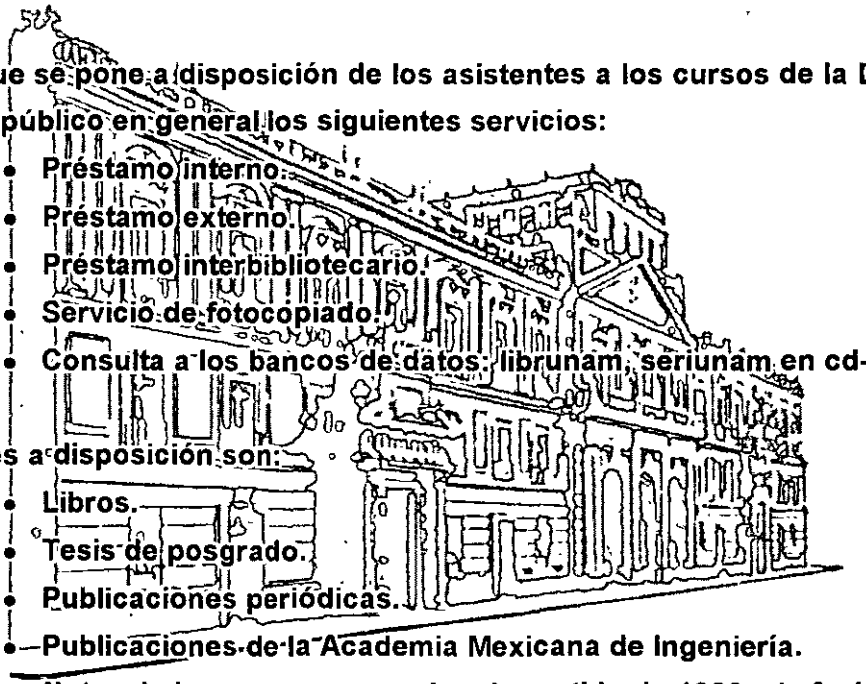
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN
" ING. BRUNO MASCANZONI "

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



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

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

Los materiales a disposición son:

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

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

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

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

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

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

A LOS ASISTENTES A LOS CURSOS

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

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

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

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

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

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

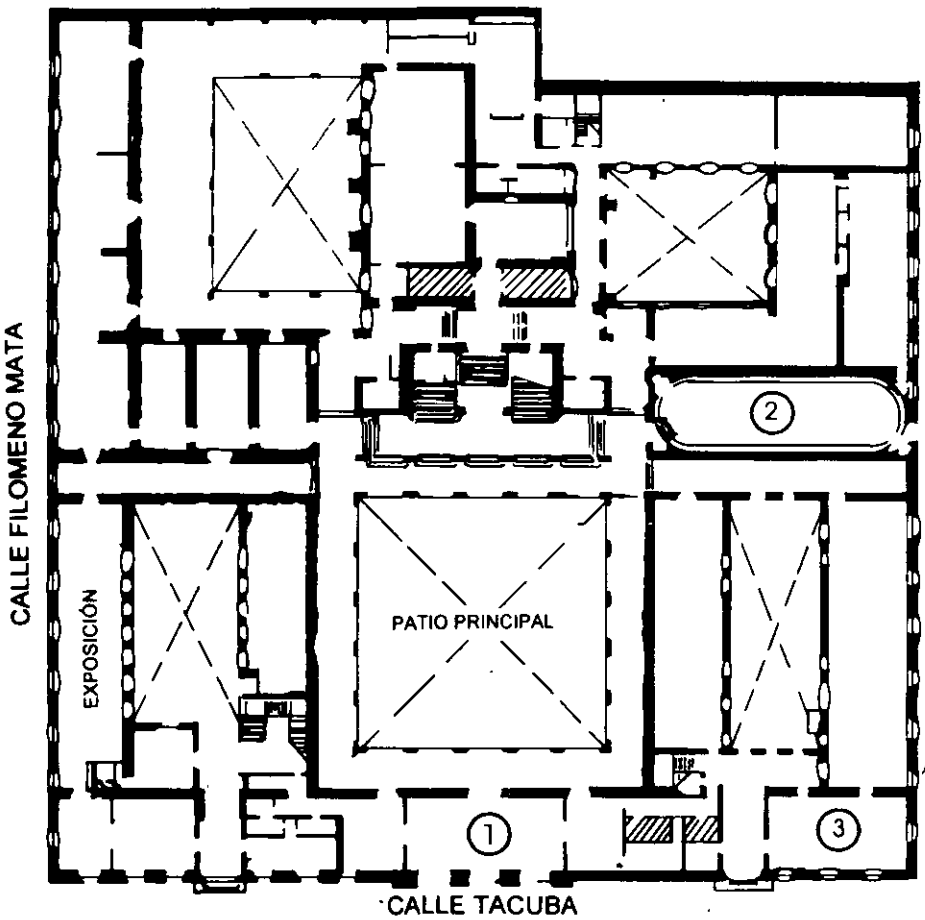
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

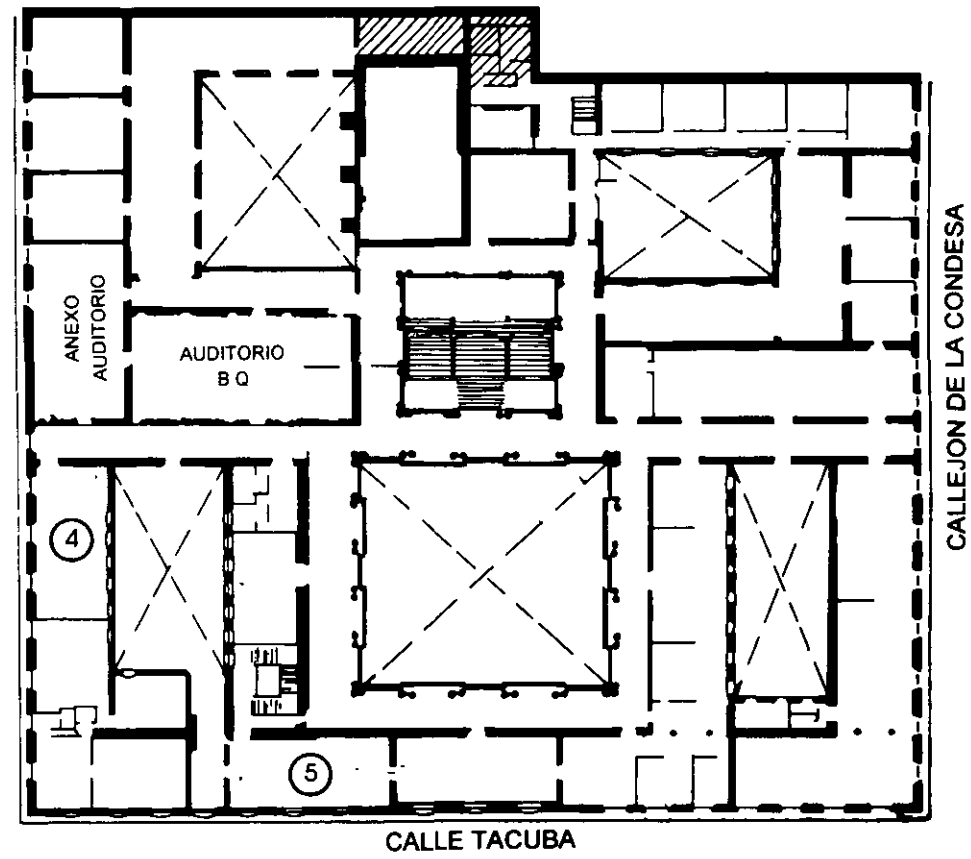
División de Educación Continua.

