1. INTRODUCCIÓN

Visual Basic 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*** es una menor velocidad o eficiencia en las aplicaciones, puesto que para convertir estas instrucciones de alto nivel a bajo nivel lo hace poniendo muchas instrucciones lo que hace que el programa vaya más lento.

Visual Basic 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*** es también un programa *basado en objetos*, aunque no *orientado a objetos* como ***C++*** o ***Java***. La diferencia está en que ***Visual Basic*** utiliza ***objetos*** con ***propiedades*** y

métodos, pero carece de los mecanismos de ***herencia*** y ***polimorfismo,****(No os preocupéis por la terminología* ***, herencia:*** *es diciéndolo de una manera sencilla aprovechar programas realizados para una cosa, para otra cosa que contiene esa cosa. Ejemplo si hacemos un programa que suma, lo podemos utilizar para hacer multiplicaciones.* ***Polimosfimos:*** *es que parte de un programa se puede modificar para hacer otro programa por si mismo, por decirlo de una manera sencilla),*propios de los verdaderos lenguajes orientados a objetos como ***Java*** y *C++*.

En este primer capítulo se presentarán las características generales de ***Visual Basic***, 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. Normalmente, 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. Un ejemplo de este tipo de programas secuenciales son todos los hechos mediante *Free Pascal* en clase.

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*** la hace especialmente sencilla y agradable.

1.2 PROGRAMAS PARA EL ENTORNO WINDOWS

Visual Basic 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*** 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***, 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***. Cada control debe tener un ***nombre*** a través del cual se puede hacer referencia a él en el programa. ***Visual Basic*** 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*** 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*** 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*** 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*** 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
chk	check
cbo	combo
cmd	command button
dir	dir list box
frm	form
fra	frame
lbl	label
mnu	menu
opt	option
pct	picture box
txt	text
tmr	timer
fil	file list box
shp	shape
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***.

Cada vez que se produce un evento sobre un determinado tipo de control, ***Visual Basic*** 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*** da ya resuelta.

