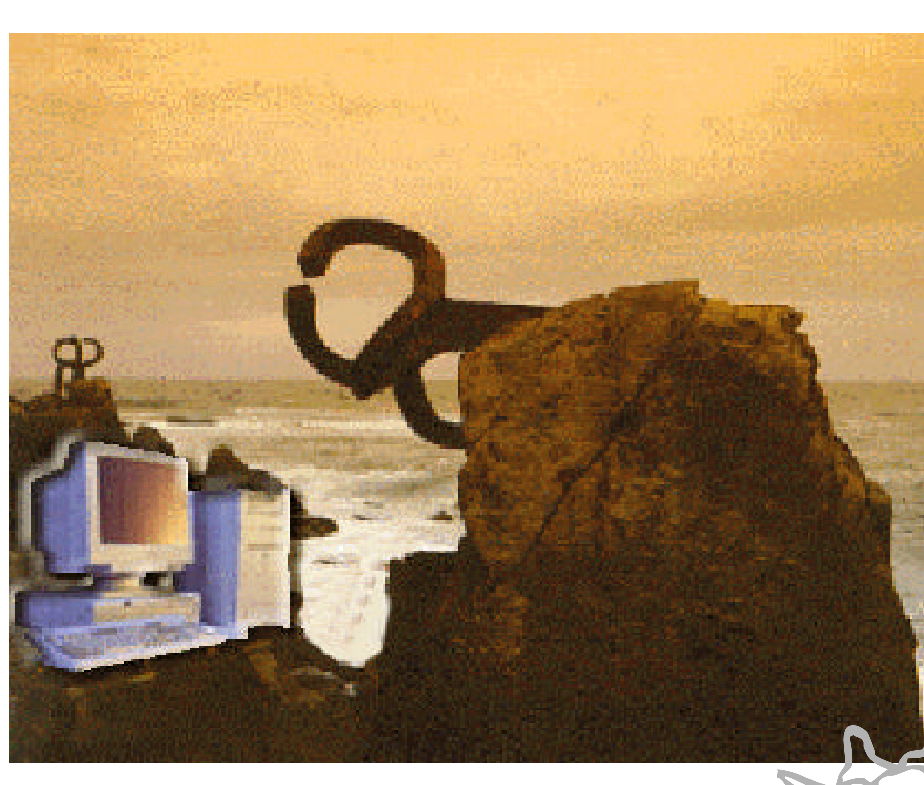
Escuela Superior de Ingenieros Industriales de San Sebastián

UNIVERSIDAD DE NAVARRA

Aprenda Visual Basic 6.0

como si estuviera en primero



San Sebastián, agosto 1999

Aprenda Informática ...

Javier García de Jalón • José Ignacio Rodríguez • Alfonso Brazález

http://www.tayuda.com/ayudainf/

Aprenda Visual Basic 6.0

como si estuviera en primero

Javier García de Jalón

José Ignacio Rodríguez

Alfonso Brazález

Perteneciente a la colección : *"Aprenda …, como si estuviera en primero"*

Editada y mantenida por Javier García de Jalón (jgjalon@etsii.upm.es)

Nuevos títulos y actualizaciones en: http://www.tayuda.com/ayudainf/index.htm

ÍNDICE

1. Introducción **1**
1.1 Programas secuenciales, interactivos y orientados a eventos 1
1.2 Programas para el entorno Windows 2
1.2.1 Modo de Diseño y Modo de Ejecución 2
1.2.2 Formularios y Controles 2
1.2.3 Objetos y Propiedades 2
1.2.4 Nombres de objetos 3
1.2.5 Eventos 4
1.2.6 Métodos 4
1.2.7 Proyectos y ficheros 4
1.3 El entorno de programación Visual Basic 6.0 5
1.4 El Help de Visual Basic 6.0 6
1.5 Ejemplos 6
1.5.1 Ejemplo 1.1: Sencillo programa de colores y posiciones 6
1.5.2 Ejemplo 1.2: Minicalculadora elemental 8
1.5.3 Ejemplo 1.3: Transformación de unidades de temperatura 9
1.5.4 Ejemplo 1.4: Colores RGB 11
2. Entorno de programación Visual Basic 6.0 **14**
2.1 Introducción: ¿Qué es Visual Basic 6.0? 14
2.2 El entorno de Visual Basic 6.0 14
2.2.1 La barra de menús y las barras de herramientas 14
2.2.2 Las herramientas (toolbox) 16
2.3 Formularios (*forms*) y módulos 16
2.4 La ventana de proyecto (*project*) 17
2.5 La ventana de propiedades (*Properties*) 17
2.6 Creación de programas ejecutables 18
2.7 Cómo utilizar el Help 19
2.8 Utilización del Code Editor 19
2.9 Utilización del Debugger 21
2.9.1 Ejecución controlada de un programa 21
2.9.2 Ventanas Immediate, Locals y Watches 22
2.9.3 Otras posibilidades del Debugger 23
3. Lenguaje BASIC **24**
3.1 Introducción 24
3.2 Comentarios y otras utilidades en la programación con visual basic 24
3.3 Proyectos y módulos 25
3.3.1 Ámbito de las variables y los procedimientos 25
3.3.1.1 Variables y funciones de ámbito local 25
3.3.1.2 Variables y funciones de ámbito global 26
3.4 Variables 27
3.4.1 Identificadores 27
3.4.2 Variables y constantes 28
3.4.3 Nombres de variables 28
3.4.4 Tipos de datos 29
3.4.5 Elección del tipo de una variable 30
3.4.6 Declaración explícita de variables 30
3.5 Operadores 31
3.6 Sentencias de control 31
3.6.1 Sentencia IF ... THEN ... ELSE ... 32
3.6.2 Sentencia SELECT CASE 33
3.6.3 Sentencia FOR ... NEXT 34
3.6.4 Sentencia DO ... LOOP 34
3.6.5 Sentencia WHILE … WEND 35
3.6.6 Sentencia FOR EACH ... NEXT 35
3.7 Algoritmos 36
3.7.1 Introducción 36
3.7.2 Representación de algoritmos 36

3.8 Funciones y Procedimientos 37
3.8.1 Conceptos generales sobre funciones 37
3.8.2 Funciones y procedimientos Sub en Visual Basic 6.0 37
3.8.3 Funciones (function) 38
3.8.4 Procedimientos Sub 39
3.8.5 Argumentos por referencia y por valor 40
3.8.6 Procedimientos recursivos 40
3.8.7 Procedimientos con argumentos opcionales 41
3.8.8 Número indeterminado de argumentos 41
3.8.9 Utilización de argumentos con nombre 41
3.9 Arrays 41
3.9.1 Arrays estáticos 42
3.9.2 Arrays dinámicos 42
3.10 Estructuras: Sentencia Type 43
3.11 Funciones para manejo de cadenas de caracteres 45
3.12 Funciones matemáticas 46

4. Eventos, Propiedades y Controles 48
4.1 Eventos 48
4.1.1 Eventos generales 48
4.1.1.1 Carga y descarga de formularios 48
4.1.1.2 Paint 50
4.1.1.3 El foco (focus) 50
4.1.1.4 KeyPress, KeyUp y KeyDown 51
4.1.2 Eventos relacionados con el ratón 51
4.1.2.1 Click y DblClick 51
4.1.2.2 MouseDown, MouseUp y MouseMove 52
4.1.2.3 DragOver y DragDrop 52
4.2 Algunas propiedades comunes a varios controles 53
4.3 Controles más usuales 54
4.3.1 Botón de comando (Command Button) 54
4.3.2 Botones de opción (Option Button) 55
4.3.3 Botones de selección (Check Box) 55
4.3.4 Barras de desplazamiento (Scroll Bars) 55
4.3.5 Etiquetas (Labels) 56
4.3.6 Cajas de texto (Text Box) 56
4.3.7 Listas (List Box) 57
4.3.8 Cajas combinadas (ComboBox) 58
4.3.9 Controles relacionados con ficheros 58
4.3.10 Control Timer 58
4.4 Cajas de diálogo estándar (Controles Common Dialog) 58
4.4.1 Open/Save Dialog Control 60
4.4.2 Print Dialog Control 60
4.4.3 Font Dialog Control 61
4.4.4 Color Dialog Control 61
4.5 Formularios múltiples 61
4.5.1 Apertura de controles en forma modal 62
4.5.2 Formularios MDI (Multiple Document Interface) 62
4.6 Arrays de controles 63

5. Menús 64
5.1 Introducción a las posibilidades de los menús 64
5.2 El editor de menús (Menu Editor) 65
5.3 Añadir código a los menús 66
5.4 Arrays de menús 66
5.5 Ejemplo: Menú para determinar las características de un texto 67
5.6 Menús contextuales (Popup Menu) 68

6. Gráficos en Visual Basic 6.0 69
6.1 Tratamiento del color 69
6.1.1 Representación hexadecimal de los colores 69
6.1.2 Acceso a los colores del sistema 69

6.1.3 Función RGB 70
6.1.4 Paleta de colores 70
6.2 Formatos gráficos 71
6.3 Controles gráficos 71
Control Line 71
Control Shape 72
6.3.3 Ejemplo 6.1: Uso de los controles Line y Shape 72
Control Image 72
6.3.5 Control PictureBox 74
6.4 Métodos gráficos 74
6.4.1 Método Print 75
6.4.2 Dibujo de puntos: método PSet 75
6.4.3 Dibujo de líneas y rectángulos: método Line 75
6.4.4 Dibujo de circunferencias, arcos y elipses: método Circle 76
6.4.5 Otros métodos gráficos 77
6.5 Sistemas de coordenadas 77
6.5.1 Método Scale 78
6.6 Eventos y propiedades relacionadas con gráficos 79
6.6.1 El evento Paint 79
6.6.2 La propiedad DrawMode 79
6.6.3 Planos de dibujo (Layers) 80
6.6.4 La propiedad AutoRedraw 80
6.6.5 La propiedad ClipControl 81
6.7 Ejemplos 81
6.7.1 Ejemplo 6.1: Gráficos y barras de desplazamiento 81
6.7.2 Ejemplo 6.2: Representación gráfica de la solución de la ecuación de segundo grado 83
6.8 Barras de Herramientas (Toolbars) 86

7. Archivos y Entrada/Salida de Datos 87
7.1 Cajas de diálogo InputBox y MsgBox 87
7.2 Método Print 88
7.2.1 Características generales 88
7.2.2 Función Format 89
7.3 Utilización de impresoras 90
7.3.1 Método PrintForm 90
7.3.2 Objeto Printer 90
7.4 Controles FileList, DirList y DriveList 91
7.5 Tipos de ficheros 92
7.6 Lectura y escritura en ficheros secuenciales 93
7.6.1 Apertura y cierre de ficheros 93
7.6.2 Lectura y escritura de datos 93
7.6.2.1 Sentencia Input 93
7.6.2.2 Función Line Input y función Input 94
7.6.2.3 Función Print # 94
7.6.2.4 Función Write # 95
7.7 Ficheros de acceso aleatorio 95
7.7.1 Abrir y cerrar archivos de acceso aleatorio 95
7.7.2 Leer y escribir en una archivo de acceso aleatorio. Funciones Get y Put 95
7.8 Ficheros de acceso binario 96

8. ANEXO A: Consideraciones adicionales sobre datos y variables 97
8.1 Caracteres y código ASCII 97
8.2 Números enteros 98
8.3 Números reales 98
8.3.1 Variables tipo Single 98
8.3.2 Variables tipo Double 99
8.4 Sistema binario, octal, decimal y hexadecimal 99

1. INTRODUCCIÓN

Visual Basic 6.0 es uno de los lenguajes de programación que más entusiasmo despiertan entre los programadores de PCs, tanto expertos como novatos. En el caso de los programadores expertos por la facilidad con la que desarrollan aplicaciones complejas en poquísimo tiempo (comparado con lo que cuesta programar en ***Visual C++***, por ejemplo). En el caso de los programadores novatos por el hecho de ver de lo que son capaces a los pocos minutos de empezar su aprendizaje. El precio que hay que pagar por utilizar ***Visual Basic 6.0*** es una menor velocidad o eficiencia en las aplicaciones.

Visual Basic 6.0 es un lenguaje de programación visual, también llamado lenguaje de 4ª generación. Esto quiere decir que un gran número de tareas se realizan sin escribir código, simplemente con operaciones gráficas realizadas con el ratón sobre la pantalla.

Visual Basic 6.0 es también un programa *basado en objetos*, aunque no *orientado a objetos* como ***C++*** o ***Java***. La diferencia está en que ***Visual Basic 6.0*** utiliza ***objetos*** con ***propiedades*** y ***métodos***, pero carece de los mecanismos de ***herencia*** y ***polimorfismo*** propios de los verdaderos lenguajes orientados a objetos como ***Java*** y *C++*.

En este primer capítulo se presentarán las características generales de ***Visual Basic 6.0***, junto con algunos ejemplos sencillos que den idea de la potencia del lenguaje y del modo en que se utiliza.

1.1 PROGRAMAS SECUENCIALES, INTERACTIVOS Y ORIENTADOS A EVENTOS

Existen distintos tipos de programas. En los primeros tiempos de los ordenadores los programas eran de tipo ***secuencial*** (también llamados tipo ***batch***) Un programa secuencial es un programa que se arranca, lee los datos que necesita, realiza los cálculos e imprime o guarda en el disco los resultados. De ordinario, mientras un programa secuencial está ejecutándose no necesita ninguna intervención del usuario. A este tipo de programas se les llama también *programas basados u orientados a procedimientos* o *a algoritmos* (*procedural languages*). Este tipo de programas siguen utilizándose ampliamente en la actualidad, pero la difusión de los PCs ha puesto de actualidad otros tipos de programación.

Los programas ***interactivos*** exigen la intervención del usuario en tiempo de ejecución, bien para suministrar datos, bien para indicar al programa lo que debe hacer por medio de menús. Los programas interactivos limitan y orientan la acción del usuario. Un ejemplo de programa interactivo podría ser ***Matlab***.

Por su parte los programas ***orientados a eventos*** son los programas típicos de ***Windows***, tales como ***Netscape***, ***Word***, ***Excel*** y ***PowerPoint***. Cuando uno de estos programas ha arrancado, lo único que hace es quedarse a la espera de las acciones del usuario, que en este caso son llamadas ***eventos***. El usuario dice si quiere abrir y modificar un fichero existente, o bien comenzar a crear un fichero desde el principio. Estos programas pasan la mayor parte de su tiempo esperando las acciones del usuario (eventos) y respondiendo a ellas. Las acciones que el usuario puede realizar en un momento determinado son variadísimas, y exigen un tipo especial de programación: ***la programación orientada a eventos***. Este tipo de programación es sensiblemente más complicada que la secuencial y la interactiva, pero ***Visual Basic 6.0*** la hace especialmente sencilla y agradable.

1.2 PROGRAMAS PARA EL ENTORNO WINDOWS

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

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

1.2.1 Modo de Diseño y Modo de Ejecución

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

La aplicación se prueba en ***modo de ejecución***. En ese caso el usuario actúa sobre el programa (introduce ***eventos***) y prueba cómo responde el programa. Hay algunas ***propiedades*** de los ***controles*** que deben establecerse en modo de diseño, pero muchas otras pueden cambiarse en tiempo de ejecución desde el programa escrito en ***Visual Basic 6.0***, en la forma en que más adelante se verá. También hay ***propiedades*** que sólo pueden establecerse en modo de ejecución y que no son visibles en modo de diseño.

Todos estos conceptos –***controles***, ***propiedades***, ***eventos***, etc.- se explican en los apartados siguientes.

1.2.2 Formularios y Controles

Cada uno de los elementos gráficos que pueden formar parte de una aplicación típica de ***Windows 95/98/NT*** es un tipo de ***control***: los botones, las cajas de diálogo y de texto, las cajas de selección desplegables, los botones de opción y de selección, las barras de desplazamiento horizontales y verticales, los gráficos, los menús, y muchos otros tipos de elementos son controles para ***Visual Basic 6.0***. Cada control debe tener un ***nombre*** a través del cual se puede hacer referencia a él en el programa. ***Visual Basic 6.0*** proporciona nombres *por defecto* que el usuario puede modificar. En el Apartado 1.2.4 se exponen algunas reglas para dar nombres a los distintos controles.

En la terminología de ***Visual Basic 6.0*** se llama ***formulario*** (*form*) a una ventana. Un formulario puede ser considerado como una especie de contenedor para los controles. Una aplicación puede tener varios formularios, pero un único formulario puede ser suficiente para las aplicaciones más sencillas. Los formularios deben también tener un nombre, que puede crearse siguiendo las mismas reglas que para los controles.

1.2.3 Objetos y Propiedades

Los formularios y los distintos tipos de controles son entidades genéricas de las que puede haber varios ejemplares concretos en cada programa. En *programación orientada a objetos* (más bien *basada en objetos*, habría que decir) se llama ***clase*** a estas entidades genéricas, mientras que se llama ***objeto*** a cada ejemplar de una clase determinada. Por ejemplo, en un programa puede haber

varios botones, cada uno de los cuales es un ***objeto*** del tipo de control ***command button***, que sería la ***clase***.

Cada formulario y cada tipo de control tienen un conjunto de ***propiedades*** que definen su aspecto gráfico (tamaño, color, posición en la ventana, tipo y tamaño de letra, etc.) y su forma de responder a las acciones del usuario (si está activo o no, por ejemplo). Cada propiedad tiene un ***nombre*** que viene ya definido por el lenguaje.

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

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

Se puede ***acceder a una propiedad*** de un objeto por medio del ***nombre del objeto*** a que pertenece, seguido de un ***punto*** y el ***nombre de la propiedad***, como por ejemplo ***optColor.objName***. En el siguiente apartado se estudiarán las reglas para dar nombres a los objetos.

1.2.4 Nombres de objetos

En principio cada objeto de ***Visual Basic 6.0*** debe tener un nombre, por medio del cual se hace referencia a dicho objeto. El nombre puede ser el que el usuario desee, e incluso ***Visual Basic 6.0*** proporciona ***nombres por defecto*** para los diversos controles. Estos nombres por defecto hacen referencia al tipo de control y van seguidos de un número que se incrementa a medida que se van introduciendo más controles de ese tipo en el formulario (por ejemplo ***VScroll1***, para una barra de desplazamiento -*scroll bar*- vertical, ***HScroll1***, para una barra horizontal, etc.).

Los ***nombres por defecto no son adecuados*** porque hacen referencia al tipo de control, pero no al uso que de dicho control está haciendo el programador. Por ejemplo, si se utiliza una barra de desplazamiento para introducir una temperatura, conviene que su nombre haga referencia a la palabra ***temperatura***, y así cuando haya que utilizar ese nombre se sabrá exactamente a qué control corresponde. Un nombre adecuado sería por ejemplo ***hsbTemp***, donde las tres primeras letras indican que se trata de una *horizontal scroll bar*, y las restantes (empezando por una mayúscula) que servirá para definir una *temperatura*.

Existe una convención ampliamente aceptada que es la siguiente: *se utilizan siempre tres letras minúsculas que indican el tipo de control, seguidas por otras letras (la primera mayúscula, a modo de separación) libremente escogidas por el usuario, que tienen que hacer referencia al uso que se va a dar a ese control*. La Tabla 1.1 muestra las abreviaturas de los controles más usuales, junto con la nomenclatura inglesa de la que derivan. En este mismo capítulo se verán unos cuantos ejemplos de aplicación de estas reglas para construir nombres.

Abreviatura	Control	Abreviatura	Control
chk	check box	cbo	combo y drop-list box
cmd	command button	dir	dir list box
drv	drive list box	fil	file list box
frm	form	fra	frame
hsb	horizontal scroll bar	img	image
lbl	label	lin	line
lst	list	mnu	menu
opt	option button	pct	pictureBox
shp	shape	txt	text edit box
tmr	timer	vsb	vertical scroll bar

Tabla 1.1. Abreviaturas para los controles más usuales.

1.2.5 Eventos

Ya se ha dicho que las acciones del usuario sobre el programa se llaman ***eventos***. Son eventos típicos el clicar sobre un botón, el hacer doble clic sobre el nombre de un fichero para abrirlo, el arrastrar un icono, el pulsar una tecla o combinación de teclas, el elegir una opción de un menú, el escribir en una caja de texto, o simplemente mover el ratón. Más adelante se verán los distintos tipos de eventos reconocidos por ***Windows 95/98/NT*** y por ***Visual Basic 6.0***.

Cada vez que se produce un evento sobre un determinado tipo de control, ***Visual Basic 6.0*** arranca una determinada ***función*** o ***procedimiento*** que realiza la acción programada por el usuario para ese evento concreto. Estos procedimientos se llaman con un nombre que se forma a partir del nombre del objeto y el nombre del evento, separados por el carácter (_), como por ejemplo ***txtBox_click***, que es el nombre del procedimiento que se ocupará de responder al evento ***click*** en el objeto ***txtBox***.

1.2.6 Métodos

Los ***métodos*** son funciones que también son llamadas desde programa, pero a diferencia de los procedimientos no son programadas por el usuario, sino que vienen ya pre-programadas con el lenguaje. Los métodos realizan tareas típicas, previsibles y comunes para todas las aplicaciones. De ahí que vengan con el lenguaje y que se libere al usuario de la tarea de programarlos. Cada tipo de objeto o de control tiene sus propios métodos.

Por ejemplo, los controles gráficos tienen un método llamado ***Line*** que se encarga de dibujar líneas rectas. De la misma forma existe un método llamado ***Circle*** que dibuja circunferencias y arcos de circunferencia Es obvio que el dibujar líneas rectas o circunferencias es una tarea común para todos los programadores y que ***Visual Basic 6.0*** da ya resuelta.

1.2.7 Proyectos y ficheros

Cada aplicación que se empieza a desarrollar en ***Visual Basic 6.0*** es un nuevo ***proyecto***. Un proyecto comprende otras componentes más sencillas, como por ejemplo los ***formularios*** (que son las ventanas de la interface de usuario de la nueva aplicación) y los ***módulos*** (que son conjuntos de funciones y procedimientos sin interface gráfica de usuario).

¿Cómo se guarda un proyecto en el disco? Un proyecto se compone siempre de ***varios ficheros*** (al menos de dos) y hay que preocuparse de guardar cada uno de ellos en el directorio

adecuado y con el nombre adecuado. Existe siempre un fichero con extensión ***.vbp*** (*Visual Basic Project*) que se crea con el comando ***File/Save Project As***. El fichero del proyecto contiene toda la *información de conjunto*. Además hay que crear *un* ***fichero por cada formulario y por cada módulo*** que tenga el proyecto. Los ficheros de los formularios se crean con ***File/Save Filename As*** teniendo como extensión ***.frm***. Los ficheros de código o ***módulos*** se guardan también con el comando ***File/Save Filename As*** y tienen como extensión ***.bas*** si se trata de un ***módulo estándar*** o ***.cls*** si se trata de un *módulo de clase* (***class module***).

Clicando en el botón ***Save*** en la barra de herramientas se actualizan todos los ficheros del proyecto. Si no se habían guardado todavía en el disco, ***Visual Basic 6.0*** abre cajas de diálogo ***Save As*** por cada uno de los ficheros que hay que guardar.

1.3 EL ENTORNO DE PROGRAMACIÓN VISUAL BASIC 6.0

Cuando se arranca ***Visual Basic 6.0*** aparece en la pantalla una configuración similar a la mostrada en la Figura 1.1. En ella se pueden distinguir los siguientes elementos:

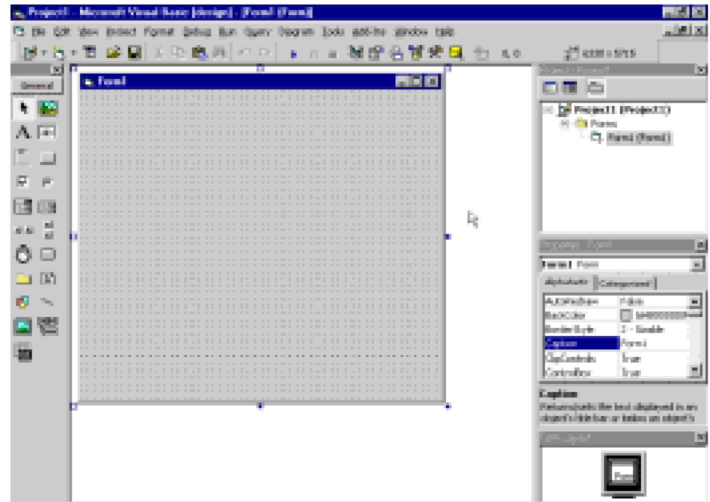


Figura 1.1. Entorno de programación de ***Visual Basic 6.0***.

1. La *barra de títulos*, la *barra de menús* y la *barra de herramientas* de ***Visual Basic 6.0*** en modo ***Diseño*** (parte superior de la pantalla).
2. *Caja de herramientas* (***toolbox***) con los controles disponibles (a la izquierda de la ventana).
3. *Formulario* (***form***) en gris, en que se pueden ir situando los controles (en el centro). Está dotado de una rejilla (***grid***) para facilitar la alineación de los controles.
4. Ventana de *proyecto*, que muestra los formularios y otros módulos de programas que forman parte de la aplicación (arriba a la derecha).
5. Ventana de *Propiedades*, en la que se pueden ver las propiedades del objeto seleccionado o del propio formulario (en el centro a la derecha). Si esta ventana no aparece, se puede hacer visible con la tecla <F4>.
6. Ventana *FormLayout*, que permite determinar la forma en que se abrirá la aplicación cuando comience a ejecutarse (abajo a la derecha).

Existen otras ventanas para edición de código (***Code Editor***) y para ver variables en tiempo de ejecución con el *depurador* o ***Debugger*** (ventanas ***Immediate***, ***Locals*** y ***Watch***). Todo este conjunto de herramientas y de ventanas es lo que se llama un *entorno integrado de desarrollo* o IDE (***Integrated Development Environment***).

Construir aplicaciones con ***Visual Basic 6.0*** es muy sencillo: basta crear los controles en el formulario con ayuda de la ***toolbox*** y del ratón, establecer sus ***propiedades*** con ayuda de la ventana de propiedades y programar el ***código*** que realice las acciones adecuadas en respuesta a los ***eventos*** o acciones que realice el usuario. A continuación, tras explicar brevemente cómo se utiliza el ***Help*** de ***Visual Basic***, se presentan algunos ejemplos ilustrativos.

1.4 EL HELP DE VISUAL BASIC 6.0

El ***Help*** de ***Visual Basic 6.0*** es de los mejores que existen. Además de que se puede buscar cualquier tipo de información con la función ***Index***, basta seleccionar una propiedad cualquiera en la ventana de propiedades o un control cualquiera en el formulario (o el propio formulario), para que pulsando la tecla <F1> aparezca una ventana de ayuda muy completa. De cada control de muestran las propiedades, métodos y eventos que soporta, así como ejemplos de aplicación. También se muestra información similar o relacionada.

Existe además un breve pero interesante curso introductorio sobre ***Visual Basic 6.0*** que se activa con la opción ***Help/Contents***, seleccionando luego MSDN ***Contents/Visual Basic Documentation/Visual Basic Start Page/Getting Started***.

1.5 EJEMPLOS

El entorno de programación de ***Visual Basic 6.0*** ofrece muchas posibilidades de adaptación a los gustos, deseos y preferencias del usuario. Los usuarios expertos tienen siempre una forma propia de hacer las cosas, pero para los usuarios noveles conviene ofrecer unas ciertas orientaciones al respecto. Por eso, antes de realizar los ejemplos que siguen se recomienda modificar la configuración de ***Visual Basic 6.0*** de la siguiente forma:

1. En el menú ***Tools*** elegir el comando ***Options***; se abre un cuadro de diálogo con 6 solapas.
2. En la solapa ***Environment*** elegir "***Promp to Save Changes***" en "***When a Program Starts***" para que pregunte antes de cada ejecución si se desean guardar los cambios realizados. En la solada ***Editor*** elegir también "***Require Variable Declaration***" en "***Code Settings***" para evitar errores al teclear los nombres de las variables.
3. En la solapa ***Editor***, en ***Code Settings***, dar a "***Tab Width***" un valor de 4 y elegir la opción "***Auto Indent***" (para que ayude a mantener el código legible y ordenado). En ***Windows Settings*** elegir "***Default to Full Module View***" (para ver todo el código de un formulario en una misma ventana) y "***Procedure Separator***" (para que separe cada función de las demás mediante una línea horizontal).

1.5.1 Ejemplo 1.1: Sencillo programa de colores y posiciones

En la Figura 1.2 se muestra el formulario y los controles de un ejemplo muy sencillo que permite mover una caja de texto por la pantalla, permitiendo a su vez representarla con cuatro colores diferentes.

En la Tabla 1.2 se describen los controles utilizados, así como algunas de sus propiedades más importantes (sobre todo las que se separan de los valores por defecto). Los ficheros de este proyecto se llamarán ***Colores0.vbp*** y ***Colores0.frm***.

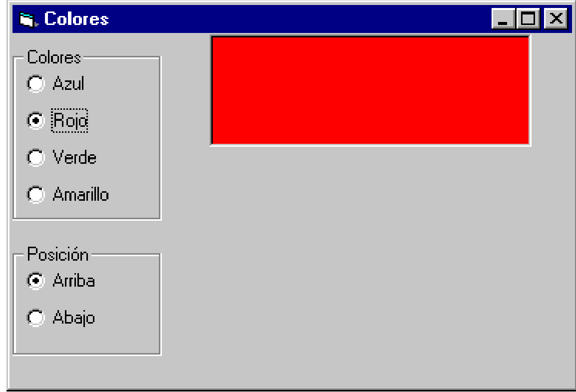


Figura 1.2. Formulario y controles del Ejemplo 1.1.

Control	Propiedad	Valor	Control	Propiedad	Valor
frmColores0	Name	frmColores0	optVerde	Name	optVerde
	Caption	Colores		Caption	Verde
fraColores	Name	fraColor	fraPosicion	Name	fraPosicion
	Caption	Colores		Caption	Posición
optAzul	Name	optAzul	optArriba	Name	optArriba
	Caption	Azul		Caption	Arriba
optRojo	Name	optRojo	optAbajo	Name	optAbajo
	Caption	Rojo		Caption	Abajo
optAmarillo	Name	optAmarillo	txtCaja	Name	txtCaja
	Caption	Amarillo		Text	“”

Tabla 1.2. Objetos y propiedades del ejemplo ***Colores0***.

A continuación se muestra el código correspondiente a los procedimientos de este ejemplo.

```
Option Explicit
Private Sub Form_Load()
    txtCaja.Top = 0
End Sub

Private Sub optArriba_Click()
    txtCaja.Top = 0
End Sub

Private Sub optAbajo_Click()
    txtCaja.Top = frmColores0.ScaleHeight - txtCaja.Height
End Sub

Private Sub optAzul_Click()
    txtCaja.BackColor = vbBlue
End Sub

Private Sub optRojo_Click()
    txtCaja.BackColor = vbRed
End Sub

Private Sub optVerde_Click()
    txtCaja.BackColor = vbGreen
End Sub

Private Sub optAmarillo_Click()
    txtCaja.BackColor = vbYellow
End Sub
```

Sobre este primer programa en ***Visual Basic 6.0*** se pueden hacer algunos comentarios:

1. El comando ***Option Explicit*** sirve para obligar a ***declarar*** todas las variables que se utilicen. Esto impide el cometer errores en los nombres de las variables (confundir *masa* con *mesa*, por ejemplo). En este ejemplo esto no tiene ninguna importancia, pero es conveniente acostumbrarse a incluir esta opción. ***Declarar una variable*** es crearla con un nombre y de un tipo determinado antes de utilizarla.

2. Cada una de las partes de código que empieza con un ***Private Sub*** y termina con un ***End Sub*** es un ***procedimiento***, esto es, una parte de código independiente y reutilizable. El nombre de uno de estos procedimientos, por ejemplo ***optAzul_Click()***, es típico de ***Visual Basic***. La primera parte es el nombre de un objeto (control); después va un separador que es el carácter de subrayado (_); a continuación el nombre de un evento ***-click***, en este caso-, y finalmente unos paréntesis entre los que irían los argumentos, en caso de que los hubiera.

3. Es también interesante ver cómo se accede desde programa a la propiedad ***backColor*** de la caja de texto que se llama ***txtCaja***: se hace utilizando el punto en la forma ***txtCaja.BackColor***. Los colores se podrían también introducir con notación hexadecimal (comenzando con &H, seguidos por dos dígitos entre 00 y FF (es decir, entre 0 y 255 en base 10) para los tres colores fundamentales, es decir para el ***Red***, ***Green*** y ***Blue*** (RGB), de derecha a izquierda. Aquí se han utilizado las constantes simbólicas predefinidas en ***Visual Basic 6.0***: ***vbRed***, ***vbGreen*** y ***vbBlue*** (ver Tabla 6.1, en la página 69).
4. Recuérdese que si se desea que el código de todos los eventos aparezca en una misma ventana hay que activar la opción ***Default to Full Module View*** en la solapa ***Editor*** del comando ***Tools/Options***. También puede hacerse directamente en la ventana de código con uno de los botones que aparecen en la parte inferior izquierda ( ).
5. ***Es muy importante*** crear primero el control ***frame*** y después, estando seleccionado, colocar los ***botones de opción*** en su interior. No sirve hacerlo a la inversa. ***Visual Basic*** supone que todos los botones de opción que están dentro del mismo ***frame*** forman parte del mismo grupo y sólo permite que uno esté seleccionado.

1.5.2 Ejemplo 1.2: Minicalculadora elemental

En este ejemplo se muestra una calculadora elemental que permite hacer las cuatro operaciones aritméticas (Figura 1.3). Los ficheros de este proyecto se pueden llamar ***minicalc.vbp*** y ***minicalc.frm***.

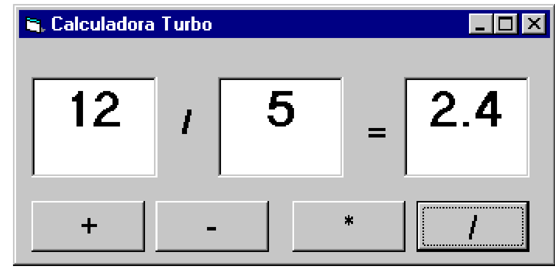


Figura 1.3. Minicalculadora elemental.

El usuario introduce los datos y clica sobre el botón correspondiente a la operación que desea realizar, apareciendo inmediatamente el resultado en la caja de texto de la derecha.

La Tabla 1.3 muestra los objetos y las propiedades más importantes de este ejemplo.

Control	Propiedad	Valor	Control	Propiedad	Valor
frmMinicalc	Name	frmMinicalc	lblEqual	Name	lblEqual
	Caption	Minicalculadora		Caption	=
txtOper1	Name	txtOper1	cmdSuma	Name	cmdSuma
	Text			Caption	+
txtOper2	Name	txtOper2	cmdResta	Name	cmdResta
	Text			Caption	-
txtResult	Name	txtResult	cmdMulti	Name	cmdProd
	Text			Caption	*
lblOp	Name	lblOp	cmdDivi	Name	cmdDiv
	Caption			Caption	/

Tabla 1.3. Objetos y propiedades del ejemplo Minicalculadora.

A continuación se muestra el código correspondiente a los procedimientos que gestionan los eventos de este ejemplo.

```
Option Explicit

Private Sub cmdDiv_Click()
   txtResult.Text = Val(txtOper1.Text) / Val(txtOper2.Text)
   lblOp.Caption = "/"
End Sub
```

```
Private Sub cmdProd_Click()
   txtResult.Text = Val(txtOper1.Text) * Val(txtOper2.Text)
   lblOp.Caption = "*"
End Sub

Private Sub cmdResta_Click()
   txtResult.Text = Val(txtOper1.Text) - Val(txtOper2.Text)
   lblOp.Caption = "-"
End Sub

Private Sub cmdSuma_Click()
   txtResult.Text = Val(txtOper1.Text) + Val(txtOper2.Text)
   lblOp.Caption = "+"
End Sub
```

En este ejemplo se ha utilizado repetidamente la función ***Val()*** de ***Visual Basic***. Esta función convierte una serie de caracteres numéricos (un texto formado por cifras) en el número entero o de punto flotante correspondiente. Sin la llamada a la función ***Val()*** el ***operador*** **+** aplicado a cadenas de caracteres las concatena, y como resultado, por ejemplo, "3+4" daría "34". No es lo mismo los caracteres "1" y "2" formando la *cadena* o *string* "12" que el número 12; la función ***val()*** convierte cadenas de caracteres numéricos –con los que no se pueden realizar operaciones aritméticas- en los números correspondientes –con los que sí se puede operar matemáticamente-. ***Visual Basic 6.0*** transforma de modo automático números en cadenas de caracteres y viceversa, pero este es un caso en el que dicha transformación no funciona porque el operador "+" tiene sentido tanto con números como con cadenas.

1.5.3 Ejemplo 1.3: Transformación de unidades de temperatura

La Figura 1.4 muestra un programa sencillo que permite ver la equivalencia entre las escalas de temperaturas en grados centígrados y grados Fahrenheit. Los ficheros de este proyecto se pueden llamar ***Temperat.vbp*** y ***Temperat.frm***.

