones de opciones sólo permiten seleccionar uno de los botones de un grupo de ellos. Esto significa que este control siempre trabaja en grupo y que al seleccionar uno de ellos inmediatamente se limpian los demás.

Por ejemplo, del grupo de botones de opciones que se muestra en la siguiente figura, el usuario puede seleccionar una de las tres opciones que se presentan.

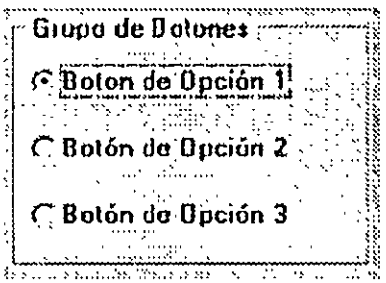


Figura 4.9

Creación de grupos de botones de opción

Todos los botones de opción que son colocados directamente sobre la forma, constituyen un grupo. Si se desea crear grupos adicionales, se debe colocar los botones dentro de controles *frames* o *picture boxes*.

Todos los botones de opción dentro de cualquier *frame* constituyen un grupo aparte, lo mismo que los botones contenido en un *picture box*. Cuando se crea un grupo separado de esta forma, siempre se debe colocar primero el *frame* o *picture box*, y después los botones de opciones dentro. La siguiente figura muestra dos grupos de botones.

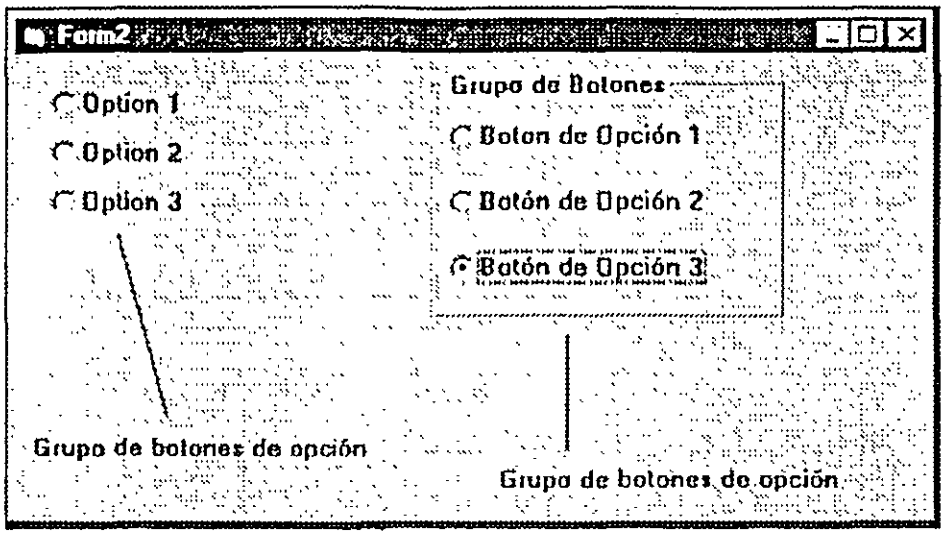


Figura 4.10

Un *frame* brinda un agrupamiento visual y funcional para controles. Los *frames* son comúnmente utilizados para agrupar botones de opciones o *check boxes*. Un usuario puede seleccionar sólo un botón de opción dentro del grupo cuando se ha colocado éste dentro del *frame*.

Para agrupar controles dentro de un *frame* :

1. Seleccionar la herramienta Frame de la caja de herramientas y dibujar el cuadro.

2. Dibujar los controles dentro del frame.

Cuando el usuario selecciona un *Option button*, el botón aparece sombreado. Un botón de opción puede ser seleccionado de las siguientes formas :

- Haciendo click sobre el con el mouse durante la ejecución de la aplicación.

- Moviéndose con el tabulador hasta el grupo de botones y después desplazándose con las flechas para seleccionar el botón adecuado dentro del grupo.

- Asignando el valor de True a la propiedad Value :
  Option1.Value = True

- Utilizando la llave de acceso especificada en la etiqueta del botón.

Para seleccionar un botón de opción por default dentro de un grupo, se debe establecer su propiedad Value como **True** durante el diseño. Esta opción permanecerá seleccionada hasta que el usuario elija otra o el mismo código lo haga.

Para deshabilitar un botón de opción, se debe establecer su propiedad Enabled como **False**. El botón aparecerá pero no estará disponible cuando se ejecute el programa.

La siguiente figura muestra una ventana donde se colocaron tres botones de opción para determinar si el número que se encuentra en el cuadro de texto debe aparecer en notación octal, decimal o hexadecimal. Cuando el usuario hace click sobre uno de esos botones, el número en el cuadro de texto es desplegado en la notación correspondiente.

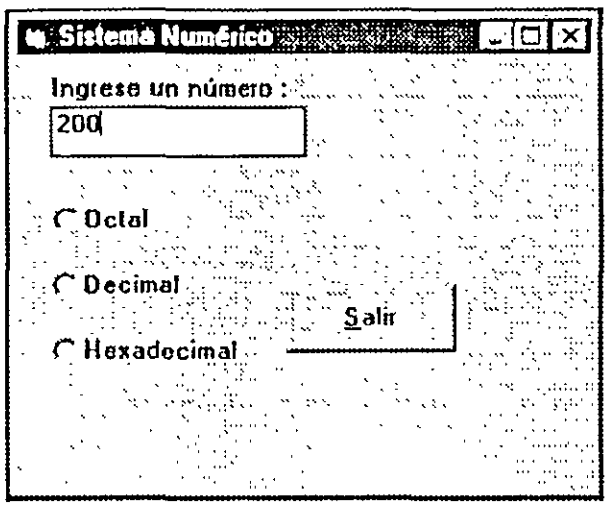


Figura 4.11

Para este ejemplo se debe crear una nueva forma con una caja de texto y tres botones de opción. Las propiedades se deben de establecer según la siguiente tabla :

Objeto	Propiedad	Valor
Text box	Name	txtNumber
	Text	(vacío)
Primer botón de opción	Name	optOctButton
	Caption	Use &octal
	Value	True
Segundo botón de opción	Name	optDecButton
	Caption	Use &decimal
Tercer botón de opción	Name	optHexButton
	Caption	Use &hexadecimal

El programa debe responder a los eventos de la siguiente forma:

- El evento Change (cambio) para la caja de texto debe leer un valor en el sistema apropiado y almacenarla en una variable llamada CurrentNum.

- El evento Click para el botón optOctButton regresa el número almacenado en CurrentNum en notación octal.

- El evento Click para el botón optDecButton regresa el número almacenado en CurrentNum en notación decimal.

- El evento Click para el botón optHexButton regresa el número almacenado en CurrentNum en notación hexadecimal.

La clave aquí esta en usar una variable a nivel de forma llamada CurrentNum. Esta variable representa el valor de la caja de texto (txtNumber) en forma numérica. El evento Change mantiene este valor siempre actualizado, así que todos los procedimientos para eventos Click deben regresar el número en el sistema numérico apropiado. CurrentNum debe declararse en la sección Declarations de la forma:

```
Dim CurrentNum As Long
```

Por default, esta variable es inicializada en 0, por lo tanto no requiere de un código para la inicialización.

El procedimiento para el evento Change verifica cual de los sistemas numéricos se encuentra activo en ese momento (octal, decimal o hexadecimal), lee el número y, si es necesario, realiza la conversión adecuada. Este procedimiento observa la condición de *optionbutton*.Value para determinar el sistema numérico que se está utilizando.

```
Sub txtNumber_Change ( )
   If optOctButton.Value = True Then
         CurrentNum = Val( "&O" & LTrim(txtNumber.Text) & "&")
   ElseIf optDecButton.Value = True Then
         CurrentNum = Val(LTrim(txtNumber.Text) + "&")
   Else
         CurrentNum = Val ("&H" & LTrim(txtNunber.Text) & "&")
   End If
End Sub
```

La función **Val** es utilizada para traducir el valor de cadena a número, y pueda reconocer las cadena octal, decimal y hexadecimal. La función **LTrim** elimina los espacios en blanco que pudiera tener el texto. El prefijo "&O" es utilizado para representar una cadena de dígitos en forma octal, y el prefijo "&H" para que sea interpretado como hexadecimal.

Los procedimientos para el evento Click de los botones de opciones utilizan las funciones **Hex** y **Oct** para desplegar CurrentNum en el sistema numérico apropiado:

```
Sub optOctButton_Click ( )
   txtNumber.Text = Oct(CurrentNum)
End Sub

Sub optDecButton_Click ( )
   txtNumber.Text = Format(CurrentNum)
End Sub

Sub optHexButton_Click ( )
   txtNumber.Text = Hex(CurrentNum)
End Sub
```

List boxes y Combo boxes

Los controles *List box* y *Combo box* presentan una lista de opciones para que el usuario elija. Por default, las opciones son desplegadas de forma vertical en una sola columna, sin embargo, se puede configurar para que sea una lista de múltiples columnas. Si el número de elementos excede el espacio en el que se despliegan, se añadirá automáticamente una barra de enrrollamiento. El usuario puede desplazarse por medio de estas barras hacia arriba o abajo y hacia la derecha o izquierda para ver toda la lista.

La siguiente figura muestra un ejemplo de un control *List box*:

Figura 4.12

Un *Combo box* es un control que combina las características de una caja de texto y de una *list box*. Este control permite al usuario realizar una selección ya sea tecleando el texto dentro de la caja o seleccionando un elemento de la lista.

Estilos de cajas combo

Existen tres estilos para las cajas combo:

- Caja combo desplegable (estilo 0)
- Caja combo simple (estilo 1)
- Lista combo desplegable (estilo 2)

Las siguientes figuras muestran cada uno de los estilos :

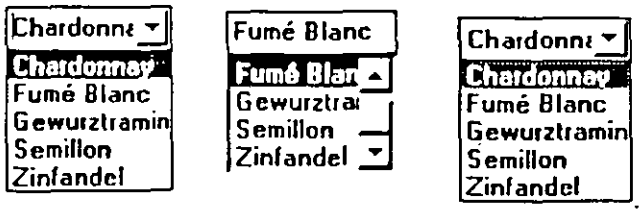


Figura 4.13

Estilo 0

Por default el estilo para el control *combo box* se encuentra en 0. El usuario puede ingresar directamente su selección o puede hacer click en la flecha adyacente a la caja para desplegar la lista de opciones de las cuales puede seleccionar una. Si se desea que el usuario seleccione opciones únicamente utilizando la lista y además se requiere conservar al máximo el espacio de la ventana, entonces se puede utilizar la lista combo (estilo 2).

Estilo 1

Si se establece la propiedad Style de un control *combo box* como 1, entonces se desplegará una lista de opciones que no puede ocultarse como en el caso anterior, y en donde además el usuario podrá ingresar texto en la caja para realizar alguna selección.

Estilo 2

Una *box list* desplegable es muy parecida a un *list box* común y corriente, la única diferencia radica en que la primera puede esconderse para aprovechar más el espacio de una ventana.

Cuando usar un Combo Box en lugar de un List Box

Generalmente, un *combo box* es apropiado cuando se trata de una lista de opciones sugeridas, y un *list box* es apropiado cuando se requiere limitar la entrada sólo a las opciones preestablecidas en la lista. Un *combo box* contiene un campo de edición, de tal forma que las opciones que no se encuentran en la lista pueden ser tecleadas por el usuario.

Además, los *combo box* ahorran espacio en una forma ya que la lista no se despliega hasta que el usuario hace click sobre la flecha que está a un lado. A excepción del estilo 1, un *combo box* puede colocarse en espacios reducidos en donde un *list box* no podría colocarse.

Nota: Utilizamos el término "el" *list box* o "el" *combo box*, refiriéndonos a el como un control. Preferimos utilizar en la mayoría de los controles, sus nombres en ingles que son

más comunes. Además en muchos casos la traducción no es siempre tan descriptiva como se quisiera.

La siguiente tabla menciona los eventos que el *combo box* no soporta según el estilo que se utilice :

Propiedad Style	Evento no aplicable
0 (Combo desplegable)	DblClick
1 (Combo simple)	DropDown
2 (lista desplegable)	Change, DblClick

Una práctica recomendable para los eventos de *list box* es añadir un *command button* para utilizarlo con la lista. El procedimiento del evento Click para este botón puede utilizar la selección de la lista, ejecutando la acción que se requiera para la aplicación.

Hacer doble click sobre un elemento de una lista tiene el mismo efecto que hacer click una vez y hacer click sobre el botón. Para hacer esto, se tiene que utilizar el procedimiento para el evento DblClick de la lista y dentro de éste llamar al procedimiento para el evento Click del botón de la siguiente forma :

```
Sub List1_DblClick ( )
   Command1_Click
End Sub
```

Adición de elementos en la lista

Para añadir elementos en la lista, se utiliza el método **AddItem**, el cual tiene las siguientes sintaxis:

box.**AddItem** [,*index*]

Argumento	Descripción
box	Nombre de la lista o combo box.
item	Expresión para añadir a la lista. Si la expresión es una constante literal se debe encerrar entre comillas.
index	Especifica en donde se debe insertar el nuevo elemento en la lista. Un índice de 0 representa la primera posición. Si se omite el índice, el elemento es insertado al final.

Por lo general los elementos de una lista son añadidos en el procedimiento para el evento Form_Load, donde se puede utilizar el método **AddItem** en cualquier momento. Esto brinda la posibilidad de añadir elementos a la lista de forma dinámica (en respuesta a las acciones del usuario).

Ejemplo: El siguiente código introduce los elementos "Germany", "India", "France" y "USA" en una lista llamada List1:

```
Sub Form_Load ( )
   List1.AddItem = "Germany"
   List1.AddItem = "India"
   List1.AddItem = "France"
   List1.AddItem = "USA"
End Sub
```

Para remover un elemento de una lista se pueden utilizar los métodos **RemoveItem** y **Clear**. **RemoveItem** tiene un argumento, *index*, el cual especifica el elemento que será eliminado:

control.**RemoveItem** *index*

Los argumentos *control* e *index* son los mismos que se utilizan con **AddItem**.

El método **Clear** permite eliminar todas las entradas de una lista y se utiliza de la siguiente forma:

```
List1.Clear
```

4.5 Obteniendo y Estableciendo el Foco de un Objeto

El foco es la habilidad de recibir la entrada del usuario a través del teclado o del mouse. Cuando un objeto tiene el foco, puede recibir la entrada de un usuario; por ejemplo, cuando una caja de texto tiene el foco, un usuario puede escribir texto en ella.

Los eventos GotFocus y LostFocus ocurren cuando un objeto recibe o pierde el foco.

Evento	Descripción
GotFocus	Ocurre cuando un objeto recibe el foco.
LostFocus	Ocurre cuando un objeto pierde el foco. Un procedimiento para el evento LostFocus es utilizado principalmente para la verificación y validación de actualizaciones, o para cambiar las condiciones que se establecieron con el procedimiento GotFocus.

Un objeto puede recibir el foco de las siguientes formas:

- Seleccionando un objeto durante la ejecución.
- Utilizando una tecla de acceso para seleccionar el control durante la ejecución.
- Utilizando el método SetFocus en el código.

4.6 Estableciendo el Orden de Tabulación

El orden de tabulación es el orden en el cual un usuario se mueve de un control a otro utilizando la tecla TAB. Normalmente, el orden de tabulación es el mismo que el orden en el que fueron creados los controles.

Por ejemplo, si se asume que primero fueron creadas dos cajas de texto, Text1 y Text2, y luego un *Command Button* llamado Command1. Cuando la aplicación se ejecuta, Text1 tendrá el foco. Presionando la tecla TAB el foco se moverá entre los controles en el orden en el que fueron creados.

Para cambiar el orden de tabulación para un control, se debe cambiar su propiedad TabIndex.

Por default, el valor de la propiedad TabIndex del primer control dibujado tiene un valor de 0, el segundo tiene un valor de TabIndex igual a 1 y así sucesivamente. Cuando se cambia el orden de tabulación, Visual Basic renumera de forma automática las posiciones de tabulación para los demás controles.

Por ejemplo, se puede establecer el valor de la propiedad TabIndex para Command1 en 0 con la siguiente instrucción :

```
Command1.TabIndex = 0
```

Los valores de los demás controles cambiarán de acuerdo a la siguiente tabla :

Control	TabIndex antes de realizar el cambio	TabIndex después de cambiar el valor de Command1
Text1	0	1
Text2	1	2
Command1	2	0

El valor más alto de TabIndex siempre es una unidad menor al número de controles que existen (debido a que empieza en 0). Aún cuando se asigne un número mayor a un control, Visual Basic convierte este valor al número de controles menos 1. Si se asigna un número negativo a una propiedad TabIndex, Visual Basic generará un error.

Para remover un control del orden de tabulación, se puede utilizar la propiedad TabStop. Es decir, cuando no se quiere que el control forme parte del orden de tabulación, se debe establecer su propiedad TabStop en **False** para que no sea considerado. De hecho, el control sigue manteniendo su orden de tabulación, pero al utilizar la tecla TAB para desplazarse entre los controles, éste control es ignorado.

4.7 Habilitación, Deshabilitación y Control de la Visibilidad en Tiempo de Ejecución

Se puede cambiar el estado de un control durante la ejecución cambiando sus propiedades en el código. En el caso de los *commands buttons*, existen varias acciones que pueden resultar muy útiles; esta acciones se presentan en la siguiente tabla:

Acción	Técnica	Propósito
Disabling	Asignando a la propiedad Enabled el valor False	Asegurarse de que el usuario no seleccionará el botón.
Enabling	Asignando a la propiedad Enabled el valor de True	Permitir que el usuario pueda seleccionar el botón.
Making invisible	Asignando a la propiedad Visible el valor de False	Deshabilitar el control y ocultarlo.