1.2.7 Proyectos y ficheros

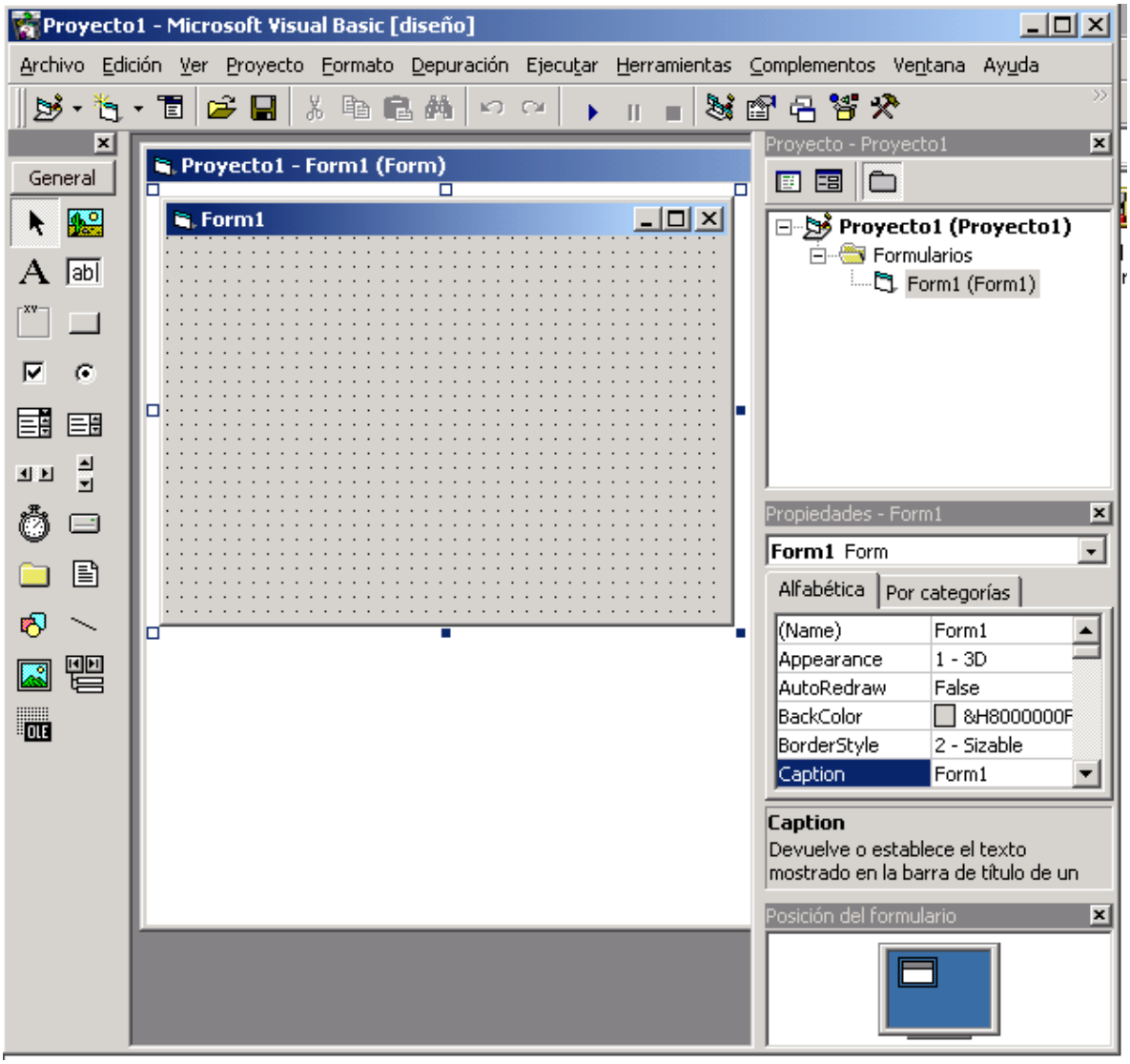
Cada aplicación que se empieza a desarrollar en el entorno de ***Visual Basic*** es un nuevo ***proyecto***. Un proyecto comprende otras componentes más sencillas, como por ejemplo los ***formularios*** (que son las ventanas de la interface de usuario de la nueva aplicación) y los ***módulos*** (que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

Picando 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*** abre cajas de diálogo ***Save As*** por cada uno de los ficheros que hay que guardar.

1.3 EL ENTORNO DE PROGRAMACIÓN VISUAL BASIC

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



- 1. La *barra de títulos*, la *barra de menús* y la *barra de herramientas* de ***Visual Basic*** en modo ***Diseño*** (partesuperior 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*** 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

El ***Help*** de ***Visual Basic*** 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*** 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*** 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*** de la siguiente forma:

1. En el menú ***Herramientas*** elegir el comando ***Opciones***; 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 "***AutoIndent***" (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***.

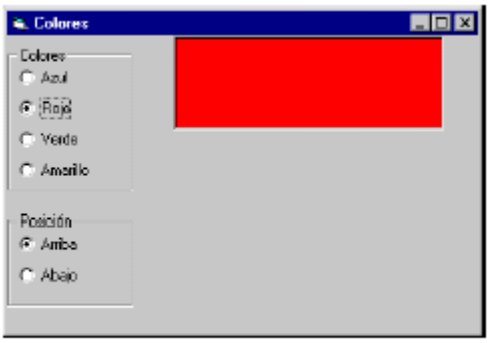


Figura 1.2. Formulario y controles del Ejemplo 1.1.

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

Control Propiedad Valor Control Propiedad Valor

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*** 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***: ***vbRed***, ***vbGreen*** y ***vbBlue*** (lo veremos en capítulos posteriores).
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 ***Herramientas/Opciones***. 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.

2. ENTORNO DE PROGRAMACIÓN VISUAL BASIC

2.1 INTRODUCCIÓN: ¿QUÉ ES VISUAL BASIC?

Visual Basic 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***: menús, opciones, herramientas, propiedades, etc.