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

PALACIO DE MINERIA

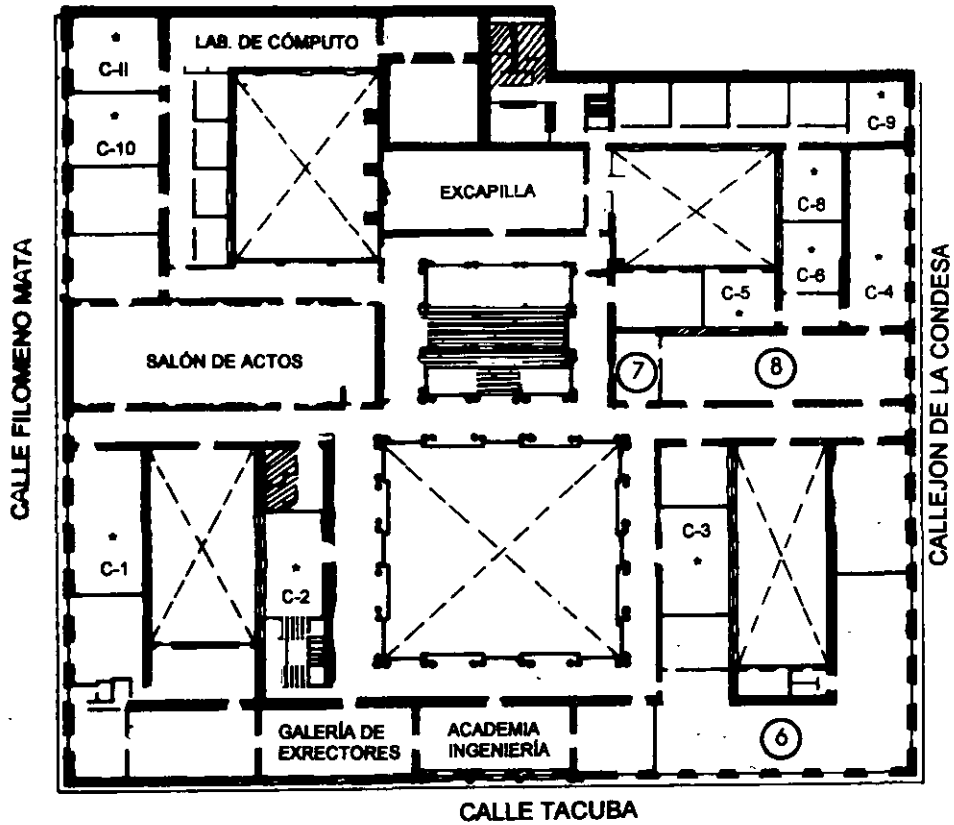


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



1er. PISO

GUÍA DE LOCALIZACIÓN

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

SANITARIOS

* **AULAS**

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

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

CURSO: ***CC020 Introducción a Visual Basic***
FECHA: ***30 de abril al 11 de mayo del 2001***

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACION: 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACION CON EL ASISTENTE	PUNTUALIDAD
TEC. YAZMIN VEGA HERNANDEZ				
JONATHAN GAMBOA BELTRAN				

Promedio _____

EVALUACIÓN DE LA ENSEÑANZA

CONCEPTO	CALIF
ORGANIZACION Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD DEL CURSO	
ACTUALIZACION DEL CURSO	
APLICACION PRACTICA DEL CURSO	

Promedio _____

EVALUACIÓN DEL CURSO

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

Promedio _____

Evaluación total del curso_____

Continúa...2

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

SI ☐ NO ☐

Si indica que "NO" diga porqué: _____

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

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

3. ¿Qué cambios sugeriría al curso para mejorarlo?

_____

_____

_____

_____

_____

_____

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

SI ☐ NO ☐

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

_____

_____

_____

_____

_____

_____

6. Otras sugerencias:

_____

_____

_____

_____

_____

FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

"Tres décadas de orgullosa excelencia" 1971 - 2001

MATERIAL DIDACTICO DEL CURSO

INTRODUCCION A VISUAL BASIC

MAYO, 2001

Palacio de Minería, Calle de Tacuba No 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Col. Centro, México D.F.,
APDO Postal M-2285 • Tels 5521 4021 al 24, 5623 2910 y 5623 2971 • Fax: 5510.0573

I. INTRODUCCIÓN.

Visual Basic es uno de los lenguajes actuales de programación que mas entusiasmo despiertan entre los programadores, tanto expertos como novatos, esto gracias a lo fácil que resulta desarrollar con el aplicaciones para WinX ahorrando así tiempo y demás recursos involucrados en la creación de un sistema.

Visual Basic 6 es una herramienta de programación Visual, (de ahí su nombre) orientada a eventos y basada en objetos (que no orientada) que nos permitirá generar aplicaciones con interfaces gráficas rápidamente y con un mínimo de programación, haciendo uso de los objetos y controles que a nuestra disposición pone el entorno de desarrollo (IDE).

I.1 Tipos de Proyectos.

Dentro del entorno de desarrollo, la primera ventana con la que tendremos contacto será con el selector de tipo de proyecto **Fig. 1.1**. Desde aquí nosotros podremos elegir que tipo de aplicación desarrollaremos dentro del IDE.

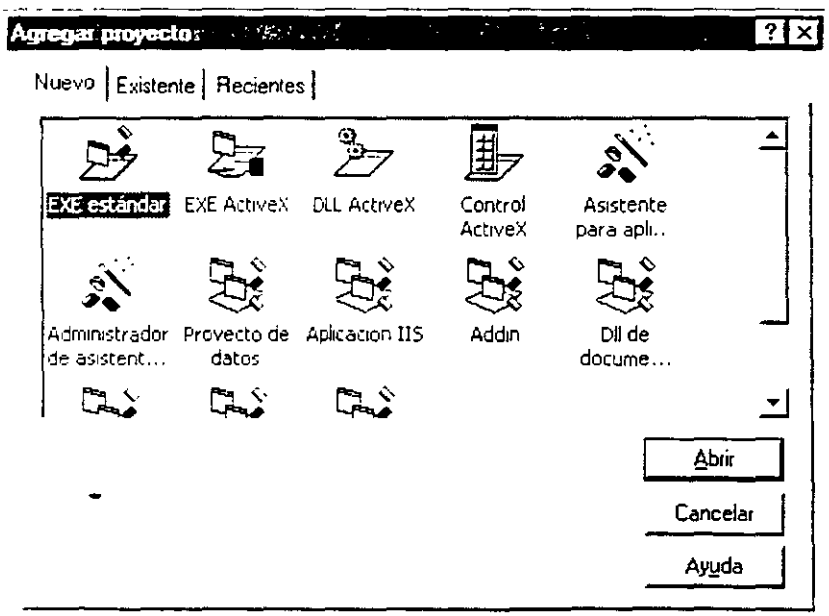


Fig. 1.1 La ventana de selección de tipo de proyecto.

Visual Basic permite construir varios tipos diferentes de aplicaciones. Las mas importantes son:

EXE estándar: Este tipo de aplicación se construye a partir de uno o mas formularios, módulos y clases.

EXE ActiveX. Crea un componente ActiveX(fichero .exe). Un componente ActiveX es una unidad de código ejecutable, como un fichero .exe, .dll u .ocx, que sigue la especificación ActiveX o para proporcionar código reutilizable en forma de objetos.

DLL ActiveX. Crea un componente ActiveX (fichero .dll).Los componentes proporcionan código reutilizable en forma de objetos.

Control ActiveX. Crea un control ActiveX. Los controles no son simplemente código. sino que tienen componentes visuales como los formularios, aunque a diferencia de estos, no pueden existir sin algún tipo de contenedor.

EXE de documento ActiveX. Se trata de un formulario que puede aparecer en un explorador Web.

Aplicación IIS. Se trata de una aplicación Visual Basic hecha para residir en un servidor Web y responder a peticiones enviadas por un explorador.

Aplicación DHTML. Se trata de una o más páginas de código HTML que utilizan código Visual Basic y el modelo de objetos HTML dinámico para responder instantáneamente a las acciones que se producen en dichas páginas.

Asistente para aplicaciones de VB. Genera una aplicación nueva completamente funcional desde la cual se pede generar una aplicación mas compleja.

I.2 El Entorno de desarrollo de Visual Basic 6.