Cuando se habilita o deshabilita un botón, se controla el acceso del usuario a dicho botón. Generalmente, se deshabilita un botón debido a que es inapropiado tenerlo activo en ese momento - por ejemplo, cuando un botón sirve para borrar un archivo especifico, pero el archivo es de sólo lectura.

La propiedad Enabled es un ejemplo de una propiedad Booleana; es decir, toma un valor de verdadero o falso. Visual Basic almacena valores Booleanos como enteros. Cualquier valor diferente de cero equivale a verdadero y el cero equivale a falso.

Por ejemplo, se puede verificar si un botón se encuentra habilitado revisando su propiedad Enabled :

```
If Command1.Enabled = True Then
   Text1.Tex = "El botón está habilitado"
Else
   Text1.Text = "El botón está deshabilitado"
End If
```

Todos los controles excepto el *frame*, *shape*, *line* y *label*, poseen la propiedad de Enabled; y se puede utilizar de forma muy similar al ejemplo anterior.

4.8 Arreglos de Controles

Un arreglo de controles es un grupo de controles que comparten el mismo nombre y tipo. Incluso comparten el mismo procedimiento para los eventos. Un arreglo de controles tiene por lo menos un elemento y puede crecer hasta tener 254 elementos. Los elementos del mismo arreglo de controles tienen su propio conjunto de propiedades. Los arreglos de controles se utilizan comúnmente en el control de menús y en grupos de botones de opciones.

Los arreglos de controles son útiles si se quiere que varios controles compartan las mismas rutinas. Por ejemplo, si tres botones de opciones son creados como parte de un arreglo de controles, se ejecutará el mismo código sin importar cual de los tres botones fue seleccionado.

Si se quiere crear un nuevo control durante la ejecución, dicho control debe ser un miembro de un arreglo de controles. Con un arreglo de controles, cada elemento nuevo hereda los procedimientos para eventos comunes del arreglo.

Sin un mecanismo para arreglos de controles, la creación de controles durante la ejecución no sería posible. Los arreglos de controles resuelven este problema, gracias a que cada control hereda los procedimientos comunes que ya están escritos para el arreglo. Por ejemplo, se puede tener en una forma varias cajas de texto que reciben una fecha, un arreglo de controles permitiría, en este caso, compartir las rutinas de validación para las cajas de texto.

Para crear un arreglo de controles durante el diseño se pueden seguir tres formas :

- Asignar el mismo nombre a más de un control.
- Copiar un control existente y pegarlo en la misma forma.
- Asignar un valor diferente de Null a la propiedad Index.

Para añadir un arreglo de controles cambiando su nombre:

1. Dibujar los controles que se desean agregar al arreglo. (Los controles deben ser del mismo tipo).
2. Seleccionar uno de los controles y cambiar su propiedad Name.
3. Cuando se ingrese un nombre de control ya existente para otro control, Visual Basic preguntará si se desea generar un arreglo con ese control. Se debe elegir Yes como respuesta.

Para añadir un control durante la ejecución se utilizan las sentencias **Load** y **Unload**. De cualquier forma, el elemento a añadir debe incluirse en un arreglo ya existente. La sintaxis es la siguiente :

Load *control* (*index%*)

Unload *control* (*index%*)

Argumento	Descripción
control	Nombre del control que se va a añadir o a eliminar del arreglo de controles.
index%	El valor del índice del control en el arreglo.

Cuando se carga un nuevo elemento a un arreglo de controles, la mayoría de las propiedades son copiadas de los elementos ya existentes en el arreglo.

4.9 Controles Gráficos

Existen dos tipos principales de objetos que se pueden dibujar con Visual Basic, Formas y cajas de imágenes. Las cajas de imágenes son simplemente cajas como cualquier otra (cajas de texto, por ejemplo), las cuales se pueden colocar en las formas. De hecho, comenzaremos trabajando con una caja de imágenes y algunos dibujos elementales como dibujar un punto sencillo.

Dibujando puntos

Las funciones gráficas de Visual Basic que dibujan en las formas o en las cajas de imágenes son bastante parecidas a las funciones de Quick Basic. La primera función gráfica que veremos será Pset(), que coloca un pixel en la pantalla.

Para usar Pset(), inicie Visual Basic, llame a este proyecto Graphics.Mak y cambie el caption de la forma predefinida (Form1) a "Graphics". A continuación encuentre la propiedad AutoRedraw en la ventana de propiedades y colóquela como True. Esta propiedad indica que Visual Basic redibujará la gráfica en la forma si parte de ella es cubierta temporalmente por otra ventana.

El sistema coordenado en Visual Basic usa twips (1/1440 de pulgada) por default. Esto se debe a que se trató de hacer independiente el sistema de medida de Visual Basic, además de que a través del uso de twips, se logra una mayor resolución al momento de usar la impresora, ya que las impresoras normalmente tienen una mayor resolución que los demás dispositivos conectados a la computadora Sin embargo, es posible cambiar las unidades de medida a pixels, centímetros u otras unidades.

El sistema coordinado comienza en (0,0) en una forma o en una caja de imágenes; la x y la y se incrementan a la derecha y hacia abajo como sigue:

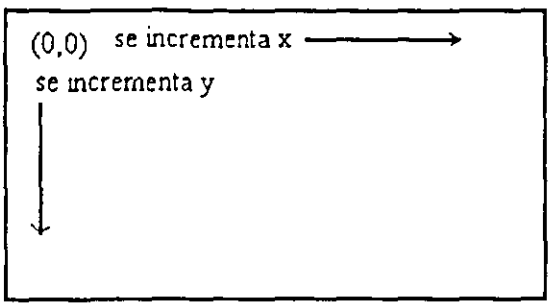


Figura 4.14

Utilizaremos este sistema coordinado para dibujar un pixel exactamente en el centro de la forma. Para hacerlo podemos obtener las dimensiones de la ventana examinando la ventana de propiedades. Por ejemplo, podemos tener una forma que mida 4320 twips de alto por 7200 twips de ancho (3 pulgadas por 5 pulgadas). En tal caso, la función Pset() aparecería como sigue:

```
Pset (4320 / 2, 7200 / 2)
```

Esta sentencia establece el pixel en el centro de la forma. La sintaxis de Pset() es como sigue:

```
PSet ( x, y )
```

Sin embargo, existe otra forma de centrar el pixel sin conocer de antemano las dimensiones de la forma, esto es, utilizando las propiedades ScaleHeight y ScaleWidth.

Cuatro propiedades preconstruidas nos pueden definir las dimensiones de una forma o caja de imágenes: Height, Width, ScaleHeight y ScaleWidth. Puede parecer que las propiedades naturales a usar para determinar como centrar un pixel son Height y Width, pero esas propiedades corresponden al ancho y alto exteriores de la ventana (incluyendo barra de título, barra de menús y demás). Las dimensiones del área cliente (el área de trabajo) son ancho ScaleWidth y alto ScaleHeight (en twips). Por esa razón, la sentencia Pset puede aparecer como sigue:

```
Pset ( ScaleWidth / 2, ScaleHeight / 2)
```

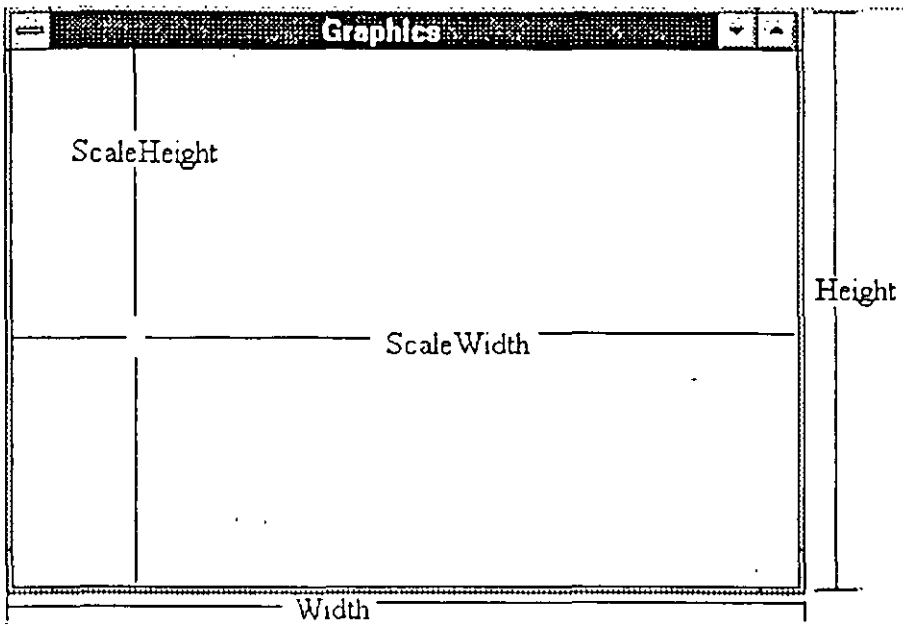


Figura 4.15

Debido a que queremos ejecutar esta línea tan pronto como la forma aparezca en la pantalla, debemos ponerla en el procedimiento Form_Load(), el cual se ejecuta cuando se despliega la forma. Coloque esa línea en la ventana de código como sigue:

```
Sub Form_Load()
        PSet ( ScaleWidth / 2, ScaleHeight / 2)
End Sub
```

Cuando usted ejecute el programa, el pixel aparecerá en el centro de la forma, como se muestra en la siguiente figura. Note que usted pudo haber usado otros eventos para hacer aparecer el punto en la forma, por ejemplo, el evento Form_Click (), que significa que el punto aparecerá en la forma cuando se haga click en cualquier punto dentro de la forma.

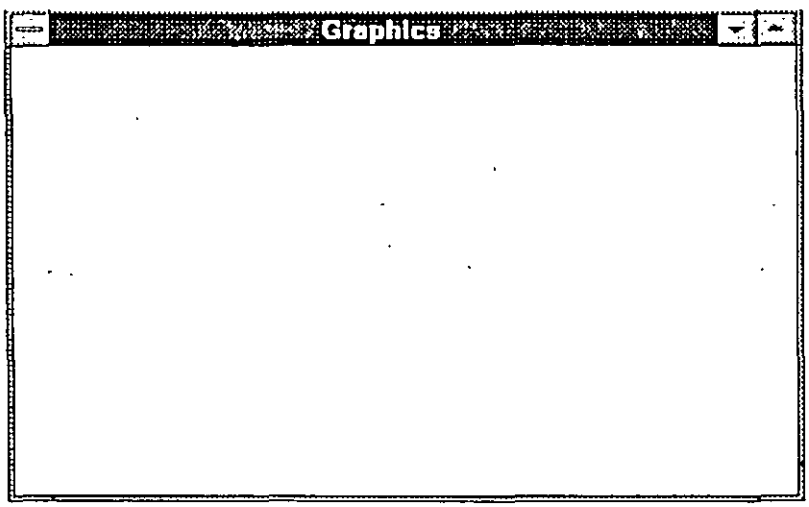


Figura 4.16

Ahora hagamos lo mismo con una caja de imágenes. Añada un objeto de tipo caja de imágenes (picture box) a la forma y establezca su propiedad AutoRedraw a True. El nombre predeterminado para este objeto es Picture1, asi que puede añadir esta linea al procedimiento Form_Load() como sigue:

```
Sub Form_Load()
        PSet ( ScaleWidth / 2, ScaleHeight / 2)
        Picture1.Pset ( Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2)
End Sub
```

En otras palabras, Pset() es un método de Visual Basic, esta conectado al objeto en el cual queremos trabajar. Cuando no se utiliza con un nombre de objeto, se asume que se quiere dibujar en la forma. Cuando se utiliza un nombre de objeto, sobre de él se aplica la función Pset(). Ahora ejecutemos el programa, podremos ver la caja de imágen con un punto en su centro, como se muestra en la siguiente figura Note que la caja de imágen oculta el punto en la forma, esto ocurre siempre; cualquier control colocado en una forma es colocado encima de los gráficos que estan en la forma, a la cual cubre.

Hasta ahora nada excitante ha pasado, hemos dibujado un punto simple en la forma. En la siguiente sección veremos como cambiarle el color.

Seleccionando colores

Existen varias formas de establecer sus propios colores en Visual Basic, la forma más fácil es usar los colores predefinidos en el archivo Constant.Txt. Cada valor de color en Visual Basic es un entero largo y muchos de tales valores estan listos para ser usados, como las constantes RED y BLUE, como se muestra en la tabla 2.1. Además, usted puede hacer que su programa se adapte a los colores establecidos por el usuario a través del Control Panel de Windows. Usted puede establecer la propiedad de color de fondo de una forma (BackColor) al estándar actual del sistema, haciéndola igual a WINDOW_BACKGROUND. Para cargar las definiciones de Constant.Txt en la sección de declaraciones de un módulo, utilice el menú Load Text... de Visual Basic. Debido a que los elementos de ese archivo estan declarados globalmente, estarán disponibles en cualquier lugar de su programa. (Debe cargarlos dentro de un módulo, ya que no se permiten declaraciones globales en una forma).

BLACK	&H0&
RED	&HFF&
GREEN	&HF00&
YELLOW	&HFFFF&
BLUE	&HFF0000
MAGENTA	&HFF00FF
CYAN	&HFFFF00
WHITE	&HFFFFFF

SCROLL_BARS	&H80000000
DESKTOP	&H80000001
ACTIVE_TILE_BAR	&H80000002
INACTIVE_TITLE_BAR	&H80000003
MENU_BAR	&H80000004
WINDOW_BACKGROUND	&H80000005
WINDOW_FRAME	&H80000006
MENU_TEXT	&H80000007
WINDOW_TEXT	&H80000008
TITLE_BAR_TEXT	&H80000009
ACTIVE_BORDER	&H8000000A
INACTIVE_BORDER	&H8000000B
APPLICATION_WORKSPACE	&H8000000C
HIGHLIGHT	&H8000000D
HIGHLIGHT_TEXT	&H8000000E
BUTTON_FACE	&H8000000F
BUTTON_SHADOW	&H80000010
GRAY_TEXT	&H80000011
BUTTON_TEXT	&H80000012

A continuación pintaremos un punto rojo. La sintaxis general de la sentencia Pset() es como sigue:

[objeto.]PSet [Step] (x!, y!) [, color&]

(x!, y!) representan la posición del pixel con el que queremos trabajar. Caba hacer notar que esas variables son medidas con respecto al objeto (forma o caja de imágen) en el que estamos dibujando. En otras palabras, cuando se establecen pixeles en una caja de imágen, (x!, y!) representan la posición del pixel con respecto a la esquina superior izquierda de la caja de imágen (la cual es (0,0)). Además, el valor de color, color&, es siempre un entero largo.

La palabra clave Step la verá en muchos métodos de dibujo. Indica que las coordenadas especificadas estan dadas respecto a la posición gráfica actual, cuyas coordenadas estan almacenadas en las propiedades CurrentX y CurrentY de las formas y cajas de imágenes. Como se verá mas adelante, podremos dibujar líneas u otras dibujos utilizando la posición del dibujo actual. En nuestro ejemplo no utilizaremos la palabra Step. Para poner el pixel de color rojo, simplemente añada la constante predefinida RED como en el siguiente ejemplo:

```
Sub Form_Load()
        PSet ( ScaleWidth / 2, ScaleHeight / 2)
        Picture1.Pset ( Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2), RED
End Sub
```

Es necesario cargar el archivo Constant Txt en un módulo debido a que la constante RED esta definida ahí. Al ejecutar este programa se cambiará el punto dibujado en la pantalla a rojo. En Visual Basic existen otras formas de indicar el color que usted desea.

Por ejemplo, usted puede utilizar la función RGB() para especificar el color. Esta función toma tres argumentos, (un valor de color rojo, un valor de color verde y un valor de color azul), como sigue:

```
RGB (RedVal%, GreenVal%, BlueVal%)
```

Cada uno de esos colores puede ir desde un mínimo de 0 (cuando se excluye el color por completo) hasta un máximo de 255 (cuando es mas fuerte). Para un punto de color rojo puro, puede usar RedVal=255, GreenVal=0 y BlueVal=0, de la siguiente manera:

```
Sub Form_Load()
   PSet ( ScaleWidth / 2, ScaleHeight / 2)
   Picture1.Pset ( Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2), RGB (255, 0, 0)
End Sub
```

La última forma de especificar colores en Visual Basic es con la función QBColor(). Quick Basic tiene 16 colores predefinidos que usted puede accesar en Visual Basic mediante la función QBColor(). Esta función devuelve un entero largo que corresponde al valor correcto RGB de Visual Basic. Utilice QBColor como sigue: QBColor(nnn%), donde nnn% es el color del número en Visual Basic del 0 al 15, asociado con el color que desee, como se muestra en la siguiente tabla:

Número de color	Color
0	Negro
1	Azul
2	Verde
3	Cyan
4	Rojo
5	Magenta
6	Amarillo
7	Blanco
8	Gris
9	Azul Claro
10	Verde Claro
11	Cyan Claro

12	Rojo Claro
13	Magenta Claro
14	Amarillo Claro
15	Blanco Brillante

Por ejemplo, usted puede poner el punto rojo en la forma siguiente:

```
Sub Form_Load()
   PSet ( ScaleWidth / 2, ScaleHeight / 2)
   Picture1.Pset ( Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2), QBColor (4)
End Sub
```

Dibujando Lineas

Una nueva herramienta en la caja de herramientas de Visual Basic le permite dibujar líneas a momento del diseño: la herramienta de-control Line. Usarla es fácil, al darle doble click aparecerá una pequeña línea con dos manijas; solo jale las manijas para colocar la línea en la posición y tamaño que desee. Sin embargo, lo que nos interesa ahora es dibujar líneas con código, por lo que utilizaremos el método Line, cuya sintaxis aparece a continuación:

```
[objeto.]Line [[Step] (x1!, y1!)] - [Step] (x2!, y2!) [, [color&],B[F]]]
```

Nuevamente, nótese que este es un método que puede ser aplicado a varios objetos especificando el nombre del objeto. Para dibujar una línea, se involucran dos puntos: el origen (x1!, y1!) y el destino (x2!, y2!). También se puede especificar el color de la línea con el parámetro color&. Los parámetros B y F se utilizan cuando se desean dibujar cajas.