2.2 EL ENTORNO DE VISUAL

Visual basic tiene todos los elementos que caracterizan a los programas de ***Windows*** e incluso alguno menos habitual. En cualquier caso, el entorno de ***Visual basic*** es muy lógico y natural, y además se puede obtener una descripción de la mayoría de los elementos picando 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*** 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*** 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***).

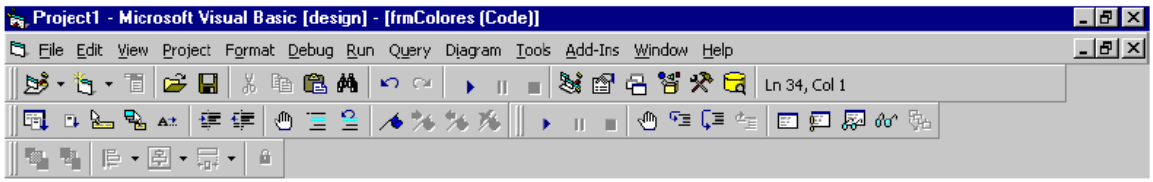


Figura 2.2. Barra de menús y barras de herramientas de Visual Basic

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.

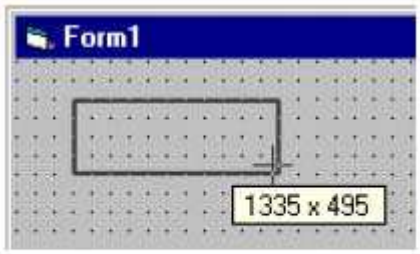


Figura 2.1. Información visual sobre el tamaño de un control.

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 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.

El menú ***Archivo*** 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*** permite tener más de un proyecto abierto simultáneamente, lo cual puede ser útil en ocasiones. Con el comando ***Agregar Proyecto*** se añade un nuevo proyecto en la ventana ***Project Manager***. Con los comandos ***Abrir Proyecto*** o ***Nuevo Proyecto*** se abre o se crea un nuevo proyecto, pero cerrando el o los proyectos que estuvieran abiertos previamente. En este menú está el comando ***Generar NombreProyecto.exe***, que permite crear ejecutables de los proyectos.

Tampoco el menú ***Edición*** aporta cambios importantes sobre lo que es habitual. Por el contrario el menú ***Ver***, generalmente de poca utilidad, es bastante propio de ***Visual basic***. 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 ***Proyecto*** permite añadir distintos tipos de elementos a un proyecto (formularios, módulos, etc.). Con ***Proyectot/Propiedades*** se puede elegir el tipo de proyecto y determinar el formulario con el que se arrancará la aplicación (***Startup Object***). Con el comando ***Componentes*** se pueden añadir nuevos controles a la ***Toolbox*** que aparece a la izquierda de la pantalla.

El menú ***Formato*** contiene opciones para controlar el aspecto de la aplicación (alinear controles, espaciarlos uniformemente, etc.). Los menús ***Depuración*** y ***Ejercutar*** permiten controlar la ejecución de las aplicaciones. Con ***Depuración*** 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 ***Herramientas*** 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 ***Herramientas/Opciones*** 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*** en dos formatos SDI (***Single Document Interface***) y MDI (***Multiple Document Interface***).

Por último, la ayuda (siempre imprescindible y en el caso de ***Visual Basic*** 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).

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.

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

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.