El entorno de programación de Visual Basic se compone de una serie de ventanas y menús que nos permitirán generar un diseño de interfaz gráfica con el usuario (**GUI** por sus siglas en ingles), agregar código a dichas interfaces y manipular propiedades de los objetos entre muchas otras. Todo esto como parte del Ambiente de Desarrollo Integrado (**IDE**) de VB6 para permitirnos de manera ágil e intuitiva desarrollar nuestras aplicaciones.

La barra de Menú.

Resulta similar a todas las presentes en los programas de Windows, contando con los elementos típicos como "**File**", "**Edit**", "**View**" etc. Mas otros propios de VB como lo son "**Debug**", "**Run**", "**Query**" etc. (ver **Fig. 1.2**)

Las barras de Herramientas.

Estas se encuentran por lo regular debajo de la barra de menú, y se activan en el menú **Ver** de esta ultima. Las barras de herramientas pueden ser: **Depuración**, **Editar**. **Editor de Formularios** y **Estandar.** Estas barras ofrecen un rápido acceso a las caracteristicas mas usuales.

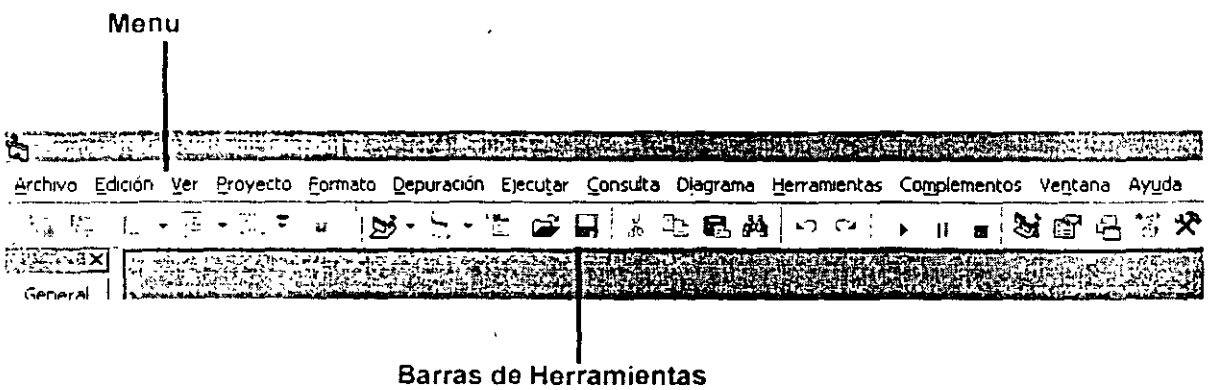
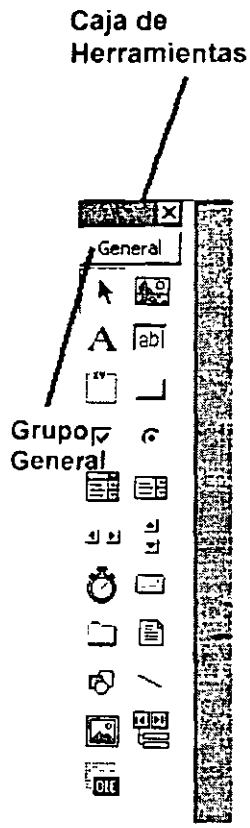


Fig. 1.2 Menú y Barras de Herramientas.

La caja de Herramientas.

Aquí se encuentran agrupados los controles que es posible usar en el diseño de nuestros formularios. Por omisión el entorno de desarrollo solo incluye en esta ventana los controles intrínsecos de VB, aunque es posible integrar en ella otro tipo de controles **ActiveX**.

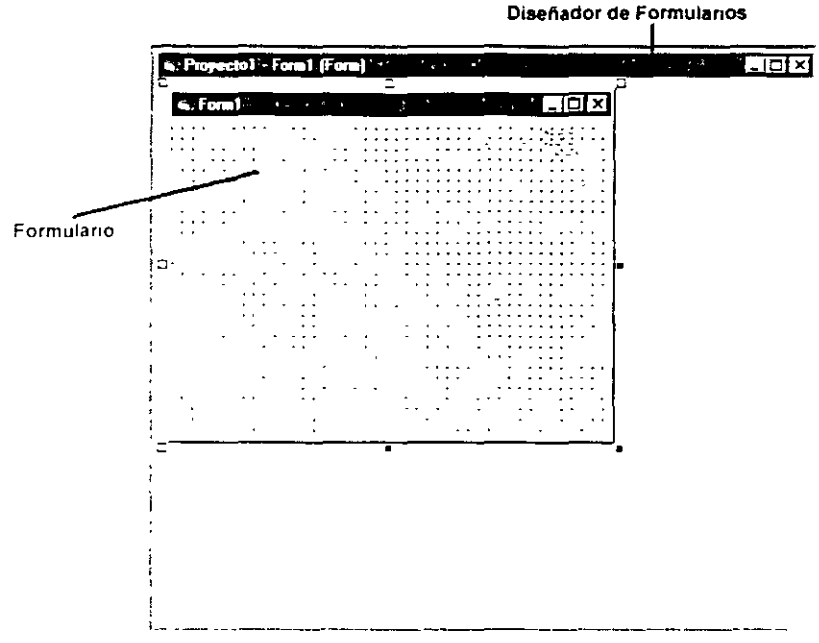


Para evitar la acumulación de controles dentro de esta ventana, es posible agruparlos bajo submenús que serán activados cuando se desee.

El diseñador de formularios.

Aquí es donde se diseña la Interfaz de Usuario (**GUI**) de nuestra aplicación. Mas adelante en este capitulo se explicara el proceso de diseño.

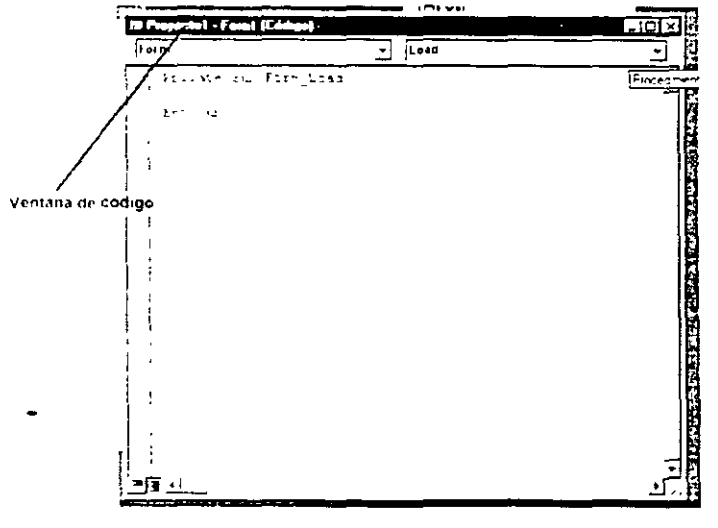
Cabe destacar que una aplicación puede contar con mas de un formulario, por lo tanto, es posible mantener mas de una ventana de diseñador de formulario abierta al mismo tiempo.



Diseñador de Formularios

La ventana de Código.

La ventana de Código se usa para especificar el comportamiento de las formas y en general de los objetos en la aplicación. Es posible mantener abierta una de estas ventanas por cada formulario, o modulo en nuestra aplicación

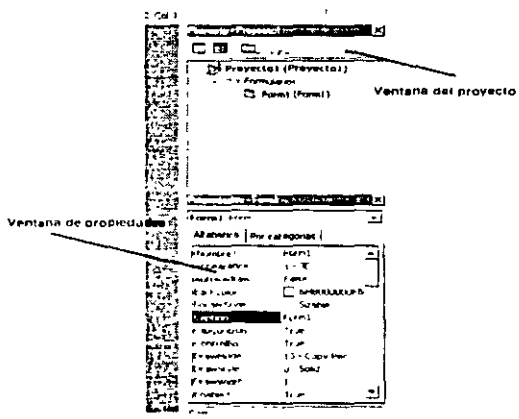


La ventada de Propiedades.

Todos los objetos en VB tienen propiedades que los definen, tanto en su aspecto, como en partes básicas de su comportamiento, en esta ventana nos es posible modificar los valores iniciales de estas propiedades, y así modificar colores, rótulos, etc.