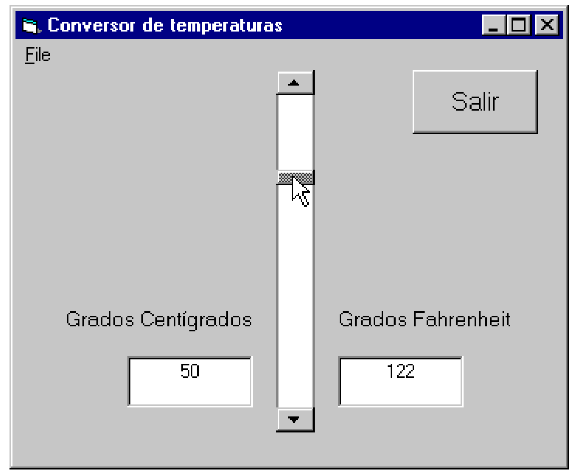


Figura 1.4. Equivalencia de temperaturas.

En el centro del formulario aparece una barra de desplazamiento vertical que permite desplazarse con incrementos pequeños de 1º C y grandes de 10º C. Como es habitual, también puede cambiarse el valor arrastrando con el ratón el cursor de la barra. Los valores máximos y mínimo de la barra son 100º C y -100º C.

A ambos lados de la barra aparecen dos cuadros de texto (color de fondo blanco) donde aparecen los grados correspondientes a la barra en ambas escalas. Encima aparecen dos rótulos (*labels*) que indican la escala de temperaturas correspondiente. Completan la aplicación un botón ***Salir*** que termina la ejecución y un menú ***File*** con la única opción ***Exit***, que termina asimismo la ejecución del programa.

La Tabla 1.4 indica los controles utilizados en este ejemplo junto con las propiedades y los valores correspondientes.

Control	Propiedad	Valor	Control	Propiedad	Valor
frmTemp	Name	frmTemp	vsbTemp	Name	vsbTemp
	Caption	Conversor de temperaturas		Min	100
mnuFile	Name	mnuFile		Max	-100
	Caption	&File		SmallChange	1
mnuFileExit	Name	mnuFileExit		LargeChange	10
	Caption	E&xit		Value	0
cmdSalir	Name	cmdSalir	lblCent	Name	lblCent
	Caption	Salir		Caption	Grados Centígrados
	Font	MS Sans Serif, Bold, 14		Font	MS Sans Serif, 10
txtCent	Name	txtCent	lblFahr	Name	lblFahr
	text	0		Caption	Grados Fahrenheit
txtFahr	Name	txtFahr		Font	MS Sans Serif, 10
	text	32			

Tabla 1.4. Controles y propiedades del Ejemplo 1.3.

Por otra parte, el código con el que este programa responde a los eventos es el contenido en los siguientes procedimientos:

```
Option Explicit

Private Sub cmbSalir_Click()
   Beep
   End
End Sub

Private Sub mnuFileExit_Click()
   End
End Sub

Private Sub vsbTemp_Change()
    txtCent.Text = vsbTemp.Value
    txtFahr.Text = 32 + 1.8 * vsbTemp.Value
End Sub
```

Sobre este tercer ejemplo se puede comentar lo siguiente:

1. Se ha utilizado la propiedad ***Value*** de la barra de desplazamiento, la cual da el valor actual de la misma con respecto a los límites inferior y superior, previamente establecidos (-100 y 100).
2. Mediante el procedimiento ***cmdSalir_Click***, se cierra el programa, gracias a la instrucción ***End***. El cometido de ***Beep*** no es otro que el de emitir un pitido a través del altavoz del ordenador, que indicará que en efecto se ha salido del programa.
3. La función ***mnuFileExit_Click()*** y activa desde el menú y termina la ejecución sin emitir ningún sonido.
4. Finalmente, la función ***vsbTemp_Change()*** se activa al cambiar el valor de la barra de desplazamiento; su efecto es modificar el valor de la propiedad ***text*** en las cajas de texto que muestran la temperatura en cada una de las dos escalas.

1.5.4 Ejemplo 1.4: Colores RGB

La Figura 1.5 muestra el formulario y los controles del proyecto ***Colores***. Los ficheros de este proyecto se pueden llamar ***Colores.vbp*** y ***Colores.frm***.

En este ejemplo se dispone de tres barras de desplazamiento con las que pueden controlarse las componentes RGB del color del fondo y del color del texto de un control ***label***. Dos botones de opción permiten determinar si los valores de las barras se aplican al fondo o al texto. Cuando se cambia del texto al fondo o viceversa los valores de las barras de desplazamiento (y la posición de los cursores) cambian de modo acorde.

Figura 1.5. Colores de fondo y de texto.

A la dcha. de las barras de desplazamiento tres cajas de texto contienen los valores numéricos de los tres colores (entre 0 y 255). A la izda. tres ***labels*** indican los colores de las tres barras. La Tabla 1.5 muestra los controles y las propiedades utilizadas en el este ejemplo.

Control	Propiedad	Valor	Control	Propiedad	Valor
frmColores	Name	frmColores	hsbColor	Name	hsbColor
	Caption	Colores		Min	0
lblCuadro	Name	lblCuadro		Max	255
	Caption	INFORMÁTICA 1		SmallChange	1
	Font	MS Sans Serif, Bold, 24		LargeChange	16
cmdSalir	Name	cmdSalir		Index	0,1,2
	Caption	Salir		Value	0
	Font	MS Sans Serif, Bold, 10	txtColor	Name	txtColor
optColor	Name	optColor		Text	0
	Index	0,1		Locked	True
	Caption	Fondo, Texto		Index	0,1,2
	Font	MS Sans Serif, Bold, 10	lblColor	Name	lblColor
				Caption	Rojo,Verde,Azul
				Index	0,1,2
				Font	MS Sans Serif, 10

Tabla 1.5. Objetos y propiedades del ejemplo ***Colores***.

Una característica importante de este ejemplo es que se han utilizado ***vectores (arrays) de controles***. Las tres barras se llaman ***hsbColor*** y se diferencian por la propiedad ***Index***, que toma los valores 0, 1 y 2. También las tres cajas de texto, las tres ***labels*** y los dos botones de opción son ***arrays de controles***. Para crear un array de controles basta crear el primero de ellos y luego hacer ***Copy*** y ***Paste*** tantas veces como se desee, respondiendo afirmativamente al cuadro de diálogo que pregunta si desea crear un array.

El ***procedimiento Sub*** que contiene el código que gestiona un ***evento*** de un array es único para todo el array, y recibe como argumento la propiedad ***Index***. De este modo que se puede saber exactamente en qué control del array se ha producido el evento. Así pues, una ventaja de los *arrays* de controles es que pueden compartir el código de los eventos y permitir un tratamiento conjunto

por medio de bucles ***for***. A continuación se muestra el código correspondiente a los procedimientos que tratan los eventos de este ejemplo.

```
Option Explicit
Public Brojo, Bverde, Bazul As Integer
Public Frojo, Fverde, Fazul As Integer

Private Sub cmdSalir_Click()
   End
End Sub

Private Sub Form_Load()
   Brojo = 0
   Bverde = 0
   Bazul = 0
   Frojo = 255
   Fverde = 255
   Fazul = 255
   lblCuadro.BackColor = RGB(Brojo, Bverde, Bazul)
   lblCuadro.ForeColor = RGB(Frojo, Fverde, Fazul)
End Sub

Private Sub hsbColor_Change(Index As Integer)
   If optColor(0).Value = True Then
      lblCuadro.BackColor = RGB(hsbColor(0).Value, hsbColor(1).Value, _
                          hsbColor(2).Value)
      Dim i As Integer
      For i = 0 To 2
         txtColor(i).Text = hsbColor(i).Value
      Next i
   Else
      lblCuadro.ForeColor = RGB(hsbColor(0).Value, hsbColor(1).Value, _
                          hsbColor(2).Value)
      For i = 0 To 2
         txtColor(i).Text = hsbColor(i).Value
      Next i
   End If
End Sub

Private Sub optColor_Click(Index As Integer)
   If Index = 0 Then  'Se pasa a cambiar el fondo
      Frojo = hsbColor(0).Value
      Fverde = hsbColor(1).Value
      Fazul = hsbColor(2).Value
      hsbColor(0).Value = Brojo
      hsbColor(1).Value = Bverde
      hsbColor(2).Value = Bazul
   Else               'Se pasa a cambiar el texto
      Brojo = hsbColor(0).Value
      Bverde = hsbColor(1).Value
      Bazul = hsbColor(2).Value
      hsbColor(0).Value = Frojo
      hsbColor(1).Value = Fverde
      hsbColor(2).Value = Fazul
   End If
End Sub
```

El código de este ejemplo es un poco más complicado que el de los ejemplos anteriores y requiere unas ciertas explicaciones adicionales adelantando cuestiones que se verán posteriormente:

1. La función ***RGB()*** crea un ***código de color*** a partir de sus argumentos: las componentes RGB (*Red*, *Green* and *Blue*). Estas componentes, cuyo valor se almacena en un byte y puede oscilar entre 0 y 255, se determinan por medio de las tres barras de desplazamiento.

2. El color ***blanco*** se obtiene con los tres colores fundamentales a su máxima intensidad. El color ***negro*** se obtiene con los tres colores RGB a cero. También se pueden introducir con las constantes predefinidas ***vbWhite*** y ***vbBlack***, respectivamente.
3. Es importante disponer de unas ***variables globales*** que almacenen los colores del fondo y del texto, y que permitan tanto guardar los valores anteriores de las barras como cambiar éstas a sus nuevos valores cuando se clica en los botones de opción. Las variables globales, definidas en la parte de definiciones generales del código, fuera de cualquier procedimiento, son visibles desde cualquier parte del programa. Las variables definidas dentro de una función o procedimiento sólo son visibles desde dentro de dicha función o procedimiento (*variables locales*).
4. La función ***hsbColor_Change(Index As Integer)*** se activa cada vez que se cambia el valor en una cualquiera de las barras de desplazamiento. El argumento ***Index***, que ***Visual Basic*** define automáticamente, indica cuál de las barras del array es la que ha cambiado de valor (la 0, la 1 ó la 2). En este ejemplo dicho argumento no se ha utilizado, pero está disponible por si se hubiera querido utilizar en el código.

2. ENTORNO DE PROGRAMACIÓN VISUAL BASIC 6.0

2.1 INTRODUCCIÓN: ¿QUÉ ES VISUAL BASIC 6.0?

Visual Basic 6.0 es una excelente herramienta de programación que permite crear aplicaciones propias (programas) para ***Windows 95/98*** o ***Windows NT***. Con ella se puede crear desde una simple calculadora hasta una hoja de cálculo de la talla de ***Excel*** (en sus primeras versiones...), pasando por un procesador de textos o cualquier otra aplicación que se le ocurra al programador. Sus aplicaciones en Ingeniería son casi ilimitadas: representación de movimientos mecánicos o de funciones matemáticas, gráficas termodinámicas, simulación de circuitos, etc.

Este programa permite crear ventanas, botones, menús y cualquier otro elemento de ***Windows*** de una forma fácil e intuitiva. El lenguaje de programación que se utilizará será el ***Basic***, que se describirá en el siguiente capítulo.

A continuación se presentarán algunos aspectos del entorno de trabajo de ***Visual Basic 6.0***: menús, opciones, herramientas, propiedades, etc.

2.2 EL ENTORNO DE VISUAL BASIC 6.0

Visual Basic 6.0 tiene todos los elementos que caracterizan a los programas de ***Windows*** e incluso alguno menos habitual. En cualquier caso, el entorno de ***Visual Basic 6.0*** es muy lógico y natural, y además se puede obtener una descripción de la mayoría de los elementos clicando en ellos para seleccionarlos y pulsando luego la tecla <F1>.

2.2.1 La barra de menús y las barras de herramientas

La ***barra de menús*** de ***Visual Basic 6.0*** resulta similar a la de cualquier otra aplicación de ***Windows***, tal y como aparece en la Figura 2.2. Bajo dicha barra aparecen las ***barras de herramientas***, con una serie de botones que permiten acceder fácilmente a las opciones más importantes de los menús. En ***Visual Basic 6.0*** existen cuatro barras de herramientas: ***Debug***, ***Edit***, ***Form Editor*** y ***Standard***. Por defecto sólo aparece la barra ***Standard***, aunque en la Figura 2.2 se muestran las cuatro. Clicando con el botón derecho sobre cualquiera de las barras de herramientas aparece un menú contextual con el que se puede hacer aparecer y ocultar cualquiera de las barras. Al igual que en otras aplicaciones de ***Windows 95/98/NT***, también pueden modificarse las barras añadiendo o eliminando botones (opción ***Customize***).

En la barra de herramientas ***Standard*** también se pueden ver a la derecha dos recuadros con números, que representan cuatro propiedades del formulario referentes a su posición y tamaño que se verán más adelante: ***Top*** y ***Left***, que indican la posición de la esquina superior izquierda del formulario, y también ***Height*** y ***Width***, que describen el tamaño del mismo en unas unidades llamadas ***twips***, que se corresponden con la vigésima parte de un ***punto*** (una pulgada tiene 72 puntos y 1440 twips). Las dimensiones de un control aparecen también cuando con el ratón se arrastra sobre el formulario, según se puede ver en la Figura 2.1. Los botones de la barra de herramientas ***Standard*** responden a las funciones más importantes: abrir y/o guardar nuevos proyectos, añadir formularios, hacer visibles las distintas ventanas del entorno de desarrollo, etc. Todos los botones tienen su correspondiente comando en

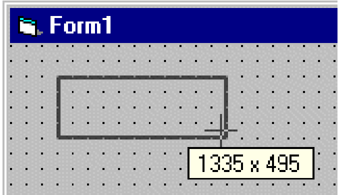


Figura 2.1. Información visual sobre el tamaño de un control.

alguno de los menús. Son importantes los botones que permiten arrancar y/o parar la ejecución de un proyecto, pasando de modo diseño a modo de ejecución y viceversa.

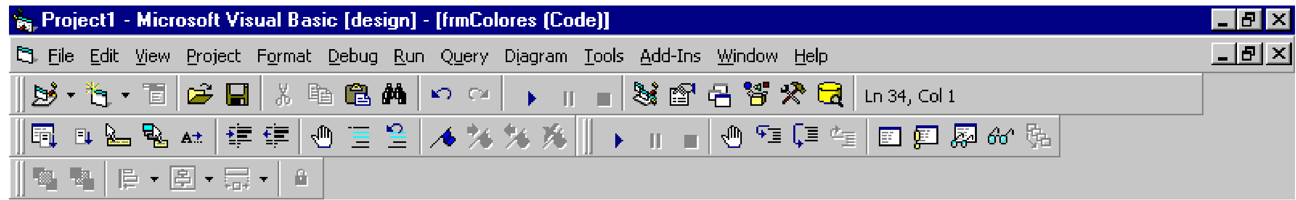


Figura 2.2. Barra de menús y barras de herramientas de Visual Basic 6.0.

El menú ***File*** tiene pocas novedades. Lo más importante es la distinción entre ***proyectos*** y todos los demás ficheros. Como ya se ha dicho, un proyecto reúne y organiza todos los ficheros que componen el programa o aplicación (hace la función de una *carpeta* que contuviera *apuntes*). Estos ficheros pueden ser *formularios*, *módulos, clases, recursos*, etc. ***Visual Basic 6.0*** permite tener más de un proyecto abierto simultáneamente, lo cual puede ser útil en ocasiones. Con el comando ***AddProject*** se añade un nuevo proyecto en la ventana ***Project Manager***. Con los comandos ***Open Project*** o ***New Project*** se abre o se crea un nuevo proyecto, pero cerrando el o los proyectos que estuvieran abiertos previamente. En este menú está el comando ***Make ProjectName.exe***, que permite crear ejecutables de los proyectos.

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

El menú ***Project*** permite añadir distintos tipos de elementos a un proyecto (formularios, módulos, etc.). Con ***Project/Properties*** se puede elegir el tipo de proyecto y determinar el formulario con el que se arrancará la aplicación (***Startup Object***). Con el comando ***Components*** se pueden añadir nuevos controles a la ***Toolbox*** que aparece a la izquierda de la pantalla.

El menú ***Format*** contiene opciones para controlar el aspecto de la aplicación (alinear controles, espaciarlos uniformemente, etc.). Los menús ***Debug*** y ***Run*** permiten controlar la ejecución de las aplicaciones. Con ***Debug*** se puede ver en detalle cómo funcionan, ejecutando paso a paso, yendo hasta una línea de código determinada, etc. Esto es especialmente útil cuando haya que encontrar algunos errores ejecutando paso a paso, o viendo resultados intermedios.

En el menú ***Tools*** se encuentran los comandos para arrancar el ***Menu Editor*** (para crear menús, como se verá en el Apartado 5, a partir de la página 64, dedicado a los ***Menús***) y para establecer las ***opciones*** del programa. En ***Tools/Options*** se encuentran las opciones relativas al proyecto en el que se trabaja. La lengüeta ***Environment*** determina las propiedades del entorno del proyecto, como las opciones para ***actualizar*** o no los ficheros antes de cada ejecución; en ***General*** se establece lo referente a la retícula o ***grid*** que aparece en el formulario; ***Editor*** permite establecer la necesidad de ***declarar*** todas las variables junto con otras opciones de edición, como si se quieren ver o no todos los procedimientos juntos en la misma ventana, y si se quiere ver una línea separadora entre procedimientos; ***Editor Format*** permite seleccionar el tipo de letra y los códigos de color con los que aparecen los distintos elementos del código. La opción ***Advanced*** hace referencia entre otras cosas a la opción de utilizar ***Visual Basic 6.0*** en dos formatos SDI (***Single Document Interface***) y MDI (***Multiple Document Interface*** descritos en el Apartado 4.5, en la página 61).

Por último, la ayuda (siempre imprescindible y en el caso de ***Visual Basic 6.0*** particularmente bien hecha) que se encuentra en el menú ***Help***, se basa fundamentalmente en una clasificación temática ordenada de la información disponible (***Contents***), en una clasificación alfabética de la información (***Index***) y en la búsqueda de información sobre algún tema por el nombre (***Search***). Como ya se ha mencionado, la tecla <F1> permite una ayuda directa sensible al contexto, esto es dependiente de donde se haya clicado con el ratón (o de lo que esté seleccionado).

Figura 2.3. Caja de componentes (***Toolbox***).

2.2.2 Las herramientas (toolbox)

La Figura 2.3 muestra la caja de herramientas, que incluye los ***controles*** con los que se puede diseñar la pantalla de la aplicación. Estos controles son por ejemplo botones, etiquetas, cajas de texto, zonas gráficas, etc. Para introducir un control en el formulario simplemente hay que clicar en el icono adecuado de la ***toolbox*** y colocarlo en el formulario con la posición y el tamaño deseado, clicando y arrastrando con el ratón. Clicando dos veces sobre el icono de un control aparece éste en el centro del formulario y se puede modificar su tamaño y/o trasladar con el ratón como se desee.

El número de controles que pueden aparecer en esta ventana varía con la configuración del sistema. Para introducir nuevos componentes se utiliza el comando ***Components*** en el menú ***Project***, con lo cual se abre el cuadro de diálogo mostrado en la Figura 2.4.

2.3 FORMULARIOS (*FORMS*) Y MÓDULOS

Los ***formularios*** son las zonas de la pantalla sobre las que se diseña el programa y sobre las que se sitúan los controles o herramientas de la ***toolbox***. Al ejecutar el programa, el ***form*** se convertirá en la ventana de la aplicación, donde aparecerán los botones, el texto, los gráficos, etc.

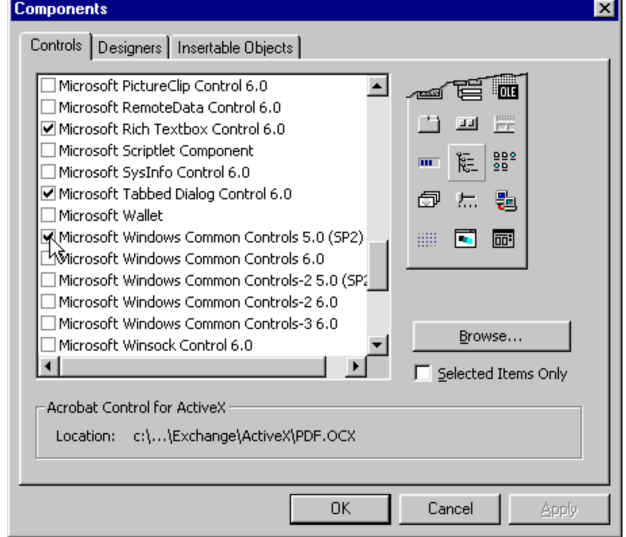


Figura 2.4. Cuadro de diálogo Components.

Para lograr una mejor presentación existe una malla o retícula (***grid***) que permite alinear los controles manualmente de una forma precisa (evitando tener que introducir coordenadas continuamente). Esta malla sólo será visible en el proceso de diseño del programa; al ejecutarlo no se verá. De cualquier forma, se puede desactivar la malla o cambiar sus características en el menú ***Tools/Options/General***, cambiando la opción ***Align Controls to Grid***.

Exteriormente, los formularios tienen una estructura similar a la de cualquier ventana. Sin embargo, también poseen un código de programación que estará escrito en ***Basic***, y que controlará algunos aspectos del formulario, sobre todo en la forma de reaccionar ante las acciones del usuario (eventos). El formulario y los controles en él situados serán el esqueleto o la base del programa. Una aplicación puede tener varios formularios, pero siempre habrá uno con el que arrancará la aplicación; este formulario se determina a partir del menú ***Project/Properties***, en ***Startup Objects***.

Resumiendo, cuando se vaya a crear un programa en ***Visual Basic 6.0*** habrá que dar dos pasos:

1. Diseñar y preparar la parte gráfica (formularios, botones, menús, etc.)
2. Realizar la programación que gestione la respuesta del programa ante los distintos eventos.

2.4 LA VENTANA DE PROYECTO (*PROJECT*)

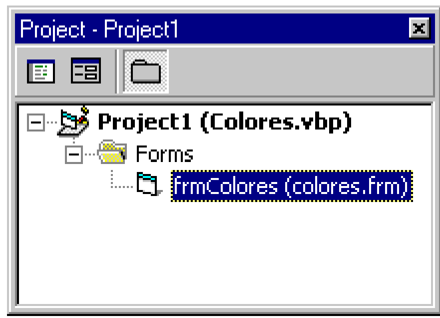


Figura 2.5. Ventana de proyecto.

Esta ventana, mostrada en la Figura 2.5, permite acceder a los distintos formularios y módulos que componen el proyecto. Desde ella se puede ver el diseño gráfico de dichos formularios (botón ***View Object***), y también permite editar el código que contienen (botón ***View Code***). Estos botones están situados en la parte superior de la ventana, debajo de la barra de títulos.

Los ***módulos estándar*** (ficheros **.bas*) contienen sólo código que, en general, puede ser utilizado por distintos formularios y/o controles del proyecto e incluso por varios proyectos. Por ejemplo puede prepararse un módulo estándar de funciones matemáticas que sea de utilidad general. Normalmente contienen siempre algunas declaraciones de variables globales o ***Public***, que serán accesibles directamente desde todos los formularios.

Los ***módulos de clase*** (ficheros ****.cls***) contienen clases definidas por el usuario. Las clases son como formularios o controles complejos, sin interface gráfica de usuario.

2.5 LA VENTANA DE PROPIEDADES (*PROPERTIES*)

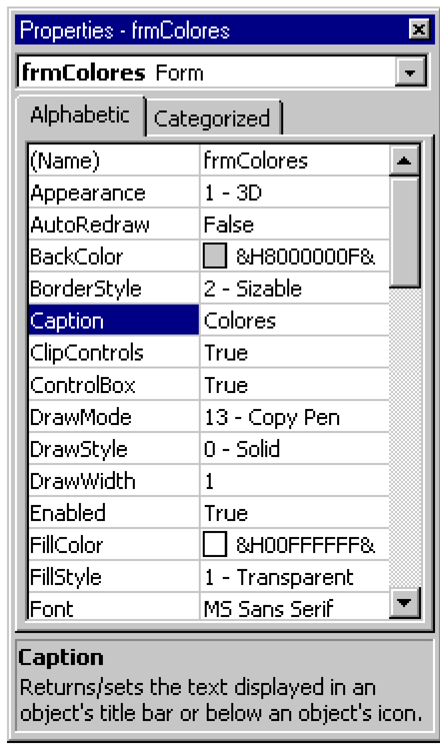


Figura 2.6. Ventana de propiedades.

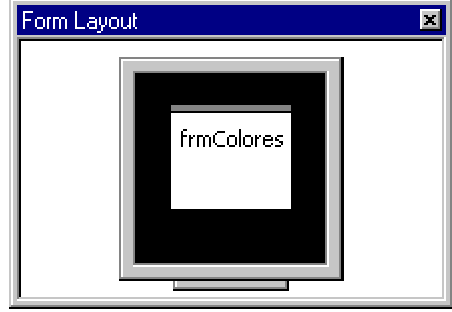


Figura 2.7. Ventana Form Layout.

Todos los objetos ***Visual Basic 6.0*** tienen unas propiedades que los definen: su *nombre* (***Name***), su *etiqueta* o *título* (***Caption***), el *texto* que contiene (***Text***), su *tamaño y posición*, su *color*, si está *activo* o no (***Enabled***), etc. La Figura 2.6 muestra parcialmente las ***propiedades*** de un formulario. Todas estas propiedades se almacenan dentro de cada control o formulario en forma de ***estructura*** (similar a las del lenguaje C). Por tanto, si por ejemplo en algún momento se quiere modificar el nombre de un botón basta con hacerlo en la ventana de propiedades (al diseñar el programa) o en el código en ***Basic*** (durante la ejecución), mediante el ***operador punto*** ***(.)***, en la forma:

```
Boton1.Name = "NuevoNombre"
```

Para realizar una modificación de las propiedades de un objeto durante el diseño del programa, se activará la ventana de propiedades (con el menú, con el botón de la

barra de herramientas o pulsando <F4>). Esta ventana tiene dos lengüetas, que permiten ordenar las propiedades ***alfabéticamente*** o por ***categorías***. Utilizando la forma que sea más cómoda se localizará con ayuda de la barra de desplazamiento la propiedad que se quiera modificar. Al clicar sobre ella puede activarse un menú desplegable con las distintas opciones, o bien puede modificarse directamente el valor de la propiedad. Si esta propiedad tiene sólo unos valores fijos (por ejemplo, los colores), puede abrirse un cuadro de diálogo para elegir un color, o el tamaño y tipo de letra que se desee si se trata de una propiedad ***Font***.

La Figura 2.7 muestra la ventana ***FormLayout***, que permite determinar la posición en la que el formulario aparecerá sobre la pantalla cuando se haga visible al ejecutar la aplicación.

2.6 CREACIÓN DE PROGRAMAS EJECUTABLES

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

Para crear un programa ejecutable se utiliza el comando ***Make nombreProyecto.exe …*** en el menú ***File.*** De esta manera se generará un fichero cuya extensión será ****.exe***. Para que este programa funcione en un ordenador solamente se necesita que el fichero ***MSVBVM60.DLL*** esté instalado en el directorio ***c:\Windows\System*** o ***c:\Winnt\System32***. En el caso de proyectos más complejos en los que se utilicen muchos controles pueden ser necesarios más ficheros, la mayoría de ellos con extensiones ****.ocx***, *****.vbx*** o ****.dll***. Para saber en cada caso cuáles son los ficheros necesarios se puede consultar el fichero ****.vbp*** que contiene la descripción completa del proyecto. Casi todos esos ficheros necesarios se instalan automáticamente al instalar el compilador de ***Visual Basic 6.0*** en el ordenador.

En el caso de el programa se vaya a utilizar en un ordenador en el que no esté instalado ***Visual Basic*** o de que en el proyecto se hayan utilizado controles comerciales (como los ***Crystal Reports*** para la creación de informes, los ***Sheridan Data Widgets*** o los ***True DBGrid*** de ***Apex*** para la gestión de bases de datos, etc.), puede resultar interesante construir unos ***disquetes de instalación*** que simplifiquen la tarea de instalar el programa en cualquier ordenador sin tener que ver en cada caso cuáles son los ficheros que faltan. ***Visual Basic 6.0*** dispone de un ***Asistente*** (***Wizard***) que, interactivamente, simplifica enormemente la tarea de creación de disquetes de instalación. Este ***Asistente*** está en el mismo grupo de programas que ***Visual Basic 6.0*** y se llama ***Package and Deployement Wizard***.

2.7 CÓMO UTILIZAR EL HELP

Visual Basic 6.0 dispone de un ***Help*** excelente, como la mayoría de las aplicaciones de ***Microsoft***. En esta nueva versión la ayuda se ofrece a través de una interface de usuario similar a la de ***Internet Explorer***. Estando seleccionado un control, una propiedad o un formulario, o estando seleccionada una palabra clave en una ventana de código, esta ayuda se puede utilizar *de modo sensible al contexto* pulsando la tecla <F1>. También se puede ver toda la información disponible de modo general y ordenado por temas con el comando ***Help/Contents***.

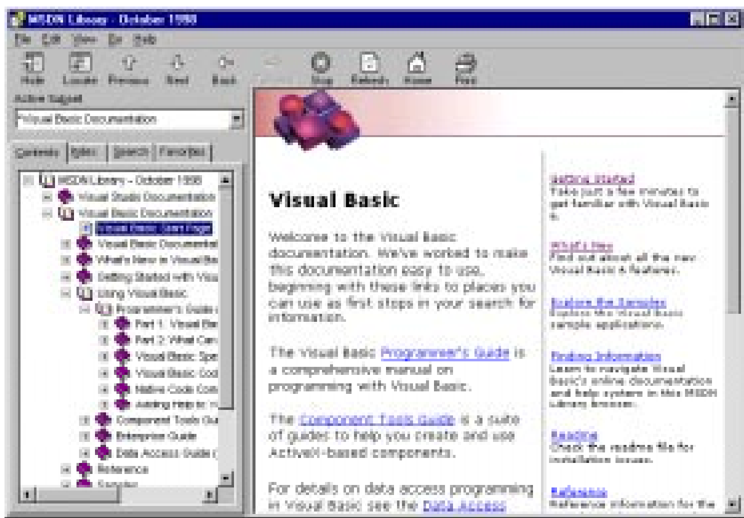


Figura 2.8. Ayuda de Visual Basic 6.0.

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

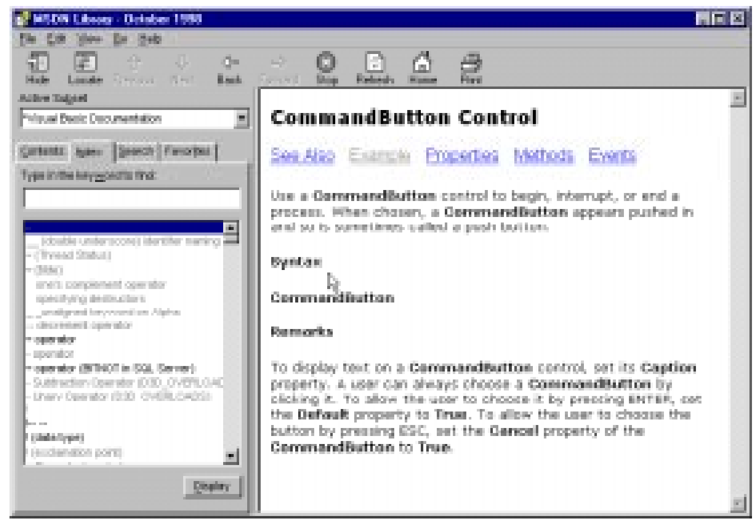


Figura 2.9. Ayuda sobre el botón de comando.

Una vez obtenida la ayuda sobre el término solicitado se pueden encontrar temas relacionados con ese término en la opción ***See Also***. En el caso de que se haya solicitado ayuda sobre un determinado tipo de control (en la Figura 2.9 se ha hecho con los botones de comando), se podría acceder también a la ayuda sobre todos y cada uno de sus propiedades, eventos y métodos con las opciones ***Properties***, ***Methods*** y ***Events***, respectivamente.

La solapa ***Contents*** de la ventana de ayuda sirve para acceder a una pantalla en la que la ayuda está ordenada por temas, la de ***Index*** sirve para acceder a una pantalla en la que se podrá realizar una búsqueda a partir de un término introducido por el usuario, entre una gran lista de términos ordenados alfabéticamente. Al teclear las primeras letras del término, la lista de palabras se va desplazando de modo automático en busca de la palabra buscada. El botón ***Back*** sirve para regresar a la pantalla desde la que se ha llegado a la actual y con el botón ***Print*** se puede imprimir el contenido de la ayuda.

2.8 UTILIZACIÓN DEL CODE EDITOR

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

Para abrir la ventana del editor de código se elige ***Code*** en el menú ***View***. También se abre clicando en el botón ***View Code*** de la ***Project Window***, o clicando dos veces en el formulario o en cualquiera de sus controles. Cada formulario, cada módulo de clase y cada módulo estándar tienen su propia ventana de código. La Figura 2.10 muestra un aspecto típico de la ventana de código. Aunque el aspecto de dicha ventana no tiene nada de particular, el ***Code Editor*** de ***Visual Basic 6.0*** ofrece muchas ayudas al usuario que requieren una explicación más detenida.

En primer lugar, el ***Code Editor*** utiliza un ***código de colores*** (accesible y modificable en ***Tools/Options/Editor Format***) para destacar cada elemento del programa. Así, el código escrito por el usuario aparece en negro, las palabras clave de ***Basic*** en azul, los comentarios en verde, los errores en rojo, etc. Esta simple ayuda visual permite detectar y corregir problemas con más facilidad.

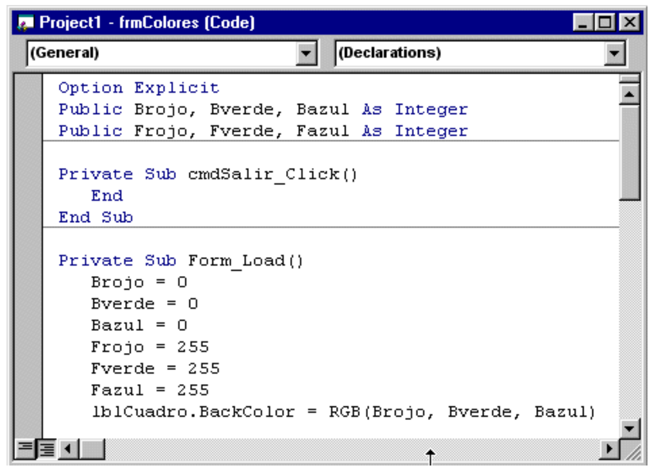


Figura 2.10. Ventana del ***Code Editor***.

En la parte superior de esta ventana aparecen dos ***listas desplegables***. La de la izquierda corresponde a los distintos elementos del formulario (la parte ***General***, que es común a todo el formulario; el propio formulario y los distintos controles que están incluidos en él). La lista desplegable de la derecha muestra los distintos procedimientos que se corresponden con el elemento seleccionado en la lista de la izquierda. Por ejemplo, si en la izquierda está seleccionado un botón de comando, en la lista de la derecha aparecerá la lista de todos los posibles procedimientos ***Sub*** que pueden generar sus posibles eventos. Estas dos listas permiten localizar fácilmente el código que se desee programar o modificar.

El código mostrado en la Figura 2.10 contiene en la parte superior una serie de declaraciones de variables y la opción de no permitir utilizar variables no declaradas (***Option Explicit***). Ésta es la parte ***General*** de código del formulario. En esta parte también se pueden definir ***funciones*** y ***procedimientos Sub*** no relacionados con ningún evento o control en particular. A continuación aparecen dos ***procedimientos Sub*** (el segundo de ellos incompleto) que se corresponden con el evento ***Click*** del botón ***cmdSalir*** y con el evento ***Load*** del formulario. Estos procedimientos están separados por una línea, que se activa con ***Procedure Separator*** en ***Tools/Options/Editor***.

Para ver todos los procedimientos del formulario y de sus controles simultáneamente en la misma ventana (con o sin separador) o ver sólo un procedimiento (el seleccionado en las listas desplegables) se pueden utilizar los dos pequeños botones que aparecen en la parte inferior izquierda de la ventana. El primero de ellos es el ***Procedure View*** y el segundo el ***Full Module View***. Esta opción está también accesible en ***Tools/Options/Editor***.

Otra opción muy interesante es la de completar automáticamente el ***código (Automatic Completion Code***). La Figura 2.11 muestra cómo al teclear el punto (o alguna letra inicial de una propiedad después del punto) detrás del nombre de un objeto, automáticamente se abre una lista con las propiedades de ese objeto. Pulsando la tecla ***Tab*** se introduce el nombre completo de la propiedad seleccionada. A esta característica se le conoce como ***AutoListMembers***.

Por otra parte, la opción ***AutoQuickInfo*** hace que al comenzar a teclear el nombre de una función aparezca información sobre esa función: nombre, argumentos y valor de retorno (ver Figura

2.12). Tanto la opción ***AutoListMembers*** como la opción ***AutoQuickInfo*** se activan en el cuadro de diálogo que se abre con ***Tools/Options/Editor.***

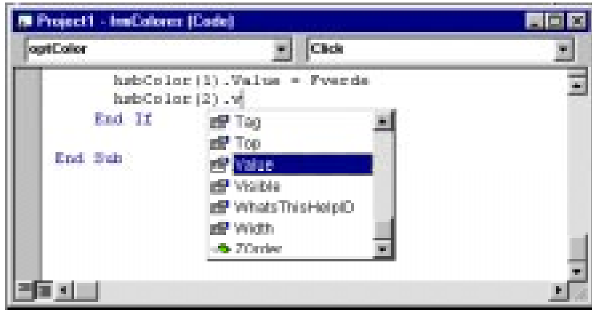


Figura 2.11. Inserción automática de propiedades.