La siguiente línea de código muestra un ejemplo de como dibujar una línea que cruce la pantalla diagonalmente:

```
Line (0, 0) - (ScaleWidth, ScaleHeight)
```

Esto es todo lo que se requiere. Para colocar este código en nuestro ejemplo, desaparezca la caja de imágenes de la forma Graphics y colóque la línea de código para dibujar una línea en la forma dentro del procedimiento Form_Load(), como se muestra a continuación:

```
Sub Form_Load()
        Line (0, 0) - (ScaleWidth, ScaleHeight)
End Sub
```

Al ejecutar el programa aparecerá una ventana como la quese muestra a continuación. Usted puede especificar el color de la línea, añadiéndo un parámetro al final de la línea de código:

```
Sub Form_Load()
        Line (0, 0) - (ScaleWidth, ScaleHeight), BLUE
End Sub
```

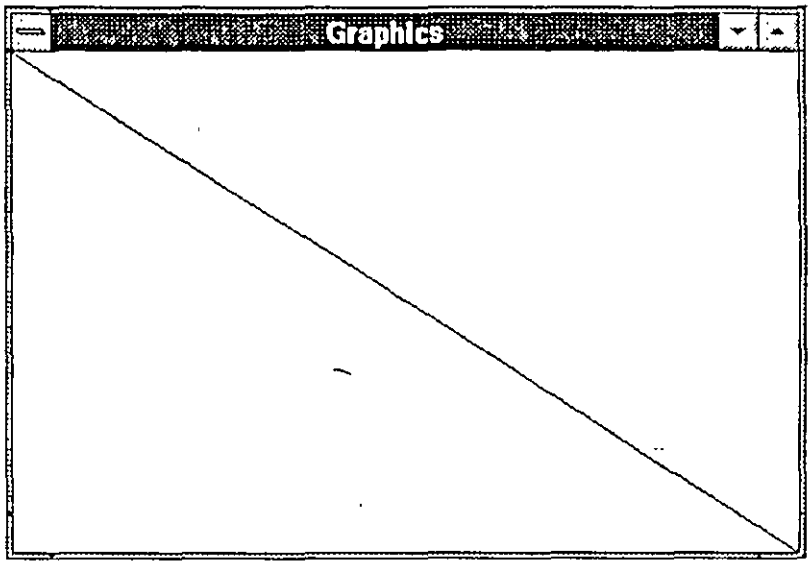


Figura 4.17

Otra forma de seleccionar el color del objeto es especificando la propiedad ForeColor al objeto antes de dibujarlo. Para hacerlo, veamos el siguiente ejemplo:

```
Sub Form_Load()
        ForeColor = BLUE
        Line (0, 0) - (ScaleWidth, ScaleHeight)
End Sub
```

Hasta que este color de dibujo del objeto sea cambiado nuevamente, el color será azul. En otras palabras, el texto y lo demás será azul cuando se coloque en la forma. Usted notará que cambiando el color del frente (Foreground) no cambiará el color del texto u otros objetos que hayan sido previamente colocados en la forma.

Las propiedades de los objetos que se utilizan cambian dependiendo del tipo de objeto, es decir, cada objeto tiene propiedades particulares que pueden examinar seleccionando el objeto y abriendo la ventana de propiedades para ese objeto.

Una pluma gráfica, por ejemplo, esta asociada con cada objeto que puede desplegar gráficos. Una de las propiedaddes asociadas con la pluma gráfica es la propiedad DrawWidth, con el cual es posible cambiar el ancho de la línea que se esta dibujando (o el tamaño del punto que dibuja Pset()). El valor predeterminado para esta propiedad gráfica es 1, que corresponde a la linea mas delgada (la que dibujamos en nuestro ejemplo). Esta linea tiene un pixel de ancho en la pantalla. Es posible alterar el ancho de esta linea cambiando el valor de DrawWidth, por ejemplo, a 2, que resultaría en una línea de 2

pixeles de ancho. Para ver algunas de estas líneas, escribamos un ciclo dentro del procedimiento Form_Load(), como se muestra en el siguiente ejemplo:

```
Sub Form_Load()
        For i = 1 to 9
          DrawWidth = i
          Line (0, i * ScaleHeight / 10) - (ScaleWidth, i * ScaleHeight / 10)
        Next i
        DrawWidth = 1
End Sub
```

Este ejemplo despliega una ventana en la cual la línea a lo largo de la forma se va ensanchando a medida que llega a la parte baja de la forma.

Usted puede afectar el estilo de la línea al igual que el ancho. En otras palabras, se puede especificar donde la línea es sólida (predefinida) o se rompe en puntos o rayas. Hay siete diferentes tipos de líneas, desde sólida hasta punteadas o transparentes.

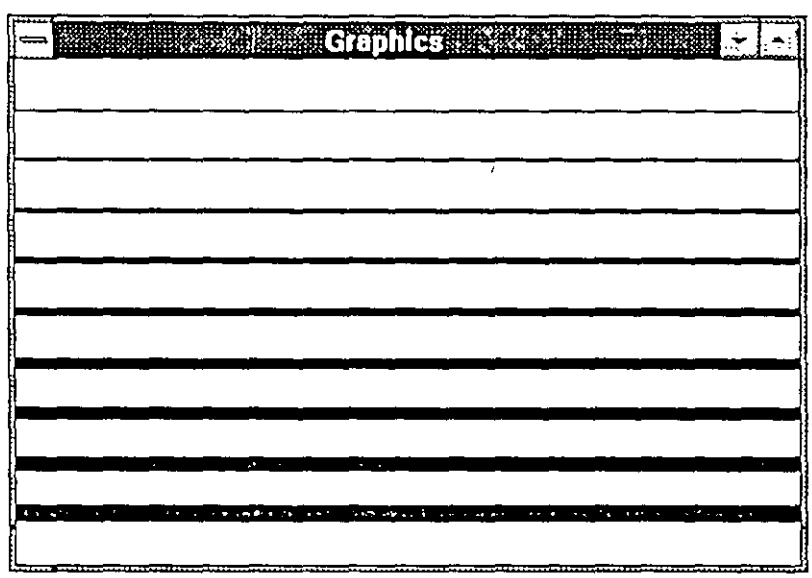


Figura 4.18

DrawStyle	Resultado
0	Línea sólida (valor predefinido de DrawStyle)
1	Línea discontinua
2	Línea punteada
3	Línea raya - punto
4	Línea raya - punto - punto
5	Línea transparente (no aparece)
6	Línea interior

Un ejemplo de como ver los diferentes estilos de línea dibujados es colocar el siguiente código en el procedimiento Form_Click(), como se muestra a continuación:

```
Sub Form_Click()
        Cls
        For i = 1 to 7
           DrawStyle = i - 1
           Line (0, i * ScaleHeight / 8) - (ScaleWidth, i * ScaleHeight / 8)
        Next i
End Sub
```

El resultado de este programa es la ventana que se muestra a continuación. Cuando usted hace click en ella, se dibujan los diferentes estilos de línea disponibles. Note que se ha incluido la instrucción Cls al principio del procedimiento, esta instrucción limpia la pantalla de los elementos que se hayan dibujado previemente, dejando la forma o la caja de imágenes en blanco.

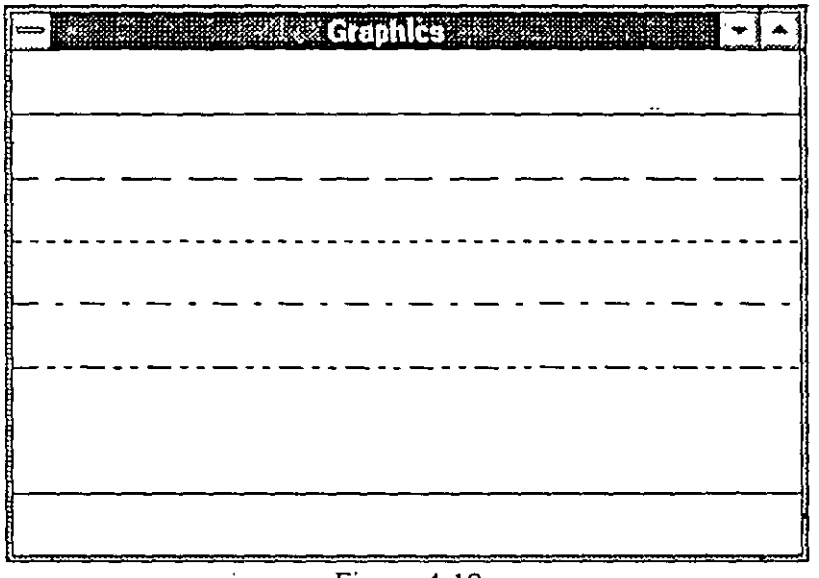


Figura 4.19

Dibujando Rectángulos

El control Linea, permite dibujar líneas al momento de diseñar la forma. El control Shape permite dibujar otros tipos de figuras, incluyendo rectángulos, óvalos, cuadrados, círculos y cuadrados redondeados. Para dubujar cualquier figura, simplemente haga click sobre la herramienta Shape de la caja de herramientas y dibújela dentro de la forma. Como ejercicio, suponga que queremos dibujar una figura que sea exactamente el doble de la figura 1, por lo que tendremos que escribir el siguiente código:

```
Sub Form_CLick()
        Shape1.Width = 2 * Shape1.Width
End Sub
```

La herramienta Shape, normalmente no se utiliza para producir una salida del programa, sin embargo nos puede ayudar a enmarcar varios objetos de dibujo dentro de la forma. Para producir rectángulos normales utilizaremos nuevamente la herramienta Line, con solo añadirle el parámetro B:

[objeto.]Line [[Step] (x1!, y1!)] - [Step] (x2!, y2!) [, [color&],B[F]]]

Cuando se especifica una línea en Visual Basic se especifican los puntos inicial y final. De la misma forma, cuando se especifica un rectángulo, se deben especificar solo dos puntos, las esquinas superior izquierda e inferior derecha., como sigue:

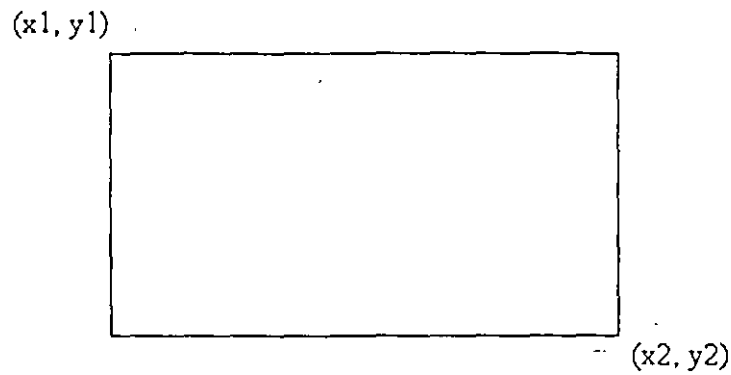


Figura 4.20

Para especificar que lo que queremos dibujar es un rectángulo y no una línea, debemos añadir el parámetro B a la instrucción Line, de la siguiente forma:

```
Sub Form_Load()
  DrawWidth = 8
  DrawStyle = 6
  Line (0, 0) - (ScaleWidth / 2, ScaleHeight / 2), , B
  Line (ScaleWidth/4, ScaleHeight/ 4) - (3*ScaleWidth/4, 3*ScaleHeight/4),, B
  Line (ScaleWidth / 2, ScaleHeight / 2) - (ScaleWidth, ScaleHeight), , B
End Sub
```

Este ejemplo genera una ventana como se muestra en la figura siguiente.

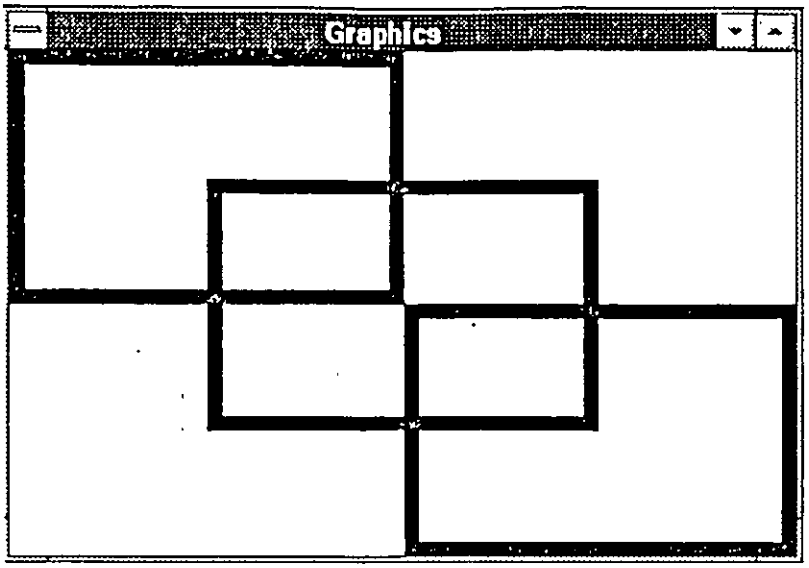


Figura 4.21

Rellenado de figuras

Las figuras cerradas que usted dibuja en Visual Basic pueden ser rellenadas automáticamente con algún patrón determinado para añadir mas efectos visuales. Existen ocho patrones de llenado de figuras, que se mencionan en la siguiente tabla:

Valor del estilo de llenado	Patrón de relleno resultante
0	Sólido
1	Transparente
2	Lineas horizontales
3	Lineas verticales
4	Diagonales hacia arriba
5	Diagonales hacia abajo
6	Cuadriculado
7	Cuadriculado diagonal

A continuación haremos un ejemplo en el que dibujaremos ocho rectángulos y cada uno tendrá un estilo de llenado distinto. El estilo de llenado se determina con la propiedad FIllStyle de lada rectángulo. El siguiente procedimiento nos muestra el código para nuestro ejercicio, que colocaremos en el procedimiento Form_Load():

```
Sub Form_Load()
   SX = ScaleWidth
   SY = ScaleHeight
   For i = 0 To 3
     FillStyle = i
     Line ((2 * i + 1) * SX / 9, SY / 5) - ((2 * i + 2) * SX / 9, 2 * SY / 5), , B
     FillStyle = i + 4
     Line ((2 * i + 1) * SX / 9, 3*SY / 5) - ((2*i + 2) *SX / 9, 4*SY / 5), , B
```

```
    Next i
End Sub
```

El resultado de este programa se muestra en la siguiente figura:

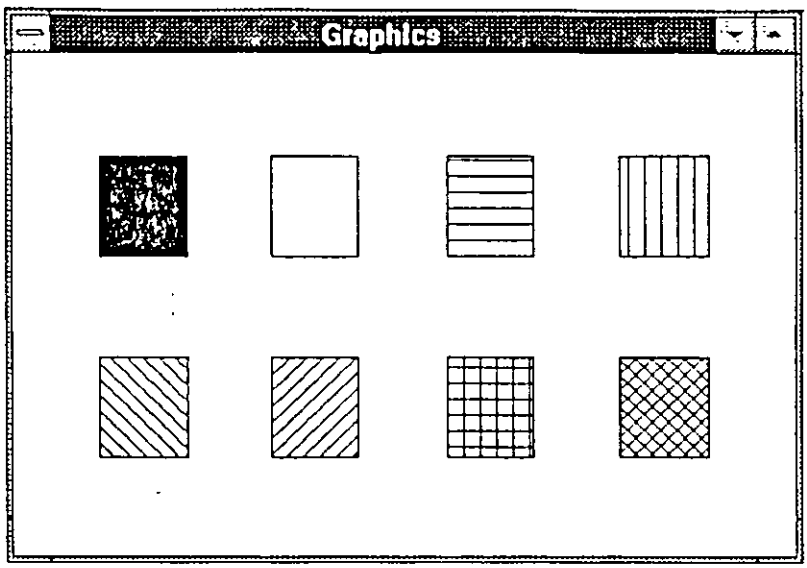


Figura 4.22

Dibujando círculos

Dibujar círculos es muy fácil utilizando el método Circle, como se muestra a continuación:

```
[objeto.]Circle [Step] (x1!, y1!), radio! [, [color&],[inicial!] [,[final!]
        [, aspecto!]]]
```

En este método, (x!, y!) representa el centro del círculo, y radio! representa se radio (todo en twips). Además, usted puede dibujar arcos especificando el ángulo inicial y el ángulo final.