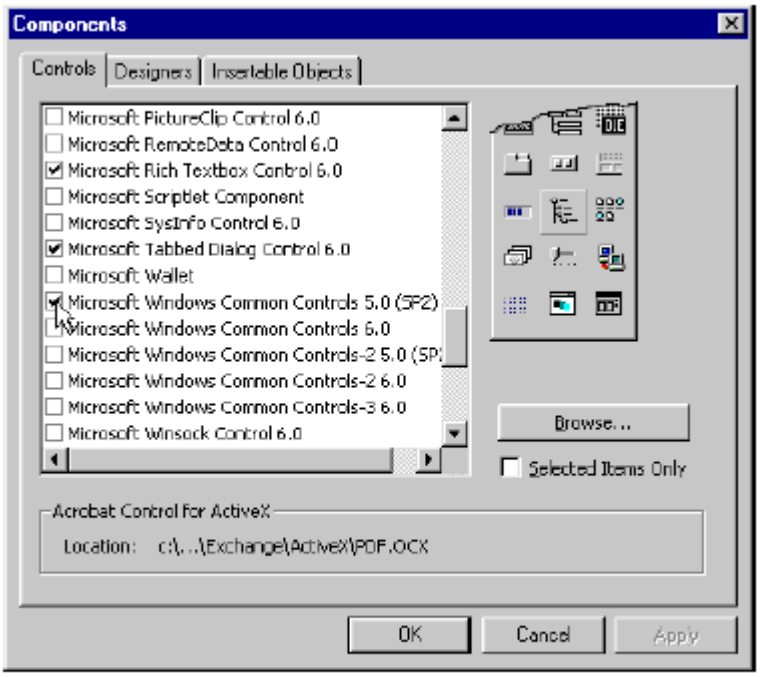


Figura 2.4. Cuadro de diálogo Components.

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.

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 ***Herramientas/Opciones/General***, cambiando la opción ***Forzar controles a la cuadrícula***.

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 ***Proyecto/Propiedades***, en ***Startup Objects***.

Resumiendo, cuando se vaya a crear un programa en ***Visual basic*** 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*)*

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.

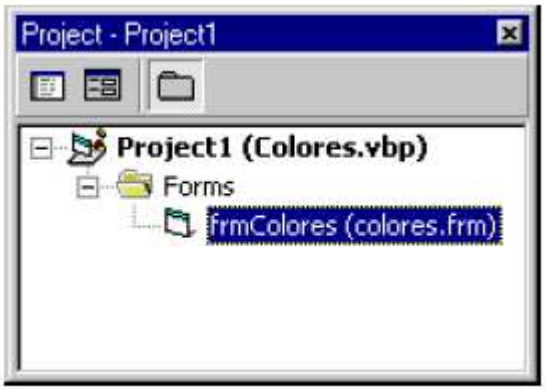


Figura 2.5. Ventana de proyecto.