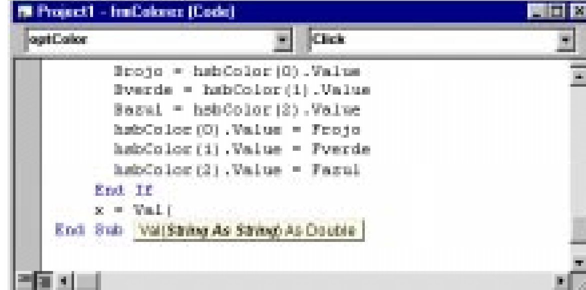


Figura 2.12. Ayuda para inserción de funciones.

2.9 UTILIZACIÓN DEL DEBUGGER

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

2.9.1 Ejecución controlada de un programa

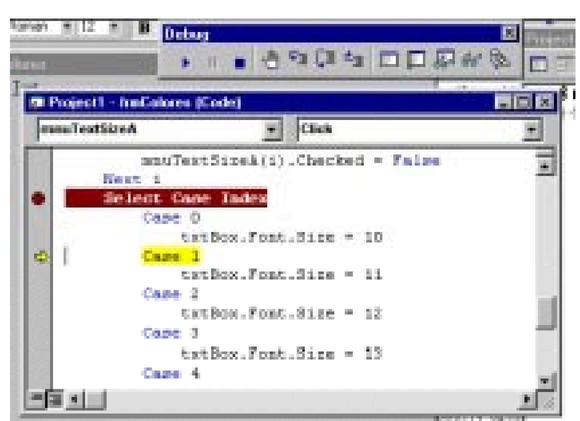


Figura 2.13. Utilización del ***Debugger***.

Para ejecutar ***parcialmente*** un programa se pueden utilizar varias formas. Una de ellas consiste en incluir ***breakpoints*** (puntos de parada de la ejecución) en determinadas líneas del código. Los breakpoints se indican con un punto grueso en el margen y un cambio de color de la línea, tal como se ve en la Figura 2.13. En esta figura se muestra también la barra de herramientas ***Debug***. El colocar un ***breakpoint*** en una línea de código implica que la ejecución del programa se detendrá al llegar a esa línea. Para insertar un ***breakpoint*** en una línea del código se utiliza la opción ***Toggle Breakpoint*** del menú ***Debug***, con el botón del mismo nombre ( ) o pulsando la tecla <F9>, estando el cursor posicionado sobre la línea en cuestión. Para borrarlo se repite esa operación.

Cuando la ejecución está detenida en una línea aparece una flecha en el margen izquierdo, tal como puede verse también en la Figura 2.13. En ese momento se puede consultar el valor de cualquier variable que sea accesible desde ese punto en la ventana de depuración (***Debug Window***). Un poco más adelante se verán varias formas de hacer esto.

En la Figura 2.13 se puede observar como la ejecución del programa está detenida en la línea coloreada o recuadrada, con una flecha en el margen izquierdo. Se puede observar también la variación del color de fondo de la línea anterior debido a que en ella hay un ***breakpoint***.

De todos modos no es estrictamente necesaria la utilización de ***breakpoints*** para la ejecución parcial de un programa. Esto se puede hacer también ejecutando el programa ***paso a paso*** (o *línea a línea*). Para hacer esto hay varias opciones: pulsando la tecla <F8>, seleccionando la opción ***Step Into*** del menú ***Run*** o clicando en el botón correspondiente ( ). Esta instrucción hace que se ejecute una línea del código. En el caso de que ésta se trate de la llamada a un procedimiento o función, la ejecución se trasladará a la primera línea de ese procedimiento o función. En el caso de que se desee ejecutar toda la función en un único paso (por ejemplo porque se tiene constancia de que esa función funciona correctamente) se puede hacer mediante la opción ***Step Over***, pulsando las teclas <mayúsculas> y <F8> simultáneamente, o clicando en el botón correspondiente ( ). En este caso la ejecución se traslada a la línea inmediatamente posterior a la llamada a la función. En el caso de que la línea a ejecutar no sea la llamada a una función ambas opciones (***Step Into*** y ***Step Over***) operan idénticamente. El comando y botón ***Step Out*** ( ) hace que se salga de la función o procedimiento que se está ejecutando y que la ejecución se detenga en la sentencia inmediatamente siguiente a la llamada a dicha función o procedimiento.

La utilización del ***Debugger*** permite también otras opciones muy interesantes como la de ejecutar el programa hasta la línea en la que se encuentre posicionado el cursor (con ***Step To Cursor*** o ***Ctrl+<F8>***); la de continuar con la ejecución del programa hasta el siguiente ***breakpoint*** en el caso de que lo haya o hasta el final del mismo si no hay ninguno (con ***Continue***, botón o <F5>); y la posibilidad de volver a comenzar la ejecución (con ***Restart*** o ***Mayúsculas + <F5>***). Además de las ya mencionadas, también existe la posibilidad de detener momentáneamente la ejecución del programa mediante el botón ***Pause*** ( ) o la combinación de teclas ***Ctrl+Pausa***.

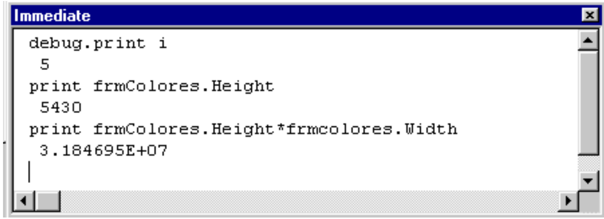


Figura 2.14. Ventana ***Immediate***.

Figura 2.15. Ventana ***Locals***.

2.9.2 Ventanas Immediate, Locals y Watches

El ***Debugger*** de ***Visual Basic 6.0*** dispone de varias formas para consultar el valor de variables y propiedades, así como para ejecutar funciones y procedimientos comprobando su correcto funcionamiento. En ello juegan un papel importante tres tipos de ventanas: ***Immediate***, ***Locals*** y ***Watch***.

La ventana ***Immediate*** (ver Figura 2.14) permite realizar diversas acciones:

1. Imprimir el valor de cualquier variable y/o propiedad accesible la función o procedimiento que se está ejecutando. Esto se puede hacer utilizando el método ***Print VarName*** (o su equivalente ***?VarName***) directamente en dicha ventana o introduciendo en el código del programa sentencias del tipo ***Debug.Print VarName***. En este último caso el valor de la variable o propiedad se escribe en la ventana ***Immediate*** sin necesidad de parar la ejecución del programa. Además esas sentencias se guardan con el formulario y no hay que volver a escribirlas para una nueva ejecución. Cuando se compila el programa para producir un ejecutable las sentencias ***Debug.Print*** son ignoradas. La utilización del método ***Print*** se explica en el Apartado 7.2, en la página 88.

2. Asignar valores a variables y propiedades cuando la ejecución está detenida y proseguir la ejecución con los nuevos valores. Sin embargo, no se pueden crear nuevas variables.
3. Ejecutar expresiones y probar funciones y procedimientos incluyendo en la ventana ***Immediate*** la llamada correspondiente.

La ventana ***Locals***, mostrada en la Figura 2.15, muestra el valor de todas las variables visibles en el procedimiento en el que está detenida la ejecución.

Otra opción que puede resultar útil es la de conocer permanentemente el valor de una variable sin tener que consultarlo cada vez. Para conocer inmediatamente el valor de una variable se puede utilizar la ventana ***Quick Watch***, mostrada en la Figura 2.16. Para observar continuamente el valor de una variable, o expresión hay que añadirla a la ventana ***Watches***. Esto se hace con la opción ***Add Watch…*** del menú ***Debug***. El valor de las variables incluidas en la ventana ***Watches*** (ver Figura 2.18) se actualiza automáticamente, indicándose también cuando no son accesibles desde el procedimiento que se esté ejecutando (***Out of Context***).

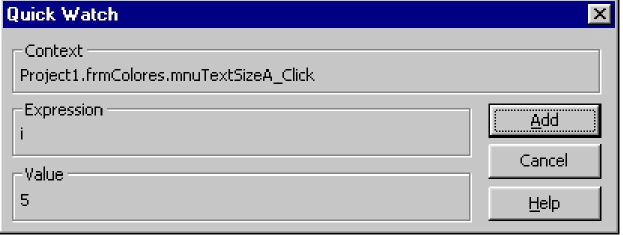


Figura 2.16. Ventana ***Quick Watch***.

La ventana ***Add Watch*** mostrada en la Figura 2.17 permite introducir ***Breaks*** o paradas del programa condicionales, cuando se cumple cierta condición o cuando el valor de la variable cambia.

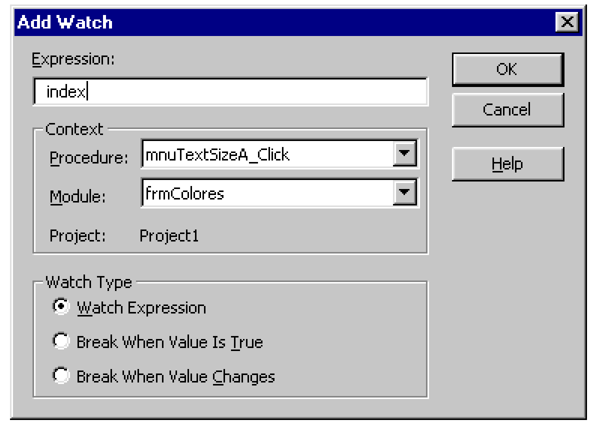


Figura 2.17. Ventana ***Add Watch***.

Las capacidades de ***Visual Basic 6.0*** para *vigilar* el valor de las variables pueden activarse desde el menú ***Debug*** o con algunos botones en la barra de herramientas ***Debug*** ( ).

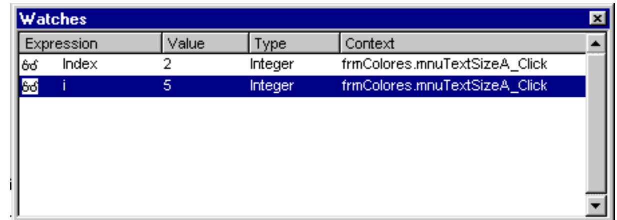


Figura 2.18. Ventana ***Watches***.

2.9.3 Otras posibilidades del Debugger

El ***Debugger*** de ***Visual Basic 6.0*** permite no sólo saber qué sentencia va a ser la próxima en ejecutarse (con ***Debug/Show Next Statement***), sino también decidir cuál va a ser dicha sentencia (con ***Debug/Set Next Statement***), pudiendo cambiar de esta forma el curso habitual de la ejecución: saltando sentencias, volviendo a una sentencia ya ejecutada, etc.

Visual Basic 6.0 puede dar también información sobre las llamadas a funciones y procedimientos. Esto se hace con el comando ***View/Call Stack*** o con el botón correspondiente de la barra ***Debug*** ( ). De esta manera puede conocerse qué función ha llamado a qué función hasta la sentencia donde la ejecución está detenida.

3. LENGUAJE BASIC

3.1 INTRODUCCIÓN

En este capítulo se explican los fundamentos del lenguaje de programación ***Basic*** utilizado en el sistema de desarrollo para ***Visual Basic 6.0*** de ***Microsoft***. En este manual se supone que el lector no tiene conocimientos previos de programación.

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

Un procesador digital únicamente es capaz de entender aquello que está constituido por conjuntos de *unos* y *ceros*. A esto se le llama *lenguaje de máquina* o *binario*, y es muy difícil de manejar. Por ello, desde casi los primeros años de los ordenadores, se comenzaron a desarrollar los llamados *lenguajes de alto nivel* (tales como el ***Fortran***, el ***Cobol***, etc.), que están mucho más cerca del lenguaje natural. Estos lenguajes están basados en el uso de *identificadores*, tanto para los *datos* como para las componentes elementales del programa, que en algunos lenguajes se llaman *rutinas*, *procedimientos*, o *funciones*. Además, cada lenguaje dispone de una *sintaxis* o conjunto de reglas con las que se indica de modo inequívoco las operaciones que se quiere realizar.

Los *lenguajes de alto nivel* son más o menos comprensibles para el usuario, pero no para el procesador. Para que éste pueda ejecutarlos es necesario traducirlos *a su propio lenguaje de máquina*. Al paso del lenguaje de alto nivel al lenguaje de máquina se le denomina ***compilación***. En ***Visual Basic*** esta etapa no se aprecia tanto como en otros lenguajes donde el programador tiene que indicar al ordenador explícitamente que realice dicha compilación. Los programas de ***Visual Basic*** se dice que son ***interpretados*** y no compilados ya que el código no se convierte a código máquina sino que hay otro programa que durante la ejecución "interpreta" las líneas de código que ha escrito el programador. En general durante la ejecución de cualquier programa, el código es cargado por el sistema operativo en la memoria RAM.

3.2 COMENTARIOS Y OTRAS UTILIDADES EN LA PROGRAMACIÓN CON VISUAL BASIC

Visual Basic 6.0 interpreta que todo lo que está a la derecha del ***carácter (')*** en una línea cualquiera del programa es un ***comentario*** y no lo tiene en cuenta para nada. El comentario puede empezar al comienzo de la línea o a continuación de una instrucción que debe ser ejecutada, por ejemplo:

```
' Esto es un comentario
A = B*x+3.4    ' también esto es un comentario
```

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

Otro aspecto práctico en la programación es la posibilidad ***de escribir una sentencia en más de una línea***. En el caso de sentencias bastante largas es conveniente cortar la línea para que entre en la pantalla. En otro caso la lectura del código se hace mucho más pesada. Para ello es necesario dejar un ***espacio en blanco*** al final de la línea y escribir el ***carácter (_)*** tal y como se muestra en el siguiente ejemplo:

```
str1 = "Londres" : str2 = "París"           'Se inicializan las variables
Frase = "Me gustaría mucho viajar a " & _
        str1 & " y a " & str2
'El contenido de Frase sería: "Me gustaría mucho viajar a Londres y a París
```

Una limitación a los comentarios en el código es que no se pueden introducir en una línea en la que se ha introducido el carácter de continuación (_).

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

```
m = a : n = b : resto = m Mod n       ' Tres sentencias en una línea
```

3.3 PROYECTOS Y MÓDULOS

Un ***proyecto*** realizado en ***Visual Basic 6.0*** es el conjunto de todos los ficheros o ***módulos*** necesarios para que un programa funcione. La información referente a esos ficheros se almacena en un fichero del tipo ***ProjectName.vbp***. La extensión ****.vbp*** del fichero hace referencia a ***Visual Basic Project***.

Si se edita este fichero con cualquier editor de texto se comprueba que la información que almacena es la localización en los discos de los módulos que conforman ese proyecto, los controles utilizados (ficheros con extensión ***.ocx***), etc. En el caso más simple un proyecto está formado por un único formulario y constará de dos ficheros: el que define el proyecto (****.vbp***) y el que define el formulario (****.frm***).

Los módulos que forman parte de un proyecto pueden ser de varios tipos: aquellos que están asociados a un formulario (****.frm***), los que contienen únicamente líneas de código ***Basic*** (****.bas***) llamados ***módulos estándar*** y los que definen agrupaciones de código y datos denominadas clases (****.cls***), llamados ***módulos de clase***.

Un módulo ****.frm*** está constituido por un ***formulario*** y toda la información referente a los ***controles*** (y a sus propiedades) en él contenidos, además de todo el código programado en los ***eventos*** de esos controles y, en el caso de que existan, las ***funciones*** y ***procedimientos*** propios de ese formulario. En general se llama ***función*** a una porción de código independiente que realiza una determinada actividad. En ***Visual Basic*** existen dos tipos de funciones: las llamadas ***function***, que se caracterizan por tener valor de retorno, y los ***procedimientos*** o *procedures*, que no lo tienen. En otros lenguajes, como C/C++/Java, las ***function*** realizan los dos papeles.

Un módulo de código estándar ****.bas*** contendrá una o varias funciones y/o procedimientos, además de las variables que se desee, a los que se podrá acceder desde cualquiera de los módulos que forman el proyecto.

3.3.1 Ámbito de las variables y los procedimientos

Se entiende por ***ámbito*** de una variable (ver Apartado 3.3.1, en la página 25) la parte de la aplicación donde la variable es *visible* (accesible) y por lo tanto puede ser utilizada en cualquier expresión.

3.3.1.1 Variables y funciones de ámbito local

Un módulo puede contener variables y procedimientos o funciones ***públicos*** y ***privados***. Los ***públicos*** son aquellos a los que se puede acceder libremente desde cualquier punto del proyecto. Para definir una variable, un procedimiento o una función como ***público*** es necesario preceder a la definición de la palabra ***Public***, como por ejemplo:

```
Public Variable1 As Integer
Public Sub Procedimiento1 (Parametro1 As Integer, …)
Public Function Funcion1 (Parametro1 As Integer, …) As Integer
```

Para utilizar una variable ***Public*** o llamar a una función ***Public*** definidas en un formulario desde otro módulo se debe preceder el nombre de la variable o procedimiento con el nombre del formulario al que pertenece, como por ejemplo:

```
Modulo1.Variable1
Call Modulo1.Procedimiento1(Parametro1, …)
Retorno = Modulo1.Funcion1(Parametro1, …)
```

Sin embargo si el módulo al que pertenecen la variable o el procedimiento ***Public*** es un módulo estándar (****.bas***) no es necesario poner el nombre del módulo más que si hay coincidencia de nombres con los de otro módulo también estándar. Una variable ***Private***, por el contrario, no es accesible desde ningún otro módulo distinto de aquél en el que se haya declarado.

Se llama variable ***local*** a una variable definida dentro de un procedimiento o función. Las variables locales no son accesibles más que en el procedimiento o función en que están definidas.

Una variable ***local*** es reinicializada (a cero, por defecto) cada vez que se entra en el *procedimiento*. Es decir, una variable ***local*** no conserva su valor entre una llamada al *procedimiento* y la siguiente. Para hacer que el valor de la variable se conserve hay que declarar la variable como ***static*** (como por ejemplo: *Static n As Integer*). ***Visual Basic*** inicializa una variable estática solamente la primera vez que se llama al *procedimiento*. Para declarar una variable estática, se utiliza la palabra ***Static*** en lugar de ***Dim.*** Un poco más adelante se verá que ***Dim*** es una palabra utilizada para crear variables. Si un procedimiento se declara ***Static*** todss sus variables locales tienen carácter ***Static***.

3.3.1.2 Variables y funciones de ámbito global

Se puede acceder a una variable o función global desde cualquier parte de la aplicación. Para hacer que una variable sea global, hay que declararla en la ***parte general*** de un módulo ****.bas*** o de un formulario de la aplicación. Para declarar una variable global se utiliza la palabra ***Public***. Por ejemplo:

```
Public var1_global As Double, var2_global As String
```

De esta forma se podrá acceder a las variables ***var1_global***, ***var2_global*** desde todos los formularios. La Tabla 3.1 muestra la accesibilidad de las variable en función de dónde y cómo se hayan declarado[1].

La diferencia entre las variables y/o procedimientos ***Public*** de los formularios y de los módulos estándar está en que las de los procedimientos deben ser cualificadas (precedidas) por el nombre del formulario cuando se llaman desde otro módulo distinto, mientras que las de un módulo estándar (****.bas***) sólo necesitan ser cualificadas si hay colisión o coincidencia de nombres.

---

[1] Las palabras ***Global*** y ***Dim*** proceden de versiones antiguas de ***Visual Basic*** y debe preferirse la utilización de las palabras clave ***Public*** y ***Private***, que expresan mejor su significado.

Tipo de variable	Lugar de declaración	Accesibilidad
Global o Public	Declaraciones de *.bas	Desde todos los formularios
Dim o Private	Declaraciones de *.bas	Desde todas las funciones de ese módulo
Public	Declaraciones de *.frm	Desde cualquier procedimiento del propio formulario y desde otros precedida del nombre del modulo en el que se ha declarado
Dim o Private	Declaraciones de *.frm	Desde cualquier procedimiento del propio formulario
Dim	Cualquier procedimiento de un módulo	Desde el propio procedimiento

Tabla 3.1. Accesibilidad de las variables.

3.4 VARIABLES

3.4.1 Identificadores

La memoria de un computador consta de un conjunto enorme de ***bits*** (1 y 0), en la que se almacenan *datos* y *programas*. Las necesidades de memoria de cada tipo de dato no son homogéneas (por ejemplo, un carácter alfanumérico ocupa un ***byte*** (*8 bits*), mientras que un número real con 16 cifras ocupa 8 *bytes*), y tampoco lo son las de los programas. Además, el uso de la memoria cambia a lo largo del tiempo dentro incluso de una misma sesión de trabajo, ya que el sistema *reserva* o *libera* memoria a medida que la va necesitando.

Cada posición de memoria en la que un dato está almacenado (ocupando un conjunto de bits) puede identificarse mediante un número o una ***dirección***, y éste es el modo más básico de referirse a una determinada información. No es, sin embargo, un sistema cómodo o práctico, por la nula relación nemotécnica que una dirección de memoria suele tener con el dato contenido, y porque –como se ha dicho antes– la dirección física de un dato cambia de ejecución a ejecución, o incluso en el transcurso de una misma ejecución del programa. Lo mismo ocurre con partes concretas de un programa determinado.

Dadas las citadas dificultades para referirse a un dato por medio de su dirección en memoria, se ha hecho habitual el uso de ***identificadores***. *Un identificador es un nombre simbólico que se refiere a un dato o programa determinado*. Es muy fácil elegir identificadores cuyo nombre guarde estrecha relación con el sentido físico, matemático o real del dato que representan. Así por ejemplo, es lógico utilizar un identificador llamado **salario_bruto** o **salarioBruto** para representar el coste anual de un empleado. El usuario no tiene nunca que preocuparse de direcciones físicas de memoria: el sistema se preocupa por él por medio de una *tabla*, en la que se relaciona cada *identificador* con el *tipo de dato* que representa y la *posición de memoria* en la que está almacenado.

Visual Basic 6.0, como todos los demás lenguajes de programación, tiene sus propias reglas para elegir los ***identificadores***. Los usuarios pueden elegir con gran libertad los nombres de sus variables y funciones, teniendo siempre cuidado de respetar las reglas del lenguaje y de no utilizar un conjunto de ***palabras reservadas*** (***keywords***), que son utilizadas por el propio lenguaje. En el Apartado 3.4.3, en la página 28, se explicarán las reglas para elegir nombres y cuáles son las palabras reservadas del lenguaje ***Visual Basic 6.0***.

3.4.2 Variables y constantes

Una ***variable*** es un nombre que designa a una zona de memoria (se trata por tanto de un ***identificador***), que contiene un valor de un tipo de información.

Tal y como su nombre indica, las variables pueden cambiar su valor a lo largo de la ejecución de un programa. Completando a las variables existe lo que se denomina ***constantes*** las cuales son identificadores pero con la particularidad de que el valor que se encuentra en ese lugar de la memoria sólo puede ser asignado una única vez. El tratamiento y tipos de datos es igual al de las variables.

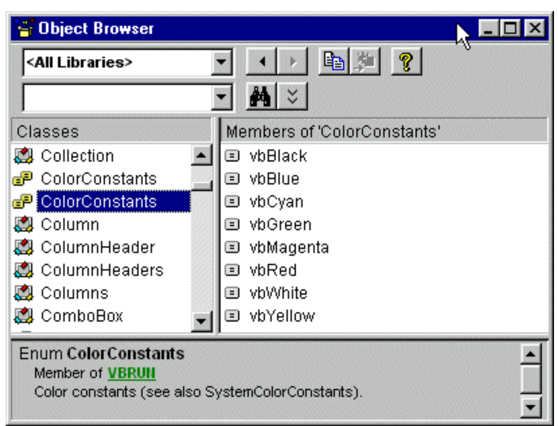


Figura 3.1. Constantes de color predefinidas.

Para declarar un dato como constante únicamente es necesario utilizar la palabra ***Const*** en la declaración de la variable. Si durante la ejecución se intenta variar su valor se producirá un error.

Ejemplos:

```
Const MyVar = 459                ' Las constantes son privadas por defecto.
Public Const MyString = "HELP"   ' Declaración de una constante pública.
Private Const MyInt As Integer = 5     ' Declaración de un entero constante.
Const Str = "Hi", PI As Double = 3.14  ' Múltiples constantes en una línea.
```

Visual Basic 6.0 tiene sus propias constantes, muy útiles por cierto. Algunas ya se han visto al hablar de los colores. En general estas constantes empiezan por ciertos caracteres como ***vb*** (u otros similares que indican a que grupo pertenecen) y van seguidas de una o más palabras que indican su significado. Para ver las constantes disponibles se puede utilizar el comando ***View/Object Browser***, tal como se muestra en la Figura 3.1.

3.4.3 Nombres de variables

El nombre de una variable (o de una constante) tiene que comenzar siempre por una letra y puede tener una longitud hasta 255 caracteres. No se admiten espacios o caracteres en blanco, ni puntos (.), ni otros caracteres especiales.

Los caracteres pueden ser letras, dígitos, el carácter de subrayado (_) y los caracteres de declaración del tipo de la variable (%, &, #, !, @, y $). El nombre de una variable no puede ser una ***palabra reservada*** del lenguaje (***For***, ***If***, ***Loop***, ***Next***, ***Val***, ***Hide***, ***Caption***, ***And***, ***...***). Para saber cuáles son las palabras reservadas en ***Visual Basic 6.0*** puede utilizarse el ***Help*** de dicho programa, buscando la referencia ***Reserved Words***. De ordinario las palabras reservadas del lenguaje aparecen de color azul en el editor de código, lo que hace más fácil saber si una palabra es reservada o no.

A diferencia de ***C***, ***Matlab***, ***Maple*** y otros lenguajes de programación, ***Visual Basic 6.0 no distingue entre minúsculas y mayúsculas***. Por tanto, las variables ***LongitudTotal*** y ***longitudtotal*** son consideradas como idénticas (la misma variable). En ***Visual Basic 6.0*** es habitual utilizar las letras mayúsculas para separar las distintas palabras que están unidas en el nombre de una variable, como se ha hecho anteriormente en la variable ***LongitudTotal***. La declaración de una variable o la primera vez que se utiliza determnan cómo se escribe en el resto del programa.

También es habitual entre los programadores, aunque no obligado, el utilizar nombres con todo mayúsculas para los nombres de las constantes simbólicas, como por ejemplo ***PI***.

3.4.4 Tipos de datos

Al igual que ***C*** y otros lenguajes de programación, ***Visual Basic*** dispone de distintos tipos de datos, aplicables tanto para constantes como para variables. La Tabla 3.2 muestra los tipos de datos disponibles en ***Visual Basic***.

Tipo	Descripción	Carácter de declaración	Rango
Boolean	Binario		True o False
Byte	Entero corto		0 a 255
Integer	Entero (2 bytes)	%	-32768 a 32767
Long	Entero largo (4 bytes)	&	-2147483648 a 2147483647
Single	Real simple precisión (4 bytes)	!	-3.40E+38 a 3.40E+38
Double	Real doble precisión (8 bytes)	#	-1.79D+308 a 1.79D+308
Currency	Número con punto decimal fijo (8 bytes)	@	-9.22E+14 a 9.22E+14
String	Cadena de caracteres (4 bytes + 1 byte/car hasta 64 K)	$	0 a 65500 caracteres.
Date	Fecha (8 bytes)		1 de enero de 100 a 31 de diciembre de 9999. Indica también la hora, desde 0:00:00 a 23:59:59.
Variant	Fecha/hora; números enteros, reales, o caracteres (16 bytes + 1 byte/car. en cadenas de caracteres)	ninguno	F/h: como Date números: mismo rango que el tipo de valor almacenado
User-defined	Cualquier tipo de dato o estructura de datos. Se crean utilizando la sentencia Type (Ver Apartado 3.10)	ninguno	

Tabla 3.2. Tipos de datos en **Visual Basic 6.0**.

En el lenguaje ***Visual Basic 6.0*** existen dos formas de agrupar varios valores bajo un mismo nombre. La primera de ellas son los ***arrays*** (vectores y matrices), que agrupan datos de tipo homogéneo. La segunda son las ***estructuras***, que agrupan información heterogénea o de distinto tipo. En ***Visual Basic 6.0*** las estructuras son verdaderos *tipos de datos definibles por el usuario*.

Para declarar las variables se utiliza la sentencia siguiente:

```
Dim NombreVariable As TipoVariable
```

cuyo empleo se muestra en los ejemplos siguientes:

```
Dim Radio As Double, Superficie as Single
Dim Nombre As String
Dim Etiqueta As String * 10
Dim Francos As Currency
Dim Longitud As Long, X As Currency
```

Es importante evitar declaraciones del tipo:

```
Dim i, j As Integer
```

pues contra lo que podría parecer a simple vista no se crean dos variables ***Integer***, sino una ***Integer*** (**j**) y otra ***Variant*** (**i**).

En ***Visual Basic 6.0*** no es estrictamente necesario declarar todas las variables que se van a utilizar (a no ser que se elija la opción ***Option Explicit*** que hace obligatorio el declararlas), y hay otra forma de declarar las variables anteriores, utilizando los caracteres especiales vistos

anteriormente. Así por ejemplo, el tipo de las variables del ejemplo anterior se puede declarar al utilizarlas en las distintas expresiones, poniéndoles a continuación el carácter que ya se indicó en la Tabla 3.2, en la forma:

```
Radio#                    doble precisión
Nombre$                   cadena de caracteres
Francos@                  unidades monetarias
Longitud&                 entero largo
```

Esta forma de indicar el tipo de dato no es la más conveniente. Se mantiene en las sucesivas versiones de ***Visual Basic*** por la compatibilidad con códigos anteriores. Es preferible utilizar la notación donde se escribe directamente el tipo de dato.

3.4.5 Elección del tipo de una variable

Si en el código del programa se utiliza una variable que no ha sido declarada, se considera que esta variable es de tipo ***Variant***. Las variables de este tipo se adaptan al tipo de información o dato que se les asigna en cada momento. Por ejemplo, una variable tipo ***Variant*** puede contener al principio del programa un *string* de caracteres, después una variable de *doble precisión*, y finalmente un número *entero*. Son pues variables muy flexibles, pero su uso debe restringirse porque ocupan ***más memoria*** (almacenan el tipo de dato que contienen, además del propio valor de dicho dato) y requieren ***más tiempo de CPU*** que los restantes tipos de variables.

En general es el tipo de dato (los valores que puede tener en la realidad) lo que determina qué tipo de variable se debe utilizar. A continuación se muestran algunos ejemplos:

- ***Integer*** para numerar las filas y columnas de una matriz no muy grande
- ***Long*** para numerar los habitantes de una ciudad o los números de teléfonos
- ***Boolean*** para una variable con sólo dos posibles valores (sí o no)
- ***Single*** para variables físicas con decimales que no exijan precisión
- ***Double*** para variables físicas con decimales que exijan precisión
- ***Currency*** para cantidades grandes de dinero

Es muy importante tener en cuenta ***que se debe utilizar el tipo de dato más sencillo que represente correctamente el dato real*** ya que en otro caso se ocupará más memoria y la ejecución de los programas o funciones será más lenta.

3.4.6 Declaración explícita de variables

Una variable que se utiliza sin haber sido declarada toma por defecto el tipo ***Variant***. Puede ocurrir que durante la programación, se cometa un error y se escriba mal el nombre de una variable. Por ejemplo, se puede tener una variable " declarada como *entera*, y al programar referirse a ella por error como "; ***Visual Basic*** supondría que ésta es una nueva variable de tipo ***Variant***.

Para evitar este tipo de errores, se puede indicar a ***Visual Basic*** que genere un mensaje de error siempre que encuentre una variable no declarada previamente. Para ello lo más práctico es establecer una opción por defecto, utilizando el comando ***Environment*** del menú ***Tools/Options***; en el cuadro que se abre se debe poner ***Yes*** en la opción ***Require Variable Declaration***. También se puede hacer esto escribiendo la sentencia siguiente en la sección de declaraciones de cada formulario y de cada módulo:

```
Option Explicit
```

3.5 OPERADORES

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

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

Tabla 3.3. Operadores de ***Visual Basic 6.0***.

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

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

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

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

```
Respuesta = Cadena1 Like Cadena2
```

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

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

3.6 SENTENCIAS DE CONTROL

Las ***sentencias de control***, denominadas también *estructuras de control,* permiten tomar decisiones y realizar un proceso repetidas veces. Son los denominados bifurcaciones y bucles. Este tipo de

estructuras son comunes en cuanto a concepto en la mayoría de los lenguajes de programación, aunque su sintaxis puede variar de un lenguaje de programación a otro. Se trata de unas estructuras muy importantes ya que son las encargadas de controlar el *flujo* de un programa según los requerimientos del mismo. ***Visual Basic 6.0*** dispone de las siguientes estructuras de control:

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

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

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

If condicion **Then** sentencia1 [**Else** sentencia2]

La segunda es más general y se muestra a continuación:

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

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

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

Si se cumple la ***condicion1*** se ejecutan las ***sentencias1***, y si no se cumple, se examinan secuencialmente las condiciones siguientes hasta ***Else***, ejecutándose las sentencias correspondientes al primer ***ElseIf*** cuya condición se cumpla. Si todas las condiciones son falsas, se ejecutan las sentencias-n correspondientes a ***Else***, que es la opción por defecto. La Figura 3.2 presenta esquemáticamente ambas formas de representar estas sentencias:

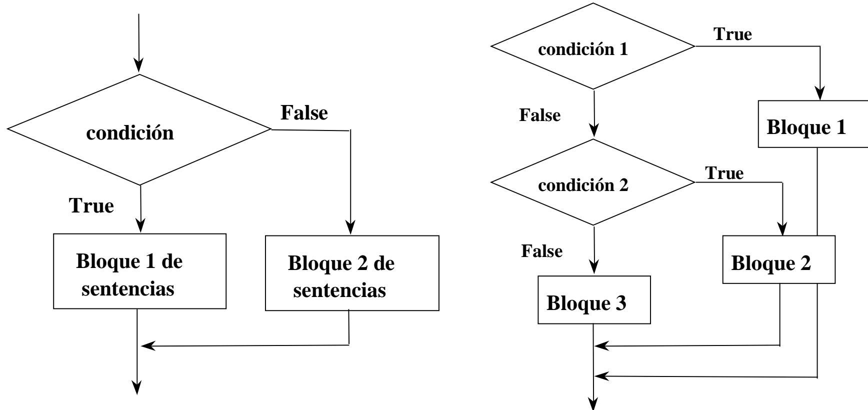


Figura 3.2. Bifurcaciones ***If*** e ***If...Else***.