A continuación haremos un ejemplo para dibujar algunos círculos:

```
Sub Form_Load()
        FillStyle = 5                       'Diagonales
        ForeColor = RGB(255, 0, 0)          'Rojo
        Circle (ScaleWidth / 4, ScaleHeight / 4), ScaleHeight / 5
End Sub
```

Para establecer el color de relleno en rojo, usaremos la instrucción FillColor, como a continuación se muestra:

```
Sub Form_Load()
        FillStyle = 5                       'Diagonales
```

```
        ForeColor = RGB(255, 0, 0)          'Rojo
        FillColor = ForeColor
        Circle (ScaleWidth / 4, ScaleHeight / 4), ScaleHeight / 5
End Sub
```

Ahora tanto el círculo como su relleno son rojos. Para dibujar elipses, usted debe usar el argumento aspecto! con la instrucción círculo. El aspecto indica la relación vertical a horizontal para dibujar elipses. Para dibujar una elipse que sea dos veces mas alta que su ancho, establezca la relación de aspecto a 2, como en el ejemplo siguiente:

```
Sub Form_Load()
        Circle (ScaleWidth / 4, ScaleHeight / 4), ScaleHeight / 5
        FillStyle = 2
        Circle (ScaleWidth / 2, ScaleHeight / 2), ScaleHeight / 3,,,,2 'Elipse
End Sub
```

Los dibujos resultantes aparecen como se muestra en la siguiente figura:

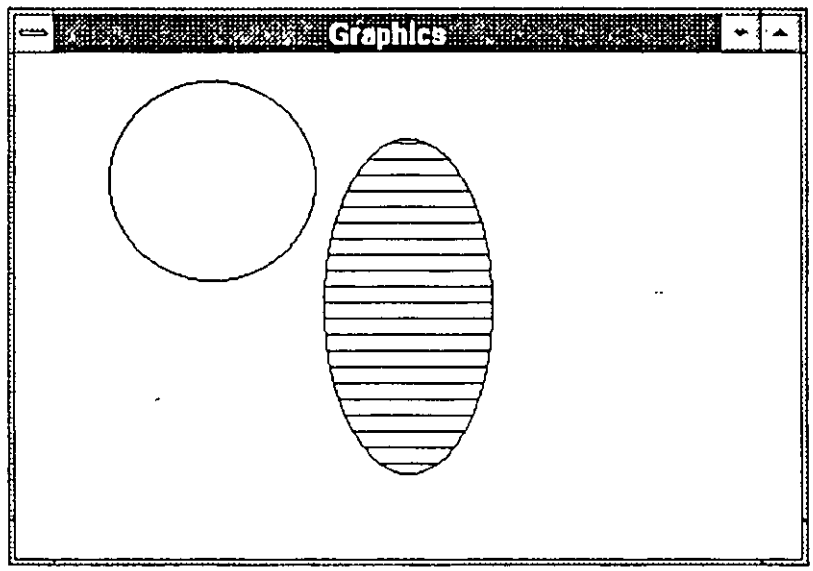


Figura 4.23

También es posible dibujar arcos especificando los argumentos inicial! y final! en radianes que van de 0 a 2 pi. Para dibujar un arco que vaya desde 0 (la posición de las 3 en el reloj) hasta pi (la posición de las 9, moviéndose en forma inversa a las manecillas del reloj), hagamos el siguiente ejemplo:

```
Sub Form_Load()
    Circle (ScaleWidth / 4, ScaleHeight / 4), ScaleHeight / 5
    FillStyle = 2
    Circle (ScaleWidth / 2, ScaleHeight / 2), ScaleHeight / 3,,,,2 'Elipse
    Circle (3*ScaleWidth/4, 3*ScaleHeight/4), ScaleHeight/5, ,0, 3.1415  'Arco
End Sub
```

Las figuras resultantes aparecen en la siguiente figura:

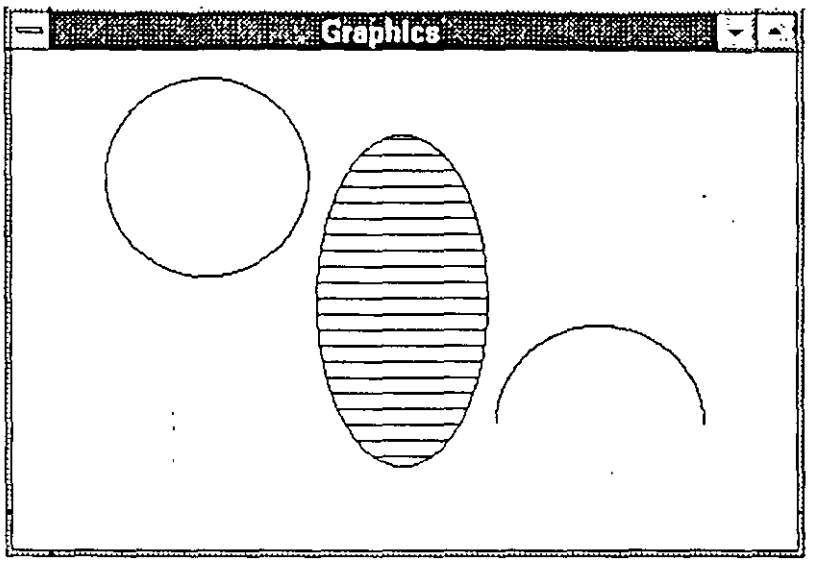


Figura 4.24

4.10 Procedimientos Aplicables a Controles Gráficos

El control del mouse le permitirá aprender cosas muy interesantes sobre la programación modular, ya que lo aplicaremos sobre un ejemplo de un programa de dibujo. El mouse es una de las dos interfaces mas importantes de Windows (la otra es el teclado) y aprenderá sobre sus eventos mas importantes y a tenerlo controlado. A continuación comenzaremos a estudiar los eventos mas importantes del mouse.

Evento MouseDown

Un evento MouseDown es generado cuando el usuario posiciona el cursor del mouse en una forma y presiona un botón. Este evento no es el mismo que el evento click. En un evento click, el usuario debe presionar y soltar el botón del mouse; un evento MouseDown ocurre cuando el usuario simplemente presiona un botón del mouse. Esas clases de eventos son reconocidas por formas, cajas de imágenes, etiquetas y cualquier control que incluya una lista. Note que los objetos tales como botones responden solo a eventos click, no a eventos MouseDown.

En un evento MouseDown, usted obtiene considerablemente mas información que la que obtiene con un evento Click. Para visualizar esto, iniciemos el ejemplo del programa Paint. Comience un nuevo proyecto en Visual Basic y dele a la forma predeterminada (Form1) el Caption Paint; establezca la propiedad AutoRedraw a True. Abra la ventana de código de la forma y carge el evento Form_MouseDown(). Este procedimiento tiene la siguiente plantilla:

```
Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)

End Sub
```

Los argumentos que se le pasan a este procedimiento son Button, Shift, X y Y. Los argumentos Button y Shift pasan información acerca del botón del mouse y el estado de la tecla Shift, los argumentos X y Y reportan la posición del cursor del mouse. Usted puede hacer uso de la información reportando la posición del cursor cuando presiona un botón del mouse. Para hacerlo, genere dos cajas de texto, Text1 y Text2, y colóquelas en la forma. Usted puede reportar la posición, (X, Y), como se muestra en el siguiente ejemplo:

```
Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
        Text1.Text = Str$(X)
        Text2.Text = Str$(Y)
End Sub
```

Ahora ejecute el programa y presione un botón del mouse (el evento MouseDown ocurre cuando cualquier botón del mouse se presiona); cuando lo haga la posición del cursor del mouse es reportada en las cajas de texto Text1 y Text2. De esta forma se puede leer directamente la información acerca de la posición del cursor del mouse. Ahora examinaremos los otros argumentos del procedimiento, Button y Shift, los cuales son enteros. El argumento Button describe cual botón fue presionado encodificando esa información en sus tres bits menos significativos, ese valor designa que botón fue presionado primero, no si se presionaron dos botones juntos. Los posibkles valores de Button se muestran en la siguiente tabla:

Valor del Botón	Binario	Significa
1	0000000000000001	Fue presionado el botón izquierdo
2	0000000000000010	Fue presionado el botón derecho
4	0000000000000100	Fue presionado el botón central

Note que Button no puede ser cero porque al menos un botón fue presionado para que ocurriera el evento MouseDown. Para saber cual botón fue presionado, añada otra caja de texto a la forma Form1 y en ella reportaremos el botón que se presionó utilizando una sentencia Select Case, como se muestra a continuación:

```
Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
        Text1.Text = Str$(X)
        Text2.Text = Str$(Y)
        Select Case Button
                Case 1
                        Text3.Text = "Botón Izquierdo"
                Case 2
                        Text3.Text = "Botón Derecho"
                Case 4
```

```
            Text3.Text = "Botón Central"
        End Select
End Sub
```

Cuando se ejecute el programa, no solo se reportará la posición del cursor del mouse, sino también el nombre del botón que se presionó. Por ejemplo, si usted presiona el botón izquierdo, verá una ventana semejante a la siguiente:

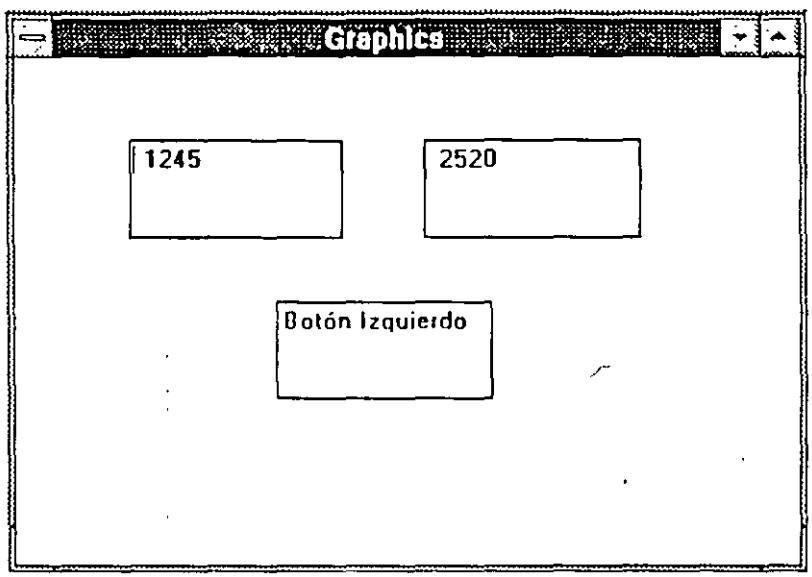


Figura 4.25

El argumento Shift también devuelve información útil. Este entero indica si se presionaron las teclas Shift o Control cuando se hizo click en el botón del mouse. Esto es útil debido a que Visual Basic no tiene un evento llamado ShiftClick. Shift puede tomar tres valores distintos, como se muestra en la tabla siguiente:

Valor de Shift	Binario	Significa
0	0000000000000000	No se presionaron si Shift ni Ctrl
1	0000000000000001	Se presionó Shift
2	0000000000000010	Se presionó Ctrl

Otra de las funciones importantes del evento MouseDown además de reportar los valores antes mencionados es que inicia la operación de dibujo.

Si usted desea dibujar una línea en un programa de dibujo, por ejemplo, puede presionar el botón izquierdo del mouse para indicar donde desea iniciar la línea, moverse al punto donde desea que la línea termine y soltar el botón; entonces la línea quedará dibujada entre esos dos puntos. En términos de Visual Basic, podemos traducir esos como salvar el punto donde ocurrió el evento MouseDown, el cual podemos llamar el punto de Anclaje (Anchor) (AnchorX, AnchorY), estableciendo la posición gráfica actual en el mismo punto y después realizando la operación de dibujo cuando el botón del mouse se levante.

Para probar lo anterior, usted puede auitar las tres cajas de texto de la forma y colocar en la ventana de código de Form_MouseDown lo siguiente:

```
Sub Form_MouseDown
        If Button = 1 Then
                AnchorX = X
                AnchorY = Y
                CurrentX = X
                CurrentY = Y
        End If
End Sub
```

Usted también puede convvertir a AnchorX y AnchorY en variables globales para que cualquier rutina de la aplicación pueda saber donde ocurrió el evento MouseDown. Para convertir esas variables en globales, declare lo siguiente en un módulo:

```
Global AnchorX As Integer
Global AnchorY As Integer
```

Ahora usted puede decir, desde cualquier punto de la aplicación, donde está el anclaje y comenzar a dibujar.

Evento MouseMove

Una capacidad de un programa de dibujo debe ser el dibujar continuamente cuando el usuario mueva el cursor del mouse, en otras palabras, el usuario debe sar capaz de presionar el botón y mover el mouse para crear un dibujo en estilo libre dejando un trazo de pixeles. Para hacerlo, debe conocer donde está el cursor en un momento en particular; y para eso nos ayuda el evento MouseMove. Cada vez que el mouse es desplazado a tarvés de una forma o control seleccionado (cajas de lista de archivos, etiquetas, cajas de lista o cajas de imágenes), es generado un evento MouseMove. Un evento MouseMove no es generado usualmente, para cada pixel sobre el que se mueva el cursor, en su lugar, Windows genera solo cierto número de esos eventos por segundo, aun asi, éstos serán suficientes para nuestra aplicación.

Además, el argumento Button en un evento MouseMove reporta el estado completo de los botones del mouse (puede reprtar si se presionó mas de un botón). Los valores que Button puede reportar se muestran a continuación:

Valor del Botón	Binario	Significa
0	0000000000000000	No se presionó ningún botón
1	0000000000000001	Solo se presionó el botón Izquierdo
2	0000000000000010	Solo se presionó el botón Derecho
3	0000000000000011	Se presionaron los botones izquierdo y derecho

4	0000000000000100	Solo se presionó el botón central
5	0000000000000101	Se presionaron los botones izquierdo y central
6	0000000000000110	Se presionaron los botones derecho y central
7	0000000000000111	Se presionaron los tres botones

Ahora utilizaremos el evento MouseMove para dibujar en la aplicación Paint. Debemos establecer el punto de anclaje y si el usuario se mueve teniendo presionado el botón del mouse, usted dibujará siguiendo los movimientos del cursor en la pantalla. Para hacerlo, necesitamos la ventana de código de Form_MouseMove() como se muestra a continuación:

```
Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)

End Sub
```

Debido a que queremos asegurarnos que solo dibujemos cuando el botón izquierdo esté presionado, podemos checar el valor de Button de la siguiente forma:

```
Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    If Button = 1 Then
    :
    End If
End Sub
```

Ahora, para dibujar en la forma, se debe seguir el cursor del mouse. Se dibuja desde la posición del ancla hasta la posición actual, podemos hacer lo anterior utilizando el método Line:

```
Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    If Button = 1 Then
        Line -(X, Y)
    End If
End Sub
```

Si usted no especifica el primer punto con el método Line, éste utiliza la posición gráfica actual, la cual se estableció con el evento MouseDown. Ahora usted esta en posibilidad de dibujar libremente simplemente presionando el botón del mouse, como se muestra en la figura siguiente:

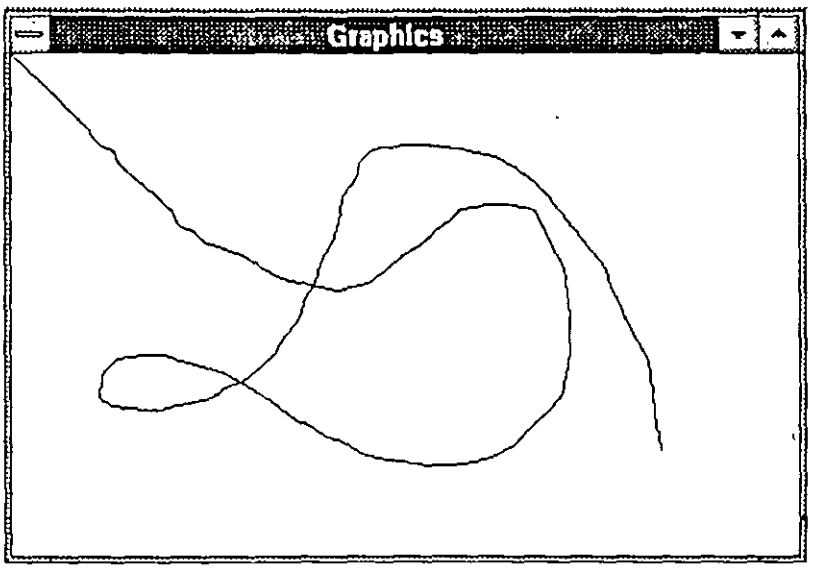


Figura 4 26

Los programas de dibujo deben tener la capacidad de dibujar figuras geométricas tales como líneas o rectángulos. Para esos casos, necesitamos determinar la posición donde el botón fue soltado.

5. MENÚS

5.1 Creación de un Menú

Muchas aplicaciones sencillas constan solo de una forma y varios controles, pero se pueden mejorar las aplicaciones Visual Basic si se utilizan menús y cajas de diálogo. En este capítulo veremos la forma de implementar un sistema de menús a una aplicación.

Creación de menús durante diseño

Si se desea brindarle al usuario un conjunto de comandos para manipular la aplicación, los menús son la opción más conveniente y consistente para agrupar comandos y la forma más fácil de que un usuario los accese.

La siguiente figura ilustra los elementos de una interface de menú muy común en los programas de Windows.