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

Todos los objetos ***Visual basic*** 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"
```

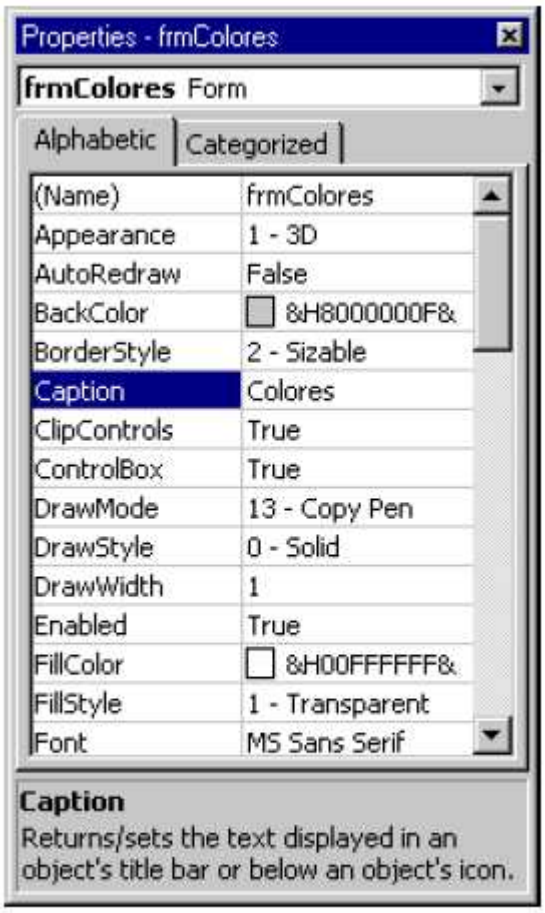


Figura 2.6. Ventana de propiedades.

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.

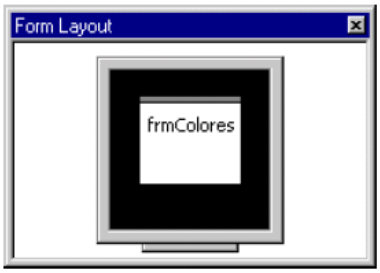


Figura 2.7. Ventana Form Layout.

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***.

Para crear un programa ejecutable se utiliza el comando ***Generar nombreProyecto.exe …*** en el menú ***Archivo.*** 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*** 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*** 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*** y se llama ***Package and Deployement Wizard***.

2.7 CÓMO UTILIZAR EL HELP

Visual basic 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***.

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.***

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, 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*** 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*** 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 ***Herramientas/Opciones/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.

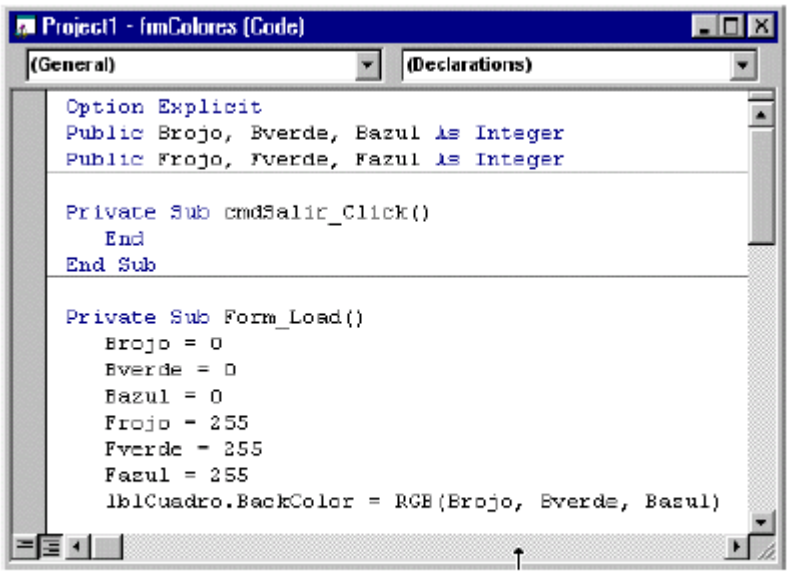


Figura 2.10. Ventana del *Code Editor*.

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 ***Herramientas/Opciones/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 ***Herramientas/Opciones/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***.

Figura 2.11. Inserción automática de propiedades.

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. Tanto la opción ***AutoListMembers*** como la opción ***AutoQuickInfo*** se activan en el cuadro de diálogo que se abre con ***Herramientas/Opciones/Editor.***

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

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 ***Depuración***. 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 ***Depuración***, 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.

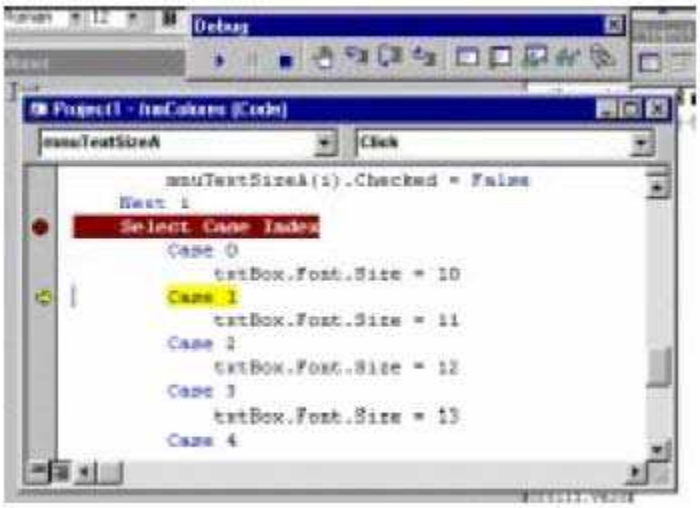


Figura 2.13. Utilización del ***Debugger***.

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***.

2.9.2 Ventanas Immediate, Locals y Watches

El ***Debugger*** de ***Visual basic*** 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.
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 ***Depuración***. 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***).

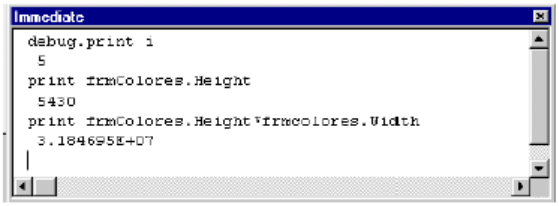


Figura 2.14. Ventana ***Immediate***.

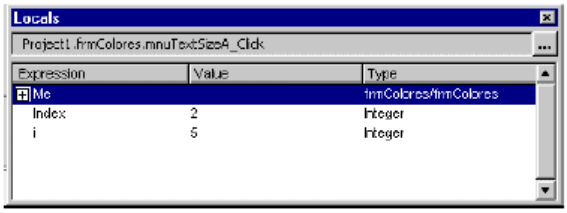


Figura 2.15. Ventana ***Locals***.

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

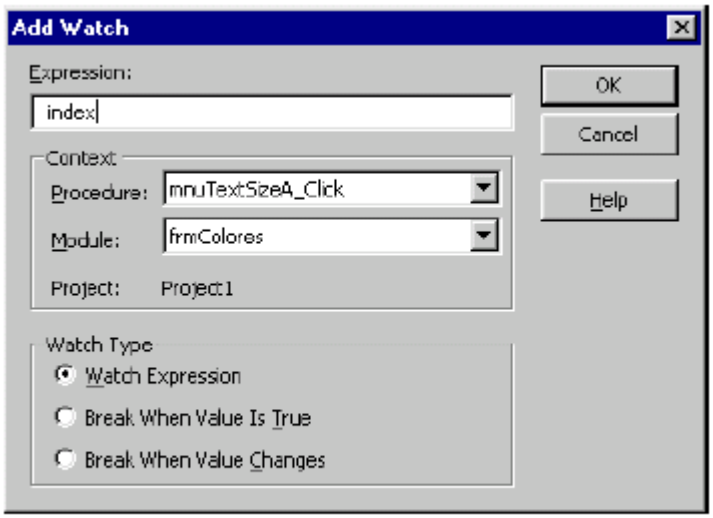


Figura 2.17. Ventana ***Add Watch***.

2.9.3 Otras posibilidades del Debugger

El ***Debugger*** de ***Visual basic*** permite no sólo saber qué sentencia va a ser la próxima en ejecutarse (con ***Depuración/Show Next Statement***), sino también decidir cuál va a ser dicha sentencia (con ***Depuración/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 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 ***Depuración*** ( ). 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*** 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 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*** permite también incluir ***varias sentencias en una misma línea***, aunque nosotros sólo pondremos una sentencia por línea para hacer el programa más legible(entendible). 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*** 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.