Por ejemplo,

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

3.6.2 Sentencia SELECT CASE

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

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

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

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

Por ejemplo,

```
Numero = 8                 ' Se inicializan las variable.
Select Case Numero         ' Se va a evaluar la variable Numero.
Case 1 To 5                ' Numero está entre 1 y 5.
    Resultado = "Se encuentra entre 1 y 5"
' Lo siguiente se ejecuta si es True la expresión.
Case 6, 7, 8               ' Numero es uno de los tres valores.
    Resultado = "Se encuentra entre 6 y 8"
```

```
Case Is = 9 , Is = 10      ' Numero es 9 ó 10.
    Resultado = "El valor es 9 o 10"
Case Else                  ' Resto de valores.
    Resultado = "El número no se encuentra entre 1 y 10"
End Select
```

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

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

3.6.3 Sentencia FOR ... NEXT

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

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

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

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

```
MyString="Informática "

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

3.6.4 Sentencia DO ... LOOP

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

```
' Formato 1:
Do [{While/Until} condicion]
    [sentencias]
    [Exit Do]
    [sentencias]
Loop
```

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

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

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

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

3.6.5 Sentencia WHILE … WEND

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

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

Por ejemplo,

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

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

3.6.6 Sentencia FOR EACH ... NEXT

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

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

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

3.7 ALGORITMOS

3.7.1 Introducción

Un ***algoritmo*** es en un sentido amplio una "*secuencia de pasos o etapas que conducen a la realización de una tarea*". Los primeros algoritmos nacieron para resolver problemas matemáticos. Antes de escribir un programa de ordenador, hay que tener muy claro el algoritmo, es decir, cómo se va a resolver el problema considerado. Es importante desarrollar buenos algoritmos (correctos y eficientes). Una vez que el algoritmo está desarrollado, el problema se puede resolver incluso sin entenderlo.

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

1. Asignar a M el valor de A, y a N el valor de B.
2. Dividir M por N, y llamar R al resto.
3. Si R distinto de 0, asignar a M el valor de N, asignar a N el valor de R,volver a comenzar la etapa 2.
4. Si R = 0, N es el m.c.d. de los números originales

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

```
Dim a, b As Integer
a = 45: b = 63            ' Estos son los valores M y N
If a < b Then             ' Se permutan a y b
   temp = a : a = b : b = temp
End If

m = a : n = b : resto = m Mod n ' Mod devuelve el valor del resto
While resto <> 0          'Mientras el resto sea distinto de 0
   m = n: n = resto:
   resto = m Mod n
Wend
' La solución es la variable n. En este caso el resultado es 9
```

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

3.7.2 Representación de algoritmos

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

- ***Detallada***: Se trata de escribir el algoritmo en un determinado lenguaje de programación (lenguaje de máquina, ensamblador, fortran, basic, pascal, C, Matlab, Visual Basic, ...).
- ***Simbólica***: Las etapas son descritas con lenguaje próximo al natural, con el grado de detalle adecuado a la etapa de desarrollo del programa.
- ***Gráfica***: por medio de diagramas de flujo.

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

En cualquier caso e independientemente del tipo de representación utilizada lo importante es tener muy claro el algoritmo a realizar y ponerlo por escrito en forma de esquema antes de ponerse a programarlo. Merece la pena pasar unos minutos realizando un esquema sobre papel antes de ponerse a teclear el código sobre un teclado de ordenador.

3.8 FUNCIONES Y PROCEDIMIENTOS

3.8.1 Conceptos generales sobre funciones

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

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

1. ***Modularización***. Cada función tiene una misión muy concreta, de modo que nunca tiene un número de líneas excesivo y siempre se mantiene dentro de un tamaño manejable. Además, una misma función (por ejemplo, un producto de matrices, una resolución de un sistema de ecuaciones lineales, ...) puede ser llamada muchas veces en un mismo programa, e incluso puede ser reutilizada por otros programas. Cada función puede ser desarrollada y comprobada por separado.
2. ***Ahorro de memoria y tiempo de desarrollo***. En la medida en que una misma función es utilizada muchas veces, el número total de líneas de código del programa disminuye, y también lo hace la probabilidad de introducir errores en el programa.
3. ***Independencia de datos y ocultamiento de información***. Una de las fuentes más comunes de errores en los programas de computador son los *efectos colaterales* o perturbaciones que se pueden producir entre distintas partes del programa. Es muy frecuente que al hacer una modificación para añadir una funcionalidad o corregir un error, se introduzcan nuevos errores en partes del programa que antes funcionaban correctamente. Una función es capaz de mantener una gran independencia con el resto del programa, manteniendo sus propios datos y definiendo muy claramente la *interfaz* o comunicación con la función que la ha llamado y con las funciones a las que llama, y no teniendo ninguna posibilidad de acceso a la información que no le compete.

3.8.2 Funciones y procedimientos Sub en Visual Basic 6.0

En ***Visual Basic 6.0*** se distingue entre ***funciones*** y ***procedimientos Sub***. En ocasiones se utiliza la palabra genérica ***procedimiento*** para ambos. La fundamental diferencia entre un ***procedimiento Sub*** y una ***función*** es que ésta última puede ser utilizada en una expresión porque tiene un ***valor de retorno***. El valor de retorno ocupa el lugar de la llamada a la función donde esta aparece. Por ejemplo, si en una expresión aparece ***sin(x)*** se calcula el seno de la variable ***x*** y el resultado es el valor de retorno que sustituye a ***sin(x)*** en la expresión en la que aparecía. Por tanto, las funciones devuelven valores, a diferencia de los procedimientos que no devuelven ningún valor, y por tanto no pueden ser utilizadas en expresiones. Un ***procedimiento Sub*** es un segmento de código independiente del resto, que una vez llamado por el programa, ejecuta un número determinado de

instrucciones, sin necesidad de devolver ningún valor al mismo (puede dar resultados modificando los argumentos), mientras que una función siempre tendrá un valor de retorno.

Los nombres de los procedimientos tienen reglas de visibilidad parecidas a las de las variables. Para llamar desde un formulario a un procedimiento ***Public*** definido en otro formulario es necesario preceder su nombre por el del formulario en que está definido. Sin embargo, si se desea llamar a un procedimiento definido en un módulo estándar (****.bas***) no es necesario precederlo del nombre del módulo más que si hay coincidencia de nombre con otro procedimiento de otro módulo estándar.

3.8.3 Funciones (function)

La sintaxis correspondiente a una función es la siguiente:

```
[Static] [Private] Function nombre ([parámetros]) [As tipo]
    [sentencias]
    [nombre = expresion]
    [Exit Function]
    [sentencias]
    [nombre = expresion]
End Function
```

donde ***nombre*** es el nombre de la función. Será de un tipo u otro dependiendo del dato que devuelva. Para especificar el tipo se utiliza la cláusula ***As Tipo*** (***Integer***, ***Long***, ***Single***, ***Double***, ***Currency***, ***String*** o ***Variant***). ***parámetros*** son los argumentos que son pasados cuando se llama a la función. ***Visual Basic*** asigna el valor de cada argumento en la llamada al parámetro que ocupa su misma posición. Si no se indica un tipo determinado los argumentos son ***Variant*** por defecto. Como se verá en un apartado posterior, los argumentos pueden ser pasados ***por referencia*** o ***por valor***.

El ***nombre de la función***, que es el valor de retorno, actúa como una variable dentro del cuerpo de la función. El valor de la variable ***expresion*** es almacenado en el propio nombre de la función. Si no se efectúa esta asignación, el resultado devuelto será 0 si la función es numérica, nulo ("") si la función es de caracteres, o *Empty* si la función es ***Variant***.

Exit Function permite salir de una función antes de que ésta finalice y devolver así el control del programa a la sentencia inmediatamente a continuación de la que efectuó la llamada a la función.

La sentencia ***End Function*** marca el final del código de la función y, al igual que la ***Exit Function***, devuelve el control del programa a la sentencia siguiente a la que efectuó la llamada, pero lógicamente una vez finalizada la función.

La ***llamada a una función*** se hace de diversas formas. Por ejemplo, una de las más usuales es la siguiente:

```
variable = nombre([argumentos])
```

donde ***argumentos*** son un lista de constantes, variables o expresiones separadas por comas que son pasadas a la función. En principio, el número de argumentos debe ser igual al número de parámetros de la función. Los ***tipos*** de los argumentos deben coincidir con los tipos de sus correspondientes parámetros, de lo contrario puede haber fallos importantes en la ejecución del programa. Esta regla no rige si los argumentos se pasan ***por valor*** (concepto que se verá más adelante).

En cada llamada a una función hay que incluir los paréntesis, aunque ésta no tenga argumentos.

El siguiente ejemplo corresponde a una función que devuelve como resultado la raíz cuadrada de un número ***N***:

```
Function Raiz (N As Double) As Double
    If N < 0 Then
        Exit Function
    Else
        Raiz = Sqr(N)
End Function
```

La llamada a esta función se hace de la forma siguiente:

```
Cuadrada = Raiz(Num)
```

A diferencia de C y C++ en ***Visual Basic 6.0*** no es necesario devolver explícitamente el valor de retorno, pues el nombre de la función ya contiene el valor que se desea devolver. Tampoco es necesario declarar las funciones antes de llamarlas.

3.8.4 Procedimientos Sub

La sintaxis que define un ***procedimiento Sub*** es la siguiente:

```
[Static] [Private] Sub nombre [(parámetros)]
    [sentencias]
    [Exit Sub]
    [sentencias]
End Sub
```

La explicación es análoga a la dada para funciones.

La llamada a un ***procedimiento Sub*** puede ser de alguna de las dos formas siguientes:

```
Call nombre[(argumentos)]
```

o bien, sin pasar los argumentos entre paréntesis, sino poniéndolos a continuación del nombre simplemente separados por comas:

```
nombre [argumentos]
```

A diferencia de una *función*, un ***procedimiento Sub*** no puede ser utilizado en una expresión pues no devuelve ningún valor. Por supuesto una función puede ser llamada al modo de un ***procedimiento Sub***, pero en esta caso no se hace nada con el valor devuelto por la función.

El siguiente ejemplo corresponde a un ***procedimiento Sub*** que devuelve una variable *F* que es la raíz cuadrada de un número *N*.

```
Sub Raiz (N As Double, F As Double)
    If N < 0 Then
       Exit Sub   'Se mandaría un mensaje de error
    Else
       F = Sqr(N)
    End If
End Sub
```

La llamada a este ***procedimiento Sub*** puede ser de cualquiera de las dos formas siguientes:

```
Raiz N, F
Call Raiz(N, F)
```

En el ejemplo anterior, el resultado obtenido al extraer la raíz cuadrada al número *N* se devuelve en la variable *F* pasada como argumento, debido a que como se ha mencionado anteriormente, un ***procedimiento Sub*** no puede ser utilizado en una expresión.

3.8.5 Argumentos por referencia y por valor

En las *funciones* (***Function***) y en los ***procedimientos Sub*** de ***Visual Basic***, por defecto los argumentos se pasan por ***referencia***[2]; de este modo, cualquier cambio de valor que sufra un parámetro dentro de la ***función*** o del ***procedimiento Sub*** también se produce en el argumento correspondiente de la llamada a la ***función*** o al ***procedimiento Sub***.

Cuando se llama a una ***función*** o a un ***procedimiento Sub***, se podrá especificar que el valor de una argumento ***no sea cambiado*** por la función o por el procedimiento, poniendo dicho argumento entre paréntesis en la llamada. Un argumento entre paréntesis en la llamada es un ***argumento pasado por valor***. Por ejemplo,

```
Raiz ((Num))     ' En el caso de la función
Raiz (Num), F    ' En el caso del procedimiento
```

El argumento ***Num*** es pasado ***por valor***. Significa que lo que se pasa es una copia de ***Num***. Si el procedimiento cambia ese valor, el cambio afecta sólo a la copia y no a la propia variable ***Num***.

Otra forma de especificar que un argumento será siempre pasado ***por valor*** es anteponiendo la palabra ***ByVal*** a la declaración del parámetro en la cabecera del procedimiento (***Sub*** o ***Function***). Por ejemplo,

```
Function Raiz (ByVal N As Double)
Sub Raiz (ByVal N As Double, F As Double)
```

Pasar argumentos ***por valor*** evita modificaciones accidentales, pero tiene un coste en tiempo y memoria que puede ser significativo cuando se pasan grandes volúmenes de información, como sucede con vectores, matrices y estructuras.

3.8.6 Procedimientos recursivos

Se dice que una ***función*** (***Function***) es *recursiva* o que un ***procedimiento Sub*** es *recursivo* si se llaman a sí mismos.

A continuación se presenta una ejemplo de una función que calcula el factorial de un número programada de forma recursiva.

```
Function Factorial (N As Integer) As Long
   If N = 0 Then
      Factorial = 1        'Condición de final
   Else
      Factorial = N * Factorial (N - 1)
   End If
End Function
```

En este ejemplo, si la variable N que ***se le pasa*** a la función vale 0, significará que se ha llegado al final del proceso, y por tanto ***se le asigna*** el valor 1 al valor del factorial (recordar que 0! = 1). Si es distinto de 0, la función se llama a ella misma, pero variando el argumento a (N-1), hasta llegar al punto en el que N-1=0, finalizándose el proceso.

---

[2] Pasar un argumento ***por referencia*** implica que en realidad se pasa a la función la variable original, de modo que la función puede modificar su valor. Pasar ***por valor*** implica crear una nueva variable dentro de la función y pasarle una copia del valor de la variable externa. Si se modifica el valor de la variable copia, la variable original queda inalterada. Cuando en la llamada a una función se ponen como argumentos constantes numéricas o expresiones los valores se pasan ***por valor***.

3.8.7 Procedimientos con argumentos opcionales

Puede haber procedimientos en los que algunos de los argumentos incluidos en su definición sean opcionales, de forma que el programador pueda o no incluirlos en la llamada de dichos procedimientos. La forma de incluir un argumento opcional es incluir la palabra ***Optional*** antes de dicho argumento en la definición del procedimiento. Si un argumento es opcional, todos los argumentos que vienen a continuación deben también ser opcionales.

Cuando un argumento es opcional y en la llamada es omitido, el valor que se le pasa es un ***Variant*** con valor *Empty*. A los argumentos opcionales se les puede dar en la definición del procedimiento un valor por defecto para el caso en que sean omitidos en la llamada, como por ejemplo:

```
Private Sub miProc(x as Double, Optional n=3 As Integer)
    sentencias
End Sub
```

3.8.8 Número indeterminado de argumentos

Este caso es similar pero diferente del anterior. En este caso no es que haya argumentos opcionales que puedan omitirse en la llamada, sino que realmente no se sabe con cuántos argumentos va a llamarse la función; unas veces se llamará con 2, otras con 3 y otras con 8. En este caso los argumentos se pasan al procedimiento por medio de un array, especificándolo con la palabra ***ParamArray*** en la definición del procedimiento, como por ejemplo:

```
Public Function maximo(ParamArray numeros())
    For Each x in numeros
        sentencias
        maximo = x
    Next x
End Function
```

3.8.9 Utilización de argumentos con nombre

Visual Basic 6.0 ofrece tambieén la posibilidad de llamar a las ***funciones*** y ***procedimientos Sub*** de una forma más libre y menos formal, pasando los argumentos en la llamada al procedimiento con un orden arbitrario. Esto se consigue incluyendo el ***nombre*** de los argumentos en la llamada y asignándoles un valor por medio de una construcción del tipo ***miArgumento:=unValor***. Unos argumentos se separan de otros por medio de comas (,). Considérese el siguiente ejemplo:

```
Public Sub EnviarCarta(direccion As String, destinatario As String)
    sentencias
End Sub
```

que se puede llamar en la forma:

```
EnviarCarta destinatario:="Mike Tyson", direccion:="Las Vegas"
```

No todas las funciones que se pueden llamar en ***Visual Basic 6.0*** admiten argumentos con nombre. Con ***AutoQuickInfo*** puede obtenerse más información al respecto.

3.9 ARRAYS

Un ***array*** permite referirse a una serie de elementos del mismo tipo con un mismo nombre, y hace referencia un único elemento de la serie utilizando uno o más índices, como un vector o una matriz en Álgebra.

Visual Basic 6.0 permite definir arrays de variables de una o más dimensiones (hasta 60) y de cualquier tipo de datos (tipos fundamentales y definidos por el usuario). Pero además ***Visual Basic*** introduce una nueva clase de arrays, *los arrays de controles* (esto es, arrays de botones, de etiquetas, de paneles, etc.) que permiten una programación más breve y clara. En este apartado sólo se tratarán los arrays de variables.

Todos los elementos de un array deben ser del mismo tipo y están almacenados de forma contigua en la memoria. Por supuesto, si el array es de tipo ***Variant*** cada elemento puede contener un dato de tipo diferente, e incluso puede contener otro array.

Entre los arrays de variables cabe distinguir dos tipos fundamentales, dependiendo de que número de elementos sea constante o pueda variar durante la ejecución del programa.

1. **Arrays estáticos**, cuya dimensión es siempre la misma.
2. **Arrays dinámicos**, cuya dimensión se puede modificar durante la ejecución del programa.

3.9.1 Arrays estáticos

La declaración de un array estático dependerá de su ámbito.

- La declaración de un array público se hace en la sección de declaraciones de un módulo utilizando la sentencia ***Public***.
- La declaración de un array a nivel del módulo o del formulario se hace en la sección de declaraciones del módulo o del formulario utilizando la sentencia ***Dim*** o ***Private***.
- Para declarar un array local a un procedimiento, se utiliza la sentencia ***Dim***, ***Private*** o ***Static*** dentro del propio procedimiento.

A continuación se presentan algunos ejemplos:

```
Dim vector(19) As Double
```

Este ejemplo declara un array de una dimensión, llamado ***vector***, con veinte elementos, *vector(0)*, *vector(1)*, ... , *vector(19)*, cada uno de los cuales permite almacenar un ***Double***. Salvo que se indique otra cosa, los índices se empiezan a contar en cero.

```
Dim matriz(3, 1 To 6) As Integer
```

Este ejemplo declara un array de dos dimensiones, llamado ***matriz***, con 4x6 elementos, *matriz(0,1)*, ... *matriz(3,6)*, de tipo entero.

```
Public cadena(1 To 12) As String
```

El ejemplo anterior declara un array de una dimensión, ***cadena***, con doce elementos, *caract(1)*, ... , *caract(12)*, cada uno de los cuáles permite almacenar una cadena de caracteres.

La declaración de los arrays estáticos es bastante cómoda. Se declaran una vez. Sin embargo tienen el inconveniente que en la mayoría de los casos están sobredimensionados y utilizan más memoria de la que realmente necesitan. Esto implica que se está malgastando memoria. Para solucionar este problema se utilizan los arrays dinámicos.

3.9.2 Arrays dinámicos

El espacio necesario para un array estático se asigna al iniciarse el programa y permanece fijo durante su ejecución. El espacio para un array dinámico se asigna durante la ejecución del

programa. Un array dinámico, puede ser redimensionado en cualquier momento de la ejecución. La forma mejor de redimensionar los arrays es mediante variables que contienen los valores adecuados.

Para crear un array dinámico primero hay que declararlo como si fuera una array estático, pero sin darle dimensión. Es decir, se deja la lista -entre paréntesis- vacía sin ponerle ningún número. Esto se hace con la sentencia ***Public*** si se quiere que sea global, con ***Dim*** o ***Private*** si se quiere a nivel de módulo o con ***Static***, ***Dim*** o ***Private*** si se quiere que sea local.

Para asignar el número actual de elementos del array se utiliza la sentencia ***ReDim***. La sentencia ***ReDim*** puede aparecer solamente en un procedimiento y permite cambiar el número de elementos del array y sus límites inferior y superior, pero no el número de dimensiones. Esto quiere decir que, por ejemplo, no se puede trasformar un vector en una matriz.

A continuación se presenta un ejemplo de cómo se declaran arrays dinámicos en ***Visual Basic***. Si se declara el array ***Matriz*** a nivel del formulario,

```
Dim Matriz( ) As Integer
```

y más tarde, un procedimiento ***Calculo*** puede asignar espacio para el array, como se indica a continuación:

```
Sub Calculo( )
   ...
   ReDim Matriz(F, C)
   ...
End Sub
```

Cada vez que se ejecuta la sentencia ***ReDim***, todos los valores almacenados en el array se pierden (si son ***Variant*** se ponen a ***Empty***; si son numéricos a cero y si son cadenas de caracteres a la cadena vacía). Cuando interese cambiar el tamaño del array conservando los valores del array, hay que ejecutar ***ReDim*** con la palabra clave ***Preserve***. Por ejemplo, supóngase un array ***A*** de dos dimensiones. La sentencia,

```
ReDim Preserve A(D1, UBound(A, 2) + 2)
```

incrementa el tamaño del array en dos columnas más. Cuando se utiliza la palabra ***Preserve*** no puede cambiarse el índice inferior del array (sí el superior). La función ***UBound*** utilizada en este ejemplo es una función que devuelve el valor más alto de la segunda dimensión de la matriz (ver el ***Help*** para más información).

3.10 ESTRUCTURAS: SENTENCIA TYPE

Una ***estructura*** (según la nomenclatura típica del lenguaje C) es un nuevo tipo de datos, un tipo definido por el usuario, que puede ser manipulado de la misma forma que los tipos predefinidos (***Int***, ***Double***, ***String***, ...). Una estructura puede definirse como una colección o agrupación de datos de diferentes tipos evidentemente relacionados entre sí.

Para crear una estructura con ***Visual Basic 6.0*** hay que utilizar la sentencia ***Type ... End Type***. Esta sentencia solamente puede aparecer en la sección ***General*** o de declaraciones de un módulo. Pueden crearse como ***Public*** o como ***Private*** en un módulo estándar o de clase y sólo como ***Private*** en un formulario. ***Dim*** equivale a ***Public***. Véase el siguiente ejemplo,

```
Public Type Alumno
   Nombre    As String
   Direccion As String *40
   Telefono  As Long
   DNI       As Long
End Type
```

Este ejemplo declara un tipo de datos denominado ***Alumno*** que consta de cuatro *miembros* o campos, denominados ***Nombre***, ***Direccion***, ***Telefono*** y ***DNI***.

Una vez definido un nuevo tipo de datos, en un módulo estándar o de clase se pueden declarar variables ***Public*** o ***Private*** de ese tipo (en un formulario sólo ***Private***), como por ejemplo:

```
Public Mikel as Alumno
```

Para referirse a un determinado miembro de una estructura se utiliza la notación *variable.miembro*. Por ejemplo,

```
Mikel.DNI = 34103315
```

A su vez, un miembro de una estructura puede ser otra estructura, es decir un tipo definido por el usuario. Por ejemplo,

```
Type Fecha
   Dia  As Integer
   Mes  As Integer
   Anio As Integer
End Type


Type Alumno
   Alta       As Fecha
   Nombre     As String
   Direccion  As String * 40
   Telefono   AS Long
   DNI        As Long
End Type
```

Dentro de una estructura puede haber arrays tanto estáticos como dinámicos.

En ***Visual Basic 6.0*** se pueden definir ***arrays de estructuras***. La declaración de un array de estructuras se hará con la palabra ***Public***, ***Private*** o ***Static***, dependiendo de su ámbito. La forma de hacerlo es la siguiente:

```
Public   grupoA (1 To 100) As Alumno
Private  grupoB (1 To 100) As Alumno
Static   grupoC (1 To 100) As Alumno
```

En ***Visual Basic 6.0***, a la hora de declarar arrays de estructuras, sucede lo mismo que al declarar arrays de cualquier tipo de variables. Con ***GrupoA(1 To 100)***, se crea un vector de estructuras de 100 elementos de tipo ***Alumno*** (*grupoA(1)*, *grupoA(2)*, ..., *grupoA(100)*). Con ***grupoB(100)***, se crearía un vector de estructuras de 101 elementos (*grupoB(0)*, *grupoB(1)*, ... *grupoB(100)*).

Es posible asignar una estructura a otra del mismo tipo. En este caso se realiza una copia miembro a miembro. Véase el siguiente ejemplo:

```
GrupoA(1) = delegado
```

Las ***estructuras*** pueden ser también ***argumentos*** en las llamadas a funciones y procedimientos ***Sub***. Siempre son pasados p***or referencia***, lo cual implica que pueden ser modificados dentro del procedimiento y esas modificaciones permenecen en el entorno de llamada al procedimiento. En el caso de las funciones, las estructuras pueden ser también ***valores de retorno***.

3.11 FUNCIONES PARA MANEJO DE CADENAS DE CARACTERES

Existen varias funciones útiles para el manejo de *cadenas de caracteres* (***Strings***). Estas funciones se utilizan para la evaluación, manipulación o conversión de cadenas de caracteres. Algunas de ellas se muestran en la Tabla 3.4.

Utilidad	Función en Visual Basic 6.0	Comentarios
Número de caracteres de una cadena	Len(string | varname)	
Conversión a minúsculas o a mayúsculas	LCase(x), UCase(x)	
Conversión de cadenas a números y de números a cadenas	Str(n), CStr(n), Val(string)	
Extracción de un nº de caracteres en un rango, de la parte derecha o izquierda de una cadena	Mid(string, ini[, n]), Right(string, length), Left(string, length)	el parámetro ***n*** de Mid es opcional e indica el número de caracteres a extraer a partir de "***ini***"
Extracción de sub-cadenas	Split(string, [[delim], n])	devuelve un array con las ***n*** (-1 para todas) subcadenas separadas por ***delim*** (por defecto, el espacio)
Unión de sub-cadenas	Join(string, [delim])	
Comparación de cadenas de caracteres	strComp(str1, str2)	devuelve -1, 0, 1 según ***str1*** sea menor, igual o mayor que ***str2***
Hallar si una cadena es parte de otra (está contenida como sub-cadena)	InStr([n], str1, str2)	devuelve la posición de ***str2*** en ***str1*** buscando a partir del carácter ***n***
Hallar una cadena en otra a partir del final (reverse order)	InstrRev(str1, str2, [n])	devuelve la posición de ***str2*** en ***str1*** buscando a partir del carácter ***n***
Buscar y reemplazar una subcadena por otra en una cadena	Replace(string, substring, replacewith)	reemplaza ***substring*** por ***replacewith***

Tabla 3.4. Funciones de manejo de cadenas de caracteres en ***Visual Basic 6.0***.

Es necesario tener presente que cuando se quieren comparar dos cadenas de caracteres, dicha comparación se realiza por defecto en función del código ASCII asociado a cada letra (ver Anexo 8.1). Esto significa que por ejemplo *caña* es posterior a *casa* debido a que la letra ***ñ*** tiene un código ASCII asociado superior a la letra ***s*** (***ñ*** es el 164; ***s*** es el 115). Esto mismo ocurre con las vocales acentuadas. Si se desea conseguir una comparación alfabética lógica es necesario incluir al comienzo del fichero de código la sentencia ***Option Compare Text*** (frente a ***Option Compare Binary*** establecida por defecto). La función ***strComp()*** admite un tercer argumento que permite especificar el tipo de comparación (constantes ***vbBinaryCompare*** o ***vbTextCompare***).

Ejemplos:

```
MyDouble = 437.324              ' MyDouble es un Double.
MyString = CStr(MyDouble)       ' MyString contiene "437.324".
MyValue = Val("2457")           ' Devuelve 2457.
MyValue = Val(" 2 45 7")        ' Devuelve 2457.
MyValue = Val("24 and 57")      ' Devuelve 24.
```

```
AnyString = "Hello World"         ' Se define el string.
MyStr = Right(AnyString, 6)       ' Devuelve " World".
MyStr = Left(AnyString, 7)        ' Devuelve "Hello W".
MyStr = Right(AnyString, 20)      ' Devuelve "Hello World".

i = StrComp("casa", "caña")       ' Devuelve –1 por defecto y 1 con Option
                                    Compare Text
MyString = "Mid Function Demo"    ' Se crea un nuevo string.
LastWord = Mid(MyString, 14, 4)   ' Devuelve "Demo".
MidWords = Mid(MyString, 5)       ' Devuelve "Function Demo".
```

El operador ***Like*** permite comparar dos cadenas de caracteres. Si son iguales devuelve ***True*** y si no lo son, ***False***. Téngase en cuenta que ***Like*** es un ***operador***, no una función.

Existe además el ***operador de concatenación &*** que puede ser utilizado con cadenas de caracteres. Se utiliza para poner una cadena a continuación de otra. Por ejemplo:

```
str1 = "My first string" 'Se inicializan los strings
str2 = "My second string"
TextoFinal = str1 & str2 'TextoFinal vale "My first stringMy second string"
```

El ***operador "+"*** opera de forma análoga, pero su uso se desaconseja pues en ciertas ocasiones convierte las cadenas en números y realiza la suma.

Para obtener más información sobre cada una de las funciones buscar ***Strings*** en el ***Help*** de ***Visual Basic 6.0***.

3.12 FUNCIONES MATEMÁTICAS

Al igual que las funciones vistas para el manejo de cadenas de caracteres, existe una serie de funciones matemáticas las cuales permiten realizar cálculos dentro de un programa de ***Visual Basic***.

Dichas funciones se muestran en la Tabla 3.5:

Función matemática	**Función en Visual Basic**	**Función matemática**	**Función en Visual Basic**
Valor absoluto	Abs(x)	Nº aleatorio	Rnd
Arco tangente	Atn(x)	Seno y coseno	Sin(x), Cos(x)
Exponencial	Exp(x)	Tangente	Tan(x)
Parte entera	Int(x), Fix(x)	Raíz cuadrada	Sqr(x)
Logaritmo	Log(x)	Signo (1, 0, -1)	Sgn(x)
Redondeo	Round(x, ndec)		

Tabla 3.5. Funciones matemáticas en ***Visual Basic 6.0***.

Ejemplos:

```
MyNumber = Abs(50.3)                ' Devuelve 50.3.
MyNumber = Abs(-50.3)               ' Devuelve 50.3.
MyAngle = 1.3                       ' El ángulo debe estar en radianes.
MySecant = 1 / Cos(MyAngle)         ' Calcula la secante.
MySqr = Sqr(4)                      ' Devuelve 2.
MySqr = Sqr(23)                     ' Devuelve 4.79583152331272.
MyVar1 = 12: MyVar2 = -2.4: MyVar3 = 0 'Declaración de las variables
MySign = Sgn(MyVar1)                ' Devuelve 1.
MySign = Sgn(MyVar2)                ' Devuelve -1.
MySign = Sgn(MyVar3)                ' Devuelve 0.
```

Las funciones trigonométricas de ***Visual Basic*** utilizan ***radianes*** para medir los ángulos.

Con el fin de completar estas funciones, se ofrece a continuación una relación de funciones que son derivadas de las anteriores. El alumno podría programar dichas funciones en un fichero ***.bas*** y así poderlas utilizar posteriormente en cualquier programa. Dichas funciones se muestran en la Tabla 3.6:

Función matemática	**Expresión equivalente**
Secante	Sec(X) = 1 / Cos(X)
Cosecante	Cosec(X) = 1 / Sin(X)
Cotangente	Cotan(X) = 1 / Tan(X)
Arcoseno	Arcsin(X) = Atn(X / Sqr(-X * X + 1))
Arcocoseno	Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1)
Arcosecante	Arcsec(X) = Atn(X / Sqr(X * X – 1)) + Sgn((X) -1) * (2 * Atn(1))
Arcocosecante	Arccosec(X) = Atn(X / Sqr(X * X - 1)) + (Sgn(X) - 1) * (2 * Atn(1))
Arcocotangente	Arccotan(X) = Atn(X) + 2 * Atn(1)
Seno Hiperbólico	HSin(X) = (Exp(X) - Exp(-X)) / 2
Coseno Hiperbólico	Hcos(X) = (Exp(X) + Exp(-X)) / 2
Tangente Hiperbólica	Htan(X) = (Exp(X) - Exp(-X)) / (Exp(X) + Exp(-X))
Secante Hiperbólica	HSec(X) = 2 / (Exp(X) + Exp(-X))
Cosecante Hiperbólica	Hcosec(X) = 2 / (Exp(X) - Exp(-X))
Cotangente Hiperbólica	Hcotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X))
Arcoseno Hiperbólico	Harcsin(X) = Log(X + Sqr(X * X + 1))
Arcocoseno Hiperbólico	Harccos(X) = Log(X + Sqr(X * X - 1))
Arcotangente Hiperbólica	Harctan(X) = Log((1 + X) / (1 - X)) / 2
Arcosecante Hiperbólica	Harcsec(X) = Log((Sqr(-X * X + 1) + 1) / X)
Arcocosecante Hiperbólica	Harccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) +1) / X)
Arcocotangente Hiperbólica	Harccotan(X) = Log((X + 1) / (X – 1)) / 2
Logaritmo en base N	LogN(X) = Log(X) / Log(N)

Tabla 3.6. Funciones auxiliares matemáticas (no las tiene ***Visual Basic 6.0***).

4. EVENTOS, PROPIEDADES Y CONTROLES

En este capítulo se pretende recoger de una manera más sistemática y general los eventos y controles más habituales de ***Visual Basic 6.0***. Hay que señalar que en ningún momento se pretende abandonar el carácter introductorio de este manual, y que ***Visual Basic 6.0*** tiene muchas más posibilidades de las que aquí se muestran. Por ejemplo, muchos de los controles y eventos de ***Visual Basic 6.0*** están relacionados con el acceso a bases de datos. Estos aspectos no se citarán en estos apuntes. Para una información más detallada se puede acudir a un buen libro de referencia o al ***Help*** del programa.

La programación en ***Visual Basic 6.0*** (al menos para ejemplos sencillos) suele proceder del siguiente modo:

1. Se definen interactivamente sobre el formulario los controles que van a constituir la aplicación.
2. Se define para cada control el código con el que se va a responder a cada uno de los eventos. Para ello basta clicar dos veces sobre el control y se abre una ventana de código como la mostrada en la Figura 4.1. En ella ***Visual Basic 6.0*** ha preparado ya el inicio y el final de la función con la que se va a responder al evento. El nombre del evento forma parte del nombre de la función, junto al nombre del control. En el ejemplo del la Figura 4.1 está preparada la función para escribir el código que se ejecutará al producirse el evento ***Click*** sobre el control ***cmbSalir***.

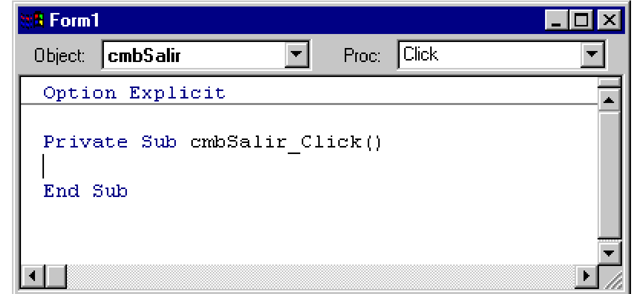


Figura 4.1. Código que gestionará el evento ***Click*** sobre el control de nombre ***cmbSalir***.

En el resto de este capítulo se verán con un cierto detalle los eventos, controles y propiedades más habituales en ***Visual Basic 6.0***.

4.1 EVENTOS

A continuación se presentan brevemente los eventos más normales que reconoce ***Visual Basic 6.0***. Es importante tener una visión general de los eventos que existen en ***Windows 95/98/NT*** porque cada control de los que se verán más adelante tiene su propio conjunto de eventos que reconoce, y otros que no reconoce. Cualquier usuario de las aplicaciones escritas para ***Windows 95/98/NT*** hace uso continuo e intuitivo de los eventos, pero es posible que nunca se haya detenido a pensar en ello.

Para saber qué eventos puede recibir un control determinado basta seleccionarlo y pulsar <F1>. De esta forma se abre una ventana del ***Help*** que explica el control y permite acceder a los eventos que soporta.

4.1.1 Eventos generales

4.1.1.1 Carga y descarga de formularios

Cuando se arranca una aplicación, o más en concreto cuando se visualiza por primera vez un formulario se producen varios eventos consecutivos: ***Initialize***, ***Load***, ***Activate*** y ***Paint***. Cada uno de

estos eventos se puede aprovechar para realizar ciertas operaciones por medio de la función correspondiente.

Al ocultar, cerrar o eliminar un formulario se producen otra serie de eventos: ***Deactivate***, ***QueryUnload***, ***Unload*** y ***Terminate*** que se verán en un próximo ejemplo.

Para inicializar las variables definidas a nivel de módulo se suele utilizar el evento ***Initialize***, que tiene lugar antes que el ***Load***. El evento ***Load*** se activa al cargar un formulario. Con el formulario principal esto sucede al arrancar la ejecución de un programa; con el resto de los formularios al mandarlos cargar desde cualquier procedimiento o al hacer referencia a alguna propiedad o control de un formulario que no esté cargado. Al descargar un formulario se produce el evento ***Unload***. Si se detiene el programa desde el botón ***Stop*** de ***Visual Basic 6.0*** (o del menú correspondiente) o con un ***End,*** no se pasa por el evento ***Unload***. Para pasar por el evento ***Unload*** es necesario cerrar la ventana con el botón de cerrar o llamarlo explícitamente. El evento ***QueryUnload*** se produce antes del evento ***Unload*** y permite por ejemplo enviar un mensaje de confirmación.

El evento ***Load*** de un formulario se suele utilizar para ejecutar una función que dé valor a sus propiedades y a las de los controles que dependen de dicho formulario. No se puede utilizar para dibujar o imprimir sobre el formulario, pues en el momento en que se produce este evento el formulario todavía no está disponible para dichas operaciones. Por ejemplo, si en el formulario debe aparecer la salida del método ***Print*** o de los métodos gráficos ***Pset***, ***Line*** y ***Circle*** (que se estudian en el Capítulo 6 de este manual) puede utilizarse el evento ***Paint*** u otro posterior (por ejemplo, el evento ***GotFocus*** del primer control) pero no puede utilizarse el evento ***Load***.

Se puede ocultar un formulario sin descargarlo con el método ***Hide*** o haciendo la propiedad ***Visible = False***. Esto hace que el formulario desaparezca de la ventana, aunque sus variables y propiedades sigan estando accesibles y conservando sus valores. Para hacer visible un formulario oculto pero ya cargado se utiliza el método ***Show***, que equivale a hacer la propiedad ***Visible = True***, y que genera los eventos ***Activate*** y ***Paint***. Si el formulario no había sido cargado previamente, el método ***Show*** genera los cuatro eventos mencionados.

Cuando un formulario pasa a ser la ventana activa se produce el evento ***Activate*** y al dejar de serlo el evento ***Deactivate***. En el caso de que el formulario que va a ser activo no estuviera cargado ya, primero sucederían los eventos ***Initialize***, ***Load*** y luego los eventos ***Activate*** y ***Paint***.

Todo esto se puede ver y entender con un simple ejemplo, mostrado en la Figura 4.2. Se han de crear dos formularios (***frmPrincipal*** y ***frmSecundario***). El primero de ellos contendrá dos botones (***cmdVerSec*** y ***cmdSalir***) y el segundo tres (***cmdHide***, ***cmdUnload*** y ***cmdTerminate***). El formulario principal será el primero que aparece, y sólo se verá el segundo si se clica en el botón ***Cargar Formulario***. Cuando así se haga, a medida que los eventos antes mencionados se vayan sucediendo, irán apareciendo en pantalla unas cajas de mensajes que tendrán como texto el nombre del evento que se acaba de producir. Según con cual de los tres botones se haga desaparecer el segundo formulario, al volverlo a ver se producirán unos eventos u otros, según se puede ver por los mensajes que van apareciendo con cada evento.