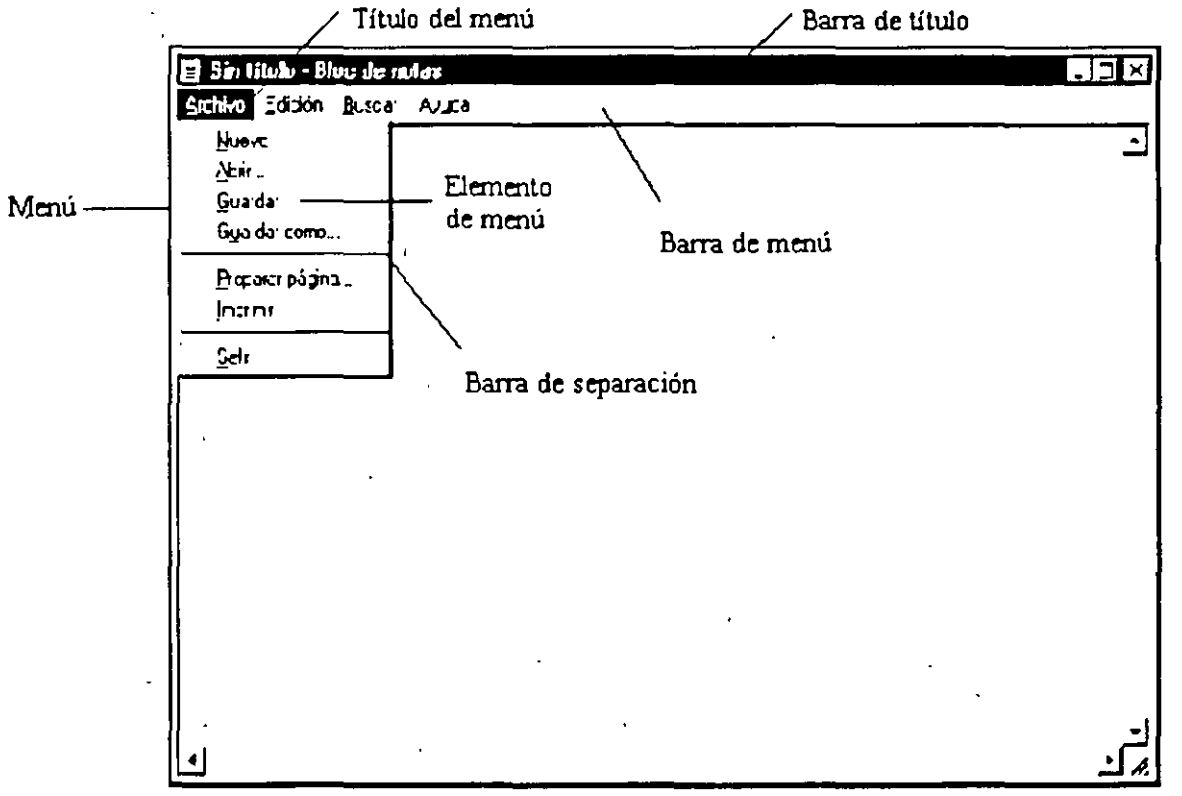


Figura 5.1

La barra de menú aparece inmediatamente debajo de la barra de título de la ventana (donde se encuentra el Caption) y puede contener uno o más títulos de menús. Cuando se hace un click sobre un título de menú (como Archivo), se despliega una lista con todas las opciones que contiene dicho menú. Los menús pueden contener comandos (como Guardar), barras separadoras y títulos para submenús. Cada elemento de un menú corresponde a un elemento que es definido durante el diseño.

Para hacer que la aplicación sea más fácil de usar, se pueden agrupar los comandos de los menús de acuerdo con su función. Por ejemplo, los del menú Archivo están agrupados porque todos ellos modifican o afectan de alguna manera los archivos de un programa.

Algunos comandos realizan una acción directamente, es decir, no requieren de información adicional para funcionar; por ejemplo, el comando Copiar no pregunta nada acerca de lo que se está copiando al portapapeles. Algunos otros comandos muestran una caja de diálogo en la que solicitan información adicional como en el caso del comando Guardar como...

5.2 Adición de Opciones para el Menú

La ventana de diseño para menús

Los menús son creados utilizando la ventana de diseño de menús. Aquí se pueden añadir elementos a un menú ya creado o crear nuevos menús.

Para mostrar la ventana de diseño de menús se puede :

- Elegir del menú Window la opción Menu Design

  - O -

- Elegir el botón Menu Design de la barra de herramientas.

De cualquier forma se abre la ventana Menu Design que se muestra en la siguiente figura:

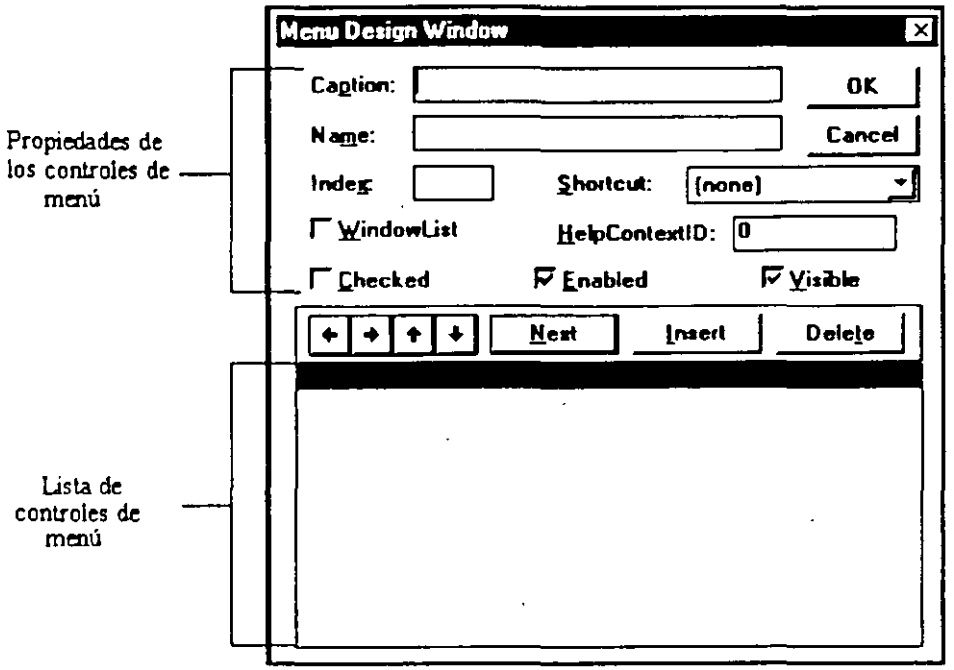


Figura 5.2

Todas las propiedades para menús disponibles durante el diseño se encuentran en esta ventana. Las propiedades más importantes son :

Name - Este es el nombre que se utiliza para hacer referencia al control menú en el código.

Caption - Este es el que aparece en el control.

La lista principal del control menú, muestra todas las opciones de los menús que se encuentran en la actual forma. Cuando se ingresa un elemento de menú en la caja Caption, ese elemento aparece en esta lista. Al seleccionar algún elemento de la lista, Visual Basic permite editar las propiedades de dicha opción.

La posición del control dentro de la lista determina si se trata del título del menú, de un elemento de menú, de un título de un submenú, o de un elemento de un submenú:

Un control que aparece pegado a la izquierda en la lista, representa un título de algún menú.

Un control que se encuentre indentado una vez, representa un elemento de menú.

Un control seguido de varios controles indentados en una posición más que el primero, representa el título de un submenú. Los controles que se encuentran indentados debajo de éste título, constituyen los elementos del submenú.

Un control que contiene un guión (-) como Caption, indica que en esa posición se insertará una barra separadora.

Para crear un menú en la ventana de diseño se pueden seguir estos pasos:

1. Seleccionar la forma.

2. Del menú Window, elegir el comando Menu Design.

3. En la caja de texto Caption, ingresar el texto para el título que aparecerá en la barra de menú.

4. En la caja de texto Name, ingresar el nombre que se utilizará para referenciar al menú en el código.

5. Utilizar la flecha a la derecha o a la izquierda para cambiar el nivel de indentación del control.

6. Establecer las propiedades del control, si se desea.

7. Seleccionar el botón Next para crear otro control.

8. Seleccionar OK para cerrar la ventana Menu Design una vez que se han creado todos los controles deseados

Escribiendo código para los controles de menús

Cuando el usuario selecciona un control de menú, ocurre un evento Click. Se requiere escribir un procedimiento para el evento Click para cada elemento del menú. Todos los elementos del menú, excepto las barras separadoras, reconocen el evento Click.

Visual Basic despliega un menú automáticamente cuando un título a sido seleccionado; por lo tanto, no es necesario escribir el código para mostrar un menú al hacer click sobre un título, a menos que se requiera ejecutar alguna otra acción, como deshabilitar algunos comandos del menú cuando éste es desplegado.

Creación de submenús

Cada menú que se crea puede incluirse en uno de los cuatros niveles de submenús. Un submenú se despliega a partir de otro menú para mostrar sus propios elementos. Se puede utilizar un submenú cuando :

- La barra de menú está llena.

- Una opción de menú es rara vez utilizada.

- Cuando se quiere enfatizar la relación entre una opción y otra.

Si existe espacio en la barra, sin embargo, es mejor crear un título adicional en lugar de un submenú. De esta forma todos los controles están visibles para el usuario cuando el menú se despliega. Es también una buena practica restringir el uso de submenús para que el usuario no se pierda navegando en la interface de la aplicación. (La mayoría de las aplicaciones utilizan un solo nivel de submenús).

En la ventana de Menu Design, cualquier elemento de un menú indentado debajo de un elemento que no es el título del menú constituye un control submenú. En general, los controles submenús pueden incluir elementos de submenú, separadores y títulos para submenús. El cuarto nivel de submenú puede incluir elementos de menú y separadores pero no títulos de submenús. La siguiente figura muestra un ejemplo de los submenús y de los títulos de los submenús.

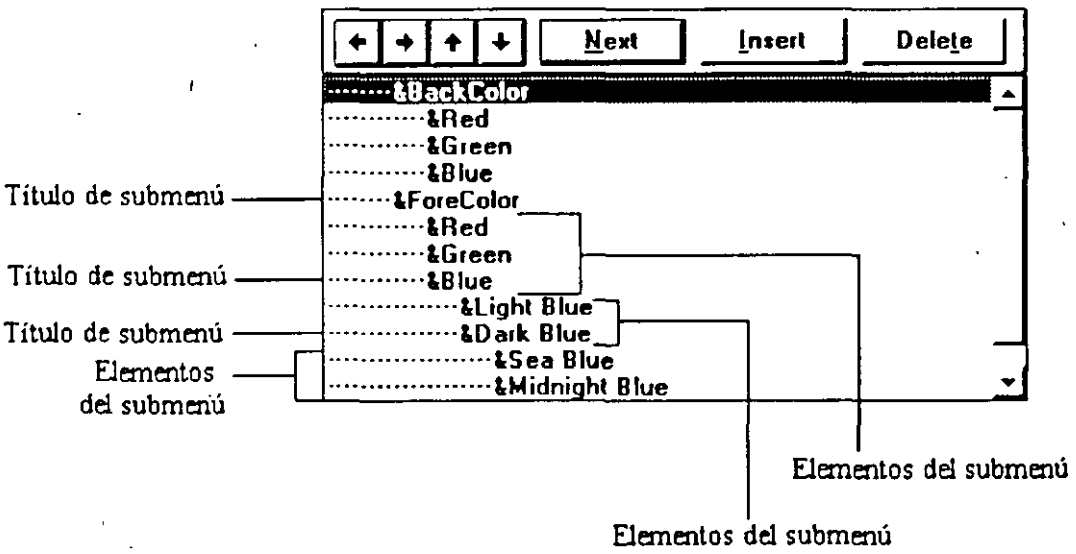


Figura 5.3

Para crear un submenú :

1. Crear el elemento del menú que se quiere como título para el submenú.

2. Crear el elemento que aparecerá en el nuevo submenú, e indentarlo seleccionando la flecha a la derecha.

5.3 Cambio de las Opciones de un Menú Durante la Ejecución

Control de los menús durante la ejecución

Los menús que se crean durante el diseño pueden responder de forma dinámica en condiciones de ejecución. Por ejemplo, si una acción de un menú es inapropiada en algún momento, se puede prevenir al usuario de utilizarla durante la ejecución si se deshabilita dicha opción.

Todos los controles de menús tienen la propiedad Enabled, y cuando esta propiedad se encuentra en **False**, el menú se deshabilita y no responde a las acciones del usuario. Un control de menú deshabilitado aparece en color gris.

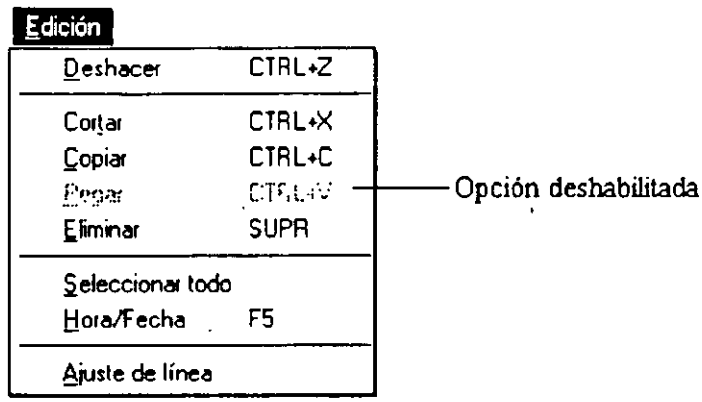


Figura 5.4

Por ejemplo, esta sentencia deshabilita la opción Close del menú File :

```
mnuFile(3).Enabled = False
```

Deshabilitar un título de un menú equivale a deshabilitar todo el menú, de tal forma que el usuario no puede accesar ninguna opción debido a que, para hacerlo, se debe hacer click primero en el título. Por ejemplo, la siguiente línea deshabilita el menú File completo :

```
mnuFile.Enabled = False
```

6. CONTROLES AVANZADOS

6.1 Interfaces con Archivos

En la mayoría de las aplicaciones para Windows es necesario almacenar los datos en disco. Con las opciones Save y Save As del menú File, podemos abrir una caja de diálogo que nos permita realizar esa acción. Para implementar esto en Visual Basic, es necesario crear una nueva forma llamada SaveForm, la cual se desplegará cuando se elija la opción Save File... del menú File de la aplicación Editor mediante el siguiente código:

```
Sub SaveItem_Click ()
        SaveForm.Show
End Sub
```

Se tiene que modificar la propiedad Caption para que diga Save File..., y salvarla en un archivo de nombre SaveForm.frm. Es necesario colocarle una caja de texto en la parte central y una etiqueta encima de esa caja que diga Save File As: (para indicarle al usuario lo que queremos que escriba), además, dos botones, uno de OK con la propiedad Default como True y otro llamado Cancel. Finalmente, debido a que ésta es una caja de diálogo, debemos deshabilitar los botones Min y Max.

La forma SaveForm debe lucir como en la siguiente figura:

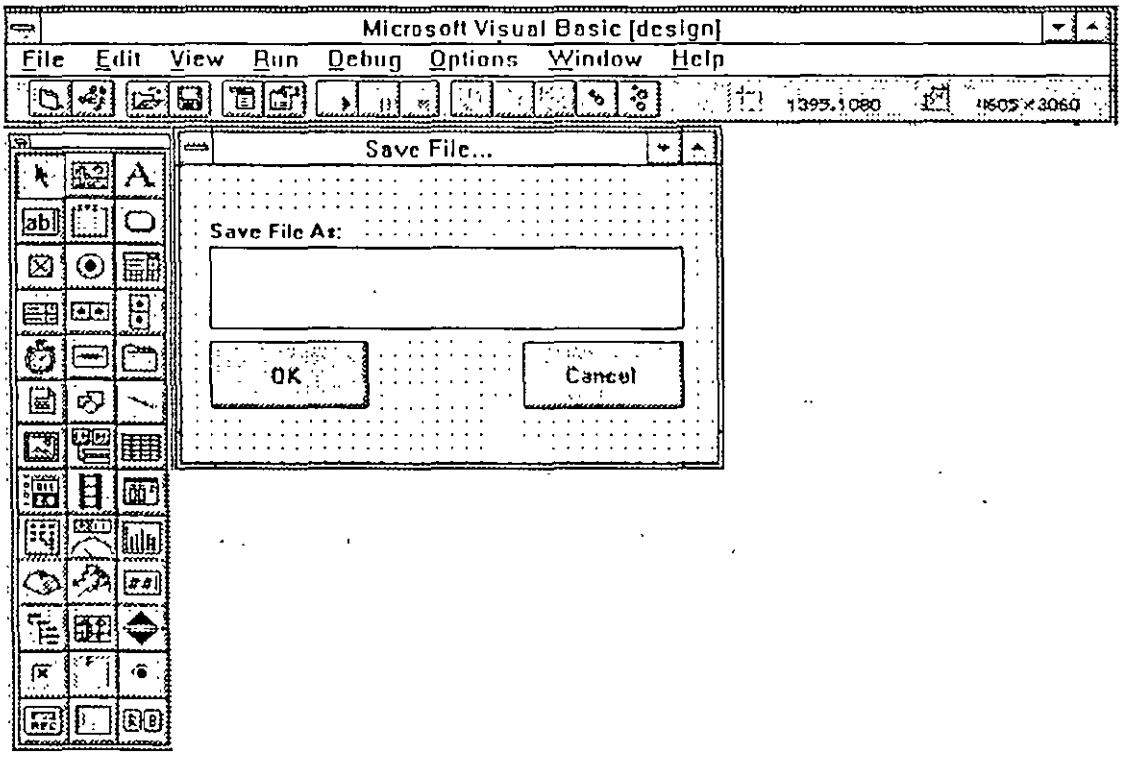


Figura 6.1

El procedimiento del botón Cancel es fácil, asi que es posible hacerlo primero. Si el usuario cancela la operación, solo se tiene que ocultar la forma SaveForm, por lo que colocaremos el siguiente código:

```
Sub _CancelButton_Click ()
    SaveForm.Hide
End Sub
```

El trabajo se realizará realmente cuando el usuario escriba el nombre del archivo y presione el botón OK. Al llegar a este punto del programa, debemos suponer que la caja de texto de la ventana contiene un nombre de archivo y que usted esta salvando el documento actual. Se involucran tres pasos en este proceso: la apertura del archivo (o creación, si es que no existe), la escritura de los datos hacia el archivo y cerrar el archivo. Las siguientes secciones describen estos pasos.

Apertura de Archivos

Para abrir o crear un archivo en Visual Basic simplemente utilice la sentencia Open. Existen cinco formas distintas de abrir archivos en Visual Basic, que son las siguientes:

Entrada Secuencial
Salida Secuencial
Adición Secuencial
Entrada/Salida Aleatoria
Entrada/Salida Binaria

Tipos de archivos en Visual Basic

Los primeros tres modos de archivos están relacionados con archivos secuenciales, estos son usualmente utilizados para archivos de texto, donde se escribe en el archivo desde el inicio hasta el final y se lee en la misma forma. El trabajar con archivos de texto es como utilizar audiocassetes, en donde, para oir alguna melodía al final, es necesario recorrer toda la cinta.