La ventana de Proyecto.

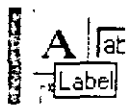
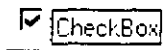
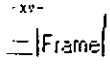
Esta ventana permite acceder a todas las formas, y módulos de nuestra aplicación, estos pueden aparecer dentro de la ventana en diferentes grupos: por tipos, orden alfabético, o por orden de aparición.

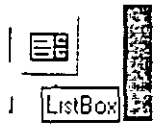


I.3 Los objetos en Visual Basic.

Los controles usados en Visual Basic representan objetos, que pueden dibujarse dentro de nuestros formularios y que conformaran la interfaz de nuestra aplicación con el usuario, por ejemplo, botones, cajas de texto, etc. Dibujar estos controles dentro del área del formulario es una tarea sencilla, no hace falta mas que seleccionarlos de la caja de herramientas descrita anteriormente, para después sobre el área de diseño arrastrarlos hasta conseguir las dimensiones deseadas.

1.4 Los controles intrínsecos.

Nombre		Descripción
Label		Sirve para mostrar texto que no puede ser modificado por le usuario.
PictureBox		Se utiliza para desplegar imágenes, BMP, DIB, ICO. CUR, WMF, EMF, GIF y JPEG.
TextBox		Contiene una cadena de caracteres que puede ser modificada por el usuario.
CommandButton		En este control se realiza la interacción mas intensa con el usuario, por lo regular mediante el evento **Click.**
CheckBox		Casilla de verificación para selección de múltiples opciones.
OptionButton		Casilla de verificación, para elecciones únicas. por ejemplo: si / no
Timer		Este control no es visible en tiempo de ejecución. Su función es realizar un proceso cada determinado tiempo
Frame		Es un contenedor de otros controles, con fin, por ejemplo de agruparlos.

ListBox		Contiene varios elementos, de los cuales el usuario podra elegir varios elementos.
ComboBox		Se trata de una lista desplegable de la que el usuario solo podrá elegir una opción.

Estos objetos de VB tienen propiedades y métodos que se modificaran, y actuaran en respuesta a los eventos que ocurran dentro de nuestra aplicación Para entender esto, es necesario definir formalmente un evento:

Evento: Un evento es un hecho asíncrono e inesperado, que generalmente el usuario realizara sobre algún elemento del formulario, o sobre el mismo formulario.

1.5 Los Eventos en Visual Basic.

Algunos de los eventos usados en Visual Basic son:

- **Click**
- **DblClick**
- **Change**
- **GotFocus**
- **KeyPress**
- **KeyDown**
- **MouseDown**
- **MouseUp**
- **MouseMove**

Después de dibujar nuestras interfaces, graficas, con los controles necesarios, incorporaremos a ellos comportamiento, para responder a los eventos que sobre ellos ocurran.

Eventos que ocurren cuando se carga el formulario

Cuando un formulario se carga la primera vez ocurren varios eventos, los cuales se resumen en la tabla siguiente:

Evento	*Ocurre cuando*
Initialize	Una aplicación crea un ejemplar de un formulario.
Load	Se carga un formulario.
Resize	El formulario se muestra o cambia el estado del mismo.
Activate	El formulario pasa a ser la ventana activa.
Paint	Un formulario se expone total o parcialmente después de haberse movido o ampliado, o después de haberse movido otro formulario que lo estaba cubriendo.

Dichos eventos ocurren en el orden expuesto.

Eventos que ocurren cuando se cierra el formulario

Cuando un formulario se cierra ocurren varios eventos, los cuales se resumen en la tabla siguiente:

Evento	**Ocurre cuando**
QueryUnload	Se cierra un formulario, pero antes de producirse este hecho.

Unload	El formulario está a punto de cerrarse.
Terminate	Todas las variables que hacen referencia al formulario pasan a valer **Nothing** o cuando la última referencia al formulario queda fuera de su ámbito.

Dichos eventos ocurren en el orden expuesto.

Eventos que pueden ocurrir en un formulario o en un control

De los muchos eventos que pueden ocurrir para un formulario o para un control merecen destacar los siguientes:

Evento	Ocurre cuando
KeyDown	El usuario pulsa una tecla.
KeyUp	El usuario suelta la tecla pulsada.
KeyPress	El usuario pulsa una tecla; este evento ocurre después del evento KeyDown y antes de KeyUp.
MouseDown	El usuario pulsa un botón del ratón.
MouseUp	El usuario suelta el botón pulsado del ratón.
Click	El usuario pulsa y suelta un botón del ratón.

Un formulario puede recibir los eventos de teclado sólo si no tiene controles visibles y habilitados o si la propiedad **KeyPreview vale True.**

1.6 El MsgBox

Esta instrucción nos permite enviar un mensaje "externo" al formulario actual, que estará vinculado a este por el valor de retorno, es decir, por el evento que se genero en el, los que podremos conocer y controlar dentro del código del formulario, a través de las siguientes constantes:.

Valor de retorno	Constante Simbólica
1	vbOK
2	vbCancel
3	vbAbort
4	vbRetry
5	vbIgnore
6	vbYes
7	vbNo

Valor tiposBotones	Constante simbólica
0	vbOKOnly
1	vbOKCancel
2	vbAbortRetryIgnore
3	vbYesNoCancel
4	vbYesNo
5	vbRetryCancel

la sintaxis para utilizar esta ventana de dialogo es la siguiente:

retorno = Msgbox("mensaje",botones y tipo,"titulo")

En *mensaje* se escribe el texto que aparecerá en la ventana.
En *botones y tipos* va la suma de las caracteriscas de los botones y el icono característico de la ventana.

16- mensaje de error critico
32- mensaje de pregunta
48- mensaje de exclamación
64- mensaje de información

y para indicar que botón aparecerá seleccionado por omisión se emplean:

0- el primero
256- el segundo
512- el tercero

1.7 Menús.

Muchas aplicaciones simples tienen un formulario y varios controles, pero no cabe duda de que su aspecto puede ser mejorado utilizando menús. Un menú es una forma de proporcionar al usuario un conjunto de órdenes, lógicamente relacionadas, agrupadas bajo un mismo título. El conjunto de todos los títulos correspondientes a los menús diseñados aparecerá en la barra de menús situada debajo del título del formulario.

Cuando el usuario haga clic en un título de un menú, se desplegará una lista visualizando los elementos que contiene el menú, o si no contiene elementos, se ejecutará directamente una acción. Los elementos de un menú pueden ser *órdenes*, *submenús* y *separadores*. Cuando se hace clic en una *orden*, o se selecciona y se pulsa *Entrar*, se ejecuta una acción o se despliega una caja de diálogo. Por convenio, una orden seguida de tres puntos (...) indica que se desplegará una caja de diálogo. Cuando se hace clic en un *submenú* se despliega una nueva lista de

elementos. Un separador tiene como finalidad separar grupos de órdenes de un mismo menú en función de su actividad.

Diseño de un menú

Para diseñar un menú, utilizaremos el *editor de menús.*
Para crear un menú, los pasos a ejecutar son los siguientes:

1. *Abrir el editor de menús.* Para ello, seleccione el formulario para el que desea crear el menú y a continuación ejecute la orden *Editor de menús* del menú *Herramientas*, o bien haga clic en el botón dispuesto para tal fin en la barra de herramientas estándar.

2. *Introducir el título del menú.* Escriba en la caja de texto **Caption** el título del menú que se desea crear, el cual aparecerá en la barra de menús. Inserte un *ampersand* (&) antes de la letra que da acceso directo al menú para que aparezca subrayada. El usuario podrá seleccionar este menú, además de con el ratón, pulsando la tecla *Alt* más la tecla correspondiente a la letra que aparece subrayada. Escriba en la caja de texto **Name** el nombre utilizado en el código para referirse al menú. A continuación, haga clic en el botón *Next* o pulse *Entrar.*

3. *Introducir los elementos que componen el menú.* Escriba en la caja de texto **Caption** el título del elemento del menú y en la caja de texto **Name** el nombre utilizado en el código para referirse a dicho elemento. Por convenio, un elemento de un menú seguido de tres puntos significa que cuando se haga clic sobre él, se desplegará una caja de diálogo.
   Para diferenciar un elemento de un menú del propio menú, hay que sangrar el título del elemento. Para ello, selecciónelo y haga clic en el botón flecha hacia la derecha (→).