```
' código del form. principal
Private Sub cmdCargar_Click()
    frmSecundario.Show
End Sub

' código del form. secundario
Private Sub cmdHide_Click()
    Hide
End Sub
```

```
Private Sub cmdUnload_Click()
    Unload Me
End Sub

Private Sub cmdTerminate_Click()
    Hide
    Set Form2 = Nothing
End Sub

Private Sub Form_Activate()
    MsgBox ("Evento Activate")
End Sub

Private Sub Form_Deactivate()
    MsgBox ("Evento Deactivate")
End Sub

Private Sub Form_Initialize()
    MsgBox ("Evento Initialize")
End Sub

Private Sub Form_Load()
    MsgBox ("Evento Load")
End Sub

Private Sub Form_Paint()
    MsgBox ("Evento Paint")
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    MsgBox ("Evento QueryUnload")
End Sub

Private Sub Form_Terminate()
    MsgBox ("Evento Terminate")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    MsgBox ("Evento Unload")
End Sub
```

Figura 4.2. Resultado del ejemplo de carga de formularios.

Es muy interesante realizar este ejemplo y seguir la secuencia de eventos que se producen al hacer aparecer y desaparecer los formularios.

4.1.1.2 Paint

El evento ***Paint*** sucede cuando hay que redibujar un formulario o ***PictureBox.*** Esto sucede cuando esos objetos se hacen visibles por primera vez y también cuando vuelven a ser visibles después de haber estado tapados por otros, tras haber sido movidos o tras haber sido modificados de tamaño.

4.1.1.3 El foco (focus)

En todas las aplicaciones de ***Windows***, en cualquiera de sus versiones, siempre hay un único control, formulario o ventana que puede recibir entradas desde teclado. En cada momento ese control, ventana o formulario es el que dispone del "foco" (***focus***). El objeto que posee el foco está caracterizado por estar ***resaltado*** con letra negrita, con un contorno más vivo o teniendo parpadeando el cursor en él. Este foco puede ser trasladado de un objeto a otro por código o por interacciones del usuario, como por ejemplo clicando con el ratón en distintos puntos de la pantalla o pulsando la tecla ***Tab***. Cada vez que un objeto pierde el foco se produce su evento ***LostFocus*** y, posteriormente, el evento ***GotFocus*** del objeto que ha recibido el foco.

El método ***SetFocus*** permite dar el focus al objeto al que se aplica.

Dos propiedades de muchos controles relacionadas con el foco son ***TabIndex*** y ***TabStop***. ***TabStop*** determina si el foco se va o no a posar en el objeto al pulsar la tecla ***Tab*** (si ***TabStop*** está a ***False*** no se puede obtener el foco mediante el tabulador) y ***TabIndex*** determina el orden en el que esto va a suceder. Así al cargar un formulario, el foco estará en aquel objeto cuyo ***TabIndex*** sea 0. Al pulsar la tecla ***Tab*** el foco irá al objeto que tenga ***TabIndex*** = 1 y así sucesivamente. Para retroceder en esta lista se pulsa ***Mayúsculas***+***Tab***. La propiedad ***TabIndex*** se puede determinar en tiempo de diseño por medio de la caja de propiedades de un control, del modo habitual.

Cuando a un control se le asigna un determinado valor de ***TabIndex***, ***Visual Basic*** ajusta automáticamente los valores de los demás controles (si tiene que desplazarlos hacia arriba o hacia abajo, lo hace de modo que siempre tengan números consecutivos). Para que un ***formulario*** reciba el foco es necesario que no haya en él ningún control que sea capaz de recibirlo.

Un grupo de ***botones de opción*** tiene un único ***TabIndex***, es decir, se comporta como si fuera un único control. Para elegir una u otra de las opciones se utilizan las flechas del teclado (↑ y ↓).

4.1.1.4 KeyPress, KeyUp y KeyDown

El evento ***KeyPress*** sucede cuando el usuario pulsa y suelta determinada tecla. En el procedimiento asociado con este evento el único argumento ***KeyAscii*** es necesario para conocer cuál es el código ASCII de la tecla pulsada. El evento ***KeyDown*** se produce cuando el usuario pulsa determinada tecla y el evento ***KeyUp*** al soltar una tecla.

Los eventos ***KeyUp*** y ***KeyDown*** tienen un segundo argumento llamado ***Shift*** que permiten determinar si esa tecla se ha pulsado estando pulsadas a la vez cualquier combinación de las teclas ***Shift***, ***Alt*** y ***Ctrl***. En un apartado próximo se explica cómo se identifican las teclas pulsadas a partir del argumento ***Shift***.

4.1.2 Eventos relacionados con el ratón

4.1.2.1 Click y DblClick

El evento ***Click*** se activa cuando el usuario pulsa y suelta rápidamente uno de los botones del ratón. También puede activarse desde código (sin tocar el ratón) variando la propiedad ***Value*** de alguno de los controles. En el caso de un formulario este evento se activa cuando el usuario clica sobre una zona del formulario en la que no haya ningún control o sobre un control que en ese momento esté inhabilitado (propiedad ***Enabled*** = *False*). En el caso de un control, el evento se activa cuando el usuario realiza una de las siguientes operaciones:

- Clicar sobre un control con el botón derecho o izquierdo del ratón. En el caso de un botón de comando, de un botón de selección o de un botón de opción, el evento sucede solamente al clicar con el botón izquierdo.
- Seleccionar un registro de alguno de los varios tipos listas desplegables que dispone ***Visual Basic***.
- Pulsar la ***barra espaciadora*** cuando el foco está en un botón de comando, en un botón de selección o en un botón de opción.
- Pulsar la tecla ***Return*** cuando en un formulario hay un botón que tiene su propiedad ***Default*** = ***True***.

- Pulsar la tecla ***Esc*** cuando en un formulario hay un botón que tiene su propiedad ***Cancel*** = ***True.***
- Pulsar una combinación de teclas aceleradoras (*Alt* + otra tecla, como por ejemplo cuando de despliega el menú ***File*** de ***Word*** con *Alt+F*) definidas para activar un determinado control de un formulario.

También se puede activar el evento ***Click*** desde código realizando una de las siguientes operaciones:

- Hacer que la propiedad ***Value*** de un botón de comando valga ***True***.
- Hacer que la propiedad ***Value*** de un botón de opción valga ***True***
- Modificar la propiedad ***Value*** de un botón de selección.

El evento ***DblClick*** sucede al clicar dos veces seguidas sobre un control o formulario con el botón izquierdo del ratón.

4.1.2.2 MouseDown, MouseUp y MouseMove

El evento ***MouseDown*** sucede cuando el usuario pulsa cualquiera de los botones del ratón, mientras que el evento ***MouseUp*** sucede al soltar un botón que había sido pulsado. El evento ***MouseMove*** sucede al mover el ratón sobre un control o formulario.

Los eventos ***MouseUp*** y ***MouseDown*** tienen algunos argumentos que merecen ser comentados. El argumento ***Button*** indica cuál de los botones del ratón ha sido pulsado o soltado, y el argumento ***Shift*** indica si además alguna de las teclas ***alt***, ***shift*** o ***ctrl*** está también pulsada. La lista con todos los posibles valores de estos argumentos se muestra en la Tabla 4.1:

Cte simbólica	Valor	Acción	Cte simbólica	Valor	Acción
vbLeftButton	1	Botón izdo pulsado o soltado	vbShiftMask	1	Tecla SHIFT pulsada
vbRightButton	2	Botón dcho pulsado o soltado	vbCtrlMask	2	Tecla CTRL pulsada
vbMiddleButton	4	Botón central pulsado o soltado	vbAltMask	4	Tecla ALT pulsada

Tabla 4.1. Valores de los argumentos de los eventos ***MouseUp*** y ***MouseDown***.

Con estos valores se aplica la aritmética booleana, lo cual quiere decir que si se pulsan simultáneamente los botones izquierdo y derecho del ratón el argumento ***Button*** valdrá 3 (1+2) y si se pulsan las tres teclas ***shift***, ***ctrl*** y ***alt*** simultáneamente el argumento ***Shift*** valdrá 7 (1+2+4). Con esta forma de combinar los valores se resuelven todas las indeterminaciones posibles.

4.1.2.3 DragOver y DragDrop

El evento ***DragOver*** sucede mientras se está arrastrando un objeto sobre un control. Suele utilizarse para variar la forma del cursor que se mueve con el ratón dependiendo de si el objeto sobre el que se encuentra el cursor en ese momento es válido para soltar o no. El evento ***DragDrop*** sucede al concluir una operación de arrastrar y soltar. El evento ***DragOver*** requiere de los argumentos que se muestran a continuación:

```
Private Sub Text1_DragOver(Source As Control, _
                           X As Single, Y As Single, State As Integer)
    ...
End Sub
```

Los argumentos de este evento son ***Source*** que contiene el objeto que está siendo arrastrado, ***X*** e ***Y*** que indican la posición del objeto arrastrado dentro del sistema de coordenadas del objeto sobre el que se está arrastrando y ***State*** (que es propio del ***DragOver***, pero no aparece en el ***DragDrop***) que vale 0, 1 ó 2 según se esté *entrando*, *saliendo* o *permaneciendo dentro* del mismo objeto, respectivamente. Es importante señalar que el evento ***DragOver*** pertenece al objeto sobre el que se arrastra, no al objeto que es arrastrado.

En el ***Help*** se puede encontrar información sobre la propiedad ***DragMode***, que puede tomar dos valores (***vbManual*** y ***vbAutomatic***). Esta constante determina cómo comienza una oparación de arrastre de un objeto. En modo manual se debe comenzar llamando al método ***Drag*** para el objeto a arrastrar. En modo automático basta clicar sobre el objeto a arrastrar, pero puede que en esta caso dicho objeto no responda del modo habitual a otros eventos.

4.2 ALGUNAS PROPIEDADES COMUNES A VARIOS CONTROLES

Hay algunas propiedades que son comunes a muchos controles. A continuación se hace una lista con las utilizadas más habitualmente:

- ***Appearance***: Establece si un objeto tiene un aspecto plano (valor 0) o tridimensional (valor 1).
- ***BackColor***: Establece el color de fondo de un objeto.
- ***Caption***: Establece el texto que aparece dentro o junto al objeto. Tiene el papel de un título.
- ***Enabled***: Establece si un objeto es accesible y modificable o no.
- ***Font***: Establece las características del tipo de letra del objeto.
- ***ForeColor***: Establece el color del texto y/o gráficos de un objeto.
- ***Height*** y ***Width***: Establecen la altura y achura de un objeto.
- ***Left*** y ***Top***: Establecen la distancia horizontal y vertical entre el origen del control y el origen del objeto que lo contiene, que puede ser un formulario, un marco (frame), etc.
- ***MousePointer:*** Establece la forma que adoptará el puntero del ratón al posicionarse sobre el objeto. Esta forma puede elegirse dentro de una lista en las que aparecen las habituales del puntero del ratón o creando iconos propios. Algunas constantes de significado inmediato que definen la forma del cursor son las siguientes: ***vbDefault***, ***vbArrow***, ***vbCrosshair***, ***vbIbeam***, ***vbSizePointer***, ***vbUpArrow***, ***vbHourglass***, etc. Para más información puede consultarse el ***Help*** de ***MousePointer***.
- ***Name***: Nombre del objeto. Todos los objetos incluidos en un formulario deben tener un nombre con el que poder referirse a él a la hora de programar la forma en que debe actuar. Existen unas reglas para definir los nombre de los controles, que ya se vieron en el Capítulo 1.
- ***Visible***: Establece si el objeto es visible o invisible.

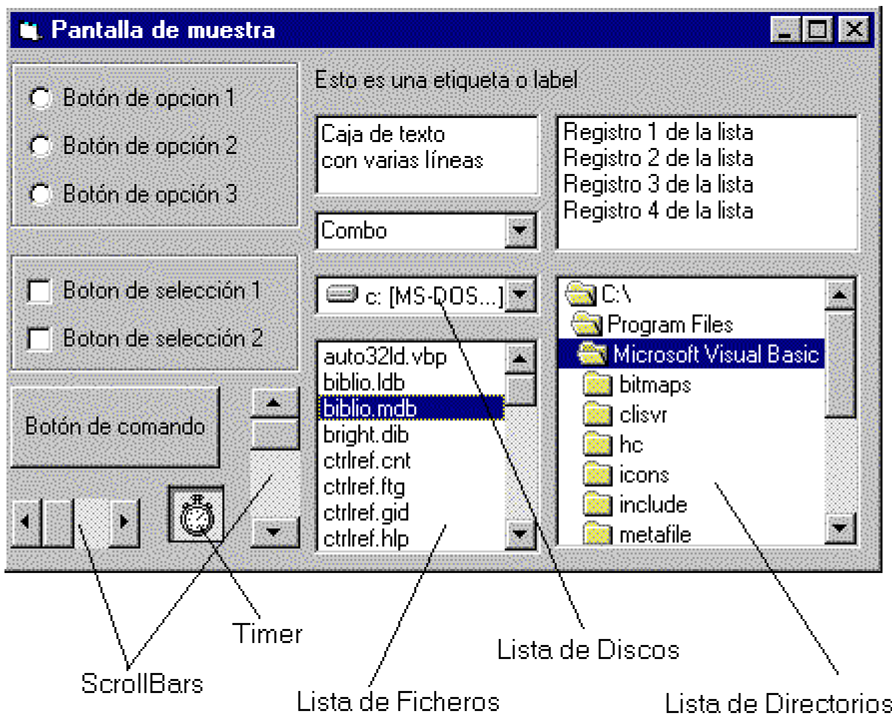
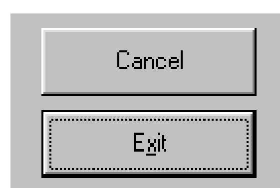


Figura 4.3. Algunos de los controles más habituales de ***Visual Basic***.

4.3 CONTROLES MÁS USUALES

En la Figura 4.3 se muestran algunos de los controles más habituales en ***Visual Basic 6.0***. Estos controles se explican a continuación con más detalle.

4.3.1 Botón de comando (Command Button)



La propiedades más importantes del botón de comando son su ***Caption***, que es lo que aparece escrito en él, las referentes a su posición (***Left*** y ***Top***) y apariencia externa (***Height***, ***Width*** y tipo de letra) y la propiedad ***Enabled***, que determina si en un momento dado puede ser pulsado o no. No hay que confundir la propiedad ***Caption*** con la propiedad ***Name***. La primera define a un texto que aparecerá escrito en el control, mientras que las segunda define el nombre interno con el que se puede hacer referencia al citado objeto.

Si en la propiedad ***Caption*** se pone el carácter (&) antes de una de sus letras, dicha letra aparece subrayyada en el botón (como la “x” en el botón ***Exit*** de la figura anexa). Esto quiere decir que, como es habitual en ***Windows***, dicho botón puede activarse con el teclado por medio de la combinación ***Alt+letra subrayada***. Esta característica es común a muchos de los controles que tienen propiedad ***Caption***.

El evento que siempre suelen tener programado los botones de comandos es el evento ***Click***.

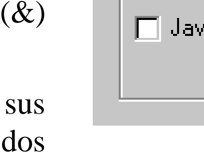
4.3.2 Botones de opción (Option Button)

Además de las mencionadas para el caso anterior estos botones tienen la propiedad ***Value***, que en un determinado momento sólo puede ser ***True*** en uno de los botones del grupo ya que se trata de opciones que se excluyen mutuamente.

Para agrupar botones se coloca primero un ***marco*** o ***frame*** en el formulario y, estando seleccionado, se colocan después cuantos botones de opción se desee. En un mismo formulario se pueden colocar cuantos grupos de botones de opción se quiera, cada uno de ellos agrupado dentro de su propio marco. Es muy importante colocar primero el ***frame*** y después los botones de opción. Con esto se consigue que los botones de opción estén agrupados, de modo que sólo uno de ellos pueda estar activado. Si no se coloca ningún ***frame*** todos los botones de opción de un mismo formulario forman un único grupo. Si los botones ya existen y se quieren introducir un un ***frame*** se seleccionan, se hace ***Cut*** y luego ***Paste*** dentro del ***frame*** seleccionado.

Sólo un grupo de botones de opción puede recibir el ***focus***, no cada botón por separado. Cuando el grupo tiene el ***focus***, con las flechas del teclado (↑ y ↓) se puede activar una u otra opción sin necesidad de usar el ratón. También se puede utilizar ***Alt+carácter*** introduciendo antes de dicho carácter un (&) en el ***Caption*** del botón de opción.

4.3.3 Botones de selección (Check Box)



La única diferencia entre estos botones y los anteriores es que en los botones de selección puede haber más de uno con la propiedad ***Value*** a ***True***. Estos botones no forman grupo aunque estén dentro de un ***frame***, y reciben el ***focus*** individualmente. Se puede también utilizar el carácter (&) en el ***Caption*** para activarlos con el teclado.

El usuario debe decidir qué tipo de botones se ajustan mejor a sus necesidades: en el caso de la edad, está claro que no se puede ser de dos edades diferentes; sí es posible sin embargo conocer varios lenguajes de programación.

4.3.4 Barras de desplazamiento (Scroll Bars)

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

El evento que se programa habitualmente es ***Change,*** que se activa cuando la barra de desplazamiento modifica su valor. Todo lo comentado en este apartado es común para las barras de desplazamiento verticales y horizontales.

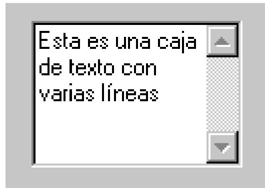
Además de las ***Scroll Bars*** horizontal y vertical, ***Visual Basic 6.0*** dispone también del control ***Slider***, utilizado en los paneles de control de ***Windows***, que tiene una función similar.

4.3.5 Etiquetas (Labels)

En las etiquetas o labels la propiedad más importante es ***Caption***, que contiene el texto que aparece sobre este control. Esta propiedad puede ser modificada desde programa, pero no interactivamente clicando sobre ella (a diferencia de las ***cajas de texto***, que se verán a continuación). Puede controlarse su tamaño, posición, color de fondo y una especie de borde 3-D. Habitualmente las ***labels*** no suelen recibir eventos ni contener código.

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

4.3.6 Cajas de texto (Text Box)



La propiedad más importante de las cajas de texto es ***Text***, que almacena el texto contenido en ellas. También se suelen controlar las que hacen referencia a su tamaño, posición y apariencia. En algún momento se puede desear impedir el acceso a la caja de texto, por lo que se establecerá su propiedad ***Enabled*** como ***False***. La propiedad ***Locked*** como ***True*** hace que la caja de texto sea de sólo lectura. La propiedad ***MultiLine***, que sólo se aplica a las cajas de texto, determina si en una de ellas se pueden incluir más de una línea o si se ignoran los saltos de línea. La justificación o centrado del texto se controla con la propiedad ***Alignment***. La propiedad ***ScrollBars*** permite controlar el que aparezca ninguna, una o las dos barras de desplazamiento de la caja.

En una caja de texto no se pueden introducir ***Intros*** con el teclado en modo de diseño. En modo de ejecución se deben introducir como caracteres ASCII (el 13 seguido del 10, *esto Carriage Return* y *Line Feed*). Afortunadamente ***Visual Basic 6.0*** dispone de la constante ***vbCrLf***, que realiza esta misión de modo automático.

Otras propiedades importantes hacen referencia a la selección de texto dentro de la caja, que sólo están disponibles en tiempo de ejecución. La propiedad ***SelStart*** sirve para posicionar el cursor al comienzo del texto que se desea seleccionar (el primer carácter es el cero); ***SelLength*** indica el número de caracteres o longitud de la selección; ***SelText*** es una cadena de caracteres que representa el texto seleccionado. Para hacer ***Paste*** con otro texto sustituyendo al seleccionado basta asignarle a esta propiedad ese otro texto (si no hay ningún texto seleccionado, el texto de ***SelText*** se inserta en la posición del cursor); para entresacar el texto seleccionado basta utilizar esta propiedad en alguna expresión.

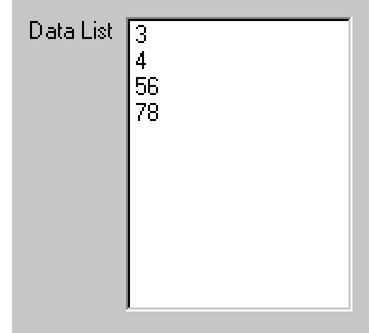
Los eventos que se programan son ***Change***, cuando se quiere realizar alguna acción al modificar el contenido de la caja, ***Click*** y ***DblClick*** y en algunos casos especiales ***KeyPress*** para controlar los caracteres que se introducen. Por ejemplo, se puede chequear la introducción del código ASCII 13 (***Intro***) para detectar que ya se finalizado con la introducción de datos. También se utiliza la propiedad ***MaxLength*** para determinar el número máximo de caracteres que pueden introducirse en la caja de texto.

En aquellos casos en los que se utilice una caja de texto como ***entrada de datos*** (es el control que se utiliza la mayoría de las veces con esta finalidad), puede ser interesante utilizar el método ***SetFocus*** para enviar el foco a la caja cuando se considere oportuno.

Otras propiedades de las cajas de texto hacen referencia a los tipos de letra y al estilo. Así la propiedad ***FontName*** es una cadena que contiene el nombre del *Font* (***Courier New***, ***Times New***

Roman, etc.), ***FontSize*** es un tipo ***Short*** que contiene el tamaño de la letra, y ***FontBold***, ***FontItalic***, ***FontUnderline*** y ***FontStrikethrough*** son propiedades tipo ***Boolean*** que indican si el texto va a tener esa característica o no.

4.3.7 Listas (List Box)



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

Para añadir o eliminar registros de la lista en modo de ejecución se utilizan los métodos ***AddItem*** y ***RemoveItem***. Las listas se suelen inicializar desde el evento ***Form_Load***.

La propiedad ***List*** es un array que permite definir el contenido de la lista en modo de diseño a través de la ventana de propiedades. ***List*** permite también acceder a los elementos de la lista en tiempo de ejecución, para utilizar y/o cambiar su valor. Para ello se pone en índice del elemento entre paréntesis (empezando a contar por cero) a continuación de ***List***, como se muestra a continuación por ejemplo, para cambiar el tercer elemento:

```
lstName.List(2) = "Tercero"
```

Para añadir un registro en tiempo de ejecución se utiliza ***AddItem***:

```
lstName.AddItem Registro_Añadido, posicion
```

donde ***posicion*** es un argumento opcional que permite especificar la posición en que se debe añadir. Si se omite el registro se añade al final de la lista. Lo anterior es válido si la propiedad ***Sorted*** está a ***False***; si está a ***True*** el nuevo registro se añade en la posición ordenada que le corresponde. Para eliminar un registro,

```
lstName.RemoveItem Posición_del_registro_en_la_lista
```

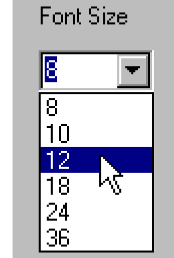
En el caso de que se quiera vaciar completamente el contenido de una lista se puede utilizar el método ***Clear***.

Dos propiedades interesantes de las listas son ***ListCount*** y ***ListIndex***. La primera contiene el número total de registros incluidos en la lista. La segunda permite acceder a una posición concreta de la lista para añadir un registro nuevo en esa posición, borrar uno ya existente, seleccionarlo, etc. Hay que recordar una vez más que los elementos de la lista ***se empiezan a numerar por cero***. El valor de propiedad ***ListIndex*** en cada momento coincide con el registro seleccionado y en el caso de no haber ninguno seleccionado esta propiedad vale -1.

Es interesante saber que al seleccionar uno de los registros de la lista se activa el evento ***Click*** de dicha lista.

Las listas permiten selecionar más de un elemento poniendo la propiedad ***MultiSelect*** a valor ***1-Simple*** o ***2-Extended***. En el primer caso los elementos se seleccionan o se elimina la selección simplemente clicando sobre ellos. En el segundo caso la forma de hacer selecciones múltiples es la típica de ***Windows***, utilizando las teclas ***Ctrl*** y ***Shift***. Con selección múltile la propiedad ***SelCount*** indica el número de elementos seleccionados, mientras que la propiedad ***Selected()*** es un array de valores boolean que indica si cada uno de los elementos de la lista está seleccionado o no.

4.3.8 Cajas combinadas (ComboBox)



Un ***ComboBox*** tiene muchas cosas en común con una ***lista***. Por ejemplo los métodos ***AddItem***, ***RemoveItem*** o ***Clear*** y las propiedades ***List***, ***ListIndex*** o ***ListCount***.

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

1. ***Style***=0 ó ***Style***=***vbComboDropDown*** (***Dropdown Combo***), Éste es el valor más habitual y corresponde con el caso en el que sólo se muestra el registro seleccionado, que es editable por el usuario, permaneciendo el resto oculto hasta que el usuario despliega la lista completa clicando sobre el botón-flecha.
2. ***Style=1*** ó ***Style***=***vbComboSimple*** (***Simple Combo***). En este caso el registro seleccionado también es editable, y se muestra una lista no desplegable dotada si es necesario de una ***scrollbar***.
3. ***Style=2 ó Style=vbComboDropDownList*** (***DropDown List***). En este último caso el registro seleccionado no es editable y la lista es desplegable.

En una ***caja combinada***, al igual que en una ***caja de texto*** sencilla, está permitido escribir con el teclado en tiempo de ejecución, si la propiedad ***Enabled*** vale ***True***. En una ***lista*** esto no es posible.

La propiedad ***Text*** corresponde con lo que aparece en la parte de caja de texto del control ***ComboBox***, bien sea porque el usuario lo ha introducido, bien porque lo haya seleccionado.

4.3.9 Controles relacionados con ficheros

Trabajando en un entorno ***Windows 95/98/NT*** es habitual tener que abrir y cerrar ficheros parar leer datos, guardar un documento, etc. Hay tres controles básicos que resultan de suma utilidad en esta tarea. Son la lista de unidades lógicas o discos (***Drive ListBox***), la lista de directorios (***Dir ListBox***) y la lista de ficheros (***File ListBox***). Estos controles se tratan con más detalle en el Capítulo 7.

4.3.10 Control Timer

Si se desea que una acción suceda con cierta periodicidad se puede utilizar un control ***Timer***. Este control produce de modo automático un evento cada cierto número de milisegundos y es de fundamental importancia para crear ***animaciones*** o aplicaciones con movimiento de objetos. La propiedad más importante de un objeto de este tipo es ***Interval***, que determina, precisamente, el intervalo en milisegundos entre eventos consecutivos. La acción que se desea activar debe programarse en el evento ***Timer*** de ese mismo control.

Si en algún momento se desea detener momentáneamente la acción periódica es suficiente con hacer ***False*** la propiedad ***Enabled*** del control ***Timer*** y para arrancarla de nuevo volver a hacer ***True*** esa propiedad. Haciendo 0 la propiedad ***Interval*** también se consigue inhabilitar el ***Timer***.

4.4 CAJAS DE DIÁLOGO ESTÁNDAR (CONTROLES COMMON DIALOG)

El control de ***cuadro de diálogo estándar*** de ***Windows 95/NT*** (***Common Dialog***) ofrece una forma sencilla y eficiente de realizar algunas de las tareas más comunes de un programa, tales como la

selección de un fichero para lectura/escritura, la impresión de un fichero o la selección de un tipo de letra o un color.

Lo primero que hay que hacer es ubicar el control en el formulario. El control se representará como un icono de tamaño invariable. No es posible especificar la ubicación que tendrá la caja de diálogo cuando se abra en la pantalla, ya que se trata de una propiedad no accesible por el usuario.

Un único ***cuadro de diálogo estándar*** puede bastar para realizar todas las funciones que se deseen, es decir, no es necesario insertar un cuadro de diálogo para imprimir un texto y otro para guardarlo, sino que ambos pueden compartir el mismo cuadro de diálogo simplemente invocando a uno u otro tipo en tiempo de ejecución (no es posible indicarlo en tiempo de diseño). Para ello se dispone de los métodos siguientes***: ShowColor, ShowFont, ShowHelp, ShowOpen, ShowPrinter*** y ***ShowSave.*** En ocasiones interesará introducir varios controles diferentes por motivos de claridad o para que ciertas propiedades sean distintas.

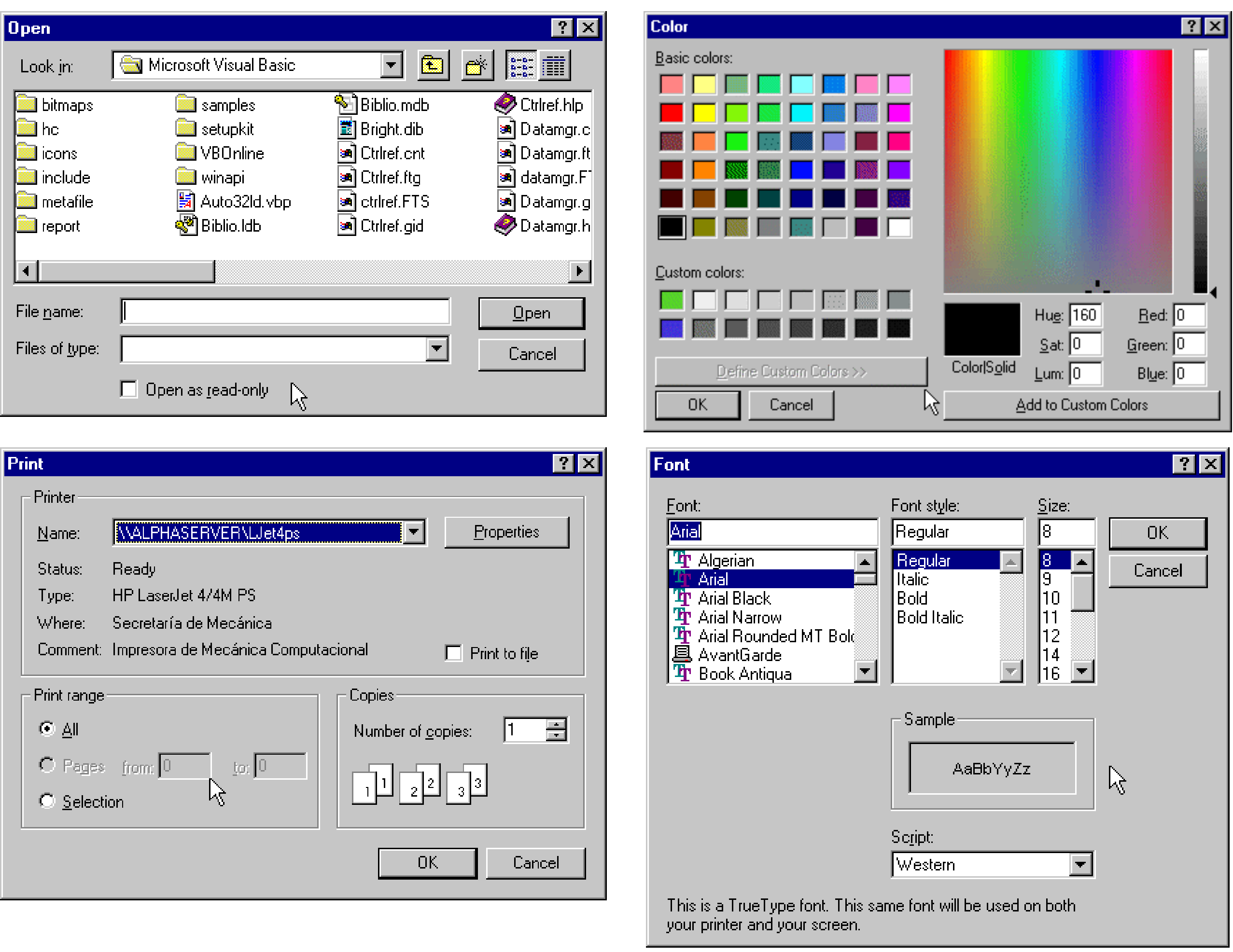


Figura 4.4. Controles Common Dialog.

En la Figura 4.4 se pueden observar distintos tipos de control ***Common Dialog***. Por ejemplo, si se desea visualizar un cuadro de diálogo para abrir un fichero, habrá que escribir:

```
dlgAbrir.ShowOpen
```

donde ***dlgAbrir*** es el nombre asignado al control ***Common Dialog***.

Las principales ***propiedades*** de este control en cada una de sus variantes se explican en los apartados siguientes. La propiedad ***Flag*** existe para todos los controles y determina algunas de sus características más importantes.

4.4.1 Open/Save Dialog Control

Las propiedades más importantes de este control son:

- ***DefaultExt***: Es la extensión por defecto a utilizar para abrir/salvar archivos. Con ***Save***, si el nombre del fichero se teclea sin extensión, se añade esta extensión por defecto.
- ***DialogTitle***: Devuelve o da valor al título de la caja de diálogo (cadena de caracteres).
- ***FileName***: Nombre completo del archivo a abrir/salvar, incluyendo el *path*.
- ***FileTitle***: Nombre del archivo a abrir/salvar pero sin la ruta de acceso correspondiente.
- ***Filter***: Contiene los filtros de selección que aparecerán indicados en la parte inferior de la pantalla en la lista de tipos de archivo. Pueden indicarse múltiples tipos de archivo, separándolos mediante un barra vertical (***Alt Gr*** +< ***1***>). Su sintaxis es la siguiente:

  ```
  Objeto.Filter = "(descripción a aparecer en la listbox)|filtro"
  ```

  Por ejemplo:

  ```
  "Texto (*.txt)|*.txt|Imágenes(*.bmp;*.ico)|*.bmp;*.ico"
  ```

- ***FilterIndex***: Indica el índice (con respecto a la lista de tipos) del filtro por defecto. Se empiezan a numerar por "1".
- ***InitDir***: Contiene el nombre del directorio por defecto. Si no se especifica, se utiliza el directorio actual.
- ***Flags***: Esta propiedad puede tomar muchos valores con objeto de fijar los detalles concretos de este control (por ejemplo, abrir un fichero en modo ***read only***, avisar antes de escribir sobre un fichero ya existente, etc.). Estos valores están definidos por constantes de ***Visual Basic 6.0*** cuyos nombres empiezan con las letras ***cdl***. Para más información en el ***Help*** de ***Common Dialog Control*** buscar ***Properties, Flags Properties (Open, Save As Dialogs)***. Por ejemplo, el valor definido por la constante ***cdlOFNOverwritePrompt*** hace que antes de escribir en un fichero ya existente se pida confirmación al usuario. Para establecer varias opciones a la vez se le asigna a ***Flags*** la suma de las constantes correspondientes. Las distintas constantes disponibles se pueden encontrar en el ***Help*** buscando ***Constants/CommonDialog Control***.

4.4.2 Print Dialog Control

Las propiedades más importantes de este control son:

- ***Copies***: Determina el número de copias a realizar por la impresora.
- ***FromPage***: Selecciona el número de página a partir del cual comienza el rango de impresión.
- ***ToPage***: Selecciona el número de página hasta la cual llega el rango de impresión.
- ***PrinterDefault***: Cuando es ***True*** se imprime en el objeto ***Visual Basic Printer***. Además las opciones actuales de impresión que se cambien serán asignadas como las opciones de impresión por defecto del sistema.
- ***Flags***: Ver con ayuda del ***Help*** los posibles valores de esta propiedad.

4.4.3 Font Dialog Control

Las propiedades más importantes de este control son:

- ***Color***: Color de impresión. Para usar esta propiedad hace falta establecer la propiedad ***Flags*** al valor de la constante ***cdlCFEffects***.
- ***FontBold, FontItalic, FontStrikethru, FontUnderline***: Devuelve o asigna los valores de los estilos de la fuente actual.
- ***FontName:*** Devuelve o asigna el nombre de la fuente en uso.
- ***FontSize:*** Devuelve o asigna el tamaño de la fuente en uso.
- ***Min y Max:*** Asigna o lee los valores del tamaño de fuente mínimo y máximo respectivamente que aparecerán en la lista de selección de tamaños de la fuente.
- ***Flags***: Indica si los tipos de letra que se van a mostrar son los de la pantalla (***cdlCFScreenFonts***), los de la impresora (***cdlCFPrinterFonts***) o ambos (***cdlCFBoth***). Con la constante ***cdlCFEffects*** se puede indicar que se permite cambiar efectos como el color, subrayado y cruzado con una línea. Si ***Flags*** vale 0 da un error en tiempo de ejecución inndicando que no hay ***fonts*** instaladas.

4.4.4 Color Dialog Control

Las propiedades más importantes de este control son:

- ***Color***: Devuelve o asigna el valor del color actual.
- ***Flags***: Ver con ayuda del ***Help*** los posibles valores de esta propiedad. Por ejemplo, con el valor ***cdlCCFullOpen*** muestra el cuadro de diálogo completo, mientras que el valor ***cdlCCPreventFullOpen*** muestra sólo los colores predefinidos, impidiendo definir otros nuevos. Con el valor ***cdlCCRGBInit*** se establece el color inicial para el cuadro de diálogo.

4.5 FORMULARIOS MÚLTIPLES

Un programa puede contener más de un formulario. De hecho, habitualmente los programas contienen múltiples formularios. Recuérdese que el formulario es la ventana de máximo nivel en la que aparecen los distintos controles.

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

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

Para activar en tiempo de ejecución un formulario distinto del inicial (o del que esté activo en ese momento), se utiliza el método ***Show*** (***frmName.Show***). El método ***Hide*** oculta el formulario, pero lo deja cargado; el método ***Activate*** lo vuelve a mostrar. El método ***Unload*** elimina los elementos gráficos del formulario, pero no las variables y el código. El método ***Unload Me*** descarga

el propio formulario que lo llama. Para eliminar completamente un formulario se puede utilizar el comando:

```
Set frmName = NOTHING
```

que llama al evento ***Terminate*** (hay que utilizar también los métodos ***Hide*** o ***Unload*** para que desaparezca de la pantalla).

Para referirse desde un formulario a los objetos y variables de otro formulario se utiliza el ***operador punto*** (*frmName.Object.Property*).

4.5.1 Apertura de controles en forma modal

En ciertas ocasiones se desea que el programa no realice ninguna acción hasta que el usuario cierre una ventana o formulario en la que se le pregunta algo o en la que tiene que tomar alguna decisión. En esos casos, al utilizar el método ***Show***, es necesario utilizar el argumento ***Style*** con valor 1. A esto se le llama mostrar una ventana en forma ***modal***. Esto quiere decir que no se permitirá al usuario hacer activa ninguna pantalla hasta que el usuario cierre esa ventana modal. Esto se hace así:

```
frmName.Show 1
```

o bien,

```
frmName.Show vbModal
```

4.5.2 Formularios MDI (Multiple Document Interface)

En algunos casos puede ser interesante establecer una jerarquía entre las ventanas o formularios que van apareciendo sucesivamente en la pantalla del ordenador, de tal manera que al cerrar una que se haya establecido como principal, se cierren también todas las que se han abierto desde ella y dentro de ella. De esta forma una misma aplicación puede tener varios documentos abiertos, uno en cada ventana hija. Así trabajan por ejemplo ***Word*** y ***Excel***, que pueden tener varios documentos abiertos dentro de la ventana principal de la aplicación. En el mundo de las ***Windows*** de ***Microsoft*** a esto se llama ***MDI*** (***Multiple Document Interface***). La Figura 4.5 muestra un ejemplo de formulario MDI.

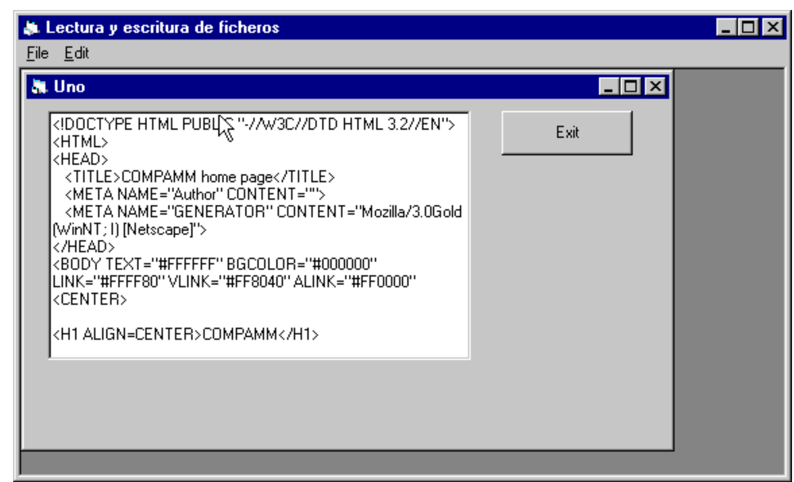


Figura 4.5. Formularios MDI (Multiple Document Interface).

En ***Visual Basic 6.0*** estos formularios que tienen sub-formularios hijos se conocen como ***MDIForms***. Los formularios ***MDI*** se crean desde el menú de ***Visual Basic 6.0*** con el comando ***Project/Add MDI Form***. En una aplicación sólo puede haber un formulario ***MDI***, pero éste puede tener varios hijos. Si se quiere que un formulario sea ***Child***, debe tener su propiedad ***MDIChild*** como ***True***.

Si al iniciar una aplicación el formulario que se carga en primer lugar es un formulario ***Child***, el formulario ***MDI*** se carga al mismo tiempo. Al cerrar un formulario ***MDIForm*** se cierran todos sus formularios ***Child***; por ejemplo, al cerrar ***Word*** también se cierran todos los documentos que

estuvieran abiertos. Los formularios ***Child*** se minimizan y maximizan dentro de los límites del formulario ***MDI***. Cuando están maximizados, su ***Caption*** aparece junto al ***Caption*** del formulario ***MDI***. Los formularios ***Child*** no tienen menús propios, sino que sus menús aparecen en la barra de menús del formulario ***MDI***.

En una aplicación con un formulario ***MDI*** y uno o más formularios ***Child***, puede haber otros formularios que no sean ***Child*** y que se abren fuera de los límites del formulario ***MDI*** cuando son requeridos.

4.6 ARRAYS DE CONTROLES

Un array de controles esta formado por controles del mismo tipo que comparten el nombre y los procedimientos o funciones para gestionar los eventos. Para identificar a cada uno de los controles pertenecientes al array se utiliza ***Index*** o *índice*, que es una propiedad más de cada control. Suponiendo que el sistema tenga memoria suficiente un array en ***Windows 95/98/NT*** podría llegar a tener hasta 32767 elementos.

La utilidad principal de los arrays se presenta en aquellos casos en los que el programa debe responder de forma semejante a un mismo evento sobre varios controles del mismo tipo. Los ejemplos más claros son los botones de opción y los menús. En estos casos el programa responde de manera semejante independientemente de cuál es la opción seleccionada. Los arrays de controles comparten código, lo cual quiere decir que sólo hay que programar una función para responder a un evento de un determinado tipo sobre cualquier control del array. Las funciones que gestionan los eventos de un array tienen siempre u argumento adicional del tipo ***Index As Integer*** que indica qué control del array ha recibido el evento.

Una opción avanzada de ***Visual Basic 6.0*** permite crear objetos en tiempo de ejecución, siempre que sean elementos de un array ya existente, con la instrucción ***Load***. De forma análoga se pueden destruir con ***Unload***.

5. MENÚS

Entre las capacidades de ***Visual Basic 6.0*** no podía faltar la de construir menús con gran facilidad. Sin embargo, hay algunas diferencias respecto al modo el que se construyen los controles. Para crear menús ***Visual Basic*** dispone de una herramienta especial que se activa mediante el comando ***Menu Editor*** del menú ***Tools***. El cuadro de diálogo que se abre se nuestra en la Figura 5.1. Más adelante se verá cómo se utiliza esta herramienta; antes, conviene recordar brevemente las características más importantes de los menús de ***Windows 95/98/NT***.

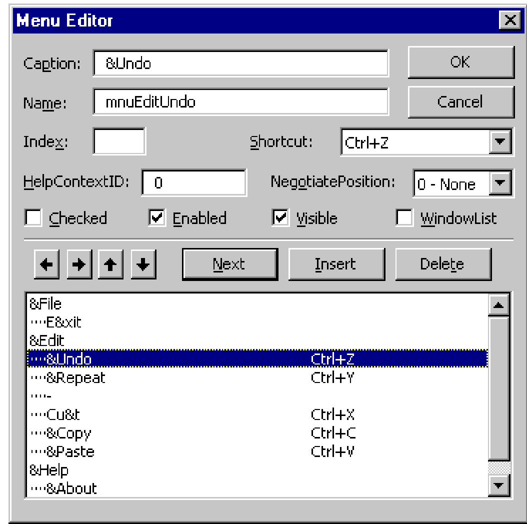


Figura 5.1. Editor de menús de ***Visual Basic***.

Los menús presentan sobre los demás controles la ventaja de que ocupan menos espacio en pantalla, pero tienen el inconveniente de que sus posibilidades no están a la vista más que cuando se despliegan.

5.1 INTRODUCCIÓN A LAS POSIBILIDADES DE LOS MENÚS

La mayor parte de las aplicaciones de ***Windows 95*** utilizan menús. Aunque todo el mundo está familiarizado con sus funciones más básicas, conviene ver algunas posibilidades menos usuales. Se utilizarán para ello unas aplicaciones tan conocidas como ***Word*** y ***Excel***.

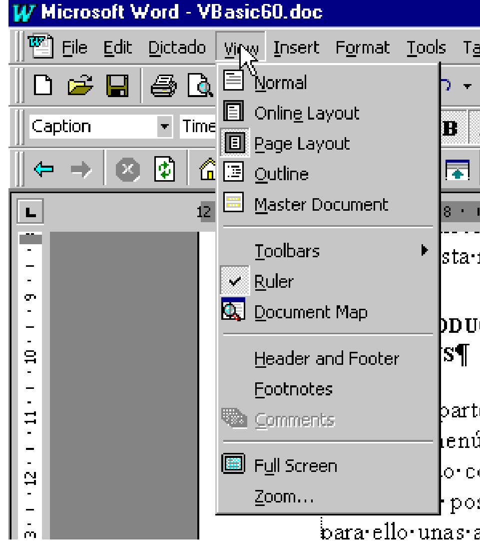


Figura 5.2. El menú ***View*** de ***Word 97***.

La Figura 5.2 recoge el aspecto del menú ***View*** de ***Word 97***, en el que conviene destacar las siguientes características:

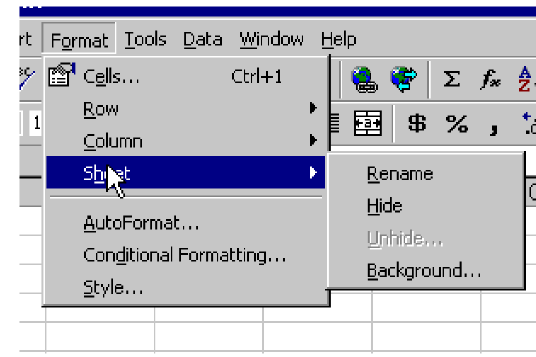


Figura 5.3. El menú ***Format/Sheet*** de ***Excel 97***.

1. Lo primero que llama la atención es que los menús aparecen divididos en grupos de opciones separados por líneas horizontales.
2. Algunos items como ***Page Layout*** tienen un icono resaltado a su izquierda. Esto quiere decir que ese ítem es la opción elegida entre los cuatro items de su grupo. En este sentido los menús se parecen a los controles ***OptionButton***. ***Visual Basic 6.0*** no permite hacer esto directamente, pero lo puede simular.
3. Otros items como ***Ruler*** tienen una marca de selección a su izquierda. En este caso el menú realiza la función de las cajas de selección (***CheckBox***).
4. Todas las opciones del menú tienen una letra subrayada. La finalidad es poder desplegar y activar los menús desde teclado, sin ayuda del ratón (con ***Alt*** y la letra subrayada).

5. También se observa que el ítem ***Comments*** aparece en gris claro. Esto quiere decir que en este momento no está activo y por tanto no es seleccionable.
6. Otros items como ***Toolbars*** están seguidos por un pequeño triángulo. Eso quiere decir que existe un menú secundario con más opciones. Otros items como ***Zoom*** aparecen seguidos por puntos suspensivos (...). Este es un convenio utilizado para indicar que eligiendo esa opción se abrirá un cuadro de diálogo en el que habrá que tomar otras decisiones.

Por lo que respecta al menú de ***Excel 97*** que aparece en la Figura 5.3 la característica más importante es que tiene ***sub-menús*** (señalados mediante un pequeño triángulo a su derecha), que se abren al colocar el cursor sobre el ítem correspondiente. Estos menús se suelen llamar ***menús en cascada***, y son muy frecuentes en ***Windows 95/98/NT***.

Otra característica de los menús, que no aparece en la Figura 5.2 ni en la Figura 5.3, es la posibilidad de definir combinaciones de teclas que realizan la misma función que una opción del menú. Por ejemplo, en muchas aplicaciones ***Ctrl+C*** equivale a ***Edit/Copy*** y ***Ctrl+V*** a ***Edit/Paste***. Estas combinaciones de teclas se llaman *accesos rápidos* (***shortcut***) y hay que distinguirlas de acceder a los menús mediante la tecla ***Alt*** y las letras subrayadas de los nombres.

5.2 EL EDITOR DE MENÚS (MENU EDITOR)

En la Figura 5.4 se vuelve a recoger -a mayor tamaño y con algunos elementos ya definidos- el editor de menús mostrado en la Figura 5.1, que se abre con ***Tools/Menu Editor*** o clicando en el botón correspondiente de la barra de herramientas.

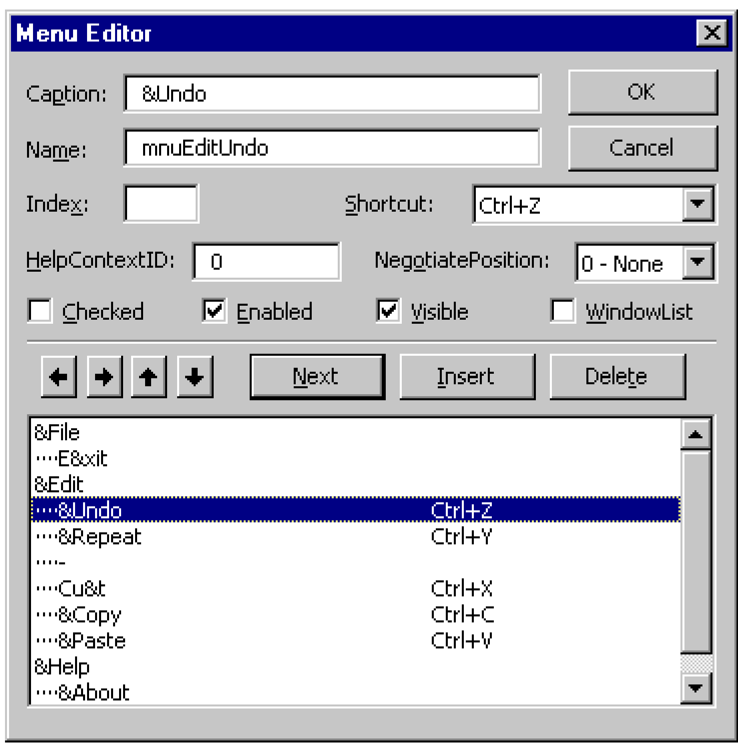


Figura 5.4. Definición de menús con ***Menu Editor***.

Se llama ***título*** a cada elemento que aparece en la barra de menús y ***línea*** o ***ítem*** a cada elemento que aparece al desplegarse un título. Para introducir un nuevo título en la barra de menús hay que definir, en la caja de texto ***Caption*** de la Figura 5.4, el nombre con el que se quiere que aparezca. Si se desea acceder a dicho título mediante teclado (***Alt+letra***), la letra que se desea utilizar deberá ir precedida por el carácter (&). Además, y al igual que todos los controles de ***Visual Basic 6.0***, conviene que el título tenga un *nombre* (caja de texto ***Name***) para que se pueda acceder a él desde programa. Los nombres de los títulos de los menús suelen comenzar por las letras ***mnu***, como por ejemplo ***mnuFile***, ***mnuEdit*** o ***mnuHelp***.

En la Figura 5.4 la caja de texto ***Index*** hace referencia a la posibilidad de crear *arrays de menús*. Se puede definir también un ***shortcut*** en la caja de texto correspondiente. En esta figura aparecen cuatro ***checkButtons*** (***Enabled***, ***Checked***, ***Visible*** y ***WindowList***) con los que se pueden especificar algunas propiedades iniciales del menú, como por ejemplo que esté activado o que sea visible.

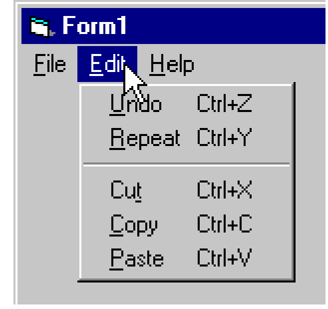


Figura 5.5. Menú ***Edit*** sencillo.

Se pueden introducir ***items*** subordinados a un ***título*** por medio de la *flecha hacia la derecha*. Para ello basta definirlos del modo habitual y luego clicar sobre dicha flecha. El resultado es que aparecen unos puntos a la izquierda del caption correspondiente. Por ejemplo, en el menú definido en la Figura 5.4, ***Exit*** es una línea subordinada del menú ***File***, mientras que ***Undo***, ***Repeat***, ***Cut***, ***Copy*** y ***Paste*** son items subordinados del menú ***Edit***. En este último caso se ha introducido una línea de separación entre ***Repeat*** y ***Cut***; para ello basta introducir un *ítem* más cuyo ***caption*** sea el carácter *menos* (-).

La Figura 5.5 muestra el resultado de ejecutar la aplicación cuyos menús están definidos en el ***Menu Editor*** de la Figura 5.4. Obsérvese la línea horizontal de separación, los shortcuts y las letras subrayadas para poder abrir el menú desde teclado.

Respecto a los nombres de los items, lo habitual es seguir la nomenclatura que ya se muestra en la Figura 5.4 para ***Undo***: primero se ponen las tres letras ***mnu***, y después los nombres del título y del ítem comenzando por mayúscula: ***mnuEditUndo***. Caso que haya menús en cascada, se ponen los distintos nombres siguiendo estas mismas normas. De esta forma siempre queda claro a partir del nombre a qué elemento del menú se está haciendo referencia.

La Figura 5.4 es bastante auto-explicativa respecto a cómo se debe proceder para estructurar un menú, añadiendo, borrando o cambiando de posición los distintos elementos. De forma resumida, se pueden establecer las siguientes normas generales:

1. Para insertar un título o ítem basta seleccionar la línea por encima de la cual se quiere insertar y clicar en el botón ***Insert***. Para añadir un nuevo ítem al final de la lista se selecciona el último elemento introducido y se clica en el botón ***Next***. Para borrar un elemento, se selecciona y se clica en el botón ***Delete***.
2. Se puede cambiar de posición un título o ítem seleccionándolo y clicando en los botones que muestran las flechas hacia arriba y hacia abajo. Para convertir un título en ítem o para cambiar el nivel de un ítem se selecciona y se utilizan las flechas hacia la derecha y hacia la izquierda. Conviene recordar que los nombres de los items (por ejemplo ***mnuEditCopy***) deben estar siempre de acuerdo con su posición, según las normas explicadas anteriormente.

5.3 AÑADIR CÓDIGO A LOS MENÚS

Los items de los menús *admiten un único evento*: el evento ***click***, que consiste en ser seleccionados por medio del ratón o del teclado. Para añadir el código correspondiente basta elegir en el menú, estando en modo diseño, el ítem correspondiente para que se abra la ventana de código en el procedimiento ligado a ese evento. También puede buscarse directamente el objeto y el evento correspondiente en las listas desplegables de la ventana de código.

En ocasiones habrá que cambiar las propiedades ***checked, active*** y ***visible*** desde los procedimientos. A estas propiedades se accede del modo habitual, con el nombre del ítem y el operador punto (.)

5.4 ARRAYS DE MENÚS

De la misma manera que pueden definirse arrays de controles, también pueden definirse arrays de items (y de títulos) en un menú. La ventaja de definir arrays de items es que *basta definir un único procedimiento* que se haga cargo del evento ***click*** de todos los items del array. Este procedimiento

recibe como parámetro la variable entera ***Index***, que indica que ítem del array ha sido seleccionado por el usuario. Dentro de este procedimiento se podrá utilizar por ejemplo la sentencia ***Select Case*** para tratar de forma adecuada cada uno de los casos.

5.5 EJEMPLO: MENÚ PARA DETERMINAR LAS CARACTERÍSTICAS DE UN TEXTO

La Figura 5.6 muestra un formulario que contiene una caja de texto con una frase (“*Visual Basic es el lenguaje de programación que hace más fácil el desarrollar aplicaciones para Windows 95*”) a la que se puede dar formato desde el menú ***Text***. El menú ***Text*** tiene tres submenús: ***Font***, ***Size*** y ***Style***. El menú ***File*** sólo tiene la opción ***Exit***, que termina la ejecución.

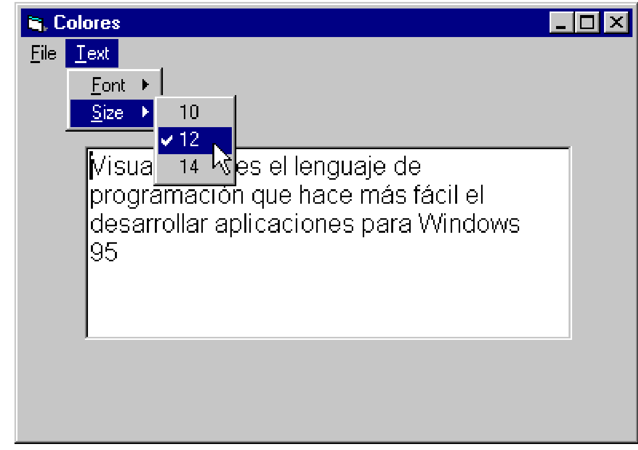


Figura 5.6. Caja de texto con formatos desde menú.

El sub-menú ***Font*** tiene tres opciones: ***Arial***, ***Courier New*** y ***Times New Roman***. El sub-menú ***Size*** tiene 5 opciones: 10, 11, 12, 13, y 14. El sub-menú ***Style*** tiene 2 opciones: ***Bold*** e ***Italic***. Los tipos de letra y los tamaños deben actuar como los *Option Buttons*: sólo una opción puede estar seleccionada para el texto de la caja. Sin embargo, los estilos ***Bold*** e ***Italic*** actúan como *Checked Boxes*: el texto puede ser a la vez ***Bold*** e ***Italic***, y puede no ser ninguna de las dos cosas.

Para los tamaños de letra se debe utilizar un array de menús con cinco elementos (propiedad ***Index*** de 0 a 4). Se deja al usuario que ponga los nombres que desee a los controles de la Figura 5.6, o que utilice los del código del programa que se muestra a continuación. Nótese que con los menús que se comportan como *Option Buttons* la propiedad ***Checked*** se pone a ***False*** en todas las opciones antes de poner a ***True*** la que el usuario ha elegido. Con el menú que se comporta como *Checked Box* simplemente se cambia la propiedad ***Checked*** de ***True*** a ***False*** o viceversa, cuando el usuario la elige. El código se muestra a continuación:

```
Option Explicit

Private Sub Form_Load()
   txtBox.Text = "Visual Basic es el lenguaje de programación " & _
      "que hace más fácil el desarrollar aplicaciones para Windows 95"
   txtBox.Font.Name = "Arial"
   mnuTextFontArial.Checked = True
   txtBox.Font.Size = 10
   mnuTextSizeA(0).Checked = True
   txtBox.Font.Bold = False
   txtBox.Font.Italic = False
End Sub

Private Sub mnuFileExit_Click()
   End
End Sub

Private Sub mnuTextFontArial_Click()
   mnuTextFontCou.Checked = False
   mnuTextFontTimes.Checked = False
   txtBox.Font.Name = "Arial"
   mnuTextFontArial.Checked = True
End Sub
```

```
Private Sub mnuTextFontCou_Click()
   mnuTextFontArial.Checked = False
   mnuTextFontTimes.Checked = False
   txtBox.Font.Name = "Courier New"
   mnuTextFontCou.Checked = True
End Sub

Private Sub mnuTextFontTimes_Click()
   mnuTextFontArial.Checked = False
   mnuTextFontCou.Checked = False
   txtBox.Font.Name = "Times New Roman"
   mnuTextFontTimes.Checked = True
End Sub

Private Sub mnuTextSizeA_Click(Index As Integer)
    Dim i As Integer
    For i = 0 To 4
        mnuTextSizeA(i).Checked = False
    Next i
    Select Case Index
        Case 0
            txtBox.Font.Size = 10
        Case 1
            txtBox.Font.Size = 11
        Case 2
            txtBox.Font.Size = 12
        Case 3
            txtBox.Font.Size = 13
        Case 4
            txtBox.Font.Size = 14
    End Select
    mnuTextSizeA(Index).Checked = True
End Sub

Private Sub mnuTextStyleBold_Click()
   txtBox.Font.Bold = Not txtBox.Font.Bold
   mnuTextStyleBold.Checked = Not mnuTextStyleBold.Checked
End Sub

Private Sub mnuTextStyleItalic_Click()
   txtBox.Font.Italic = Not txtBox.Font.Italic
   mnuTextStyleItalic.Checked = Not mnuTextStyleItalic.Checked
End Sub
```

5.6 MENÚS CONTEXTUALES (POPUP MENU)

Los ***menús contextuales*** aparecen cuando el usuario clica con el botón derecho sobre un elemento de la aplicación. El programa debe reconocer el evento ***MouseUp*** o ***MouseDown***, ver si el usuario ha clicado con el botón derecho (argumento ***Button*** igual a 2) y llamar al método PopupMenu, que tiene la siguiente forma general:

```
PopupMenu menuName [,flags[,x[,y]]]
```

donde ***menuName*** ee el nombre de un menú (con al menos un elemento), ***x*** e ***y*** son las coordenadas base para hacer aparecer el menú contextual, y ***flags*** son unas constantes que determinan más en concreto dónde y cómo se muestra el menú. Las constantes que determinan dónde aparece el menú son: *vbPopupMenuLeftAlign* (default), *vbPopupMenuCenterAlign* y *vbPopupMenuRightAlign*. Por otra parte *vbPopupMenuLeftButton* (default) y *vbPopupMenuRightButton* determinan si el comando se activa con el botón izquierdo o con cualquiera de los dos botones. Las constantes se combina con el operador ***Or***. El nombre del menú que aparece en el método ***PopupMenu*** debe haber sido creado con el ***Menu Editor***, aunque puede tener la propiedad ***Visible*** a ***False***, si no se desea que se vea.

6. GRÁFICOS EN VISUAL BASIC 6.0

Visual Basic 6.0, además de hacer fácil la construcción de interfaces gráficas de usuario, tiene también grandes posibilidades gráficas en lo que se refiere a dibujo de líneas y formas geométricas, así como en lo referente a la introducción de gráficos y figuras realizados con otras aplicaciones. En este capítulo se presentarán brevemente las posibilidades gráficas más importantes de ***Visual Basic 6.0***.

6.1 TRATAMIENTO DEL COLOR

Antes de ver cómo se dibuja en ***Visual Basic 6.0*** se verá cómo se definen los colores. Al igual que en tantas aplicaciones informáticas, los colores de ***Visual Basic*** se definen por medio de las componentes fundamentales RGB (***Red***, ***Green*** and ***Blue***). La intensidad de cada color fundamental se define con un ***byte***, es decir con un número entero entre 0 y 255. Se utilizan pues tres bytes para definir los tres colores. ***Visual Basic 6.0*** utiliza un entero *long* (32 bits, 4 bytes) para guardar un color, lo cual quiere decir que existe un byte adicional donde se podrá guardar alguna otra información (ver Apartado 6.1.2).

6.1.1 Representación hexadecimal de los colores

Para los números enteros entre 0 y 255 se utilizan dos dígitos hexadecimales. Con esta notación el cero es el “00” y el 255 el “FF”. El número que indica el color va precedido por el carácter “&” y la letra “H”. Así, el color verde se define en la forma: &H00FF00. Con esta notación es posible prescindir de los ceros situados a la izquierda. Por ejemplo, el color rojo se puede escribir como &H0000FF y como &HFF.

Visual Basic 6.0 dispone también de nombres para los colores fundamentales y los que son combinación de los colores fundamentales, según puede verse en la Tabla 6.1.

Nombre	Código HEX	Color
vbBlack	&H000000	Negro
vbRed	&H0000FF	Rojo.
vbGreen	&H00FF00	Verde.
vbYellow	&H00FFFF	Amarillo.
vbBlue	&HFF0000	Azul.
vbMagenta	&HFF00FF	Magenta.
vbCyan	&HFFFF00	Cyan.
vbWhite	&HFFFFFF	Blanco.

Tabla 6.1. Nombres de colores.

6.1.2 Acceso a los colores del sistema

El cuarto byte (en el entero *long* que contiene el color) puede utilizarse para hacer referencia a los ***colores del sistema***. Los colores del sistema son aquellos colores con los que ***Windows 95/98/NT*** representa las ventanas y sus bordes, las barras de desplazamiento, etc. Dichos colores se eligen en el panel de control ***Display/Appearance***, y ***Visual Basic 6.0*** permite acceder a ellos a través de su nombre o de su valor hexadecimal, que empieza por “&H8” y utiliza el cuarto byte. La Tabla 6.2 muestra algunos de estos valores. Para una descripción completa buscar ***Color Constants*** en el ***Help*** de ***Visual Basic 6.0***.

Nombre	Valor	Descripción
vbScrollBars	&H80000000	Scroll bar color.
vbDesktop	&H80000001	Desktop color.
vbActiveTitleBar	&H80000002	Color of the title bar for the active window.
vbInactiveTitleBar	&H80000003	Color of the title bar for the inactive window.
vbMenuBar	&H80000004	Menu background color.
vbWindowBackground	&H80000005	Window background color.
vbWindowFrame	&H80000006	Window frame color.
vbMenuText	&H80000007	Color of text on menus.
vbWindowText	&H80000008	Color of text in windows.
vbTitleBarText	&H80000009	Color of text in caption, size box, and scroll arrow.
...	...	...

Tabla 6.2. Colores del sistema.

6.1.3 Función RGB

Esta función devuelve un número que representa un color a partir de tres argumentos enteros entre 0 y 255, que son sus componentes RGB. Como ejemplo de uso:

```
form1.BackColor = RGB(127, 127, 64)
```

Si alguno de los argumentos tiene un valor mayor que 255, se toma como 255.

6.1.4 Paleta de colores

Elegir adecuadamente un color a partir de sus componentes RGB no es una tarea fácil. Por eso ***Visual Basic 6.0*** proporciona una paleta de 64 colores predefinidos, 16 de los cuales pueden ser definidos a medida por el usuario. La Figura 6.1 muestra la paleta de colores, que aparece con ***View/Color Palette***.

Figura 6.1. Paleta de colores.

La Figura 6.2 muestra el cuadro de diálogo que se abre al pulsar el botón ***Define Colors...*** en la parte inferior dcha. de la Figura 6.1. Para elegir un color se pueden introducir directamente los valores RGB, pero también se puede clicar en el mapa de colores de la parte superior izda. y luego mover el cursor de la parte superior dcha. Finalmente, clicando en el botón ***Add Color***. El color seleccionado se añade en la parte inferior de la paleta de colores (Figura 6.1).

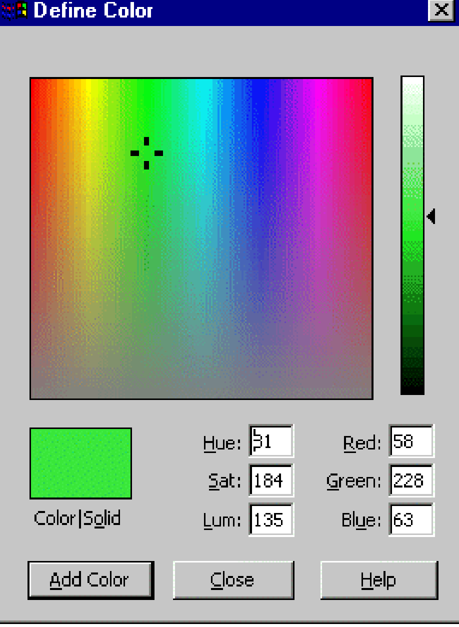


Figura 6.2. Creación de colores a medida.

Por supuesto es necesario tener en cuenta el número de colores soportado por la tarjeta gráfica del PC en el que se esté trabajando. Lo más frecuente es que los PCs estén configurados para soportar 256 colores (8 bits por pixel), 65.536 colores (16 bits por pixel) o 16.777.216 colores (24 bits por pixel). Si la tarjeta gráfica soporta 65.536 colores se elige el color más cercano al que el usuario ha querido

representar, con la función RGB por ejemplo. Si la tarjeta gráfica soporta sólo 256 colores se utiliza el *dithering*, que consiste en mezclar pixels de distintos colores con objeto de obtener un efecto lo más parecido posible al color solicitado.

Una vez añadidos los colores a la paleta, al clicar en el pequeño triángulo que aparece en cualquier propiedad de color en la ventana ***Properties*** aparecerán una ventana donde es posible elegir entre los *colores de la paleta* y los denominados *colores del sistema* (Figura 6.3).

El Ejemplo 1.5.4 (Colores RGB), mostrado en la página 11 de este manual, es un buen ejemplo de la utilización de los colores en ***Visual Basic 6.0***.

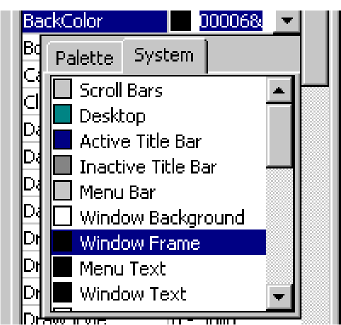


Figura 6.3. Colores de sistema

6.2 FORMATOS GRÁFICOS

En un formulario de ***Visual Basic 6.0*** -y en los controles ***Image*** y ***PictureBox***- es posible insertar gráficos, tanto de tipo *bitmap* (los producidos por aplicaciones como ***Paint***, ***Paintbrush***, ***Paint Shop Pro***, etc.), como de tipo *vectorial* (los producidos por las herramientas gráficas de ***Word*** y ***PowerPoint***).

Visual Basic 6.0 admite varios formatos de ficheros gráficos: los ficheros ****.bmp*** y ****.ico*** para los gráficos de tipo *bitmap,* los ficheros ****.wmf*** (*Windows Meta File*) y ****.emf*** (***Enhanced Meta File***) para los gráficos de tipo vectorial y ****.jpg*** (*JPEG o Joint Photographic Experts Group*) y ****.gif*** (*Graphic Interchange Format*). Los ficheros ****.ico*** son ficheros *bitmap* de pequeño tamaño (32 por 32) destinados a contener iconos. Los ficheros JPEG y GIF son formatos gráficos comprimidos que soportan respectivamente color de 24 bit (~16 millones de colores) y 8 bit (256 colores). Ambos formatos son los utilizados en Internet para mostrar imágenes.

Si se desea insertar ficheros gráficos que estén en otros formatos, habrá que convertirlos previamente a uno de estos formatos usando el programa adecuado.

6.3 CONTROLES GRÁFICOS

Visual Basic 6.0 dispone de varios controles con los que insertar gráficos en un formulario. Algunos tienen más posibilidades que otros y es necesario conocerlos bien. A continuación se verán los controles ***Line***, ***Shape***, ***Image*** y ***PictureBox***.

6.3.1 Control Line

Es el control gráfico más elemental, ya que carece de propiedades como *Text*, *Caption* y *Value*. Además no reconoce ningún evento, por lo que su misión es casi exclusivamente decorativa.

El control ***Line*** permite dibujar líneas en un ***formulario*** o en un control ***PictureBox***. Las propiedades más importantes son las coordenadas de los puntos extremos (***X1***, ***Y1***, ***X2*** e ***Y2***), la anchura en pixels (***BorderWidth***), el estilo de la línea (***BorderStyle***) -continua, a trazos, etc.- que sólo está activo cuando la anchura es 1 pixel, el color (***BorderColor***) y el nombre (***Name***). La línea puede estar visible o no (***Visible***), y existe la propiedad ***Index***, que permite crear *arrays* de líneas.

6.3.2 Control Shape

Este control es en muchos aspectos similar al control ***Line***: tampoco tiene las propiedades *text*, *Caption* y *Value*, ni reconoce eventos. Se diferencia en que admite formas geométricas más complejas, que vienen definidas por la propiedad ***Shape***, que admite los valores siguientes: cuadrado (***Square***), rectángulo (***Rectangle***), círculo (***Circle***), elipse (***Oval***), cuadrado redondeado (***Rounded Square***) y rectángulo redondeado (***Rounded Rectangle***).

Además de las propiedades correspondientes al tamaño y posición, las propiedades más interesantes del control ***Shape*** son las siguientes: *BackColor*, *BackStyle*, *BorderColor*, *BorderStyle*, *BorderWidth*, *FillColor*, *FillStyle*, *DrawMode*. Un control ***Shape*** puede estar visible o no (***Visible***), y existe la propiedad ***Index***, que permite crear *arrays* de ***Shapes***.

6.3.3 Ejemplo 6.1: Uso de los controles Line y Shape

La Figura 6.4 muestra un formulario en el que se han dibujado tres controles ***Line*** y dos controles ***Shape***. Las tres líneas se han dibujado con la propiedad ***BorderWidth=1***, pues si no la propiedad ***BorderStyle*** no surte efecto. La propiedad ***BorderStyle*** es *2-Dash* para la segunda línea y *3-Dot* para la tercera.

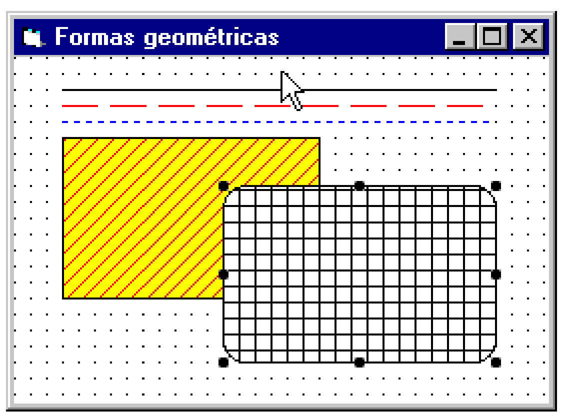


Figura 6.4. Controles ***Line*** y ***Shape***.

Después se han dibujado dos controles ***Shape*** llamados ***shpRect*** y ***shpRRec***, cuyas propiedades *Shape* están respectivamente a *0-Rectangle* y a *4-Rounded Rectangle*. La propiedad ***BackColor*** está en amarillo para ***shpRect*** y en blanco para ***shpRRect***. En ambos casos ***BackStyle*** está en *1-Opaque*, pues si no el color de fondo no surte efecto. La propiedad ***FillColor*** (que determina el color de las líneas de rayado) está en rojo para ***shpRect*** y en negro para ***shpRRect***. Finalmente, la propiedad ***FillStyle*** que determina el tipo de rayado está en *5-Downward Diagonal* para ***shpRect*** y en *6-Cross* para ***shpRRect***. Como la propiedad ***DrawMode*** está en *13-Copy Pen* para ambos controles, ***shpRRect*** se superpone sobre ***shpRect*** porque ha sido creada sobre él con posterioridad.

6.3.4 Control Image

El control ***Image*** es un contenedor de gráficos ***bitmap***, ***iconos***, ***metafile***, ***enhanced metafile***, ***GIF*** y ***JPEG***. Este control admite ya una amplia colección de eventos, por lo que es ya un control con un papel mucho más activo que los anteriores.

Las propiedades más propias e importantes de este control son las propiedades ***Picture*** y ***Stretch***. La propiedad ***Picture*** sirve para relacionar este control con el fichero que contiene el gráfico que se desea representar, a través del cuadro de diálogo ***Load Picture*** que permite elegir el fichero a insertar. El fichero deberá ser de uno de los tipos admitidos. Según el fichero elegido, la propiedad ***Picture*** tendrá uno de los tres valores siguientes: ***icon*** (ficheros *cur*, *ico*), ***bitmap*** (*bmp*, *gif*, *jpg*) o ***metafile*** (*wmf*, *emf*).

La propiedad ***Stretch*** indica cómo se comporta el control ***Image*** al introducir en él el contenido del fichero gráfico. Por defecto, cuando se crea un control ***Image*** arrastrando en el formulario con el ratón esta propiedad tiene el valor ***False***. Estando la propiedad ***Stretch*** en ***False*** el tamaño del control se ajusta al tamaño del *bitmap* o del ***metafile*** que se introduce en dicho control.

Por el contrario, si dicha propiedad está en ***True*** el gráfico que proviene del fichero se adapta al tamaño de control.

Se puede tratar de modificar el tamaño del gráfico en modo de diseño (con el ratón o cambiando las propiedades de tamaño del control). Si el gráfico es un *bitmap* y la propiedad ***Stretch*** está en ***False***, el tamaño de la imagen no cambia aunque cambie el del control (quedando en la esquina superior izquierda si el control se hace más grande, o quedando parcialmente oculta si alguna de las dimensiones del control se hace más pequeña que la del *bitmap*. Si la propiedad ***Stretch*** está en ***True***, el *bitmap* se adapta al tamaño del control y su tamaño se cambia con el de éste. Los gráficos *metafile* siempre se pueden cambiar de tamaño en modo de diseño, tanto si ***Stretch*** está en ***True*** como si está en ***False***.

Existen otras formas de cargar un gráfico en un control ***Image***, además de utilizar la propiedad ***Picture*** en modo de diseño, como se ha visto anteriormente. Una segunda forma, utilizable también en modo de diseño, es hacer ***Copy*** y ***Paste*** a partir de un gráfico contenido en otra aplicación como ***Paint Shop Pro*** o ***Excel***.

En modo de ejecución se puede copiar el contenido de un control ***Image*** en otro control del mismo tipo por medio de una sentencia de asignación en la forma:

```
imgCuadro.picture = imgCaja.picture
```

y se puede también cargar una imagen de un fichero utilizando el procedimiento ***LoadPicture***, por ejemplo en la forma siguiente (habrá que estar seguro de que existe el fichero):

```
imgCuadro.picture = LoadPicture("G:\graficos\pc.wmf")
```

Aunque el control ***Image*** admite algunos eventos (*Click*, *DblClick*, *DragDrop*, *DragOver*, *MouseUp*, *MouseDown*, *MouseMove*), sus posibilidades son también limitadas. Por la forma en que se dibuja, el control ***Image*** no puede estar sobre otro control, como por ejemplo un botón (ver los *layers*, más adelante en este capítulo). Tampoco puede contener otros controles en su interior: sólo puede contener gráficos. Finalmente, este control no puede obtener el ***focus*** y por tanto no puede responder a acciones desde el teclado. El control ***PictureBox***, que se verá a continuación, resuelve estas limitaciones aunque presenta la desventaja de ser más lento en dibujar que el control ***Image***.

6.3.5 Control PictureBox

Este es el control gráfico ( ) más potente y general de ***Visual Basic 6.0***. Se trata de una especie de *formulario reducido*, pues puede contener imágenes y otros tipos de controles tales como botones, shapes, labels, cajas de texto, etc.

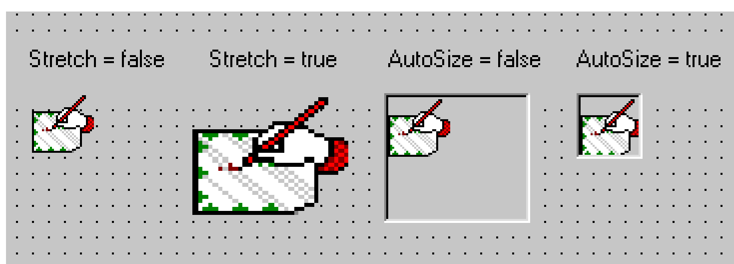


Figura 6.5. Comparación entre ***Image*** y ***PictureBox*** con *bitmaps*.

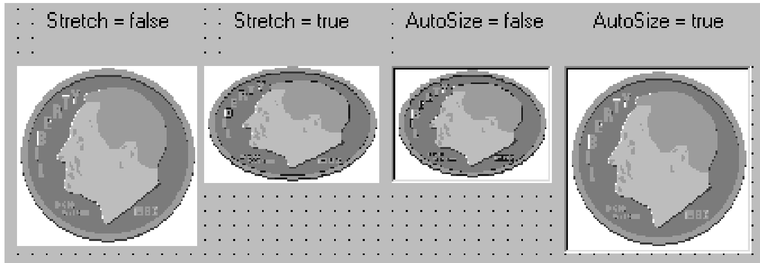


Figura 6.6. Comparación entre ***Image*** y ***PictureBox*** con *metafiles*.

Con respecto a los *bitmaps*, el control ***PictureBox*** se comporta de modo diferente que el control ***Image***. El control ***PictureBox*** no tiene propiedad ***Stretch***, con lo cual al cargar un icono o un bitmap siempre aparecen con su tamaño natural (tal y como se puede observar en la Figura 6.5). Sin embargo el control ***PictureBox*** tiene la propiedad ***AutoSize***, que por defecto está en ***False***. Cuando se carga un *bitmap* con ***AutoSize*** en ***False*** el gráfico aparece en la esquina superior izquierda del control; sin embargo, si ***AutoSize*** está en ***True*** el control ***PictureBox*** adapta su tamaño al del *bitmap* que es cargado. La Figura 6.5 muestra los resultados de introducir un icono en un control ***Image*** (***Stretch***: *False* y *True*) y en un control ***PictureBox*** (***AutoSize***: *False* y *True*).

Los gráficos *metafile* se comportan de un modo diferente, según puede verse en la Figura 6.6. En el control ***Image*** se cargan con su verdadero tamaño si la propiedad ***Stretch*** es ***False***, mientras que se adaptan al tamaño del control si dicha propiedad es ***True***. Con el control ***PictureBox*** se adaptan al tamaño del control si ***AutoSize*** es ***False***, mientras que se cargan con su propio tamaño si es ***AutoSize*** es ***True***.

En el control ***PictureBox*** (al igual que en los formularios) son importantes las cuatro propiedades relacionadas con el color: *BackColor*, *ForeColor*, *FillColor* y *FillStyle*. La propiedad ***BackColor*** controla el color de fondo del control. La propiedad ***ForeColor*** controla el color del texto que se escribe en el control (con el método ***Print***, por ejemplo, como luego se verá). Las propiedades ***FillColor*** y ***FillStyle*** no afectan directamente al control sino a los elementos gráficos que se dibujen sobre él con métodos tales como ***Line*** y ***Circle***, que se verán a continuación. ***FillStyle*** determina el tipo de relleno o *pattern* (líneas horizontales, verticales, inclinadas, cruzadas, ...), mientras que ***FillColor*** determina el color de estas líneas del relleno.

6.4 MÉTODOS GRÁFICOS

Sólo los ***formularios*** y los controles ***PictureBox*** pueden albergar otros tipos de controles. Además es posible escribir texto y dibujar directamente sobre ellos por medio de ciertos ***métodos***[3] de ***Visual Basic***. Por defecto estos métodos actúan sobre el ***formulario activo***. Si se desea que actúen sobre un control ***PictureBox*** hay que precederlos por el nombre del control y el operador punto.

---

[3] Los ***métodos*** son procedimientos que ***Visual Basic*** ofrece ya programados. El usuario sólo tiene que llamarlos pasándoles los argumentos apropiados.

6.4.1 Método Print

En tiempo de ejecución se puede escribir texto en un ***formulario*** o en un control ***PictureBox*** por medio del método ***Print***. La forma general de este método es la siguiente:

```
objeto.Print {spc(n)|tab(n)} expresion poschar
```

donde ***spc(n)*** es opcional y sirve para insertar ***n*** caracteres en la salida; ***tab(n)*** es también opcional y sirve para posicionar la salida en una posición absoluta determinada por ***n*** con un tabulador. Si ***tab*** se utiliza sin argumentos lleva al comienzo de la siguiente ***región de salida***[4]; ***expresion*** representa cualquier expresión cuyo resultado sea un número o una cadena de caracteres. ***poschar*** indica dónde se imprimirá el siguiente carácter. Si es un ***punto y coma*** (;) la impresión se hace a continuación del último carácter impreso; si es un ***tab(n)*** o un ***tab*** tiene el efecto antes descrito; si se omite, la impresión comienza en una nueva línea.

El color, la fuente y el tamaño del texto se toman de las correspondientes propiedades del formulario o control ***PictureBox***.

6.4.2 Dibujo de puntos: método PSet

El método ***PSet*** sirve para dibujar puntos en un formulario o en un control ***PictureBox***. Su forma general es la siguiente:

```
object.PSet Step (x, y), color
```

donde:

object es opcional y representa el objeto (***form*** o ***PictureBox***) en el que se va a dibujar el punto. Si se omite, el punto se dibuja en el formulario activo (el que tiene el *focus*).

Step es opcional. Si se introduce las coordenadas que le siguen son relativas respecto a las propiedades ***CurrentX*** y ***CurrentY*** de la ***PictureBox***. Al dibujar un punto, estas propiedades se actualizan a las coordenadas de dicho punto.

(x, y) son las coordenadas absolutas o relativas del punto a dibujar (expresiones, variables o constantes *single*). Tanto las coordenadas como la coma, como los paréntesis son obligatorios. Las unidades dependen de la propiedad ***ScaleMode*** del objeto en que se dibuja.

Color es opcional y es un nombre de color (***vbRed***, ***vbBlue***, etc.) o un ***long*** conteniendo el código de color hexadecimal (puede ser el valor de retorno de la función ***RGB***). Si se omite, se utiliza la propiedad ***ForeColor*** del objeto en el que se dibuja.

El tamaño del punto viene determinado por la propiedad ***DrawWidth*** del objeto en que se dibuja. Si el tamaño es mayor que uno, el punto se dibuja centrado en las coordenadas suministradas a ***PSet***. Si se desea eliminar un punto previamente dibujado es necesario volver a pintar ese punto con el color de fondo del objeto (***BackColor***).

6.4.3 Dibujo de líneas y rectángulos: método Line

El método ***Line*** dibuja líneas y -en ciertas condiciones- cajas rectangulares de lados horizontales y verticales. Su forma general es la siguiente:

```
object.Line Step (x1, y1) - Step (x2, y2), color, BF
```

---

[4] En ***Visual Basic*** se comienza una ***región de salida*** cada 14 caracteres, si se utiliza un tipo de letra de anchura constante. Con otros tipos de letra esta medida es sólo aproximada.

donde ***object***, ***step*** y ***color*** tienen el mismo significado que en ***PSet***, y

(x1, y1) son opcionales y son las coordenadas del punto inicial de la línea. Si se omiten la línea comienza en las coordenadas definidas por ***CurrentX*** y ***CurrentY***.

(x2, y2) son obligados y contienen las coordenadas del punto final de la línea.

B es un carácter opcional. Si se incluye se dibuja un rectángulo (*Box*) con los puntos dados como extremos de una de sus diagonales.

F es también un carácter opcional, que sólo se puede incluir si se ha incluido ***B***. Si se incluye, la caja rectangular se rellena (*Fill*) con el mismo color del contorno. Si se omite la caja se rellena con las propiedades ***FillColor*** y ***FillStyle*** del objeto en el que se dibuja.

Después de ejecutarse este método las propiedades ***CurrentX*** y ***CurrentY*** tienen el valor del punto final de la línea. Es necesario introducir el carácter (-), aunque se omita el primero de los puntos que definen la línea.

Valor	Nombre	Estilo de línea
0	vbSolid	continua (valor por defecto)
1	vbDash	trazos (continua si w>1)
2	vbDot	puntos (continua si w>1)
3	vbDashDot	raya-pto (continua si w>1)
4	vbDashDotDot	raya-pto-pto (continua si w>1)
5	vbInvisible	transparente (continua si w>1)
6	vbInsideSolid	continua interna

Tabla 6.3. Valores de ***DrawStyle***.

Las propiedades ***DrawWidth*** y ***DrawStyle*** determinan cómo se dibujan las líneas rectas o curvas en ***Visual Basic 6.0***. Más en concreto, la propiedad ***DrawWidth*** determina el grosor en pixels, mientras que ***DrawStyle*** determina el tipo de línea. La Tabla 6.3 considera los posibles valores de la propiedad ***DrawStyle***.

Los tipos de raya discontinua no permiten que el grosor sea mayor que 1 pixel. Si el grosor es superior, la línea se dibuja de modo continuo.

Ejemplo:

```
Line (0 ,0 )-(100 , 0) ' Línea del punto (0,0) al (100,0)
Line -(100 , 100)      ' Línea del punto (100,0) al (100,100)
Line -Step (20 , 80)   ' Línea del punto (100,100) al (120,180)
Line (100,100)-(200 , 200), vbRed, BF ' Rectángulo rojo del punto
                                      ' (100,100) al (200,200)
```

6.4.4 Dibujo de circunferencias, arcos y elipses: método Circle

El método ***Circle*** permite dibujar circunferencias, elipses y arcos. Su forma general es la siguiente:

```
object.Circle Step (x, y), radius, color, start, end, aspect
```

donde ***object***, ***step*** y ***color*** tienen el mismo significado que en ***PSet*** y ***Line***, y:

(x, y) son obligatorias, y contienen las coordenadas del centro de la circunferencia.

Radius es obligatoria y define el radio de la circunferencia.

Start, end son opcionales, y permiten definir arcos por medio del ángulo inicial (***start***) y final (***end***). Los ángulos se miden siempre en ***radianes*** y en sentido contrario a las agujas del reloj. Sus valores deben estar entre -2π y 2π. En principio se dibuja solamente el arco, pero si uno o ambos valores son negativos se tratan como positivos, pero se

dibuja una línea que une el centro de la circunferencia con el origen o el extremo del arco.

Aspect es también opcional y se utiliza para dibujar elipses. Es la relación entre el diámetro vertical y el horizontal. El valor por defecto es 1.0, lo que corresponde a una circunferencia. Cuando ***aspect*** es distinto de 1.0, el parámetro ***radius*** define el mayor de los dos diámetros.

Sólo las figuras cerradas (no los arcos sin líneas que unan los extremos con el centro) pueden ser rellenadas con el color determinado por las propiedades ***FillColor*** y ***FillStyle*** del objeto en que se dibuja). El grosor y estilo de las líneas se determina con las propiedades ***DrawWidth*** y ***DrawStyle***. Después de ejecutarse este método, las propiedades ***CurrentX*** y ***CurrentY*** tienen el valor del centro de la circunferencia. Si se omite algún argumento (excepto los que van al final), deben respetarse las comas de separación entre argumentos.

6.4.5 Otros métodos gráficos

Existen algunos otros métodos gráficos de interés. Por ejemplo, el método ***Cls*** cuya forma general es

```
object.Cls
```

borra del formulario o control ***PictureBox*** todos los resultados de los métodos gráficos y del método ***Print***, al mismo tiempo que pone las propiedades ***CurrentX*** y ***CurrentY*** a cero. No afecta a los gráficos introducidos en modo de diseño (por ejemplo con la propiedad ***Picture***). Tampoco se borran con este método el texto y gráficos que se hayan creado con la propiedad ***AutoRedraw*** en ***True***, si dicha propiedad se pone a ***False*** antes de llamar al método ***Cls***. De esta forma se pueden realizar ***borrados selectivos***.

El método ***Point*** devuelve, como entero *long*, el color (RGB) del punto especificado en un formulario o control ***PictureBox***. Su forma general es:

```
object.Point(x, y)
```

Si se desea, el entero *long* devuelto por ***Point*** puede convertirse a la notación hexadecimal que se usa para los colores utilizando la función ***Hex***.

6.5 SISTEMAS DE COORDENADAS

Un punto de particular importancia con ***Visual Basic 6.0*** es el que hace referencia a la posición y tamaño de los formularios y de los demás controles, así como a las unidades en que se expresan y determinan.

Visual Basic 6.0 permite elegir entre distintas unidades, e incluso utilizar distintas unidades para distintos elementos de la aplicación. Las unidades se especifican con la propiedad ***ScaleMode***, que es propia de los formularios y controles ***PictureBox***.

Valor	Nombre	Unidades
0	vbUser	definidas por el usuario
1	vbTwips	twips (1440 por pulgada)
2	vbPoints	puntos (72 por pulgada)
3	vbPixels	pixels
4	vbCharacters	caracteres (120x240 twips)
5	vbInches	pulgadas
6	vbMillimeters	milímetros
7	vbCentimeters	centímetros

Tabla 6.4. Valores de ***ScaleMode***.

La Tabla 6.4 especifica los posibles valores de esta propiedad. La unidad por defecto es el ***twip***, que es la vigésima parte del ***punto*** o ***pixel***.

En un formulario las propiedades relacionadas con la escala y las dimensiones, agrupadas de cuatro en cuatro, son las siguientes: (***top***, ***left***, ***height*** y ***width***) y (***scaleTop***, ***scaleLeft***, ***scaleHeight*** y ***scaleWidth***). Su significado de explica a continuación con ayuda de la Figura 6.7.

En esta figura se muestra la *pantalla* y un *formulario*. La posición y dimensiones del formulario vienen dadas por las propiedades (***top***, ***left***, ***height*** y ***width***). Para un formulario, estas propiedades se definen ***siempre*** en ***twips***. Obsérvese que se miden a partir de la esquina superior izquierda.

Sin embargo, el formulario puede tener su propio sistema de coordenadas interno, definido por las propiedades (***scaleTop***, ***scaleLeft***, ***scaleHeight*** y ***scaleWidth***), para lo cual su propiedad ***ScaleMode*** debe estar puesta a cero. Las propiedades ***scaleLeft*** y ***scaleTop*** determinan las coordenadas de la esquina superior izquierda en el *propio sistema de coordenadas*, mientras que ***scaleWidth*** y ***scaleHeight*** determinan su anchura y altura en dichas coordenadas. En realidad estas propiedades determinan indirectamente la posición del origen de coordenadas y la escala y orientación de los ejes. Si ***scaleHeight*** es positiva el eje de ordenadas va hacia abajo, mientras que si es negativa estará orientado hacia arriba. El eje horizontal va hacia la derecha si ***scaleWidth*** es positiva, y hacia la izquierda si es negativa.

El método ***Scale*** permite establecer las cuatro propiedades (***scaleTop***, ***scaleLeft***, ***scaleHeight*** y ***scaleWidth***) conjuntamente, como se verá en el siguiente apartado. Sólo los formularios y los controles ***PictureBox*** pueden tener las propiedades ***scaleTop***, ***scaleLeft***, ***scaleHeight*** y ***scaleWidth***.

Si las propiedades (***top***, ***left***, ***height*** y ***width***) no se aplican a un formulario sino a un control, ya no es obligatorio medirlas en ***twips***, sino que se miden en las unidades determinadas por la propiedad ***scaleMode*** del ***formulario*** o ***pictureBox*** que las contiene. Cuando estas propiedades se utilizan sin anteponerles el nombre de un objeto se aplican al formulario activo. Para que se apliquen a un objeto cualquiera basta anteponerles el nombre del objeto separado por el operador punto (.).

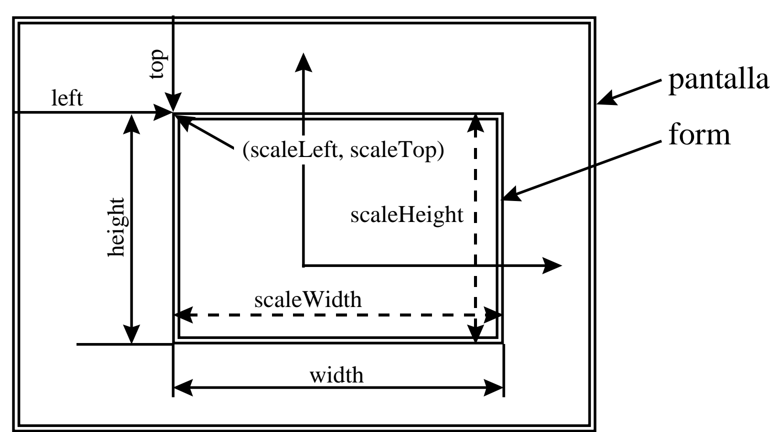


Figura 6.7. Posición y tamaño de una caja ***PictureBox***.

6.5.1 Método Scale

El método ***Scale*** permite definir las cuatro propiedades (***scaleTop***, ***scaleLeft***, ***scaleHeight*** y ***scaleWidth***) de un formulario o ***PictureBox*** simultáneamente. Su forma general es:

```
object.Scale (x1, y1) - (x2, y2)
```

donde ***object*** es el nombre del control ***PictureBox*** (si se omite, el método se aplica al formulario activo). Las coordenadas ***(x1, y1)*** son las coordenadas del vértice superior izquierdo del formulario o ***PictureBox***, mientras que ***(x2, y2)*** corresponden al vértice inferior derecho. Por ejemplo, el siguiente método:

```
pctCaja.Scale (-100, 100) - (100, -100)
```

establece unos ejes en el centro de la ***PictureBox***, con los sentidos ordinarios, que varían entre -100 y 100, tal como puede verse en la Figura 6.8. Este método equivale establecer las cuatro propiedades siguientes:

```
pctCaja.scaleTop = 100
pctCaja.scaleLeft = -100
pctCaja.scaleHeight = -100
pctCaja.scaleWidth = 100
```

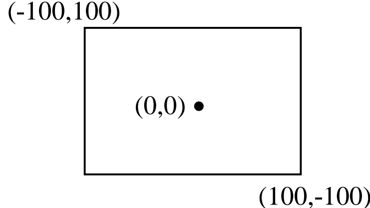


Figura 6.8. Método ***Scale***.

6.6 EVENTOS Y PROPIEDADES RELACIONADAS CON GRÁFICOS

6.6.1 El evento Paint

El evento ***Paint*** se ejecuta cuando un objeto -de tipo ***form*** o ***PictureBox***- se hace visible. Su finalidad es que el resultado de los ***métodos gráficos*** y del método ***Print*** aparezcan en el objeto correspondiente. Hay que tener en cuenta que si se introducen métodos gráficos en el procedimiento ***Form_load*** su resultado no aparece al hacerse visible el formulario (es como si se dibujara sobre el formulario antes de que éste existiera). Para que el resultados de ***Print*** y de los métodos gráficos aparezcan al hacerse visible el formulario, deben introducirse en el procedimiento ***Paint_form***. También los controles ***pictureBox*** tienen evento ***Paint***, que se ejecuta al hacerse visibles.

El evento ***Paint*** tiene mucha importancia en relación con el refresco de los gráficos y con la velocidad de ejecución de los mismos. En los apartados siguientes se completará la explicación de este tema.

6.6.2 La propiedad DrawMode

Esta es una propiedad bastante importante y difícil de manejar, sobre todo si se quieren realizar cierto tipo de acciones con los métodos gráficos. La opción por defecto es la nº 13: ***Copy Pen***.

La propiedad ***DrawMode*** controla cómo se dibujan los controles ***Line*** y ***Shape***, así como los resultados de los métodos gráficos ***PSet***, ***Line*** y ***Circle***. La opción por defecto hace que cada elemento gráfico se dibuje con el color correspondiente (por defecto el ***foreColor***) sobre lo dibujado anteriormente. En ocasiones esto no es lo más adecuado pues, por ejemplo, si se superponen dos figuras del mismo color o si se dibuja con el ***backColor***, los gráficos resultan indistinguibles.

Para entender cómo funciona ***DrawMode*** es necesario tener claros los conceptos de ***color complementario*** y ***combinación de dos colores***. El color complementario de un color es el color que sumado con él da el ***blanco*** (&HFFFFFF&). Por ejemplo, el color complementario del ***rojo*** (&H0000FF&) es el ***cyan*** (&HFFFF00&).

El ***color complementario*** se puede obtener mediante la simple resta del color blanco menos el color original. Por su parte la ***combinación de dos colores*** es el color que resulta de aplicar el operador lógico ***Or***: el color resultante tiene sus *bits* a 1 si alguno o los dos de los colores originales tiene a 1 el *bit* correspondiente. La explicación de los distintos valores de la propiedad ***DrawMode*** que se obtiene del ***Help*** es la siguiente:

Constant	Setting	Description
VbBlackness	1	Blackness.
VbNotMergePen	2	Not Merge Pen— Inverse of setting 15 (Merge Pen).
VbMaskNotPen	3	Mask Not Pen — Combination of the colors common to the background color and the inverse of the pen.
VbNotCopyPen	4	Not Copy Pen — Inverse of setting 13 (Copy Pen).
VbMaskPenNot	5	Mask Pen Not — Combination of the colors common to both the pen and the inverse of the display.
VbInvert	6	Invert — Inverse of the display color.
VbXorPen	7	Xor Pen — Combination of the colors in the pen and in the display color, but not in both.
VbNotMaskPen	8	Not Mask Pen — Inverse of setting 9 (Mask Pen).
VbMaskPen	9	Mask Pen — Combination of the colors common to both the pen and the display.
VbNotXorPen	10	Not Xor Pen — Inverse of setting 7 (Xor Pen).
VbNop	11	Nop — No operation — output remains unchanged. In effect, this setting turns drawing off.
VbMergeNotPen	12	Merge Not Pen — Combination of the display color and the inverse of the pen color.
VbCopyPen	13	Copy Pen (Default) — Color specified by the **ForeColor** property.
VbMergePenNot	14	Merge Pen Not — Combination of the pen color and the inverse of the display color.
VbMergePen	15	Merge Pen — Combination of the pen color and the display color.
VbWhiteness	16	Whiteness.

El explicar más a fondo las distintas aplicaciones de esta propiedad esta fuera del alcance de este manual introductorio.

6.6.3 Planos de dibujo (Layers)

Visual Basic 6.0 considera ***tres planos superpuestos*** (layers): el ***plano frontal***, el ***plano intermedio*** y el ***plano de fondo***. Es importante saber en qué plano se introduce cada elemento gráfico para entender cuándo unos elementos se superpondrán a otros en la pantalla. En principio, los tres planos se utilizan del siguiente modo:

1. En el ***plano frontal*** (***Front***) se dibujan *todos los controles*, excepto los controles gráficos y las labels.
2. En el ***plano intermedio*** se representan los *controles gráficos* y *labels*.
3. En el ***plano de fondo*** se representa el *color de fondo* y el resultado de los *métodos gráficos*.

Estas reglas tienen excepciones que dependen de la propiedad ***AutoRedraw***, de la propiedad ***ClipControl*** y de si los métodos gráficos se utilizan o no asociados al evento ***Paint***.

6.6.4 La propiedad AutoRedraw

Esta propiedad tiene una gran importancia. En principio, todas las aplicaciones de ***Windows*** permiten superponer ventanas y/u otros elementos gráficos, recuperando completamente el contenido de cualquier ventana cuando ésta se selecciona de nuevo y viene a primer plano (es la ventana activa). A esto se llama ***redibujar*** (*redraw*) la ventana. Cualquier aplicación que se desarrolle en ***Visual Basic 6.0*** debe ser capaz de redibujarse correctamente, pero para ello el programador debe conocer algo de la propiedad ***AutoRedraw***.

Por defecto, ***Visual Basic 6.0*** redibuja siempre los controles que aparecen en un formulario. Esto no sucede sin embargo con el resultado de los ***métodos gráficos*** y de ***Print***. Para que la salida de estos métodos se redibuje es necesario adoptar uno de los dos métodos siguientes:

1. Si en el ***form*** o ***pictureBox*** la propiedad ***AutoRedraw*** está en ***False***:
   - Si los ***métodos gráficos*** y ***Print*** están en el procedimiento correspondiente al evento ***Paint*** se redibujan en el *plano de fondo* (los métodos vuelven a ejecutarse, por lo que el proceso puede ser lento en ciertos casos).
   - Si los ***métodos gráficos*** y ***Print*** están fuera del evento ***Paint*** no se redibujan.
2. Si en el ***form*** o ***pictureBox*** la propiedad ***AutoRedraw*** está en ***True***:
   - Si los ***métodos gráficos*** y ***Print*** están en el evento ***Paint*** se ignoran.
   - Si los ***métodos gráficos*** y ***Print*** están fuera del evento ***Paint*** se redibujan guardando en memoria una copia de la zona de pantalla a refrescar. Este es la forma más rápida de conseguir que los gráficos y el texto se redibujen. Tiene el inconveniente de necesitar más memoria.

La propiedad ***AutoRedraw*** de los ***forms*** y de las ***pictureBox*** es independiente, por lo que las dos formas anteriores de conseguir que los gráficos se redibujen se pueden utilizar conjuntamente, por ejemplo una en el formulario y otra en las ***pictureBox***.

6.6.5 La propiedad ClipControl

Por defecto esta propiedad de las ***forms*** y ***pixtureBox*** está en ***True***. En este caso los controles están siempre por encima de la salida de los métodos gráficos, por lo que nunca por ejemplo una línea se dibujará sobre un botón o una barra de desplazamiento (los controles están siempre en el plano frontal o en el plano intermedio, según se ha explicado antes).

Cuando la propiedad ***ClipControl*** se pone a ***False*** se produce una doble circunstancia:

- Los métodos gráficos situados en un evento ***Paint*** siempre se dibujan en el *plano de fondo* y por tanto respetan los controles.
- Los métodos gráficos situados fuera de un evento ***Paint*** se dibujan sobre cualquier elemento que esté en la pantalla, incluidos los controles.

6.7 EJEMPLOS

A continuación se muestra dos ejemplos que hacen uso de algunos de los controles y métodos gráficos explicados previamente.

6.7.1 Ejemplo 6.1: Gráficos y barras de desplazamiento

Este primer programa, cuyo formulario se muestra en la Figura 6.9, es un ejemplo sencillo que permite utilizar algunas de las herramientas gráficas de ***Visual Basic***. Para ello se han utilizado dos barras de desplazamiento que, junto a otras dos cajas de

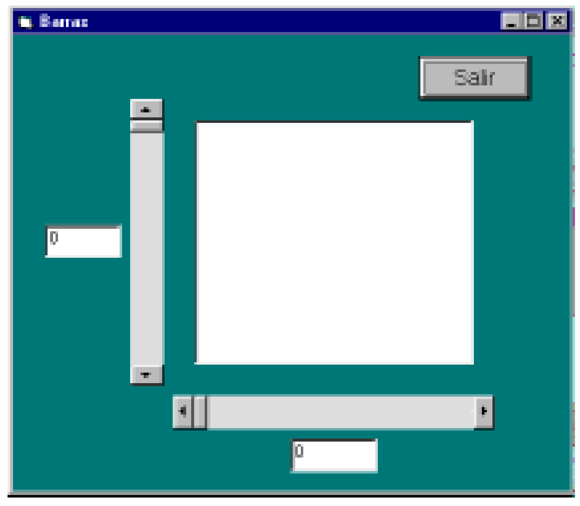


Figura 6.9. Movimiento de un punto con ***PSet***.

texto, modificarán y visualizarán las coordenadas del punto a dibujar.

La Tabla 6.5 muestra los objetos y las propiedades a considerar en este ejemplo.

Control	**Propiedad**	**Valor**	**Control**	**Propiedad**	**Valor**
Hscrollbar	Name	HScroll1	TextBox	Name	txtCaja2
	LargeChange	5		Text	0
	Max	100	TextBox	Name	txtCaja3
	Min	0		Text	0
	SmallChange	1	PictureBox	Name	PctBox
VScrollbar	Name	VScroll1		BackColor	&H00FFFFFF&
	LargeChange	5	CommandButton	Name	Command1
	Max	100		Caption	Salir
	Min	0			
	SmallChange	1			

Tabla 6.5. Controles y propiedades del Ejemplo 6.2.

Se presenta a continuación el código del programa:

```
Private Sub cmdSalir_Click()
    End
End Sub

Private Sub Form_Load()
   pctBox.Scale (0, 0)-(100, 100)
End Sub

Private Sub hsbX_Change()
    txtCaja3.Text = Format(hsbX.Value)
    pctBox.PSet (hsbX.Value, vsbY.Value), vbRed
End Sub


Private Sub txtCaja2_KeyPress(KeyAscii As Integer)
    Dim valor As Integer
    valor = Val(txtCaja2.Text)
    If KeyAscii = 13 Then
        If valor <= vsbY.Max And valor >= vsbY.Min Then
            vsbY.Value = valor
        ElseIf valor > vsbY.Max Then
            vsbY.Value = vsbY.Max
        Else
            vsbY.Value = vsbY.Min
        End If
    End If
End Sub


Private Sub txtCaja3_KeyPress(KeyAscii As Integer)
    Dim valor As Integer
    valor = Val(txtCaja3.Text)
    If KeyAscii = 13 Then
        If valor <= hsbX.Max And valor >= hsbX.Min Then
            hsbX.Value = valor
        ElseIf valor > hsbX.Max Then
            hsbX.Value = hsbX.Max
        Else
            hsbX.Value = hsbX.Min
        End If
    End If
End Sub
```

```
Private Sub vsbY_Change()
    txtCaja2.Text = Format(vsbY.Value)
    pctBox.PSet (hsbX.Value, vsbY.Value), vbRed
End Sub
```

6.7.2 Ejemplo 6.2: Representación gráfica de la solución de la ecuación de segundo grado

En este segundo ejemplo, cuyo formulario se muestra en la Figura 6.10, se representa el lugar de raíces de la ecuación de segundo grado en función de los coeficientes, o más en concreto en función de los cocientes B/A y C/A. El valor de estas relaciones se cambia interactivamente por medio de dos barras de desplazamiento.

El programa permite además la posibilidad de mantener dibujadas las soluciones anteriores de la ecuación, o borrarlas y dibujar sólo las últimas raíces calculadas borrando las anteriores. Para finalizar el programa basta presionar el botón ***Salir***.

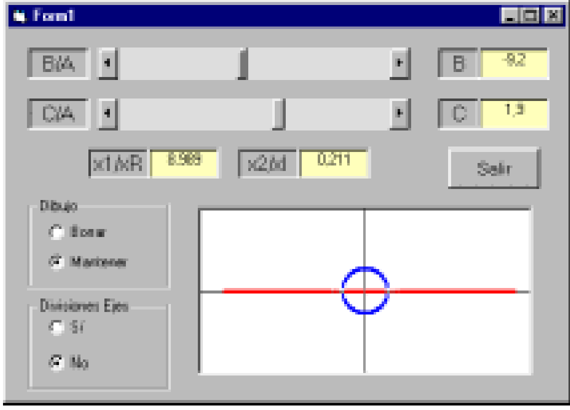


Figura 6.10. Raíces de una ecuación de 2º grado.

La Tabla 6.6 muestra los nombres y los valores de las principales propiedades de los objetos que aparecen en la Figura 6.10.

Control	**Propiedad**	**Valor**	**Control**	**Propiedad**	**Valor**
Frame	Name	fraDib	Label	Name Caption	Label2 C/A
	Caption	Dibujo	Label	Name Caption	Label3 B
Frame	Name	fraEjes	Label	Name Caption	Label4 C
	Caption	Divisiones Ejes	Label	Name Caption	Label5 X1/XR
HScrollBar	Name	hsbBA	Label	Name Caption	Label6 X2/XI
	LargeChange	10	CommandButton	Name	CmdSalir
	Max	1000		Caption	Salir
	Min	-1000	Label	Name	lblBA, lblCA, lblX1, lblB2
	SmallChange	1		BackColor	&H00C0FFFF&
HScrollBar	Name	hsbCA	Option	Name	optD1
	LargeChange	10		Caption	Borrar
	Max	100	Option	Name	optD2
	Min	-100		Caption	Mantener
	SmallChange	1	Option	Name	OptNo
PictureBox	Name	pctBox		Caption	No
	BackColor	&H00FFFFFF&	Option	Name	OptSi
Label	Name Caption	Label1 B/A		Caption	Si

Tabla 6.6. Controles y propiedades del Ejemplo 6.3.

Todas las labels que aparecen tienen la propiedad ***BorderStyle*** igual a *1- Fixed Single*.

El código del programa es el siguiente:

```
Option Explicit
Dim a, b, c As Double
Dim x1, x2, dis, xr, xi As Double

Private Sub divisiones(nx As Integer, ny As Integer)
   Dim i As Integer
   Dim x, xinc, y, yinc As Single
   pctBox.DrawWidth = 1
   xinc = 20 / (nx - 1)
   x = -10
   For i = 1 To nx
      pctBox.Line (x, 0)-(x, -1)
      x = x + xinc
   Next i
   yinc = 10 / (ny - 1)
   y = -5
   For i = 1 To ny
      pctBox.Line (-1, y)-(0, y)
      y = y + yinc
   Next i
   pctBox.DrawWidth = 2
End Sub

Private Sub cmdSalir_Click()
   End
End Sub

Private Sub Form_Load()
   pctBox.Scale (-10, 5)-(10, -5)
End Sub

Private Sub hsbBA_Change()
   a = 1
   b = hsbBA.Value / 10#
   c = hsbCA.Value / 10#
   lblBA.Caption = b
   lblCA.Caption = c
   dis = b ^ 2 - 4 * a * c
   If optD2.Value = True Then    'mantener
      pctBox.AutoRedraw = True
   Else                          'borrar
      pctBox.AutoRedraw = False
      pctBox.Cls
   End If
   If dis > 0 Then
      x1 = (-b + Sqr(dis)) / (2 * a)
      x2 = (-b - Sqr(dis)) / (2 * a)
      lblX1.Caption = Format(x1, "###0.000")
      lblX2.Caption = Format(x2, "###0.000")
      pctBox.PSet (x1, 0), vbRed
      pctBox.PSet (x2, 0), vbRed
   ElseIf dis = 0 Then
      x1 = -b / (2 * a)
      x2 = x1
      lblX1.Caption = Format(x1, "###0.000")
      lblX2.Caption = ""
      pctBox.PSet (x1, 0), vbGreen
   Else
      xr = -b / (2 * a)
      xi = Sqr(-dis) / (2 * a)
      lblX1.Caption = Format(xr, "###0.000")
      lblX2.Caption = Format(xi, "###0.000")
      pctBox.PSet (xr, xi), vbBlue
      pctBox.PSet (xr, -xi), vbBlue
   End If
```