Si los archivos secuenciales son como audiocassetes, los archivos de acceso aleatorio son como CD's, en donde se puede elegir una melodia sin tener que recorrer las anteriores. La condición para trabajar con archivos de acceso aleatorio es que la información tiene que estar seccionada en registros para saber donde están exactamente los datos buscados y asi evitamos recorrer registros innecesarios. Debido a que en nuestro ejemplo estamos usando un editor de texto, nuestro archivo será secuencial.

El tercer tipo de archivos son los archivos binarios, en este caso, Visual Basic no interpreta todos los datos claramente. Por ejemplo, los archivos ejecutables (.EXE) son archivos binarios y deben ser tratados byte por byte.

Cada uno de estos tipos tiene sus propias sentencias de control dentro de Visual Basic. A continuación se muestra una tabla de las sentencias de manejo de archivos en Visual Basic.

Acceso	Sentencias comunes en Visual Basic
Secuencial	Open, Line Input #, Print #, Write #, Input$, Close
Aleatorio	Type...End Type, Open, Put #, Len, Close, Get #
Binario	Open, Get #, Put #, Close, Seek, Input$

Nuestro trabajo aquí es salvar el documento de la aplicación Editor. Esto se logrará abriendo un archivo secuencial. Existen tres formas de abrir archivos secuenciales: para Entrada, para Salida y para Adición. Un archivo se abre para Entrada si se desea leer algo de él, para Salida si desea escribir algo en él y para Anexión si desea añadir algo al final del archivo.

Si usted abre un archivo para salida secuencial, escribe una cadena en él y después escribe una segunda cadena, la segunda cadena irá inmediatamente después de la primera y asi sucesivamente. Para leerlas, tiene que cerrar el archivo y abrirlo para Entrada, entonces podrá leer los datos desde el inicio hasta el final. En los archivos que se abren para acceso aleatorio no existen tales restricciones, ya que se abren para Entrada y Salida. En general, la sintaxis de la sentencia Open es la siguiente:

Open fff$ [For mmm] [Access aaa] [lll] As [#] nnn% [Len = rrr%]

La siguiente es una lista de argumentos:

fff$	El nombre del archivo (incluyendo una ruta opcional).
mmm	Modo: puede ser Append, Binary, Input, Output o Random.
aaa	Acceso: puede ser Read, Write o Read Write.
lll	Lock: restringe el acceso de otras aplicaciones a este archivo como Shared, Lock Read, Lock Write, Lock Read Write.
nnn%	Número de archivo (1-255): el número de archivo que usará para referirse a este archivo de ahora en adelante.
rrr%	Largo del registro para archivos de acceso aleatorio o largo del tamaño del buffer que Visual Basic usará para archivos secuenciales.

En nuestro caso, el usuario desea escribir al nombre del archivo dado en FileNameBox.Text, asi que usted puede utilizar la siguiente sentencia para abrir el archivo:

Open FileNameBox.Text For Output As # 1

De hecho, este archivo puede no existir -el usuario puede querer crearlo. Esto es manejado automáticamente por la instrucción Open. Si el archivo no existe y usted esta

tratando de abrirlo para cualquier cosa excepto Input, Visual Basic creará el archivo por usted. Note que cuando usted abre un archivo para Output y escribe en él, el contenido original del archivo será destruido. Ahora ya podemos escribir la línea de código para el botón OK de la ventana SaveForm:

```
Sub OKButton_Click ()
        Open FileNameBox.Text For Output As # 1          'Abrir Archivo
End Sub
```

Como en Basic estándar, de ahora en adelante usted puede referirse a este archivo como file # 1 cuando quiera escribir en él o cerrarlo.

Escritura en Archivos

La forma usual de escribir a un archivo secuencial es utilizando las sentencias Print # o Write #, como sigue:

```
Print # nnn%, lista
Write # nnn%, lista
```

Aquí, nnn% es el número del archivo (1, en nuestro ejemplo), y lista es una lista de variables (incluyendo cadenas) que desee escribir al archivo. Las dos sentencias, Print # y Write #, son diferentes; Write # inserta comas entre los elementos separados en la lista mientras los escribe al archivo, coloca comillas encerrando a las cadenas e inserta una línea en blanco al final del archivo. Debido a que usted no quiere ninguna de las marcas antes mencionadas, utilice la sentencia Print #. De hecho, como en nuestro ejemplo solo queremos enviar una linea de texto -Form1 PadText.Text- al archivo, su sentencia Print# deberá aparecer como sigue:

```
Sub OKButton_Click ()
        Open FileNameBox.Text For Output As # 1          'Abrir Archivo
        Print # 1, Form PadText.Text                     'Escribe el documento
        :
        :
        Exit Sub
End Sub
```

Esto es todo lo que hay que hacer para escribir el texto en el archivo. Cerrar el archivo no es demasiado difícil, use la sentencia Close de la siguiente manera:

```
Sub OKButton_Click ()
        Open FileNameBox.Text For Output As # 1          'Abrir archivo
        Print # 1, Form.PadText.Text                     'Escribe el documento
        Close # 1                                        'Cierra archivo
        :
```

```
        Exit Sub
End Sub
```

Close # 1 cierra el archivo número 1, que es sobre el que estamos trabajando. Después de cerrar el archivo salga del procedimiento Sub con la sentencia Exit Sub. Hasta este punto, el archivo ha sido escrito exitosamente al disco.

Nota: Para cerrar todos los archivos a la vez, utilice la sentencia Close sin número, para que Visual Basic cierre todos los archivos que están abiertos.

Si todo ha salido bien, el paso final es ocultar la caja de diálogo Save File..., en la siguiente forma:

```
Sub OKButton_Click ()
        Open FileNameBox.Text For Output As # 1          'Abrir archivo
        Print # 1, Form.PadText.Text                     'Escribe el documento
        Close # 1                                        'Cierra archivo
        SaveForm.Hide
        Exit Sub
End Sub
```

Para probar los procesos anteriores, escriba algún texto en el Editor y sálvelo a algún archivo. El texto es almacenado como una cadena de caracteres contínua sin Retornos de Carro (a menos que el texto los contenga). De esta forma es posible almacenar archivos de texto en el disco. Para recuperar archivos previamente salvados, veamos las siguientes secciones.

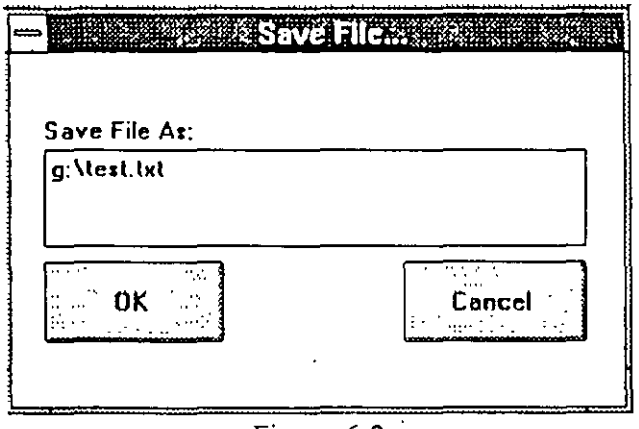


Figura 6.2

Uso de Controles de Archivos

El primer paso para leer el contenido de un archivo es obtener el nombre de ese archivo. Sin embargo, Esto no es simplemente preguntarle al usuario el nombre del archivo en una caja de texto, nuestra aplicación debe ser capaz de buscar el archivo en el disco (como otras aplicaciones similares en Windows) y permitirle al usuario seleccionar su

archivo que ya se encuentre ahí. Visual Basic provee tres controles especiales que permiten hacer eso exactamente: cajas de lista de discos, cajas de lista de directorios y cajas de lista de archivos.

Las herramientas para crear esos controles de archivos están en la parte baja de la Caja de Herramientas de Visual Basic, esos controles harán mucho trabajo por usted. Los controles buscarán discos y directorios automáticamente y usted será capaz de trabajar con múltiples propiedades asociadas con ellos.

Para emplear los controles en nuestra aplicación, es necesario crear una forma que sea útil para recuperar un archivo, a la que llamaremos LoadFile y estará almacenada como LoadFile.Frm. Primero es necesario conectarla al elemento del menú Load File..., haciendo un click en ese elemento para desplegar su sección de código y aparezca como sigue:

```
Sub LoadItem_Click ()
End Sub
```

Para desplegar la caja de diálogo Load File... (LoadForm), simplemente muéstrela de la siguiente manera:

```
Sub LoadItem_Click ()
    LoadForm.Show
End Sub
```

Para diseñar la forma LoadForm, utilice el elemento New Form del menú File de Visual Basic. Para crear la forma, asígnele el Nombre LoadForm, el título (Caption) Load File..., deshabilite los botones Max y Min, cambie la propiedad BorderStyle a double fixed, y coloque un botón OK y un Cancel.

A continuación, añada una caja de lista de drives haciendo doble click en la herramienta correspondiente en la Caja de Herramientas. Note que la herramienta caja de lista de drives es una caja de lista de cascada, que le ahorrará un poco de espacio. También necesitará una caja de lista de directorio y una caja de lista de archivos, añádalas y acomódelas en la forma que prefiera (Ver figura 6.3). Note que las cajas tienen los directorios y archivos actuales.

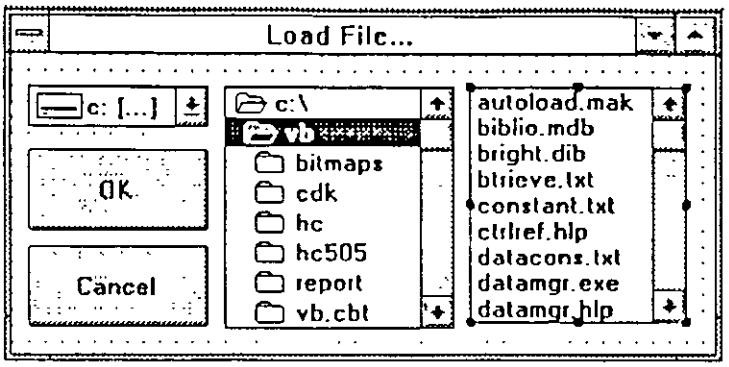


Figura 6.3

En esta forma, el usted será capaz de cargar un archivo existente combinando el drive, el directorio y el disco. Usted podrá usar la caja de lista de drives para elegir la unidad de disco, la caja de lista de directorio para especificar el directorio en esa unidad de disco y la caja de lista de archivos para indicar el archivo actual a abrir. Ese archivo puede ser abierto de dos formas: haciendo doble click en el nombre del archivo o seleccionándolo en la caja de lista de archivos y después haciendo click en el botón OK.

Como es usual, hacer el código del botón Cancel es fácil, sólo es necesario ocultar la caja de diálogo cuando se presione este botón, por ejemplo:

```
Sub CancelButton_Click ()
        LoadForm.Hide
End Sub
```

Ahora vamos a los controles del archivo. Hasta este punto, Las tres cajas de lista (drive, directorio y archivo) no se están comunicando una con otra; esto es, están mostrando información independiente para el directorio actual en el disco. Si usted ejecuta este programa y cambia de unidad de disco, las otras dos cajas no responderán a este cambio. Para hacer que se comuniquen necesitamos conocer otros eventos importantes de esas herramientas, las que se describen en las siguientes secciones.

Cajas de Lista de Drives

La caja de lista de drives es una lista de cascada. El drive actual es indicado en ella, cuando el usuario hace click en la flecha lateral, la lista se desenrolla mostrando cuales otros drives están disponibles para elegir. Cuando el usuario selecciona uno, ocurre un evento Change en esa caja de lista. Debido a que usted no ha cambiado el nombre de la herramienta caja de lista de drives, ésta todavía conserva el nombre de Drive1, asi que el procedimiento para este evento es Drive1_Change (). La propiedad de su caja Drive1 que contiene el drive es simplemente Drive1.Drive. El siguiente paso es pasar este nuevo drive a la caja de lista de directorios, que todavía tiene su nombre predefinido Dir1, para hacerlo, solo necesitamos pasar la propiedad Drive1.Drive a la propiedad Dir1.Path, como sigue:

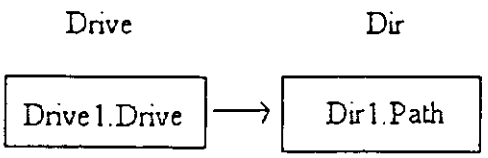


Figura 6.4

Usted puede hacerlo haciendo click en la caja de lista de drives, la cual traerá el procedimiento Drive1_Change ():

```
Sub Drive1_Change ()

End Sub
```

Solo asigne la propiedad Drive de Drive1 a la propiedad Path de Dir1, de la siguiente forma:

```
Sub Drive1_Change ()
        Dir1.Path = Drive1.Drive
End Sub
```

Esto es todo lo necesario para conectar las cajas de lista y de drives. El siguiente paso es conectar la caja de lista de directorios con la caja de lista de archivos, para que al seleccionar un nuevo directorio, podamos visualizar los archivos correspondientes a ese directorio.

Cajas de Lista de Directorios

La caja de lista de directorios despliega los directorios disponibles en cierto drive. Esta es una caja de lista simple, esto es, que siempre esta desplegada, no es una lista de cascada. El drive de trabajo es desplegado en la linea superior, con los directorios de ese drive debajo de él. Si existen mas directorios de los que caben en la ventana, se añadirá una barra de desplazamiento vertical a la derecha de esta caja de lista de directorios. El directorio actual aparece como un folder sombreado abierto; sus subdirectorios aparecen como folders cerrados justo debajo de él. Cuando el usuario cambia de directorio haciendo click en un nuevo directorio, ocurre un evento Dir1_Change (). En otras palabras, el único evento que nos importa es el evento Change, que ahora maneja los cambios en el drive y directorio.

Cuando tal evento ocurre, hacer click en la lista de directorio traerá la plantilla del procedimiento Sub en la ventana de código, como sigue:

```
Sub Dir1_Change ()

End Sub
```

Nuestra meta aquí es conectar cualquier cambio ocurrido en la propiedad Dir1.Path a la propiedad File1.Path, lo que se puede hacer de la siguiente forma:

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

De esta forma, cada vez que ocurra un cambio en el directorio de trabajo (o en el drive de trabajo), la caja de lista de archivos también se modificará. Los eventos importantes aquí son: Drive1_Change y Dir1_Change, y las propiedades importantes a ser transferidas son: Drive1.Drive ---> Dir1.Path y Dir1.Path ---> File1.Path. La forma en que usted lee el nombre del archivo que el usuario desea cargar es:

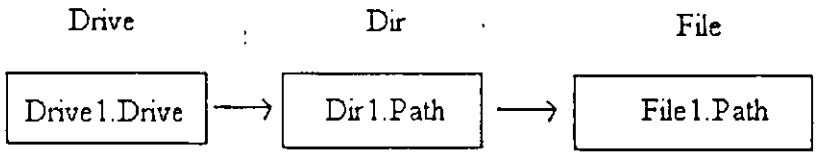


Figura 6.5

El siguiente paso es integrar la caja de lista de archivos al programa, cuando el usuario haga doble click en un nombre de archivo, usted deberá abrir ese archivo para cargarlo en el Editor. A continuación veremos las cajas de lista de archivos.

Cajas de Lista de Archivos

La caja de lista de archivos muestra los archivos del directorio actual. Al igual que la caja de lista de directorios, ésta es una caja de lista simple, que siempre esta desplegada. Si la lista es demasiado larga para la caja, aparecerá una barra de desplazamiento vertical.

Los archivos mostrados en la caja corresponden a dos propiedades: Path y Pattern. La propiedad Path contiene un nombre de ruta para el directorio cuyos archivos usted quiere desplegar. La propiedad Pattern contiene las especificaciones del archivo, tales como "*.Exe" (el patrón de default es "*.*").

Las cajas de lista de archivos pueden responder a dos eventos: click y doble click. En particular es importante añadir código al procedimiento File1_DblClick () debido a que el usuario seleccionará un nombre de archivo y cerrará la caja de diálogo Load File....Como se seguirá el mismo procedimiento si se elige el nombre del archivo y se presiona el botón OK, usted escribirá un solo procedimiento que se utilice en ambos eventos. Se hará eso escribiendo un procedimiento en un módulo de código y después llamando a ese procedimiento desde ambos eventos. Existe otra forma de hacer lo mismo, primero, haga click en la caja de lista de archivos y seleccione el procedimiento File1_DblClick () en la siguiente ventana de código, como a continuación se muestra:

```
Sub File1_DblClick ()

End Sub
```

Como deseamos hacer lo mismo que el procedimiento OKButton_Click (), pero éste ya esta hecho, simplemente llámelo de la siguiente forma:

```
Sub File1_DblClick ()
        OKButton_Click ()
End Sub
```

Esto es todo lo que hay que hacer aquí. Ahora escribiremos el procedimiento OKButton_Click (), que es el que realizará todo el trabajo. Abra la sección de código para el botón OK hasta ver lo siguiente:

```
Sub OKButton_Click ()

End Sub
```

Hasta este punto, el usuario esta tratando de abrir un archivo, el drive correcto esta en Drive1.Drive y la ruta correcta es Dir1.Path. Ahora necesitamos el nombre del archivo actual de la caja de lista de archivos. El nombre del archivo actual seleccionado esta guardado en la propiedad File Name de la caja de lista de archivos, para que usted tenga la especificación completa del archivo.

Una forma de abrir el archivo requerido es cambiar el nuevo drive (Drive1.Drive) con la sentencia de Basic ChDrive, cambiar la nueva ruta con (Dir1 Path) con la sentencia de Basic ChDir, y después abrir el archivo por si mismo (File1.FileName). No hay necesidad de cambiar el drive y directorio predeterminado a nivel de sistema operativo; sin embargo, puede ensamblar la especificación completa del archivo por usted mismo.