4. *Crear un submenú.* Un elemento puede ser una orden si el elemento siguiente aparece sangrado al mismo nivel, o el título de un submenú si el elemento siguiente aparece sangrado un nivel más.

5. *Añadir un separador.* Utilizando separadores puede agrupar las órdenes en función de su actividad. Para insertar un separador, escriba un único guión (—) en la caja **Caption** del editor de menús. Tiene que especificar también un nombre cualquiera (**Name**).

6. *Cerrar el editor de menús.* Una vez que haya finalizado el diseño, pulse la tecla *Aceptar* y observe cómo en la barra de menús del formulario aparecen los menús diseñados.

Lógicamente. los elementos que componen cada menú no estarán operativos hasta que no se una el código correspondiente. Para definir cómo debe responder una orden de un menú cuando reconozca el evento **Click**, hay que escribir un procedimiento para que esa orden responda a ese evento.

Las órdenes de un menú sólo responden al evento **Click.**

Para crear un procedimiento para una orden de un menú, haga clic sobre el menú y a continuación haga clic sobre la orden. Observe que se visualiza el esqueleto del procedimiento, lo que le permite escribir el cuerpo del mismo. Otra forma de realizar este mismo proceso es la siguiente:

1. Abra la ventana de código ejecutando la orden *Código* del menú *Ver*, pulsando *F7*. o bien pulsando el botón *Ver código* del *Explorador de proyectos*.

2. Seleccione el nombre del control en la lista *Objeto;* en nuestro caso seleccionaremos el nombre de la orden del menú.

3. Escriba el código entre **Sub** y **End Sub**.

Propiedades de un menú

Las propiedades que pueden seleccionarse en el editor de menús para un elemento de un menú son **Index, Shortcut, HelpContextID, NegotiatePosition, Checked, Enabled, Visible** y **WindowList**. A continuación, comentamos algunas de ellas.

La propiedad **Index** permite que un conjunto de órdenes sean agrupadas en una matriz de controles. Cuando un usuario selecciona una orden de una matriz de controles, Visual Basic utiliza el valor **Index** para identificar qué orden de la matriz fue seleccionada.

La propiedad **Shortcut** define un acelerador; esto es, una tecla o combinación de teclas que permiten activar un elemento. Las combinaciones válidas se visualizan en la lista asociada con esta propiedad. Por ejemplo, abra el editor de menús, seleccione la orden *Salir* del ejemplo anterior y asigne a la propiedad **Shortcut** la combinación de teclas *Ctrl+S*. Cierre el editor de menús y ejecute la aplicación. Compruebe cómo al pulsar *Ctrl+S* se ejecuta la orden *Salir*.

La propiedad **Checked** es útil para indicar si una orden está activa o no lo está. Cuando se especifica esta propiedad aparece una marca (_) a la izquierda del elemento del menú.

La propiedad **Enabled** es útil para desactivar una orden en un momento en el cual no tiene sentido que esté activa. Por ejemplo, si estamos trabajando con un editor y no tenemos seleccionado un bloque de texto, no tiene sentido que la orden *Copiar* del menú *Edición* esté activa. Cuando se hace clic en una orden de un menú, ésta será ejecutada si su propiedad **Enabled** está señalada. Si la propiedad **Enabled** no está señalada, la orden se muestra en tono gris y no se puede ejecutar.

La propiedad **Visible** es útil cuando durante la ejecución se desea ocultar una orden. Cuando esta propiedad no está señalada, la orden no aparece y por lo tanto no puede ejecutarse.

1.8 Cajas de dialogo.

El control *Microsoft Common Dialog* (diálogo común) permite visualizar las cajas de diálogo más comúnmente empleadas en el diseño de aplicaciones. tales como:

Abrir (Open), Guardar como (Save As), Imprimir (Print), Fuente Çtont) y *Color (Color).*

Este control viene incluido con todas las ediciones de Visual Basic.

Para utilizar una de estas cajas de diálogo, ejecute los pasos siguientes:

1. Seleccione el control *diálogo común* en la caja de herramientas y dibújelo sobre el formulario. Este control fija su tamaño automáticamente, y durante la ejecución es invisible.

   Si este control no se encuentra en la caja de herramientas, entonces hay que añadirlo. Esto puede hacerlo ejecutando la orden *Componentes...* del menú *Proyecto* y eligiendo el control *diálogo común* de Microsoft *(Microsoft Common Dialog Control 6.0).* Observe que se añade el Archivo *comdlg32.ocx* al proyecto; este tipo de Archivos se localizan en el directorio *..\system.*

2. Para visualizar alguna de las cajas de diálogo mencionadas, invoque durante la ejecución de la aplicación a alguno de los métodos siguientes:

Método de diálogo	*Caja*
ShowOpen	Abrir
ShowSave	Guardar como
ShowColor	Color
ShowFont	Fuente
ShowPrinter	Imprimir
ShowHelp	Invoca al motor de ayuda de Windows

Cajas de diálogo Abrir y Guardar como

La caja de diálogo *Abrir* permite al usuario seleccionar una unidad de disco, un directorio, una extensión de Archivo y un nombre de Archivo, así como indicar si éste va a ser abierto sólo para lectura. Una vez realizada la selección, la propiedad **FileTitle** de la caja de diálogo contiene el nombre del Archivo elegido.

La caja de diálogo *Guardar como* es idéntica a la caja de diálogo *Abrir*, excepto en el título.

Para visualizar la caja de diálogo *Abrir* (o *Guardar como*):

1. Especifique la lista de tipos de Archivos que desea visualizar. Para ello, asigne a la propiedad **Filter** una cadena de caracteres de la forma:

   descripción1 | filtro1 | descripción2 |filtro2...

2. Asigne a la propiedad **FilterIndex** el filtro por omisión de los de la lista especificada anteriormente (el primero es el 1).

3. Visualice la caja de diálogo y seleccione o escriba un nombre de Archivo.

```
CommonDialog1.ShowOpen 'visualiza Abrir
```

```
CommonDialog1.ShowSave 'visualiza Guardar como
```

4. Una vez elegido el Archivo, la propiedad **FileTitle** contiene el nombre del Archivo.

Cuando el usuario visualiza una caja de diálogo común y pulsa el botón *Cancelar*, se produce un error que interrumpe la aplicación. Para evitarlo, hay que generar un error interceptable justo cuando el usuario haga clic en el botón *Cancelar* de una caja de diálogo predefinida. Para ello, asigne a la propiedad **CancelError** el valor **True** y proceda de forma similar a como se indica en el ejemplo siguiente. Un error interceptable es aquel que podemos tratar a voluntad utilizando la sentencia **On Error** que estudiaremos mas adelante. El error que se genera es el *32755 (cdlCancel)*.

El control diálogo común tiene una propiedad, *(Personalizado)*, que le permite establecer algunas propiedades de las mencionadas durante el diseño.

2. INTRODUCCIÓN AL LENGUAJE VISUAL BASIC.

Visual Basic adopta los tipos de datos, estructuras y muchas de las reglas sintácticas de su principal predecesor, BASIC. Sin embargo implementa muchas cosas nuevas, que iremos estudiando a partir de ahora.

2.1 Comentarios.

En Visual Basic todas las líneas que comiencen con " ' " no son tomadas en cuenta por el compilador, y nos sirven para escribir comentarios que clarifiquen el sentido del código.

```
Dim i As Integer
```

Las líneas de comentarios solo podrán encontrarse dentro de algún modulo de codigo. ya sea función , procedimiento o zona de declaraciones.

2.2 Constantes Numéricas y de Caracteres.

Una constante es un dato cuyo valor no cambia durante la ejecución de un programa. Y son aceptados valores decimales, hexadecimales, octales y cadenas alfanuméricas.
Es importante aclarar que en las constantes octales y hexadecimales el valor debe ir precedido por un & . Y en el caso de las alfanuméricas deben ir entre comillas dobles.