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.

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.

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, 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, se explicarán las reglas para elegir nombres y cuáles son las palabras reservadas del lenguaje ***Visual Basic***.

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.

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 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 ***Ver/Examinador de Objetos***.

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*** puede utilizarse el ***Help*** de dicho programa, buscando la referencia ***Palabras Reservadas (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 no distingue entre minúsculas y mayúsculas***. Por tanto, las variables ***LongitudTotal*** y ***longitudtotal*** son consideradas como idénticas (la misma variable). En ***Visual Basic*** 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 determina 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** .

En el lenguaje ***Visual Basic*** 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*** 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*** 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, que lo activaremos nosotros). Cuando no se declaran variables en ***Visual Basic*** lo que hace para saber el tipo de la variable es coger el primer valor que le asignamos a dicha variable y la convierte a ese tipo. 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 ***Herramientas/Opciones***; en el cuadro que se abre se debe poner ***Si*** en la opción ***Requerir Declaración de Variable***. 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

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***.

Veremos la tabla que muestra todos los operadores que soporta ***Visual Basic***:

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*

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*** 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
```

Vemos que es muy parecido a Pascal con la diferencia que en ***Visual Basic*** no se ponen las sentencias begin y end para esta instrucción.

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.

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

En ***Visual Basic*** 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*** 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 ***argumentopasado 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.

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 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*** 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 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 .

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*** 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*** 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***, 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)*, g*rupoB(1), ...* g*rupoB(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* .

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. 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
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.

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***.

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*

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.