La propiedad Path de la caja de lista del directorio Dir1.Path, usualmente representa una ruta completa, incluyendo la letra del drive, por ejemplo: "C:\vb2\icons\arrows". Usted puede añadirla a su nombre de archivo, File1.FileName, si añade un backslash después de la ruta, como sigue:

```
Filename$ = Dir1.Path + "\" + File1.FileName
```

Aquí, Filename$ es una variable local que usted puede usar en el procedimiento OKButton_Click(). Sin embargo, esto no es lo suficientemente bueno, si sucede que usted tiene el directorio raíz de un drive como d:\, entonces Dir1.Path sería "d:\"; si el nombre del archivo fuera Novel.Txt, entonces Filename$ sería igual a Dir1.Path + "\" + File1.FileName, esto es "d:\\Novel.Txt". En otras palabras, tendría un backslash de más. Para evitar esto, chequemos el último caracter de Dir1.Path, si éste ya es un backslash, no tendremos que añadirlo, como sigue:

```
If (Right$(Dir1.Path,1) = "\") Then
        Filename$ = Dir1.Path + File1.FileName
Else
        Filename$ = Dir1.Path + "\" + File1.FileName
End If
```

Ahora Filename$ tiene la especificación completa del archivo que supuestamente esta usted abriendo y puede abrirlo con la especificación Open. Debido a que usted esta usando archivos de acceso secuencial, y quiere leer el archivo, puede abrirlo para Input de la siguiente forma:

```
If (Right$(Dir1.Path,1) = "\") Then
        Filename$ = Dir1.Path + File1.FileName
Else
        Filename$ = Dir1.Path + "\" + File1.FileName
End If
Open Filename$ For Input As # 1
```

Hasta este punto el archivo ha sido seleccionado y esta abierto, el siguiente paso es leer los datos.

Lectura de Archivos de Visual Basic

Las formas estándar de leer archivos secuenciales en Visual Basic son Input #, Line Input #, e Input$. Usted las podrá usar de la siguiente forma:

```
Input # nnn%, expresionlist
Line Input # nnn%, stringvariable
Input$ (bbb%, [#] nnn%)
```

Aquí, nnn% es el número del archivo (1 para nuestro ejercicio), bbb% es el número de bytes por leer, stringvariable es el nombre de una variable cadena en la que se van a colocar los datos y expresionlist es una lista de expresiones donde serán colocados los datos.

Por ejemplo, si usted utiliza Input # para llenar Form1.PadText.Text, ésta se podría ver como sigue: Input # 1 Form1.PadText.Text. El problema con Input # sin embargo, es que éste espera que los elementos en el archivo estén separados por comas, espacios u otros fines de línea.

Similarmente, la función Line Input # lee cadenas de los archivos hasta que encuentra un retorno de carro (CR) y entonces termina. Esto significa que usted tendría que leer cada línea del archivo separadamente. Una forma de hacerlo es la siguiente:

```
Do Until EOF(1)
        Line Input # 1, Dummy$
        Form1.PadText.Text = Form1.PadText.Text + Dummy$ + Chr(13) + Chr(10)
Loop
```

Aquí, usted estará leyendo líneas del archivo hasta que llegue al final del archivo. Además, está añadiendo retornos de carro (CR) e inicios de línea (LF) cada vez que lee una nueva línea.

Una mejor opción es la función Input$, que esta hecha especialmente para leer cadenas de caracteres que no contienen CR ni LF. Sin embargo, para utilizar esta función, usted tiene que especificar el número exacto de bytes que quiere leer, cuando lo haga, Input$ devuelve una cadena (que usted puede colocar en Form1.PadText.Text). El número de bytes que usted quiere leer e simplemente el tamaño del archivo en bytes; usted puede utilizar otra función de archivo, LOF(), para que obtenga ese dato por usted. Como EOF(), LOF() toma un número de archivo como argumento y devuelve el largo en bytes del archivo (el archivo debe ser abierto por LOF para trabajar), asi usted puede leer el archivo en un solo paso, como sigue:

```
Form1.PadText.Text = Input$(LOF(1), # 1)
```

De hecho, debido a que usted especifica el número de bytes a leer, también puede usar esta sentencia para leer datos de archivos binarios. Usted puede añadir su sentencia Input$ a OKButton_Click() de esta forma:

```
Sub OKButton_Click()
        If (Right$(Dir1.Path,1) = "\") Then
                Filename$ = Dir1.Path + File1.FileName
        Else
                Filename$ = Dir1.Path + "\" + File1.FileName
        End If
        Open Filename$ For Input As # 1
        Form1.PadText.Text = Input$(LOF(1), # 1)

        Exit Sub
End Sub
```

Todo lo que resta ahora es cerrar el archivo y ocultar la caja de diálogo (LoadForm) al mismo tiempo. Para hacerlo añada el siguiente código como se muestra a continuación:

```
Sub OKButton_Click()
        If (Right$(Dir1.Path,1) = "\") Then
                Filename$ = Dir1.Path + File1.FileName
        Else
                Filename$ = Dir1.Path + "\" + File1.FileName
```

```
        End If
        Open Filename$ For Input As # 1
        Form1.PadText.Text = Input$(LOF(1), # 1)
        Close # 1
        LoadForm.Hide
        Exit Sub
End Sub
```

Ahora la caja de diálogo de Load File... está completa. Para usarla, simplemente inicie el programa y seleccione Load File... en el menú File. La caja de diálogo Load File... se abrirá como se muestra en la siguiente figura. Como puede ver, la caja de lista de archivos presenta los nombres de archivos en orden alfabético. Para abrir uno de ellos, solo haga doble click sobre él o selecciónelo y haga click en el botón OK. Cuando lo haga el archivo aparecerá en la ventana del editor y podrá salvarlo a disco con la opción Save As... del menú File.

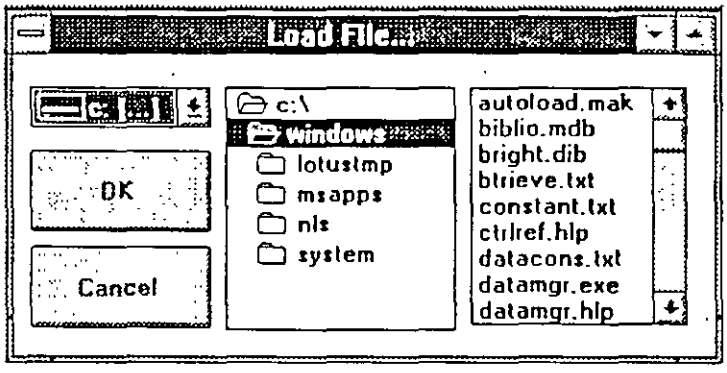


Figura 6.6

6.2 Grid

Otro ejemplo de controles avanzados es la utilización de un programa de hoja de cálculo. La creación de una hoja de cálculo es fácil por medio del control Grid, el cual está contenido en el archivo Grid.Vbx y se carga automáticamente dentro de Visual Basic. Como ejemplo, iniciar un nuevo programa y darle el nombre de Spread.Mak. Salvar la forma como Spread.Frm. Posteriormente dar un click a la herramienta Grid (la que contiene un pequeño emparrillado en su botón) y dibujar un emparrillado cubriendo una buena parte de la forma. Utilizar la ventana de propiedades para poder asignar el número de renglones y columnas de este nuevo emparillado (grid), tanto para renglones como para columnas dar 7. Tambien dar el *caption* Spreadsheet. El resultado será semejante a la figura siguiente.

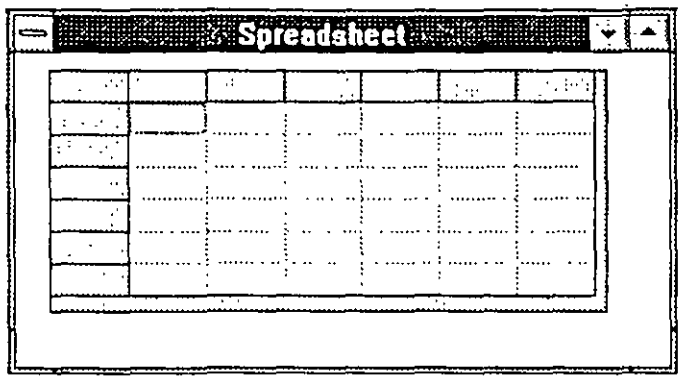


Figura 6.7

Ahora se puede hacer referencia a una celda individual en la hoja de cálculo haciendo una celda la *celda actual*. Unicamente se necesitan establecer las propiedades de Grid1.Row y Grid1.Col para hacerlo. Supóngase que se quiere llevar el seguimiento de los gastos de la siguiente forma:

$400	Rent
$100	Food
$80	Car
$15	Phone
$10	Gas
?	Total

Para introducir la información dentro de la hoja se utiliza el evento Form_Load (). Lo que se puede hacer es etiquetar los renglones con números y las columnas con letras (de la misma forma que en las hojas de cálculo), además de poner el concepto de cada gasto. El proceso se puede realizar con el siguiente programa:

```
Sub Form_Load ()
        Static Gastos(6) As String
        Gastos(1) = "Rent"
        Gastos(2) = "Food"
        Gastos(3) = "Car"
        Gastos(4) = "Phone"
        Gastos(5) = "Gas"
        Gastos(6) = "Total"
        Grid1.Row = 1
        For loop_index% = 1 To 6
                Grid1.Col = 0
                Grid1.Row = loop_index%
                Grid1.Text = Str$(loop_index%)
                Grid1.Col = 2
                Grid1.Text = Gastos(loop_index%)
        Next loop_index%
        Grid1.Col = 1
```

```
        Grid1.Row = 0
        For loop_index% = 1 To 6
                Grid1.Col = loop_index%
                Grid1.Text = Chr$(Asc("A") - 1 + loop_index%)
        Next loop_index%
        Grid1.Row = 1
        Grid1.Col = 1
End Sub
```

Dando como resultado la siguiente figura:

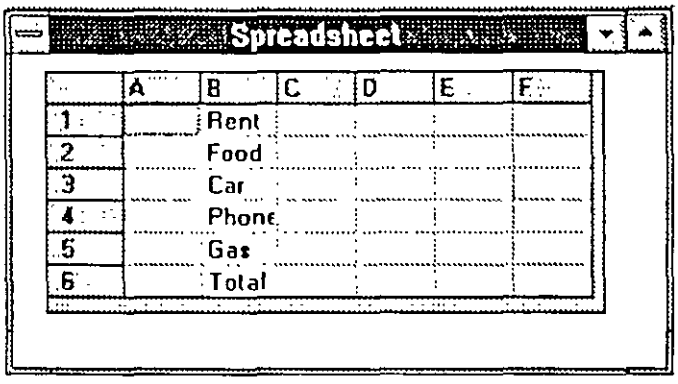


Figura 6.8

Ahora se pueden leer los gastos de la misma forma como introducir datos a una celda (a excepción de la celda marcada como Total), incluso se puede tener la suma de los gastos en la celda Total. Esto se logra con el evento Grid1's KeyPress(); siempre que un evento KeyPress() ocurre se puede colocar el dato en la celda adecuada y actualizar el resultado de manera automática. Primero adicionar los datos digitados a la celda actual, de la siguiente forma:

```
Sub Grid1_KeyPress (KeyAscii As Integer)
        Grid1.Text = Grid1.Text + Chr$(KeyAscii)

End Sub
```

Al digitar los valores, estos aparecen en la celda actual pudiendo ser cambiados utilizando el mouse o con las teclas de desplazamieto. Para actualizar el Total, primero salvar los valores de Row y Col de la celda actual de la siguiente manera:

```
Sub Grid1_KeyPress (KeyAscii As Integer)
        Grid1.Text = Grid1.Text + Chr$(KeyAscii)
        OldRow = Grid1.Row
        OldCol = Grid1.Col
```

```
End Sub
```

Posteriormente se adicionarán todos los números de la columna 1, realizando un ciclo de la siguiente forma:

```
Sub Grid1_KeyPress (KeyAscii As Integer)
        Grid1.Text = Grid1.Text + Chr$(KeyAscii)
        OldRow = Grid1.Row
        OldCol = Grid1.Col
        Grid1.Col = 1                    'Adiciona números en la primera columna
        Grid1.Row = 0
        Sum% = 0
        For row_index% = 1 To 5
                Grid1.Row = row_index%
                Sum% = Sum% + Val(Grid1.Text)
        Next row_index%

End Sub
```

Lo único qre resta es introducir el resultado de Sum% en la celda Total y restablecer la celda original como actual. Lo anterior se realiza de la siguiente forma:

```
Sub Grid1_KeyPress (KeyAscii As Integer)
        Grid1.Text = Grid1.Text + Chr$(KeyAscii)
        OldRow = Grid1.Row
        OldCol = Grid1.Col
        Grid1.Col = 1                    'Adiciona números en la primera columna
        Grid1.Row = 0
        Sum% = 0
        For row_index% = 1 To 5
                Grid1.Row = row_index%
                Sum% = Sum% + Val(Grid1.Text)
        Next row_index%
        Grid1.Row = 6
        Grid1.Text = Str$(Sum%)
        Grid1.Row = OldRow
        Grid1.Col = OldCol
End Sub
```

Con el código anterior la hoja de cálculo trabaja correctamente. Cuando un usuario digite los valores en la primera columna, se van actualizando los totales de manera automática. El programa resultante aparece en la siguiente figura. Como se puede ver, este puede ser fácilmente adaptable para otras aplicaciones del tipo hoja de cálculo. A continuación aparece el código completo de la aplicación:

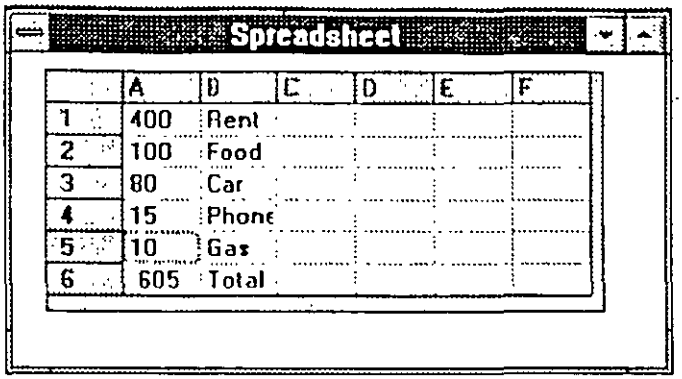


Figura 6.9

```
Form Form1 ----------------------------------------
        Caption = "Spreadsheet"

Grid Grid1
        Cols = 7
        Rows = 7
        ScrollBars = 0  ' Ninguna

Sub Form_Load ()
        Static Gastos(6) As String
        Gastos(1) = "Rent"
        Gastos(2) = "Food"
        Gastos(3) = "Car"
        Gastos(4) = "Phone"
        Gastos(5) = "Gas"
        Gastos(6) = "Total"
        Grid1.Row = 1
        For loop_index% = 1 To 6
                Grid1 Col = 0
                Grid1 Row = loop_index%
                Grid1.Text = Str$(loop_index%)
                Grid1.Col = 2
                Grid1.Text = Gastos(loop_index%)
        Next loop_index%
        Grid1.Col =  1
        Grid1.Row = 0
        For loop_index% = 1 To 6
                Grid1.Col = loop_index%
                Grid1.Text = Chr$(Asc("A") - 1 + loop_index%)
        Next loop_index%
        Grid1.Row = 1
        Grid1.Col = 1
End Sub
```

```
Sub Grid1_KeyPress (KeyAscii As Integer)
        Grid1.Text = Grid1.Text + Chr$(KeyAscii)
        OldRow = Grid1.Row
        OldCol = Grid1.Col
        Grid1.Col = 1                    'Adiciona números en la primera columna
        Grid1.Row = 0
        Sum% = 0
        For row_index% = 1 To 5
                Grid1.Row = row_index%
                Sum% = Sum% + Val(Grid1.Text)
        Next row_index%
        Grid1.Row = 6
        Grid1.Text = Str$(Sum%)
        Grid1.Row = OldRow
        Grid1.Col = OldCol
End Sub
```

6.3 Timer

Un timer permite producir un evento específico, llamado evento Timer en un predeterminado intervalo de tiempo. El símbolo de control es un pequeño reloj, ya sea dentro de la forma (aunque no sea visible en tiempo de ejecución) o en la caja de herramientas. Al dar doble click sobre la herramienta Timer se coloca el timer aproximadamente en la misma posición que se muestra en la siguiente figura. Por default toma el nombre de Timer1.

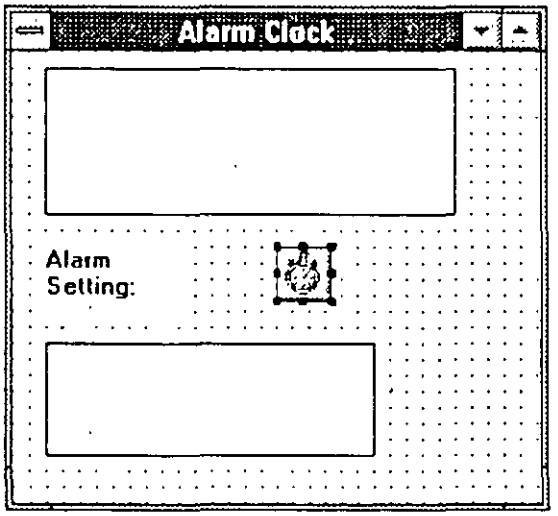


Figura 6.10

El siguiente paso es activar la propiedad de Intervalo del timer para indicar cada cuando el evento del timer ocurre. Para establecerlo seleccionar el timer, y abrir la lista de

propiedades en la ventana propiedades. Seleccionar la propiedad Interval y moverse a la caja de edición para establecer el tiempo, el cual está dado en milisegundos. Por ejemplo, para actualizar el reloj una vez por segundo, especificar 1000 en la propiedad Interval del timer.