2.3 Variables

Una variable es un dato que puede cambiar su valor durante la ejecución del programa. Estos datos tienen ciertas características que forman parte de su funcionamiento:

Nombre: Para poder hacer referencia a ella durante el programa. Es importante aclarar que el nombre debe empezar con una letra, y puede tener hasta 255 caracteres. los caracteres (excepto el primero) pueden ser %&!#$@_ sin embargo deben estar en la ultima posición (excepto el _). No se pueden utilizar los caracteres :,. (); ni las palabras reservadas por el lenguaje como **For**, **Hide**, **Caption**, **Long** y **And** entre otras.

***Tipo*:** Determina cuales valores pueden ser almacenados en ella.

***Ámbito*:** Especifica en que parte del proyecto podremos hacer referencia a ella. Se debe tener cuidado de no declarar dos variables con el mismo nombre dentro del mismo ámbito, pues esto generará un error en la compilación.

Tipo	Descripción	Carácter reser.	Rango
Integer	Entero (2 bytes)	%	-32.768 a 32.767
Long	Entero largo (4 bytes)	&	-2.147.483.648 a 2.147.483.647
Single	Precision simple (4)	¡	-3,40E+38 a 3,40E+38
Double	Precision doble (8 bytes)	#	-1,79D+308 a 1,79D+308
Currency	Numero con punto decimal fijo (8)	@	+/- 922.337.203.685.477,5807
String	Cadena de	Ninguno	Hasta 64 K

	longitud fija (1 byte por caracter)		
String	Cadena de longitud Variable (10 +1 por carácter)	$	Hasta 2^{31} caracteres
Byte	Carácter (1 byte)	Ninguno	0 a 255
Boolean	Boolean (2 byte)	Ninguno	True o False
Date	Fecha/Hora(8 bytes)	Ninguno	1/Enero/100 a 31/Diciembre/9999
Object	Referencia a un objeto (4 bytes)	Ninguno	Cualquier referencia a tipo objeto
Variant	Con números 16 bytes	Ninguno	Cualquier valor numerico hasta el Double
Vatiant	Con caracteres 22 + 1 por caracter	Ninguno	El mismo que para un string de longitud variable.

2.3.1 Declaración de Variables

Lo mejor es declarar de que tipo serán las variables que utilicemos ya que esto nos evitara problemas de mezclar datos no compatibles mas adelante.
La sintaxis es:

Dim I as Integer
Dim Y as Integer
Dim I, Y as Integer

La declaración de las dos primeras lineas es similar ambas indican que se utilizaran las variables I e Y de tipo entero sin embargo la tercera no es eso lo que

dice, esta indica que se empleara una variable I de tipo Variant y una Y de tipo Integer.

2.4 Estructuras de Control.

Las sentencias de control, denominadas también *estructuras de control*, permiten tomar decisiones y realizar un proceso repetidas veces. Visual Basic dispone de las siguientes estructuras, algunas de ellas idénticas a las de sus predecesores, como QuickBasic:

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

Veamos a continuación la sintaxis correspondiente a cada una de ellas; cualquier expresión entre corchetes –[]-- es opcional.

If ... Then ... Else

Esta estructura permite ejecutar condicionalmente una o más sentencias y puede escribirse de las dos formas siguientes:

If *condición* **Then** *sentencia(s)* **[Else** *sentencia(s)*]

If *condición* **Then**
sentencia(s)

[Else
sentencia(s)]
End If

La condición es generalmente una expresión de tipo *Booleano*, pero puede ser cualquier expresión que sea evaluada a un valor numérico, ya que un valor cero es considerado como **False** y un valor distinto de cero como **True.** Si la *condición* es **True**. se ejecutan las sentencias que están a continuación de **Then y** si la condición es **False** . se ejecutan las sentencias que están a continuación de **Else**. si esta cláusula ha sido especificada.

Una expresión de tipo *Booleano* da un resultado **True o False** . Desde un análisis riguroso, los operadores booleanos son los operadores lógicos **And, Or** y **Not** Ahora bien, por ejemplo, observe que las comparaciones producen un resultado de tipo **Boolean.** Quiere esto decir que el resultado de una comparación puede utilizarse como operando en una expresión *Booleana.* Según esto, podemos enunciar que los operadores que intervienen en una expresión de tipo Booleana pueden ser de relación (=, <>,<, <=, >, y >=), lógicos **(And, Or, Xor, Eqv, Imp,** y **Not**) y otros, como **Like** e **Is.**

Para indicar que se quiere ejecutar uno de varios bloques de sentencias dependientes cada uno de ellos de una condición, la estructura adecuada es la siguiente:

If *condición-1* **Then**
sentencias-1
[ElseIf *condición-2* **Then**
sentencias-2]
...**[Else** *sentencias-n]*
End If

Si se cumple la *condición-1*, se ejecutan las *sentencias-1* y si no se cumple, se examinan secuencialmente las condiciones siguientes hasta **Else**, ejecutándose las sentencias correspondientes al primer **ElseIf** cuya condición se cumpla. Si todas las condiciones son falsas, se ejecutan las *sentencias-n* correspondientes a **Else** En cualquier caso, se continúa en la sentencia que sigue a **End If**

Select Case

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

Select *Case* *expr-test*
Case *lista-1*
[sentencias-1]
[Case *lista-2*
[sentencias-2]] ...
[CaseElse
[sentencias-nl]
End Select

donde *expr-test* es una expresión numérica o alfanumérica.

Cuando se ejecuta una sentencia **Select Case**. Visual Basic evalúa la *expr-test* y busca el primer **Case** que incluya el valor evaluado, ejecutando a continuación el correspondiente bloque de sentencias. Si no existe un valor igual a la *expr-test.* entonces se ejecutan las sentencias a continuación de ***CaseElse***. En cualquier caso, el control pasa a la siguiente sentencia a End Select

Do... Loop

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

Formato 1:

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

Formato **2:**

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

donde *condición* es cualquier expresión que se evalúe **a True o a False.**

Esta sentencia permite realizar varias estructuras diferentes. Permite, como se puede apreciar en los formatos, crear bucles con la condición de terminación al final o al principio del bloque de sentencias.

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

éste.

For... Next

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

For *variable* = *expresión-1***To** *expresión 2* **[Step** *expresión 3] [sentencias]*
[Exit For]
[sentencias]
Next *[variable [,variable]...]*

Cuando se ejecuta una sentencia **For** en la que el valor de la *expresión 3* es positivo o no se ha especificado, primero se asigna el valor de la *expresión1* a la *variable* y a continuación se comprueba si la *variable* es mayor que la *expresión 2*, en cuyo caso se salta el cuerpo del bucle y se continúa en la línea que esté a continuación de la sentencia Next. En otro caso, se ejecutan las líneas de programa que haya entre la sentencia **For** y la sentencia Next. Por último, la *variable* se incrementa en el valor de la *expresión 3*, o en 1 si Step no se especifica, volviéndose a efectuar la comparación entre la *variable* y la *expresión 2*, y asi sucesivamente.

La sentencia Exit **For** permite salir de un bucle **For...** Next antes de que éste finalice.

Un bucle **For** ... **Next** se ejecuta más rápidamente cuando la *variable* es entera, y las *expresiones 1,2, 3*, constantes.

For Each .... Next

For Each .... Next es similar a la sentencia **For,** con la diferencia de que ahora se repite un grupo de sentencias por cada elemento de una colección de objetos o de una matriz (excepto matrices de elementos de un tipo definido por el usuario). Esto es especialmente útil cuando no conocemos cuántos elementos hay en la colección o en la matriz. Su sintaxis es la siguiente:

For Each *elemento* **In** *grupo*
[sentencias]
[Exit For]
[sentencias]
Next *elemento*

donde *elemento* es una variable de tipo **Variant** que representa a cada uno de los elementos de la colección o de la matriz representada por *grupo*.

2.5 Tipo de datos definidos por el usuario.

Una estructura o registro es un tipo de dato nuevo, definido por el usuario y que puede estar formado por datos de los tipos predefinidos, o por otras estructuras, con la caracteristica de que pueden ser tratados como si de datos predefinidos se tratase. Las sentencias para generar una estructura son **Type** y **End Type**:

Ejemplo:

Type *Alumno*
Nombre As String
Edad As integer
End Type.

Con estas sentencias se ha declarado el tipo de dato *Alumno*.

Una vez que se haya definido una estructura, podrán crearse variables de este nuevo tipo igual que lo haría normalmente con los tipos nativos de Visual Basic.

Ejemplo:

Dim Juan **As** Alumno
Juan.Nombre = "Juan"
Juan.Edad="21"

2.6 Funciones.

Las funciones son subrutinas parecidas a los procedimientos que hemos usado hasta ahora. con la diferencia de que estas regresaran un valor a quien las haya invocado. la declaración general de una función y las palabras clave involucradas son:

[Private | Public] Function *nombre* (*parametros*) As *tipo*
instrucciones
nombre = expresión
End Function

En el la declaración anterior, **[Private | Public]** indican el grado de acceso que se tendrá a la función, **Function** es la palabra reservada que nos permite declarar la función, *nombre* es el nombre con el que se invocara a la función y a la ves dentro del cuerpo de la función nos servirá para regresar el valor de la función, *parametros* es la lista de argumentos separados por coma, con los que trabajara

nuestra función, finalmente As *tipo* nos sirve para declarar el tipo de retorno que tendrá la función.

Respecto al manejo de funciones, VB es mas flexible que otros lenguajes, pues permite contar con argumentos opcionales de la siguiente forma:

```
Function Division( div As Integer, div2 As Integer, Optional b As String )
```

En la declaración anterior la palabra reservada **Optional** nos permite establecer que el parámetro que le sigue puede o no proporcionarse.

Otra de las ventajas que aporta el sistema de funciones y procedimientos de VB es el de permitir crear funciones con un numero indeterminado de argumentos esto de la siguiente forma:

```
Function Division( ParamArray args() As Variant )
```

La palabra clave **ParamArray** nos permite establecer que no existe un numero definido de argumentos y que los que se reciban se almacenaran en un arreglo, con el cual se podrá hacer referencia a ellos dentro de la función.

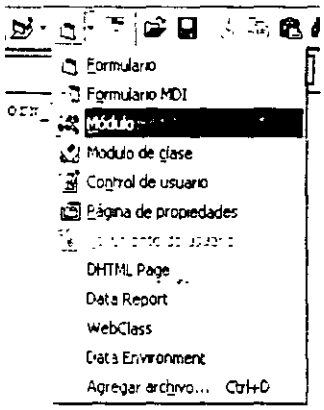
Existen algunas restricciones para esta modalidad:

- Solo puede existir un **ParamArray** por cada declaración, y debe de ser el ultimo de la declaración de parámetros.
- El arreglo declarado por la palabra **ParamArray** solo puede ser de tipo **Variant.**
- La palabra clave **ParamArray** no puede venir precedida por el parámetro **Optional**

2.7 Módulos Estándar.

Hasta el momento solamente hemos utilizado formularios dentro de nuestros programas. Pero dentro de la variedad de módulos con los que es posible trabajar en Visual Basic se encuentran los Módulos estándar donde almacenaremos exclusivamente código de la aplicación que será visible en todos el resto de los módulos.

Para agregar un módulo estándar , se hace uso del botón agregar seleccionándolo dentro de las opciones.



3. MANEJO DE ERRORES.

El manejo de los errores ocurridos dentro de la ejecución de un programa es uno de los aspectos mas importantes en el desarrollo de Software comercial. pues se debe de garantizar en la medida de lo posible, que se terminara de manera correcta la ejecución de nuestro programa, para cubrir con estas posibilidades Visual Basic cuenta con un esquema de manipulación de errores que permite al programador tomar el control de lo que ocurre, tras haberse desarrollado un error. esto mediante instrucciones que se ven mezcladas con la estructura de las funciones y procedimientos, estas sentencias son:

On Error Resume Next.

Esta instrucción obliga al programa a continuar con la sentencia siguiente a aquella donde se produjo el error.

On Error Goto 0..

Esta instrucción anula la manipulación de errores.

On Error Goto <etiqueta>..

Esta instrucción obliga al programa a saltar al punto del mismo donde se encuentra la <*etiqueta*> donde se procesaran los errores.

Existen diferentes maneras de procesar el error ocurrido en una instrucción *On Error Goto <etiqueta>*

- Puede ejecutarse la función *Resume* para ejecutar nuevamente la línea donde se genero el error.

- Puede ejecutarse la función *Resume Next* para continuar el programa a partir de la siguiente línea después de donde se genero el error.
- Puede ejecutarse *Resume <Line>* para continuar la ejecución a partir de una línea dada. donde *<Line>* puede ser un numero de línea o una etiqueta.
- Se puede reportar a la función que invoco a la generadora del error. que este a ocurrido, mediante el metodo *Err.Raise*.

Se puede terminar la ejecución del procedimiento con un *Exit Sub* o *Exit Function*.

3.2 El objeto Err.

Visual Basic almacena información útil sobre el error que ha ocurrido. dentro del objeto **Err** a continuación se enlistan sus propiedades y metodos:

.Number = Almacena el código numérico del error.
.Source = Se almacena el lugar donde se produjo el error.
.Description = Se almacena una cadena con la descripción del error.
.HelpFile y *.HelpContext* = Guardan descripciones secundarias del error.
.LastDllError = Almacena información en caso de que se produzca un error en el proceso de una rutina

Err.Raise

Este método nos permite generar un error asignando valores a sus propiedades.

Err.Clear

Este metodo limpia todas las propiedades del objeto **Err**.

3.3 El Objeto Collection.

Este objeto predefinido en VBA nos permite almacenar datos referenciados no solo por un índice numérico, si no también por una llave o clave, relacionada al dato, y que nos permitirá acceder a el de manera mas flexible. La declaración de un objeto de este tipo es:

Dim libreria **As New Collection**

Declaración muy parecida a las que hasta ahora conocíamos, con excepción del operador **New**, este operador nos permite crear un nuevo objeto de determinado tipo, en este caso del tipo **Collection**.

Las ventajas del objeto **Collection** no terminan ahí, este objeto también nos permite insertar y borrar elementos de el de manera automática, sin tener que preocuparnos de recorrerlo o agrandarlo, como sucedía con los arreglos y matrices:

Para agregar elementos:

Colección.**Add** *"Item" , "Llave"*

Para borrar elementos:

Colección.**Remove** *"Llave"*

Para acceder a un elemento:

Colección.**Item** *"Llave"* o *Colección*.**Item** (*i*)

4. MANEJO DE ARCHIVOS.

4.1 Apertura y cierre de archivos

Para poder leer o escribir en un archivo antes debe ser abierto con la sentencia ***Open***, cuya forma general es la siguiente:

Open filename For modo As # fileNo

donde:

filename es el nombre del archivo a abrir. Será una variable ***String*** o una cadena entre dobles comillas (" ").

modo Para acceso secuencial existen tres posibilidades: ***Input*** para leer, ***Output*** para escribir al comienzo de un archivo y ***Append*** para escribir al final de un archivo ya existente. Si se intenta abrir en modo ***Input*** un archivo que no existe, se produce un error. Si se abre para escritura en modo ***Output*** un archivo que no existe se crea, y si ya existía se borra su contenido y se comienza a escribir desde el principio. El modo ***Append*** es similar al modo ***Output***, pero respeta siempre el contenido previo del archivo escribiendo a continuación de lo último que haya sido escrito anteriormente.

fileNo es un número entero (o una variable con un valor entero) que se asigna a cada archivo que se abre. En todas las operaciones sucesivas de lectura y/o escritura se hará
referencia a este archivo por medio de este número. No puede haber dos archivos abiertos con el mismo número. ***Visual Basic*** dispone de una función llamada

FreeFile que devuelve un número no ocupado por ningún archivo.

A continuación puede verse un ejemplo de archivo abierto para lectura:

Open "C:\usuarios\PRUEBA1.txt" For Input as #1

Después de terminar de leer o escribir en un archivo hay que cerrarlo. Para ello se utilizara el comando ***Close***, que tiene la siguiente forma:

Close # fileNo

donde el ***fileNo*** es el número que se la había asignado al abrirlo con la instrucción ***Open***.

Lectura y escritura en un archivo de acceso secuencial.

La función ***Line Input #*** lee una línea completa del archivo y devuelve su contenido como valor de retorno. Su forma general es:

varString = Line Input #fileNo

Conviene recordar que en los archivos de texto se suele utilizar el carácter ***return*** (o ***Intro***) para delimitar las distintas líneas. Este es el carácter ASCII nº 13, que por no ser un carácter imprimible se representa en ***Visual Basic 6.0*** como *chr(13)* En muchas ocasiones (como herencia del MS-DOS) se utiliza como delimitador de líneas una combinación de los caracteres ***return*** y ***linefeed***, representada en ***Visual Basic 6.0*** como *chr(13)+chr(10)*. En la cadena de caracteres que devuelve ***Line*** no se incluye el carácter de terminación de la línea.
Para leer todas las líneas de un archivo se utiliza un ciclo ***for*** o ***while***. ***Visual Basic 6.0***
dispone de la función ***EOF*** (*End of File*) que devuelve ***True*** cuando se ha llegado al final del archivo.

```
Do While Not EOF(fileNo)
miLinea = Line Input #fileNo
...
Loop
```

También se puede utilizar la función ***Input***, que tiene la siguiente forma general:

```
varString = Input(nchars, #fileNo)
```

donde ***nchars*** es el número de caracteres que se quieren leer y ***varString*** es la variable donde se almacenan los caracteres leídos por la función. Esta función lee y devuelve todos los caracteres que encuentra, incluidos los ***intro*** y ***linefeed***. Para ayudar a utilizar esta función existe la función ***LOF(fileNo)***, que devuelve el nº total de caracteres del archivo. Por ejemplo, para leer todo el contenido de un archivo y escribirlo en una caja de texto se puede utilizar:

```
txtCaja.text = Input(LOF(fileNo), #fileNo)
```

Print #

Para escribir el valor de unas ciertas variables en un archivo previamente abierto en modo ***Output*** o ***Append*** se utiliza la instrucción ***Print #***, que tiene la siguiente forma.

```
Print #fileNo, var1, var2, var2, ...
```

donde ***var1***; ***var2***,... pueden ser variables, expresiones que dan un resultado numérico o
alfanumérico, o cadenas de caracteres entre dobles comillas, tales como *"El valor de x es..."*.Considérese el siguiente ejemplo:

Print #1, "El valor de la variable I es: ", I

donde *I* es una variable con un cierto valor que se escribe a continuación de la cadena.

FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

"Tres décadas de orgullosa excelencia" 1971 - 2001

INTRODUCCION A VISUAL BASIC

C O M P L E M E N T O

MAYO DEL 2001

Palacio de Minería, Calle de Tacuba No. 5 Primer piso, Delegación Cuauhtémoc, CP 06000, Col. Centro, México D.F.
APDO Postal M-2285 • Tels. 5521 4021 al 24, 5623 2910 y 5623 2971 • Fax. 5510 0573

APÉNDICE A. Manipulación de cadenas numéricas.

En este apéndice se explican algunas funciones para el manejo de cadenas, que forman parte de la librería de funciones de VB, estas se encuentran agrupadas según sus características, y fines.

1. Existen funciones que nos permitirán cambiar de base el valor numérico representado en una cadena, a saber:

Hex$(cadena_Decimal) ***cadena_Decimal → cadenaHexadecimal***

Ejemplo:

```
Dim cadHexadecimal As String
Dim cadDecimal As  String

cadDecimal = "255"

cadHexadecimal = Hex$( cadDecimal )
```

Oct$(cadena_Decimal) ***cadena_Decimal → cadenaOctal***

Ejemplo:

```
Dim cadOctal As String
```

Dim cadDecimal As String

cadDecimal = "80"

*cadOctal = **Oct$**(cadDecimal)*

Val(cadena_Octal o cadena_Hexadecimal) cadenal → cadenaDecimall

Ejemplo:

Dim cadOctal As String
Dim cadDecimal As String

cadOctal = "120"

*cadDecimal = **Val**("&O" & cadOctal)*

En esta función cabe destacar la necesidad de anexar a la cadena a convertir los caracteres de especificación de tipo de datos, "&H" para hexadecimal o "&O" para octal.

2. Existen funciones que nos permiten extraer secciones de una cadena, estas son:

- La función ***Mid$*** nos permitirá extraer una sección intermedia de una cadena.

Mid$(cadena, inicio_de_sección As integer, longitud_de_seccion As Integer)

cadena	es el texto del que se generara la subcadena.
inicio_de_sección	es la posición de inicio a partir de la cual se formara la subcadena.
longitud_de_seccion	es la cantidad de caracteres a partir de la posición de inicio que formaran la subcadena.

Ejemplo:

Subcadena As String

*Subcadena = **Mid$**("12345678",3,4)*

- La función ***Left$*** nos permitirá extraer una sección a partir del extremo izquierdo de la cadena

Left$(cadena, longitud_de_seccion As Integer)

cadena	es el texto del que se generara la subcadena.

longitud_de_seccion	es la cantidad de caracteres a partir del extremo izquierdo (inicio) de la cadena que formaran la subcadena.

Ejemplo:

Subcadena As String

Subcadena = LeftS("12345678",3)

- La función ***RightS*** nos permitirá extraer una sección a partir del extremo derecho (fin) de la cadena

RightS (cadena, longitud_de_seccion As Integer)

cadena	es el texto del que se generara la subcadena.
longitud_de_seccion	es la cantidad de caracteres a partir del extremo derecho (fin) de la cadena que formaran la subcadena.

Ejemplo:

Subcadena As String

Subcadena = RightS ("12345678",3)

3. Para convertir una cadena de texto, entre minúsculas y mayúsculas se utilizan las siguientes funciones:

UcaseS("texto") "texto" → "TEXTO"

LcaseS("TEXTO") "TEXTO" → "texto"

Estas funciones serán útiles por ejemplo cuando deseemos homogeneizar el formato de un texto.

4. Existen funciones para quitar espacios en blanco a los lados de una cadena, estas son muy útiles cuando es necesario realizar comparaciones entre expresiones exactas tomadas de cajas de texto en la interfaz del programa, pues nunca estaremos seguros de la forma en que el usuario a llenado los campos de nuestros formularios:

LTrimS(" **texto** ") " texto " → "texto "

RTrimS(" **texto** ") " texto " → " texto"

TrimS(" **texto** ") " texto " → "texto"

5. Para trabajar con caracteres siempre se hará necesario el uso del código de los caracteres. Existen en Visual Basic ciertas funciones que nos permiten manejar de manera indirecta estos valores, estas son:

Asc("texto") Toma el primer carácter de ***texto*** y regresa su código ascii

Ejemplo:

```
Dim codigo As integer

Codigo = Asc("Hola")    ' Ahora codigo= 72 que es H en código ascii
```

CharS(numero) Regresa el carácter correspondiente al entero que se le ha pasado como parámetro.

```
Dim caracter As String

caracter = CharS(65)    ' Ahora carácter tiene el carácter "A".
```

5. En Visual Basic para buscar subcadenas dentro de otras existen algunas funciones:

InStr (Inicio de Búsqueda, Cadena , subcadena a buscar)

Donde:

Inicio de Búsqueda Es la posición de la cadena donde se iniciara la búsqueda.

Cadena Es la cadena donde se realizara la búsqueda.

subcadena a buscar Es la cadena que se buscara.

Esta función nos devolverá la posición donde se encontró por primera vez la subcadena buscada.

Ejemplo:

```
z = InStr(" Esta es una PRUEBA de una BÚSQUEDA", "prueba" )
'No encontrara "prueba"

z = InStr(" Esta es una PRUEBA de una BÚSQUEDA", "PRUEBA" )
'z vale 14
```