```
   If optSi = True Then
      Call divisiones(10, 5)
   End If
End Sub

Private Sub hsbCA_Change()
   a = 1
   b = hsbBA.Value / 10#
   c = hsbCA.Value / 10#
   lblBA.Caption = b
   lblCA.Caption = c
   dis = b ^ 2 - 4 * a * c
   If optD2.Value = True Then    'mantener
      pctBox.AutoRedraw = True
   Else                          'borrar
      pctBox.AutoRedraw = False
      pctBox.Cls
   End If
   If dis > 0 Then
      x1 = (-b + Sqr(dis)) / (2 * a)
      x2 = (-b - Sqr(dis)) / (2 * a)
      lblX1.Caption = Format(x1, "###0.000")
      lblX2.Caption = Format(x2, "###0.000")
      pctBox.PSet (x1, 0), vbRed
      pctBox.PSet (x2, 0), vbRed
   ElseIf dis = 0 Then
      x1 = -b / (2 * a)
      x2 = x1
      lblX1.Caption = Format(x1, "###0.000")
      lblX2.Caption = ""
      pctBox.PSet (x1, 0), vbGreen
   Else
      xr = -b / (2 * a)
      xi = Sqr(-dis) / (2 * a)
      lblX1.Caption = Format(xr, "###0.000")
      lblX2.Caption = Format(xi, "###0.000")
      pctBox.PSet (xr, xi), vbBlue
      pctBox.PSet (xr, -xi), vbBlue
   End If
   If optSi = True Then
      Call divisiones(10, 5)
   End If
End Sub

Private Sub optD1_Click()
   pctBox.AutoRedraw = True
   pctBox.Cls
   pctBox.DrawWidth = 1
   pctBox.Line (-90, 0)-(90, 0), vbBlack
   pctBox.Line (0, -45)-(0, 45), vbBlack
   pctBox.DrawWidth = 2
End Sub

Private Sub pctBox_Paint()
   pctBox.AutoRedraw = True
   pctBox.Line (-90, 0)-(90, 0), vbBlack
   pctBox.Line (0, -45)-(0, 45), vbBlack
   pctBox.DrawWidth = 2
End Sub
```

6.8 BARRAS DE HERRAMIENTAS (TOOLBARS)

JGJCon ***Visual Basic 6.0*** es fácil crear ***barras de herramientas*** constituidas por botones clicables, al estilo de las aplicaciones de ***Windows***. De ordinario las barras de herramientas dan acceso a las funciones o comandos más comunes de los menús de la aplicación.

Se puede crear una barra de herramientas por medio de un ***PictureBox*** colocado en un formulario. En este ***PictureBox*** se pueden colocar controles ***CommandButton*** o ***Image*** en los que se programa el evento ***click***. La propiedad ***Picture*** del control ***Image*** puede contener la dirección de alguno de los iconos estándar que vienen con ***Visual Basic*** (extensión ****.ico***) o la de un icono construido por el programador.

En el caso de los formularios MDI se puede colocar una barra de herramientas en el ***MDIform***, que automáticamente adquiere la anchura del formulario.

7. ARCHIVOS Y ENTRADA/SALIDA DE DATOS

En este capítulo se van a describir varias formas de introducir información en el programa, así como de obtener resultados en forma impresa o mediante escritura en un fichero. Se va a presentar una nueva forma interactiva de comunicarse con el usuario, como son las cajas de diálogo ***MsgBox*** e ***InputBox***. Particular interés tiene la lectura y escritura de datos en el disco, lo cual es necesario tanto cuando el volumen de información es muy importante (la memoria RAM está siempre más limitada que el espacio en disco), como cuando se desea que los datos no desaparezcan al terminar la ejecución del programa. Los ficheros en disco resuelven ambos problemas.

También se verá en este capítulo cómo obtener resultados alfanuméricos y gráficos por la impresora.

7.1 CAJAS DE DIÁLOGO INPUTBOX Y MSGBOX

Estas cajas de diálogo son similares a las que se utilizan en muchas aplicaciones de ***Windows***. La caja de mensajes o ***MsgBox*** abre una ventana a través de la cual se envía un mensaje al usuario y se le pide una respuesta, por ejemplo en forma de clicar un botón ***O.K./Cancel***, o ***Yes/No***. Este tipo de mensajes son muy utilizados para confirmar acciones y para decisiones sencillas. La caja de diálogo ***InputBox*** pide al usuario que teclee una frase, por ejemplo su nombre, un título, etc.

La forma general de la función ***MsgBox*** es la siguiente:

```
respuesta = MsgBox("texto para el usuario", tiposBotones, "titulo")
```

donde ***respuesta*** es la variable donde se almacena el valor de retorno, que es un número indicativo del botón clicado por el usuario, de acuerdo con los valores de la Tabla 7.1. La constante simbólica que representa el valor de retorno indica claramente el botón clicado. Los otros dos argumentos son opcionales. El parámetro ***tiposBotones*** es un entero que indica la combinación de botones deseada por el usuario; sus posibles valores se muestran en la Tabla 7.2. También en este caso la constante simbólica correspondiente es suficientemente explícita. Si este argumento se omite se muestra sólo el botón ***O.K.*** El parámetro ***titulo*** contiene un texto que aparece como título de la ventana; si se omite, se muestra en su lugar el nombre de la aplicación.

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

Tabla 7.1. Botón clicado por el usuario.

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

Tabla 7.2. Botones mostrados en ***MsgBox***.

Se puede modificar el valor de ***tiposBotones*** de modo que el botón que se activa por defecto cuando se pulsa la tecla ***Intro*** (el botón que tiene el *focus*) sea cualquiera de los botones de la caja. Para ello basta sumar a ***tiposBotones*** otra constante que puede tomar uno de los tres valores siguientes: ***0*** (*vbDefaulButton1*, que representa el primer botón), ***256*** (*vbDefaulButton2*, que representa el segundo botón) y ***512*** (*vbDefaulButton3*, que representa el tercer botón).

Figura 7.1. Ejemplo de caja ***MsgBox***.

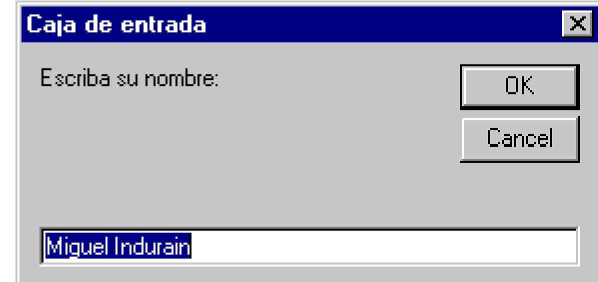


Figura 7.2. Ejemplo de caja de ***InputBox***.

Finalmente, se puede incluir en el mensaje un *icono ad-hoc* por el mismo procedimiento de sumarle al argumento ***tiposBotones*** una nueva constante numérica con los siguientes valores y significados definidos por la constante simbólica apropiada: 16 (*vbCritical*), 32 (*vbQuestion*), 48 (*vbExclamation*) y 64 (*vbInformation*). Es obvio que, por los propios valores considerados, al sumar estas constantes o las anteriores al argumento ***tiposBotones***, la información original descrita en la Tabla 7.2 no se pierde. La Figura 7.1 muestra un ejemplo de caja ***MsgBox*** resultado de ejecutar el comando siguiente:

```
lblBox.Caption = MsgBox("Pulse un botón: ", 2 + 256 + 48, _
                        "Caja de mensajes")
```

donde el “2” indica que deben aparecer los botones ***Abort***, ***Retry*** y ***Cancel***, el “256” indica que el *botón por defecto* es el segundo (***Retry***) y el “48” indica que debe aparecer el *icono de exclamación*.

Por otra parte, la forma general de la función ***InputBox*** es la siguiente:

```
texto = InputBox("texto para el usuario", "titulo", "default", left, top)
```

donde ***texto*** es la variable donde se almacena el valor de retorno, que es el texto tecleado por el usuario. Los parámetros "*texto para el usuario*" y ***titulo*** tienen el mismo significado que en ***MsgBox***. El parámetro ***default*** es un texto por defecto que aparece en la caja de texto y que el usuario puede aceptar, modificar o sustituir; el contenido de esta caja es lo que en definitiva esta función devuelve como valor de retorno. Finalmente, ***left*** y ***top*** son las coordenadas de la esquina superior izquierda de la ***InputBox***; si se omiten, ***Visual Basic 6.0*** dibuja esta caja centrada en horizontal y algo por encima de la mitad de la pantalla en vertical. La Figura 7.2 muestra un ejemplo de caja ***InputBox*** resultado de ejecutar el comando siguiente:

```
lblBox.Caption = InputBox("Escriba su nombre: ", _
                          "Caja de entrada", "Miguel Indurain")
```

donde el nombre que aparece por defecto es el del mejor ciclista de los últimos tiempos. Este nombre aparece seleccionado y puede ser sustituido por otro que teclee el usuario.

7.2 MÉTODO PRINT

Este método permite escribir texto en ***formularios***, cajas ***pictureBox*** y en un objeto llamado ***Printer*** que se verá un poco más adelante, en el Apartado 7.3.

7.2.1 Características generales

La forma general del método ***Print*** se explica mejor con algunos ejemplos como los siguientes:

```
pctBox.Print "La distancia es: "; Dist; " km."
pctBox.Print 123; 456; "San"; "Sebastián"
```

```
pctBox.Print 123, 456, "San", "Sebastián"
pctBox.Print -123; -456
```

cuyo resultado se puede ver en la Figura 7.3 (puede variar dependiendo del tipo y tamaño de las letras):

Figura 7.3: Ejemplo del método ***Print***.

De estos ejemplos se pueden ya sacar algunas conclusiones:

1. El método ***Print*** recibe como datos una *lista de variables y/o cadenas de caracteres*. Las cadenas son impresas y las variables se sustituyen por su valor.
2. Hay dos tipos básicos de ***separadores*** para los elementos de la lista. El carácter ***punto y coma*** (;) hace que se escriba inmediatamente a continuación de lo anterior. La ***coma*** (,) hace que se vaya al comienzo de la siguiente ***área de salida***. Con letra de paso constante como la Courier las áreas de salida empiezan cada 14 caracteres, es decir en las columnas 1, 15, 29, etc. Con letras de paso variable esto se hace sólo de modo aproximado.
3. Las constantes numéricas positivas van precedidas por un espacio en blanco y separadas entre sí por otro espacio en blanco. Si son negativas el segundo espacio es ocupado por el signo menos (-).
4. El tipo y tamaño de letra que se utiliza depende de la propiedad ***Font*** del formulario, objeto ***PictureBox*** u objeto ***Printer*** en que se esté escribiendo.

Existen otros separadores tales como ***Tab(n)*** y ***Spc(n)***. El primero de ellos lleva el punto de inserción de texto a la columna ***n***, mientras que el segundo deja ***n*** espacios en blanco antes de seguir escribiendo. ***Tab*** sin argumento equivale a la coma (,). Estos espaciadores se utilizan en combinación con el punto y coma (;), para separarlos de los demás argumentos.

Por defecto, la salida de cada método ***Print*** se escribe en una nueva línea, pero si se coloca un punto y coma al final de un método ***Print***, el resultado del siguiente ***Print*** se escribe en la misma línea.

Puede controlarse el lugar del formulario o control donde se imprime la salida del método ***Print***. Esta salida se imprime en el lugar indicado por las propiedades ***CurrentX*** y ***CurrentY*** del formulario o control donde se imprime. Cambiando estas propiedades se modifica el lugar de impresión, que por defecto es la esquina superior izquierda. Existen unas funciones llamadas ***TextWidth(string)*** y ***TextHeight(string)*** que devuelven la anchura y la altura de una cadena de caracteres pasada como argumento. Estas funciones pueden ayudar a calcular los valores más adecuados para las propiedades ***CurrentX*** y ***CurrentY***.

La función ***str(valor_numérico)*** convierte un número en cadena de caracteres para facilitar su impresión. En realidad, es lo que ***Visual Basic 6.0*** ha hecho de modo implícito en los ejemplos anteriores. En versiones anteriores del programa era necesario que el usuario realizase la conversión de modo explícito.

7.2.2 Función Format

La función ***Format*** realiza las conversiones necesarias para que ciertos datos numéricos o de otro tipo puedan ser impresos con ***Print***. Como se ha visto, en el caso de las variables numéricas esto no es imprescindible, pero la función ***Format*** permite controlar el número de espacios, el número de decimales, etc. En el caso de su aplicación a objetos tipo *fecha* (***date***) y *hora* (***time***) la aplicación de

Format es imprescindible, pues ***Print*** no los escribe directamente. La forma general de esta función es la siguiente:

```
Format(expresion, formato)
```

donde ***expresion*** es una variable o expresión y ***formato*** -que es opcional- describe el formato deseado para el resultado. El valor de retorno es una cadena de caracteres directamente utilizable en ***Print***. Para *fechas* existen formatos predefinidos tales como *"General Date"*, *"Long Date"*, *"Medium Date"* y *"Short Date"*; para la *hora* los formatos predefinidos son *"Long Time"*, *"Medium Time"* y *"Short Time"*. Además existe la posibilidad de que el usuario defina sus propios formatos (ver ***User-Defined Date/Time Formats (Format Function)***, en el ***Help*** del programa). El usuario también puede definir sus propios formatos numéricos y de cadenas de caracteres.

A diferencia de la función ***Str***, la función ***Format*** no deja espacio en blanco para el signo de los números positivos.

7.3 UTILIZACIÓN DE IMPRESORAS

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

7.3.1 Método PrintForm

El método ***PrintForm*** permite imprimir un formulario con sus controles y con los resultados de los métodos gráficos (***PSet***, ***Line*** y ***Circle***) y del método ***Print***. Para ello la propiedad ***AutoRedraw*** del formulario tiene que estar puesta a ***True***, y los métodos citados tienen que estar llamados desde un evento distinto del ***Paint***. Lo único que no se dibuja del formulario es la *barra de títul*o.

Este sistema de impresión es muy sencillo de utilizar, pero tiene el inconveniente de que el resultado se imprime con la misma resolución de la pantalla (entre 50 y 100 puntos por pulgada), no aprovechando por tanto la mayor resolución que suelen tener las impresoras (300, 600 ó más puntos por pulgada).

7.3.2 Objeto Printer

Este segundo sistema tiene la ventaja de que permite aprovechar plenamente la resolución de la impresora, pero ***no permite dibujar controles*** sino sólo los métodos gráficos habituales (***PSet, Line*** y ***Circle***), el método ***Print*** y un método no visto hasta ahora que es ***PaintPicture***.

Para ***Visual Basic 6.0*** la impresora es un objeto gráfico más, similar a los formularios y a las cajas gráficas ***PictureBox***. Como tal objeto gráfico tiene sus propiedades generales (*DrawStyle*, *BackColor*, *ForeColor*, etc.), además de otras propiedades específicas de la impresora, como *DeviceName*, *DriverName*, *Orientation*, *Copies*, etc. Para más información puede utilizarse el ***Help***, buscando ***Printer object***. En principio se utiliza la impresora por defecto del PC, pero ***Visual Basic*** mantiene una ***Printers Collection***, que es algo así como un array de impresoras disponibles. A partir de esta ***Printers Collection*** se puede cambiar a la impresora que se desee.

El objeto ***Printer*** tiene un método llamado ***EndDoc*** para enviar realmente a la impresora el resultado de los métodos gráficos. El método ***PaintPicture*** permite incorporar el contenido de ***ficheros gráficos*** a un *formulario*, ***PictureBox*** o ***Printer***. Su forma general es:

```
object.PaintPicture pictProp X, Y, Width, Height
```

donde ***pictProp*** indica el gráfico (coincide con la propiedad ***Picture*** de ***PictureBox***), ***X*** e ***Y*** indican las coordenadas de inserción y los dos últimos parámetros las dimensiones (opcionales).

7.4 CONTROLES FILELIST, DIRLIST Y DRIVELIST

Uno de los problemas que hay que resolver para leer o escribir en ficheros de disco es ser capaces de localizar interactivamente los correspondientes ficheros, de modo análogo a como se realiza con los comandos ***File/Open*** o ***File/Save As*** de ***Word***, ***Excel*** o de cualquier otra aplicación. Este tipo de operaciones se pueden hacer mucho más fácilmente con los ***Common Dialog Controls*** vistos en el Apartado 4.4, en la página 58, ***aconsejando por lo tanto su uso***. A pesar de ello, aquí se van a explicar los controles específicos de que dispone ***Visual Basic 6.0.***

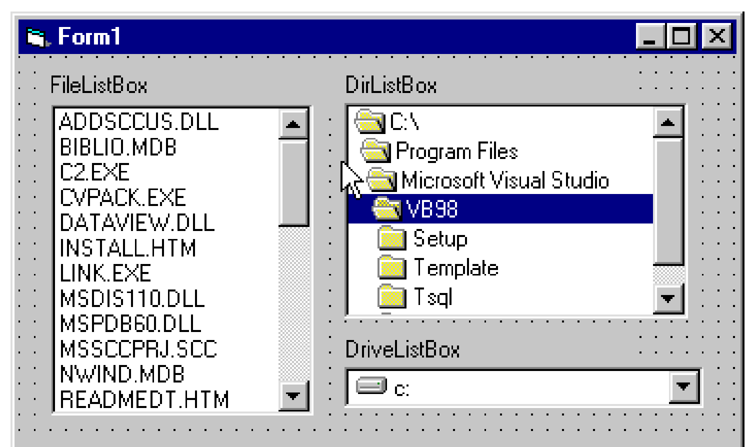


Figura 7.4. Cajas ***DriveListBox***, ***DirListBox*** y ***FileListBox***.

Visual Basic 6.0 dispone de tres controles que facilitan el recorrer el árbol de ficheros y de directorios, localizando o creando interactivamente un fichero determinado. Estos controles son el ***FileListBox*** (para ficheros), el ***DirListBox*** (para directorios) y el ***DriveListBox*** (para unidades de disco). La Figura 7.4 muestra estos tres controles, junto con unas etiquetas que los identifican. Los dos primeros son ***listas***, mientras que el tercero es una caja de tipo ***ComboBox***.

En principio estos controles, cuando se colocan en un formulario tal como se muestra en la Figura 7.4, están *desconectados*. Quiere esto decir que al cambiar la unidad de disco (***drive***) no se muestran en la caja ***dirListBox*** los directorios correspondientes a la nueva unidad de disco. Por otra parte, al cambiar de directorio tendrán que cambiar de modo acorde los ficheros en la caja ***fileListBox***. La dificultad de conectar estas cajas no es grande, pero sí hay que saber cómo se hace pues depende de propiedades de estas cajas que no aparecen en la ventana de propiedades (ventana ***Properties***) en modo de diseño, y que ***sólo están accesibles en modo de ejecución***. De entre estas propiedades las más importantes son las siguientes:

1. La ***DriveListBox*** tiene una propiedad llamada ***Drive*** que recoge la unidad seleccionada por el usuario (puede ser una unidad física como el disco ***c:*** o una unidad lógica asignada por el usuario a otro disco o directorio en un servidor o en otro ordenador de la red).
2. La propiedad ***path*** de la caja ***DirListBox*** determina el ***drive*** seleccionado y por tanto qué directorios se muestran en dicha caja.
3. Finalmente, una propiedad también llamada ***path*** de la caja ***FileListBox*** determina el directorio que contiene los ficheros mostrados.

Para enlazar correctamente las cajas de discos, directorios y ficheros se puede utilizar el evento ***Change***, de tal forma que cada vez que el usuario cambia la unidad de disco se cambia el ***path*** del directorio y cada vez que se cambia el directorio se cambia el ***path*** de los ficheros. Esto puede hacerse con el código siguiente:

```
Private Sub dirPrueba_Change()
   filPrueba.Path = dirPrueba.Path
End Sub

Private Sub drvPrueba_Change()
   dirPrueba.Path = drvPrueba.Drive
End Sub
```

La caja ***FileListBox*** tiene una propiedad llamada ***FileName*** que contiene el nombre del fichero seleccionado por el usuario. Para tener el ***path*** completo del fichero basta anteponerle la propiedad ***Path*** de la ***fileListBox***, que incluye el directorio y el drive, y la barra invertida (\). Si el usuario introduce ***FileName*** incluyendo el ***path***, ***Visual Basic*** actualiza también de modo automático la propiedad ***Path*** de ***FileListBox***. El usuario se debe preocupar de utilizar el evento ***Change*** para actualizar el ***Path*** de la caja ***DirListBox*** y la propiedad ***Drive*** de ***DriveListBox***.

Otra propiedad importante es la propiedad ***Pattern***, que indica los tipos de ficheros que se mostrarán en la caja. El valor por defecto es ***"*.*"***, lo cual hace que se muestren todos los ficheros. Si su valor fuese ***"*.doc"*** sólo se mostrarían los ficheros con esta extensión. La propiedad ***Pattern*** admite varias opciones separadas por untos y coma (***"*.doc; *.dot"***).

7.5 TIPOS DE FICHEROS

Tanto en ***Windows*** como en ***Visual Basic 6.0*** existen, principalmente, dos tipos de archivos:

1. ***Ficheros ASCII*** o ficheros de texto. Contienen caracteres codificados según el código ASCII y se pueden leer con cualquier editor de texto como ***Notepad***. Suelen tener extensión ****.txt*** o ****.bat***, pero también otras como ****.m*** para los programas de ***Matlab***, ****.c*** para los ficheros fuente de C, ****.cpp*** para los ficheros fuente de C++ y ****.java*** para los de ***Java***.
2. ***Ficheros binarios***: Son ficheros imagen de los datos o programas tal como están en la memoria del ordenador. No son legibles directamente por el usuario. Tienen la ventaja de que ocupan menos espacio en disco y que no se pierde tiempo y precisión cambiándolos a formato ASCII al escribirlos y al leerlos en el disco.

Con ***Visual Basic 6.0*** se pueden leer tanto ficheros ASCII como ficheros binarios. Además el acceso a un fichero puede ser de tres formas principales.

1. ***Acceso secuencial***. Se leen y escriben los datos como si se tratara de un libro: siempre a continuación del anterior y sin posibilidad de volver atrás o saltar datos. Si se quiere acceder a un dato que está hacia la mitad de un fichero, habrá que pasar primero por todos los datos anteriores. Los ficheros de texto tienen acceso secuencial.
2. ***Acceso aleatorio*** *(random)*: Permiten acceder directamente a un dato sin tener que pasar por todos los demás, y pueden acceder a la información en cualquier orden. Tienen la limitación de que los datos están almacenados en unas unidades o bloques que se llaman ***registros***, y que todos los registros que se almacenan en un fichero deben ser del mismo tamaño. Los ficheros de acceso aleatorio son ficheros binarios.
3. ***Acceso binario***. Son como los de acceso aleatorio, pero el acceso no se hace por *registros* sino por ***bytes***.

Antes de poder leer o escribir en un fichero hay que abrirlo por medio de la sentencia ***Open***. En esta sentencia hay que especificar qué tipo de acceso se desea tener, distinguiendo también si es para lectura (***input***), escritura (***output***) o escritura añadida (***append***).

7.6 LECTURA Y ESCRITURA EN FICHEROS SECUENCIALES

7.6.1 Apertura y cierre de ficheros

Para poder leer o escribir en un fichero 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 fichero a abrir. Será una variable ***string*** o un nombre entre dobles comillas (“ ”).

modo Para acceso secuencial existen tres posibilidades: ***Input*** para leer, ***Output*** para escribir al comienzo de un fichero y ***Append*** para escribir al final de un fichero ya existente. Si se intenta abrir en modo ***Input*** un fichero que no existe, se produce un error. Si se abre para escritura en modo ***Output*** un fichero 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 fichero 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 fichero que se abre. En todas las operaciones sucesivas de lectura y/o escritura se hará referencia a este fichero por medio de este número. No puede haber dos ficheros abiertos con el mismo número. ***Visual Basic*** dispone de una función llamada ***FreeFile*** que devuelve un número no ocupado por ningún fichero.

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

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

Después de terminar de leer o escribir en un fichero 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***.

7.6.2 Lectura y escritura de datos

7.6.2.1 Sentencia Input

Existen varias formas de leer en un fichero de acceso secuencial. Por ejemplo, para leer el valor de una o más variables se utiliza la sentencia ***Input***:

```
Input # fileNo, varName1, varName2, varName3, ...
```

donde el ***fileNo*** es el número asignado al archivo al abrirlo y *varName1, varName2, ...* son los nombres de las variables donde se guardarán los valores leídos en el fichero. Debe haber una correspondencia entre el orden y los tipos de las variables en la lista, con los datos almacenados en el fichero. No se pueden leer directamente vectores, matrices o estructuras. Si los datos del disco han de ser escritos por el propio programa, conviene utilizar la sentencia ***write*** (mejor que ***Print***) para garantizar que los valores están convenientemente separados. La sentencia ***Write*** se verá posteriormente.

7.6.2.2 Función Line Input y función Input

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 ficheros 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 fichero se utiliza un bucle ***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 fichero. Véase el siguiente ejemplo:

```
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 fichero. Por ejemplo, para leer todo el contenido de un fichero y escribirlo en una caja de texto se puede utilizar:

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

7.6.2.3 Función Print #

Para escribir el valor de unas ciertas variables en un fichero 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. Las reglas para determinar el formato de la función ***Print #*** son las mismas que las del método ***Print*** visto previamente.

7.6.2.4 Función Write #

A diferencia de ***Print #***, la función ***Write #*** introduce comas entre las variables y/o cadenas de caracteres de la lista, además encierra entre dobles comillas las cadenas de caracteres antes de escribirlas en el fichero. La función ***Write #*** introduce un carácter ***newline***, esto es, un ***return*** o un ***return+linefeed*** después del último carácter de las lista de variables. Los ficheros escritos con ***Write #*** son siempre legibles con ***Input #***, cosa que no se puede decir de ***Print #***. Véase el siguiente ejemplo:

```
' Se abre el fichero para escritura
Open "C:\Temp\TestFile.txt" For Output As #1
Write #1, "Hello World", 234           ' Datos separados por comas
MyBool = False: MyDate = #2/12/1969#   ' Valores de tipo boolean y Date
Write #1, MyBool; " is a Boolean value"
Write #1, MyDate; " is a date"
Close #1                               ' Se cierra el fichero
```

El fichero ***TestFile.txt*** guardado en ***C:\Temp*** contendrá:

```
"Hello World",234
#FALSE#," is a Boolean value"
#1969-02-12#," is a date"
```

7.7 FICHEROS DE ACCESO ALEATORIO

Los ficheros de acceso aleatorio se caracterizan porque en ellos se puede leer en cualquier orden. Los ficheros de acceso aleatorio son ficheros binarios. Cuando se abre un fichero se debe escribir ***For Random***, al especificar el modo de apertura (si el fichero se abre ***For Binary*** el acceso es similar, pero no por ***registros*** sino por ***bytes***; este modo es mucho menos utilizado).

7.7.1 Abrir y cerrar archivos de acceso aleatorio

Estos archivos se abren también con la sentencia ***Open***, pero con modo ***Random***. Al final se añade la sentencia ***Len=longitudRegistro***, en bytes. Véase el siguiente ejemplo:

```
fileNo = FreeFile
size = Len(unObjeto)
Open filename For Random as #fileNo Len = size
```

donde *filename* es una variable que almacena el nombre del archivo. Se recuerda que la función ***FreeFile*** devuelve un número entero válido (esto es que no está siendo utilizado) para poder abrir un fichero. El último parámetro informa de la longitud de los registros (todos deben tener la misma longitud). ***Visual Basic 6.0*** dispone de la función ***Len(objetoName)***, que permite calcular la dimensión en bytes de cualquier objeto perteneciente a una clase o estructura.

De ordinario los ficheros de acceso directo se utilizan para leer o escribir de una vez todo un bloque de datos. Este bloque suele ser un *objeto* de una *estructura*, con varias *variables miembro*. Los ficheros abiertos para acceso directo se cierran con ***Close***, igual que los secuenciales.

7.7.2 Leer y escribir en una archivo de acceso aleatorio. Funciones Get y Put

Se utilizan las funciones ***Get*** y ***Put***. Su sintaxis es la siguiente:

```
Get #fileNo, registroNo, variableObjeto

Put #fileNo, registroNo, variableObjeto
```

La instrucción ***Get*** lee un registro del fichero y almacena los datos leídos en una variable, que puede ser un *objeto* de una determinada *clase* o *estructura*. La instrucción ***Put*** escribe el contenido

de la variable en la posición determinada del fichero. Si se omite el número de registro se lee (escribe) a continuación del registro leído (escrito) anteriormente. Véase el siguiente ejemplo:

```
FileNo=FreeFile
size=Len(unObjeto)
Open filename for Random as #fileNo Len=size
Get #fileNo, 3, size
```

Con este ejemplo, se ha abierto el fichero ***filename*** de la misma forma que se realizó en el ejemplo anterior, pero ahora, además se ha leído un registro de longitud ***size***, y más en concreto, el tercer registro. Si se quisiera modificar el valor de este registro, no habría más que asignarle el valor que se quisiera, para a continuación introducirlo en el fichero mediante la sentencia siguiente:

```
Put #fileNo, 3, size
```

7.8 FICHEROS DE ACCESO BINARIO

La técnica a emplear es básicamente la misma que con los ficheros de acceso aleatorio, con la salvedad de que en lugar de manejar ***registros***, en los ficheros de acceso binario se trabaja con ***bytes***. Véase el siguiente ejemplo:

```
FileNo=FreeFile
Open filename for Binary as #fileNo
Get #1, 4, dato
dato = 7
Put #1, 4, dato
Close #1
```

En el anterior ejemplo se puede observar como primero se introduce en la variable ***dato*** el valor del cuarto byte del fichero ***filename***, para posteriormente asignarle el valor 7, e introducirlo de nuevo en el cuarto byte de ***filename***.

8. ANEXO A: CONSIDERACIONES ADICIONALES SOBRE DATOS Y VARIABLES

En este ***Anexo*** se incluyen algunas consideraciones de interés para personas que no han programado antes en otros lenguajes. A continuación se explican las posibilidades y la forma de almacenar los distintos tipos de variables.

8.1 CARACTERES Y CÓDIGO ASCII

Las variables ***string*** (cadenas de caracteres) contienen un conjunto de caracteres que se almacenan en ***bytes*** de memoria. Cada carácter es almacenado en un ***byte*** (8 bits). En un bit se pueden almacenar dos valores (0 y 1); con dos bits se pueden almacenar 2^2 = 4 valores (00, 01, 10, 11 en binario; 0, 1 2, 3 en decimal). Con 8 bits se podrán almacenar 2^8 = 256 valores diferentes (normalmente entre 0 y 255; con ciertos compiladores entre -128 y 127).

En realidad, cada ***letra*** se guarda en un solo ***byte*** como un número entero, el correspondiente a esa letra en el código ASCII (una correspondencia entre números enteros y caracteres, ampliamente utilizada en informática), que se muestra en la Tabla 8.1 para los caracteres estándar (existe un código ASCII extendido que utiliza los 256 valores y que contiene caracteres especiales y caracteres específicos de los alfabetos de diversos países, como por ejemplo las *vocales acentuadas* y la *letra ñ* para el castellano).

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	“	#	$	%	&	‘
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

Tabla 8.1. Código ASCII estándar.

Esta tabla se utiliza de la siguiente forma. La primera cifra (las dos primeras cifras, en el caso de los números mayores o iguales que 100) del número ASCII correspondiente a un carácter determinado figura en la primera columna de la Tabla 8.1, y la última cifra en la primera fila de dicha Tabla. Sabiendo la fila y la columna en la que está un determinado carácter puede componerse el número ASCII correspondiente. Por ejemplo, la letra A está en la fila 6 y la columna 5. Su número ASCII es por tanto el 65. El carácter % está en la fila 3 y la columna 7, por lo que su representación ASCII será el 37. Obsérvese que el código ASCII asocia números consecutivos con las letras mayúsculas y minúsculas ordenadas alfabéticamente. Esto simplifica notablemente ciertas operaciones de ordenación alfabética de nombres. Nótese que todas las mayúsculas tienen código ASCII anterior a cualquier minúscula.

En la Tabla 8.1 aparecen muchos caracteres no imprimibles (todos aquellos que se expresan con 2 ó 3 letras). Por ejemplo, el **ht** es el *tabulador horizontal*, el **nl** es el carácter *nueva línea*, etc.

En realidad la ***Versión 6.0*** de ***Visual Basic*** no utiliza para representar caracteres el código ASCII sino el ***Unicode***, que utiliza dos bytes por carácter, con lo cual tiene capacidad para representar 65536 caracteres, pudiéndose adaptar a las lenguas orientales. Para los caracteres latinos el byte más significativo coincide con el del código ASCII, mientras que el segundo byte está a cero.

8.2 NÚMEROS ENTEROS

Los números enteros en ***Visual Basic 6.0*** se guardan en 1, 2 ó 4 bytes.

- Con 8 bits (1 byte) se podrían guardar 2^8 números: desde 0 hasta 255.
- Con 16 bits (2 bytes) se podrían guardar 2^{16} números: desde 0 hasta 65.535. Si se reserva un bit para el signo se tendrían 2^{15} números: desde **-32768** hasta **32767**
- Con 32 bits (4 bytes) se podrían guardar 2^{32} números: desde 0 hasta 4.294.967.295. Si se reserva un bit para el signo se tendrían 2^{31}: desde **-2.147.483.648** hasta **2.147.483.647**

8.3 NÚMEROS REALES

En muchas aplicaciones hacen falta variables reales, capaces de representar magnitudes que contengan *una parte entera y una parte fraccionaria* o *decimal*. Estas variables se llaman también de ***punto flotante***. De ordinario, en base 10 y con notación científica, estas variables se representan por medio de la ***mantisa***, que es un número mayor o igual que 0.1 y menor que 1.0, y un ***exponente*** que representa la potencia de 10 por la que hay que multiplicar la mantisa para obtener el número considerado. Por ejemplo, el número π se representa como $0.3141592654 \cdot 10^1$. Tanto la ***mantisa*** como el ***exponente*** pueden ser positivos y negativos.

8.3.1 Variables tipo Single

Los computadores trabajan en base 2. Por eso un número con decimales de tipo ***Single*** se almacena en 4 bytes (32 bits), utilizando *24 bits para la mantisa* (1 para el signo y 23 para el valor) y *8 bits para el exponente* (1 para el signo y 7 para el valor). Es interesante ver qué clase de números de punto flotante pueden representarse de esta forma. En este caso hay que distinguir el ***rango*** de la ***precisión***. La *precisión* hace referencia al número de cifras con las que se representa la *mantisa*: con 23 bits el número más grande que se puede representar es,

$$2^{23} = 8.388.608$$

lo cual quiere decir que se pueden representar todos los números decimales de 6 cifras y la mayor parte –aunque no todos– de los de 7 cifras (por ejemplo, el número 9.213.456 no se puede representar con 23 bits). Por eso se dice que las variables tipo ***Single*** tienen entre 6 y 7 cifras decimales equivalentes de precisión.

Respecto al *exponente de dos* por el que hay que multiplicar la *mantisa* en base 2, con 7 bits el número más grande que se puede representar es 127. El ***rango*** vendrá definido por la potencia,

$$2^{127} = 1.7014 \cdot 10^{38}$$

lo cual indica el orden de magnitud del número más grande representable de esta forma.

En el caso de los números ***Single*** (4 bytes) el rango es: desde **-3.402823E38** a **-1.401298E-45** para los valores negativos y desde **1.401298E-45** a **3.402823E38** para los valores positivos.

8.3.2 Variables tipo Double

Así pues, las variables tipo ***Single*** tienen un *rango* y –sobre todo– una *precisión* muy limitada, insuficiente para la mayor parte de los cálculos técnicos y científicos. Este problema se soluciona con el tipo ***Double***, que utiliza 8 bytes (64 bits) para almacenar una variable. Se utilizan *53 bits para la mantisa* (1 para el signo y 52 para el valor) y *11 para el exponente* (1 para el signo y 10 para el valor). La ***precisión*** es en este caso,

$$2^{52} = 4.503.599.627.370.496$$

lo cual representa entre 15 y 16 cifras decimales equivalentes. Con respecto al ***rango***, con un exponente de 10 bits el número más grande que se puede representar será del orden de 2 elevado a 2 elevado a 10 (que es 1024):

$$2^{1024} = 1.7977 \cdot 10^{308}$$

Si se considera el caso de los números declarados como ***Double*** (8 bytes) el rango va desde **-1.79769313486232E308** a **-4.94065645841247E-324** para los valores negativos y desde **4.94065645841247E-324** a **1.79769313486232E308** para los valores positivos.

8.4 SISTEMA BINARIO, OCTAL, DECIMAL Y HEXADECIMAL

A contnuación se presentan los primeros números naturales expresados en distintos sistemas de numeración (bases 10, 2, 8 y 16). Para expresar los números en base 16 se utilizan las seis primeras letras del abecedario (A, B, C, D, E y F) para representar los números decimales 10, 11, 12, 13, 14 y 15. La calculadora que se incluye como accesorio en ***Windows 95*** ofrece posibilidades de expresar un número en distintos sistemas de numeración.

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11

Tabla 3.4 Expresión de un número en los distintos sistemas.