Una vez establecido el intervalo, se procede a escribir el procedimiento que será ejecutado cada vez que se cumpla el tiempo. Dar doble click sobre el timer para abrir la ventana de código. El procedimiento que se despliega es el siguiente:

```
Sub Timer1_Timer ()

End Sub
```

Un ejemplo puede ser el checar una hora determinada, de tal modo que al ser mas tarde suene un beep. La hora puede ser dada por medio de la función Time$ y la hora de alarma por medio de una constante o incluso por medio de una variable, de tal modo que se puede tener Time$ igual o mayor a constante o variable. Si la hora es mayor, entonces sonará la alarma. El código queda de la siguiente forma para empezar a sonar cada segundo a partir de las 9 de la noche:

```
Sub Timer1_Timer ()
        If (Time$ >= "21:00:00") Then
                Beep
        End If
End Sub
```

7. Interface con ODBS

7.1 El acceso a las bases de datos utilizando Visual Basic

Existen tres tipos de bases de datos que se pueden accesar desde Visual Basic (fig 7.1):

- Bases nativas de Microsoft Access. Estas bases de datos son accesadas directamente por Visual Basic.
- Bases de datos indexadas con método de acceso secuencial (ISAM). Por ejemplo bases de datos de Dbase, Paradox y Btrieve. Visual Basic obtiene estas tablas a través de drivers instalados por el usuario que ligan a Visual Basic a bases de datos especificas.
- Bases de datos en sistemas abiertos (ODBC). Estas incluyen sistemas de administración de bases de datos Cliente-Servidor tales como Microsoft SQL Server y Oracle. Visual Basic obtiene estas bases a través de los drivers ODBC correspondientes.

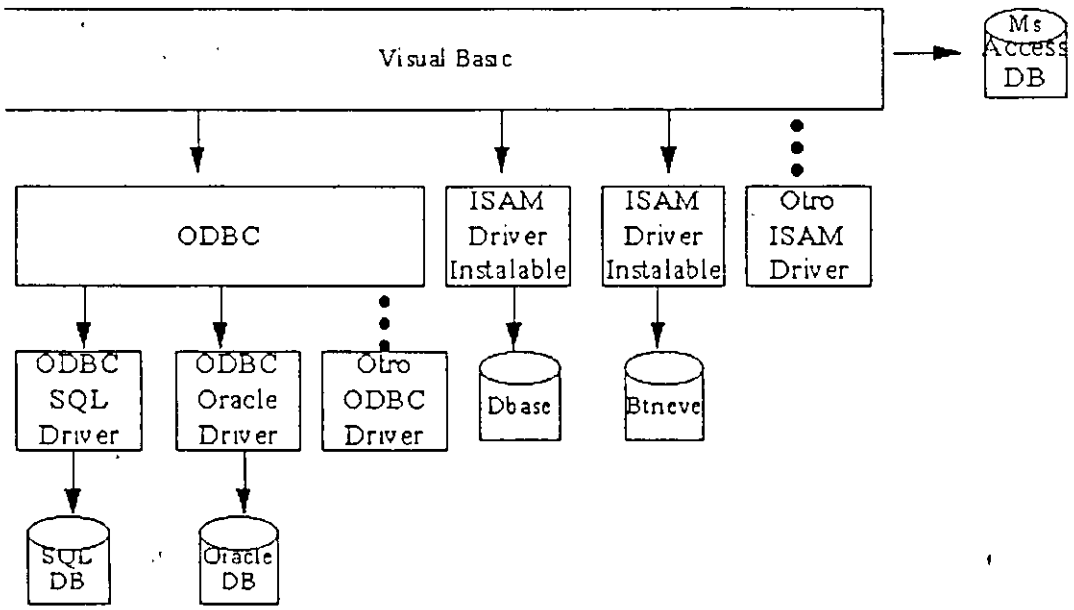


fig. 7.1

Si la aplicación espera abrir una base de datos Access que ha sido asegurada que tiene definida un esquema de permisos, es necesario añadir una linea en el archivo VB.INI o en el archivo APP.INI para indicar la ubicación del archivo .MDA, por ejemplo:

[options]

SystemDB=C:\ACCESS\SYSTEM.MDA

Además la aplicación debe de incluir la sentencia de **SetDefaultWorkspace** para indicar un nombre de usuario y un password apropiado. Esta sentencia debe de ir antes de cualquier otra sentencia que haga empezar a la inicialización del motor de bases de datos de Visual Basic .

```
SetDefaultWorkspace "myusername", "mypassword"
```

Bases de Datos Soportadas por Visual Basic

Los tres controladores ISAM de bases de datos provistos con Visual Basic (XBS110.DLL, PDX110.DLL y BTRV110.DLL), le permiten conectarse con archivos de dBASE III y IV, FoxPro 2.x, Paradox 3.x y Btrieve 5.1+. El mecanismo de bases de datos de Access y los tres controladores de ISAM le permiten emplear archivos que son usados por lo menos por el 90% de las aplicaciones de bases de datos de escritorio actuales. Cada uno de esos tipos de bases de datos foraneos tiene conexión con métodos que difieren.

Bases de Datos de dBASE y FoxPro

Para utilizar bases de datos de dBASE y FoxPro con el mecanismo de bases de datos de Access, sus archivos VB.INI o APPNAME.INI deben contener al menos alguna de las entradas que identifiquen la ruta y el nombre del controlador ISAM para el tipo de bases de datos:

```
[Installable ISAMs]
dBASE III=d:\path\xbs110.dll
dBASEIV=d:\path\xbs110.dll
FoxPro 2.0= d:\path\xbs110.dll
FoxPro 2.5= d:\path\xbs110.dll
```

Los valores del nombre de elemento que preceden al signo igual en el ejemplo anterior, corresponden exactamente al valor strDBType (sin el punto y coma) que debe ser incluído en la cedena de conexión de su sentencia OpenDatabase().

Para abrir una base de datos de dBASE III en modo compartido para operaciones de lectura-escritura con los archivos .DBF y .NDX que constituyen la base de datos en su directorio D:\DBASE, use las siguientes sentencias:

```
Dim dbXBase As Database
Dim strDBPath As String
Dim strDBType As String
strDBPath = "d:\dbase"
strDBType = "dBASE III"
Set dbXbase = OpenDatabase(strDBPath, False, False, strDBType)
```

Una vez que usted ha abierto la base de datos dbXBase puede obtener y establecer los valores de las propiedades y aplicar cualquiera de los métodos de acceso a datos que son aplicables a las bases de datos de Access.

Bases de Datos de Paradox

Para utilizar bases de datos de dBASE y FoxPro con el mecanismo de bases de datos de Access, sus archivos VB.INI o APPNAME.INI deben contener al menos alguna de las entradas que identifiquen la ruta y el nombre del controlador ISAM para el tipo de bases de datos de Paradox:

```
[Installable ISAMs]
Paradox 3.x=d:\path\pdx110.dll
```

Para abrir una base de datos de Paradox 3.x en modo compartido para operaciones de lectura-escritura con los archivos en su directorio D:\PARADOX, use las siguientes sentencias:

```
Dim dbPdox As Database
Dim strDBPath As String
Dim strDBType As String
strDBPath = "d:\paradox"
strDBType = "Paradox"
Set dbXbase = OpenDatabase(strDBPath, False, False, strDBType)
```

Paradox aporta la especificación del campo llave primaria para todas las tablas de Paradox que usted genera. Paradox automáticamente crea un índice único en el campo llave primaria de la tabla El nombre del archivo índice de llave primaria para FILENAME.DB es FILENAME.PX. SI su tabla Paradox fue creada con un índice de llave primaria, el archivo índice debe estar localizado en el directorio de su base de datos. Si éste archivo índice no esta o está localizado en otro directorio, no será posible abrir la tabla.

Bases de Datos Btrieve

El abrir una base de datos Novell Btrieve es un proceso mas complejo que abrir una base de datos de xBASE o Paradox, por lo que no nos enfocaremos a bases de datos de este tipo en este manual; sin embargo, a continuación se mencionan los requerimientos para utilizar las bases de datos de Btrieve con Visual Basic:

- Debe tener una copia de la librería WBTRCALL.DLL de Btrieve para Windows localizada en su directorio \WINDOWS o \WINDOWS\SYSTEM. WBTRCALL.DLL no es provista con Visual Basic.

Un archivo de llaves (DATABASE.LDB). Cuando usted abre un archivo de bases de datos de Access por primera vez, la base de datos de Access crea un archivo de llaves con el mismo nombre que el archivo de bases de datos, pero con la extensión .LDB, éste archivo contiene referencias binarias a las páginas cerradas por su aplicación y otros usuarios del archivo.

Un archivo SYSTEM.MDA. Las bases de datos aseguradas de Access requieren este archivo, éste archivo contiene los nombres de los usuarios y grupos de usuarios en la tabla del sistema MSysAccounts.

Los desarrolladores profesionales de Access usualmente crean dos tablas para cada aplicación. Una base de datos (la base de datos tabla) contiene los datos en tablas de Access y la otra base de datos (la base de datos aplicación) contiene otros objetos de base de datos que usted puede crear con Access 1.x: consultas, formas, reportes, macros y módulos (objetos aplicación). Las tablas en la base de datos Tabla están conectadas a la base de datos Aplicación. Este método permite a los desarrolladores cualquiera o todos los objetos de la aplicación simplemente copiando una nueva base de datos aplicación sobre la ya existente. La alternativa es un laborioso proceso de borrado y después importación de los objetos de la aplicación actualizados en una base de datos sencilla.

Cuando usted crea un nuevo archivo .MDB, el mecanismo de bases de datos de Access generalmente reserva 64K de espacio en disco. La mayoría de este espacio esta vacío cuando se genera la base de datos. El contenido inicial de las bases de datos es una colección de tablas del sistema. Una vez que usted llena el espacio vacío restante con datos en las tablas, Access automáticamente expande el espacio en bloques de 2K, el tamaño de un cluster de un disco duro formateado convencionalmente para acomodar 2K nuevos de datos en páginas de índices. A diferencia de muchas bases de datos cliente-servidor, usted no necesita reservar espacios en disco adicionales para aumentar los requerimientos de espacio en disco

- Para cada base de datos de Btrieve que abra, necesita dos diccionarios de datos de Btrieve, FILE.DDF y FIELD.DDF, en el mismo directorio que el archivo de bases de datos FILENAME.DAT de Btrieve especificado en FILE.DDF.

- Su archivo de Btrieve debe ser de la versión 1.5 o posterior.

- Si esta utilizando Btrieve en un ambiente multiusuario, necesitará el archivo de transacciones BTRIEVE.TRN para los demás usuarios del archivo.

- No es posible mantener índices de Btrieve que tengan los establecidos los atributos Manual o Null. El mecanismo de bases de datos de Access no soporta valores de tipo Null en índices.

- Al igual que otras bases de datos foráneas, necesitará añadir la ruta y el nombre del controlador ISAM para la bases de datos en la sección [Installable ISAMs] de sus archivos de inicialización VB.INI o APPNAME.INI .

7.2 Arquitectura de las Bases de Datos de Access

La mayoría de las aplicaciones complejas de Windows usan una estructura de capas comprimida en un archivo ejecutable (.EXE) y una o más librerías dinámicas ligadas (DLL's) que la aplicación llama cuando las necesita.

Los archivos ejecutables de Visual Basic que usted "compila" contienen el equivalente de código objeto de Visual Basic para las formas y código fuente de su aplicación. La principal diferencia entre librerías DOS y Windows es que el código objeto en librerías para las aplicaciones DOS se convierte en una parte permanente del archivo .EXE (llamadas ligas estáticas), mientras que las DLLs de Windows son ligadas en tiempo de ejecución (llamadas librerías dinámicas). Una de las ventajas de las ligas dinámicas es que una simple .DLL, tal como VBRUN300.DLL, puede servir a todas las aplicaciones de Visual Basic en su computadora. Si usted ejecuta mas de un aaplicación de Visual Basic al mismo tiempo, Windows crea instancias adicionales de VBRUN300.DLL en memoria para servir a las otras aplicaciones.

La figura 7.2 ilustra la estructura de capas de una aplicación de base de datos de Visual Basic. La siguiente lista explica la lista de eventos que ocurren cuando su aplicación llama auna operación deacceso de datos,usando la terminología de la figura 7.2:

- El archivo .EXE de su aplicación envía un requerimiento a VBRUN300.DLL para realizar una operación en un objeto de acceso a datos.

- VBRUN300.DLL pasa el requerimiento de acceso a datos al DLL de Soporte de Acceso a Datos, VBDB300.DLL.

- VBDB300.DLL traduce el requerimiento a llamadas a función que son compatibles con las llamadas a funciones de la librería JET Engine, MSAJT110.DLL, ésta determina el tipo de base de datos que corresponde al objeto, abre una instancia del controlador de base de datos apropiado y pasa la llamada al controlador.El controlador devuelve el dato (suponiendo que la petición fue para alimentar valores contenidos en los campos de la tabla) respaldando la cadena de DLLs a su aplicación.

- Si su aplicación utiliza en ODBC API, otra capa (el ODBC.DLL) es interpuesto entre el DLL JET Engine y el controlador ODBC que se conecta a la base de datos.

Tres de los elementos mostrados en la figura 7.2 son opcionales para las aplicaciones de bases de datos de Visual Basic, ésos son descritos en la siguiente lista:

- La aplicación ODBC Administrator provista con Visual Basic viene en dos versiones: una aplicación stand-alone, ODBCADM.EXE y ODBCINST.DLL, la que está diseñada para ser añadida al panel de control.

- Si su base de datos de dBASE o FoxPro incluye archivos de índices, necesitará un archivo .INF en su directorio de la base de datos que especifique el índice para cada tabla.

- Access 1.x requiere un archivo SYSTEM.MDA que contiene información acerca de los grupos de usuarios de las bases de datos de Access. Siusted aplica cualquiera de las opciones de seguridad de Access 1.x a un archivo de base de datos de Access, un archivo SYSTEM.MDA sencillose requiere para el mecanismo de base de datos de Access para abrir el archivo.Si usted no ha establecido seguridad en el acceso a las bases de datos con Access 1.x, no necesitará SYSTEM.MDA. En este caso, todos los usuarios son identificados como Admin (un miembro del grupo Admins) y tienen un password (Null).

La siguiente figura nos muestra la estructura de los manejadores de bases de datos respecto a Visual Basic:

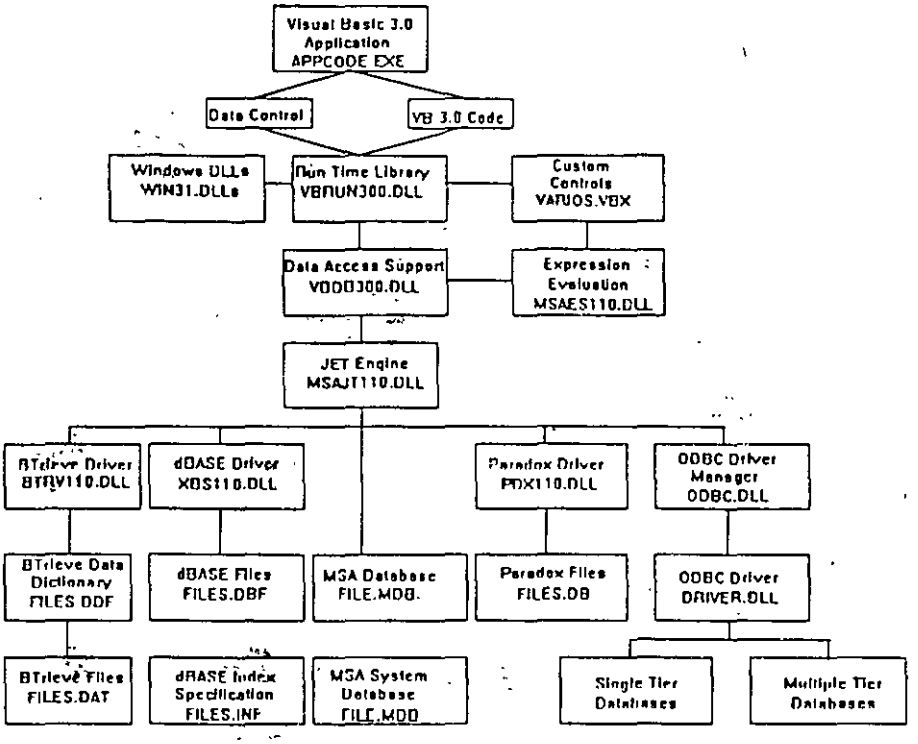


Fig. 7.2

Uso de Bases de Datos de Access con Visual Basic

Visual Basic 3.0 está fuertemente orientado hacia las bases de datos .MDB de Access; éste es el tipo de bases de datos "nativo" de Visual Basic .Usted no necesita un controlador extra para crear objetos Database de Access. A diferencia de Access 1.x, que solo soporta archivos .MDB como colecciones de tablas o bases nativas, Visual Basic 3.0 ocupa las basesde datos por igual. A excepción de los conjuntos de características específicas ofrecidos por los controladores ISAM individuales para otras bases de datos, los objetos Database creados por Visual Basic tienen un conjunto común de colecciones y comparten muchas propiedades y métodos. Access 1.x requiere que usted conecte tablas de bases de datos externas, incluyendo tablas en bases de datos conectadas con ODBC API, a una base de datos de Access y que usted tenga activa solo una base de datos .MDB a la vez. Visual Basic no impone esas limitantes.

La estructura de los archivos de Bases de Datos de Access

Además del archivo DATABASE.MDB, las bases de datos de Access tienen los siguientes archivos opcionales: