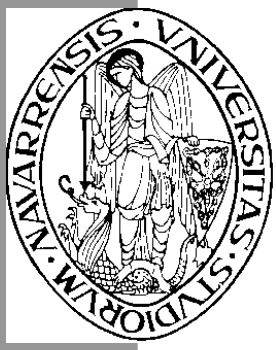
Practique Informática ...

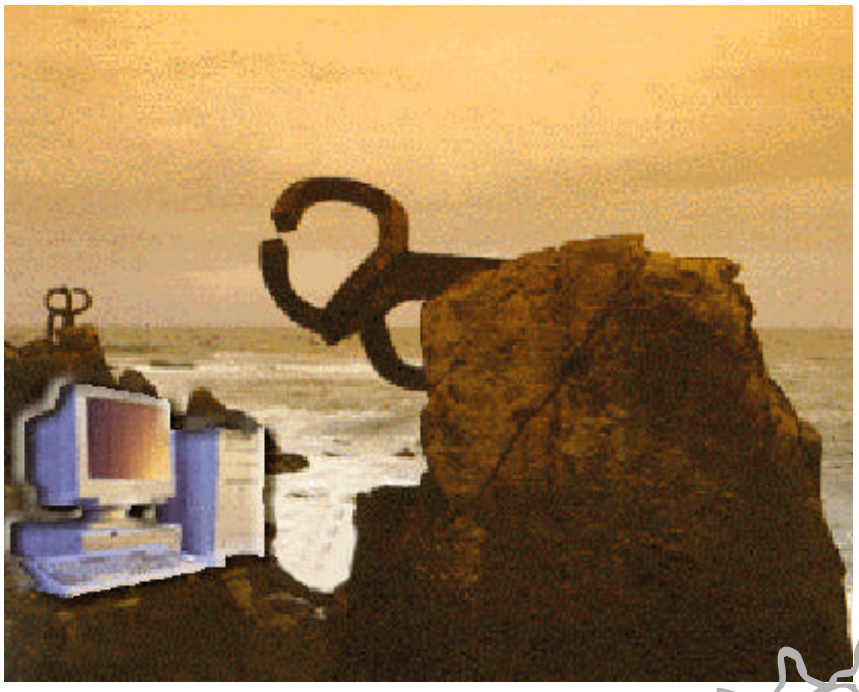


Escuela Superior de Ingenieros Industriales

UNIVERSIDAD DE NAVARRA

Practique Visual Basic 6.0

como si estuviera en primero



Madrid, junio 2003

Javier García de Jalón • José Ignacio Rodríguez • Alfonso Brazález

Practique Visual Basic 6.0

como si estuviera en primero

Javier García de Jalón
José Ignacio Rodríguez
Alfonso Brazález

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

ÍNDICE

Como recomendación general, antes de comenzar cada práctica abre el ***Windows Explorer*** y crea en tu disco un directorio llamado ***Prac01***, ***Prac02***, etc. Por motivos de orden es importante que todos los ficheros de esta práctica se creen dentro de este directorio.. Todos los proyectos deberán estar dentro del directorio de la práctica, en un sub-directorio especial para cada ejercicio.

Mantén abierto el ***Windows Explorer*** y comprueba de vez en cuando que los proyectos de los distintos ejercicios se están guardando correctamente.

0 INTRODUCCIÓN

Este manual recoge los ejercicios de programación en lenguaje Visual Basic 6.0 realizados en las prácticas de la asignatura ***Informática 1***, en el Primer Curso de la Escuela Superior de Ingenieros Industriales de San Sebastián (Universidad de Navarra), desde el curso 1997-98 al curso 1999-2000.

Esta colección de ejercicios nunca llegó a publicarse en Internet. Sin embargo, es lógico considerarla como el complemento imprescindible a los apuntes "Aprenda Visual Basic 6.0 como si estuviera en Primero", que no contienen ejemplos o ejercicios resueltos.

Aunque con cierto retraso, estos ejemplos se publican ahora en formato PDF, esperando que ayuden a aprender a programar a muchos estudiantes o simples aficionados a la informática.

Los distintos ejercicios están agrupados en "prácticas". De cada uno de ellos se incluye:

- Un enunciado que describe el programa a realizar, tal como se planteaba a los alumnos.
- El programa correspondiente al ejercicio resuelto.
- Unos breves comentarios sobre los aspectos del ejercicio resuelto a los que convenga prestar más atención.

A lo largo de estas páginas se utilizan con frecuencia las unidades de disco ***Q:*** y ***G:***, y los directorios ***Q:\Infor1\Prac%***% y ***G:\Infor1\Prac%%***. El disco ***Q:*** de la red Novell de la ESII de San Sebastián era un disco compartido, visible desde todos los ordenadores de la red, en el que los profesores ponían los ficheros de sólo lectura que querían compartir con los alumnos. El disco ***G:*** era una partición del servidor propia de cada alumno que se establecía como tal unidad cuando el alumno iniciaba sesión desde cualquier ordenador de la red. Las prácticas se recogían automáticamente a partir de estas particiones propias de cada alumno.

Para facilitar la tarea a los usuarios de esta colección de ejercicios se facilita un directorio llamado ***programas*** en el que se incluyen los ficheros correspondientes a todos los ejercicios resueltos, de forma que el lector no necesite teclear o escanear ningún programa. Es posible que alguno de los programas incluidos contenga algún error; se agradecerá recibir noticia de ello para corregirlo y facilitar el trabajo a los futuros lectores..

Madrid, junio de 2003

Javier García de Jalón de la Fuente (jgjalon@etsii.upm.es)

1 PRIMERA PRÁCTICA

1.1 INTRODUCCIÓN

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

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

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

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

1.2 PROGRAMAS SECUENCIALES, INTERACTIVOS Y ORIENTADOS A EVENTOS

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

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

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

1.3 PROGRAMAS PARA EL ENTORNO WINDOWS

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

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

1.3.1 Modo de Diseño y Modo de Ejecución

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

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

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

1.3.2 Formularios y Controles

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

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

1.3.3 Objetos y Propiedades

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

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

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

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

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

1.3.4 Nombres de objetos

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

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

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

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

Tabla 1.1. Abreviaturas para los controles más usuales.

1.3.5 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/XP*** y por ***Visual Basic 6.0***.

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

1.3.6 Métodos

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

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

1.3.7 Proyectos y ficheros

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

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

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

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

1.4 EL ENTORNO DE PROGRAMACIÓN VISUAL BASIC 6.0

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

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

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

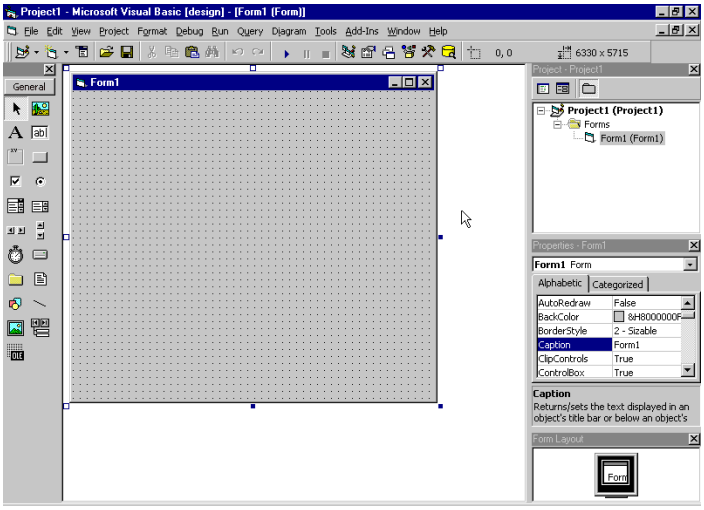


Figura 1.1. Entorno de programación de ***Visual Basic 6.0***.

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

1.5 EL HELP DE VISUAL BASIC 6.0

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

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

1.6 EJEMPLOS

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

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

1.6.1 Ejemplo 1.1: Sencillo programa de colores y posiciones

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

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

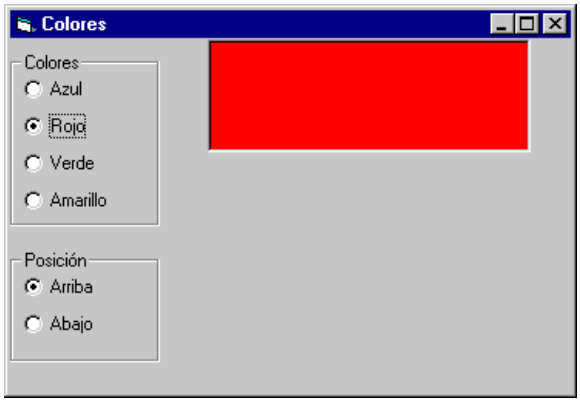


Figura 1.2. Formulario y controles del Ejemplo 1.1.

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

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

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

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

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

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

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

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

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

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

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

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

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

3. Es también interesante ver cómo se accede desde programa a la propiedad ***backColor*** de la caja de texto que se llama ***txtCaja***: se hace utilizando el punto en la forma ***txtCaja.BackColor***. Los colores se podrían también introducir con notación hexadecimal (comenzando con &H, seguidos por dos dígitos entre 00 y FF (es decir, entre 0 y 255 en base 10) para los tres colores fundamentales, es decir para el ***Red***, ***Green*** y ***Blue*** (RGB), de derecha a izquierda. Aquí se han utilizado las constantes simbólicas predefinidas en ***Visual Basic 6.0***: ***vbRed***, ***vbGreen*** y ***vbBlue*** (ver página 70, en ***Aprenda Visual Basic como*** ...).

4. Recuérdese que si se desea que el código de todos los eventos aparezca en una misma ventana hay que activar la opción ***Default to Full Module View*** en la solapa ***Editor*** del comando ***Tools/Options***. También puede hacerse directamente en la ventana de código con uno de los botones que aparecen en la parte inferior izquierda ().

5. ***Es muy importante*** crear primero el control ***frame*** y después, estando seleccionado, colocar los ***botones de opción*** en su interior. No sirve hacerlo a la inversa. ***Visual Basic*** supone que todos los botones de opción que están dentro del mismo ***frame*** forman parte del mismo grupo y sólo permite que uno esté seleccionado.

1.6.2 Ejemplo 1.2: Minicalculadora elemental

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

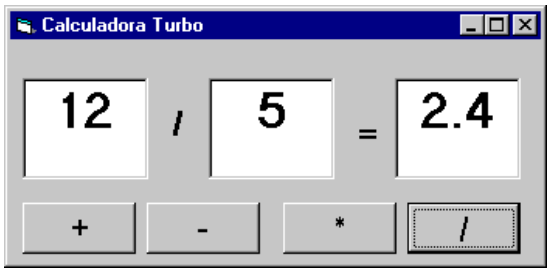


Figura 1.3. Minicalculadora elemental.

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

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

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

Tabla 1.3. Objetos y propiedades del ejemplo Minicalculadora.

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

```
Option Explicit

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

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

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

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

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

1.6.3 Ejemplo 1.3: Transformación de unidades de temperatura

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

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

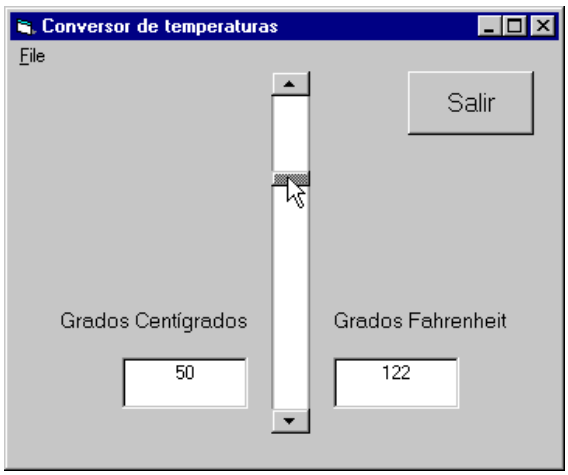


Figura 1.4. Equivalencia de temperaturas.

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

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

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

Tabla 1.4. Controles y propiedades del Ejemplo 1.3.

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

```
Option Explicit

Private Sub cmbSalir_Click()
   Beep
   End
End Sub

Private Sub mnuFileExit_Click()
   End
End Sub

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

Sobre este tercer ejemplo se puede comentar lo siguiente:

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

1.6.4 Ejemplo 1.4: Colores RGB

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

Figura 1.5. Colores de fondo y de texto.

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

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

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

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

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

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

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

Private Sub cmdSalir_Click()
   End
End Sub

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

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

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

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

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

2. El color ***blanco*** se obtiene con los tres colores fundamentales a su máxima intensidad. El color ***negro*** se obtiene con los tres colores RGB a cero. También se pueden introducir con las constantes predefinidas ***vbWhite*** y ***vbBlack***, respectivamente.

3. Es importante disponer de unas ***variables globales*** que almacenen los colores del fondo y del texto, y que permitan tanto guardar los valores anteriores de las barras como cambiar éstas a sus nuevos valores cuando se clica en los botones de opción. Las variables globales, definidas en la parte de definiciones generales del código, fuera de cualquier procedimiento, son visibles desde cualquier parte del programa. Las variables definidas dentro de una función o procedimiento sólo son visibles desde dentro de dicha función o procedimiento (*variables locales*).

4. La función ***hsbColor_Change(Index As Integer)*** se activa cada vez que se cambia el valor en una cualquiera de las barras de desplazamiento. El argumento ***Index***, que ***Visual Basic*** define automáticamente, indica cuál de las barras del array es la que ha cambiado de valor (la 0, la 1 ó la 2). En este ejemplo dicho argumento no se ha utilizado, pero está disponible por si se hubiera querido utilizar en el código.

2 SEGUNDA PRÁCTICA

2.1 EJERCICIO 1. EVENTOS EN FORMULARIOS.

Este ejercicio está descrito en el Apartado 4.1.1.1 (página 48 y siguientes) de la última edición *"Aprenda Visual Basic 6.0 como si estuviera en Primero"*. Aquí se recuerdan algunos puntos de dicho Apartado.

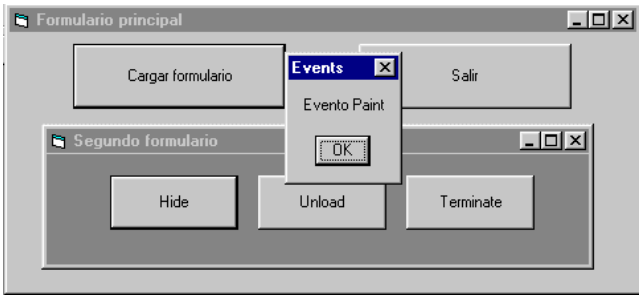


Figura 2.1. Eventos en formularios.

Cuando se arranca una aplicación, o más en concreto cuando se visualiza por primera vez un formulario se producen varios eventos consecutivos: ***Initialize***, ***Load***, ***Activate*** y ***Paint***. Cada uno de estos eventos se puede aprovechar para realizar ciertas operaciones por medio de la función correspondiente.

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

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

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

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

Todo esto se puede ver y entender con un simple ejemplo, mostrado en la Figura 2.1 Se han de crear dos formularios (***frmPrincipal*** y ***frmSecundario***). El primero de ellos contendrá dos botones (***cmdCargar*** y ***cmdSalir***) y el segundo tres (***cmdHide***, ***cmdUnload*** y ***cmdTerminate***). El formulario principal será el primero que aparece, y sólo se verá el segundo si se clica en el botón ***Cargar Formulario***. Cuando así se haga, a medida que los eventos antes mencionados se vayan sucediendo, irán apareciendo en pantalla unas cajas de mensajes que tendrán como texto el nombre del evento que se acaba de producir. Según con cual de los tres botones se haga desaparecer el segundo formu-

lario, al volverlo a ver se producirán unos eventos u otros, según se puede ver por los mensajes que van apareciendo con cada evento.

El fichero ejecutable de este ejercicio está en el directorio ***Q:\Infor1\Prac02\Ejer1*** y se llama ***Eventos.exe***. El resultado de este ejercicio constará de los ficheros ***Events.vbp***, ***Principal.frm***, ***Secundario.frm***. Se guardará en el directorio ***G:\Infor1\Prac02\Ejer1***. El código de este ejemplo es:

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

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

Private Sub cmdUnload_Click()
    Unload Me
End Sub

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

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

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

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

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

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

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

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

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

2.2 EJERCICIO 2. LISTAS (EXAMEN DE SEPTIEMBRE 1997)

En este ejercicio se trata de incluir en un *formulario* una *caja de texto* para introducir cantidades numéricas y una *lista*[1] en la que se podrán ir guardando las cantidades introducidas (ver Figura 2.2 n la parte inferior de la página). Se deberán introducir los 5 botones siguientes:

1. Un botón que permita añadir números de la caja de texto a la lista (***Add***),
2. Un botón que borre el último número añadido a lista (***Delete Last***),
3. Un botón que borre todos los elementos de la lista (***Delete All***),
4. Un cuarto botón que calcule la ***media m*** y la ***desviación típica*** *s* (***Compute***)**.** La *media* ***m*** es la suma de los elementos incluidos en la lista hasta ese momento dividida por el número de elementos. La *desviación típica* *s* es la raíz cuadrada de la suma de los cuadrados de cada elemento menos el valor medio, dividida por el número de elementos menos uno. Ambas magnitudes se calculan según las fórmulas siguientes:

$$m = \frac{\sum_i x_i}{n} \qquad s = \sqrt{\frac{\sum_i (x_i - m)^2}{n-1}}$$

5. Un último botón para salir de la aplicación (***Exit***).

Para pasar a la ***lista*** los números tecleados en la ***caja de texto*** se deberá poder utilizar también la tecla ***Intro***. Se chequeará que lo tecleado en la caja de texto corresponde a una cantidad numérica, enviándose un mensaje al usuario en caso contrario para que vuelva a teclear el valor en la caja de texto (se puede utilizar la función ***IsNumeric(texto)*** que permite chequear si un texto representa o no un valor numérico).

Es importante que la caja de texto tenga el ***focus*** en todo momento, de modo que se puedan seguir metiendo datos con la máxima facilidad (se puede utilizar para ello el método ***SetFocus***).

El ejecutable ***lista.exe*** indica cómo debe funcionar exactamente la aplicación a desarrollar. Dicho programa está en el directorio ***Q:\Infor1\Prac02\Ejer2***.

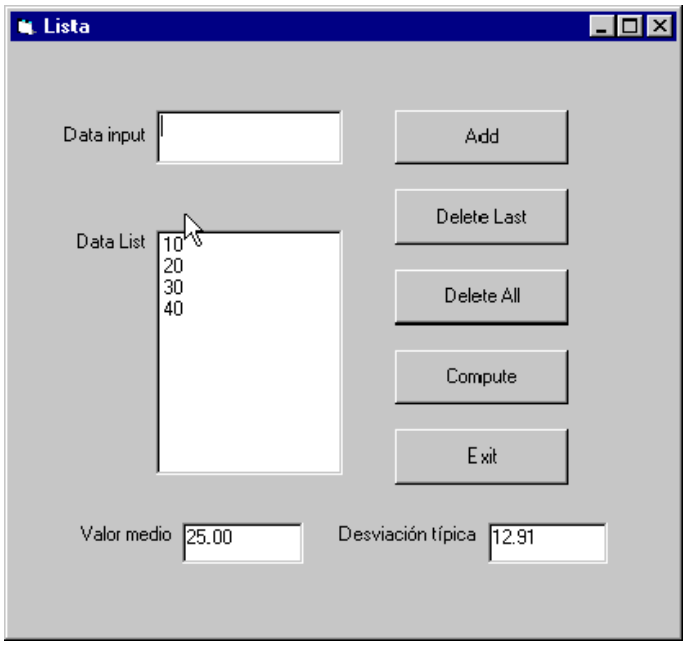


Figura 2.2. Ejemplo de lista.

Obsérvese por ejemplo que al añadir o quitar un elemento a la lista se borran los resultados de los cálculos que se hayan hecho antes (ya no tienen sentido, pues han cambiado los datos con los que se habían calculado).

El resultado de este ejercicio se guardará en el directorio ***G:\Infor1\Prac02\Ejer2***.

---

[1] Como ayuda respecto a las listas se recuerda que ***ListCount*** es una propiedad que indica el número de elementos de una lista (numerados desde 0) y que ***List(I)*** es la propiedad que permite acceder al elemento ***I*** de la lista.

2.3 EJERCICIO 3. OPERACIONES CON NÚMEROS

Este ejercicio es bastante similar al anterior, aunque hay también diferencias significativas. En el directorio ***Q:\Infor1\Prac02\Ejer3*** se encuentra un fichero ejecutable llamado ***Numeros.exe*** que se corresponde con la solución del programa que hay que realizar en este ejercicio.

Al ejecutar ***Numeros.exe***, tras introducir algunos números enteros, se obtiene una figura similar a la mostrada. Este programa permite añadir números a la lista, así como utilizar botones para borrar el último número introducido o para borrar todos los números de la lista.

Desde el punto de vista de cálculo, este programa permite ordenar los números de menor a mayor (***sort***), así como calcular el máximo, el mínimo y la suma de los números introducidos hasta ese momento. En este caso, el programa estará preparado solamente para trabajar con números enteros, aunque es fácil modificarlo para trabajar con cualquier clase de números.

La caja de texto que sirve para introducir los números tiene el ***focus*** en todo momento, de modo que se puede empezar a teclear otro número en cualquier momento.

Cuando cambia la lista, es decir cuando se introduce o se borra algún número, los resultados de las cajas de texto que muestran el máximo, el mínimo y la suma se ponen en blanco ya que esos resultados no tienen sentido si la lista ya no es la misma que la que sirvió para calcular.

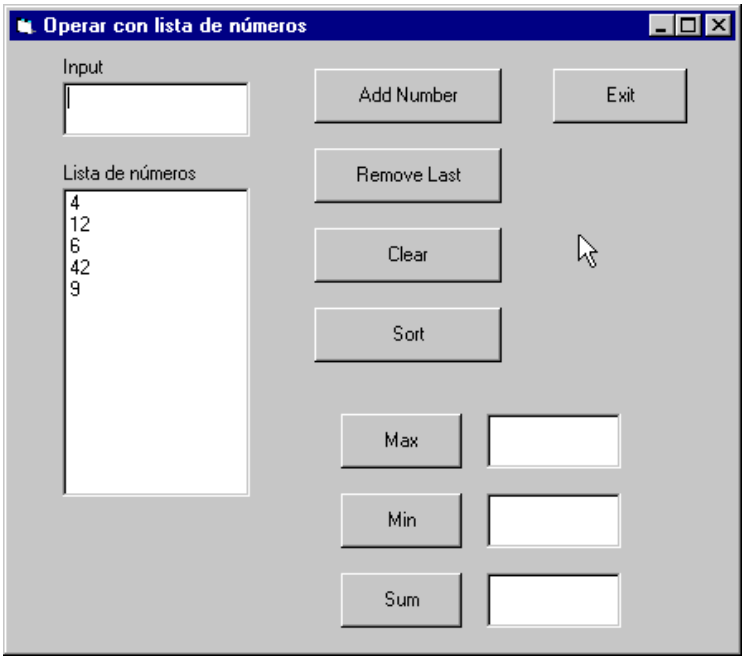


Figura 2.3. Ejemplo de lista con números.

Para hacer este ejercicio hay que programar un procedimiento llamado ***ordenar( )*** que ordena lor elementos de un vector (array) de enteros. También hay que programar dos funciones llamadas ***maximum( )*** y ***minimum( )*** que calculan el máximo y el mínimo elemento de un vector, respectivamente. En los tres casos hay dos argumentos: el vector de enteros y el número de elementos de dicho vector. Estos tres programas se deben realizar en un módulo distinto del formulario de la aplicación. Este módulo se llamará ***Numeros.bas***.

Así pues, en este ejercicio se deberán guardar tres ficheros: el proyecto (***Numeros.vbp***), el formulario (***Numeros.frm***) y el módulo de código (***Numeros.bas***). Deberán ser guardados en el directorio ***G:\Infor1\Prac02\Ejer3***.

3 TERCERA PRÁCTICA

3.1 EJERCICIO 1. APLICACIÓN CON DIVERSOS CONTROLES

Este ejemplo tiene un formulario principal y cuatro formularios secundarios, contenidos respectivamente en ficheros llamados ***main.frm***, ***semaforo.frm***, ***label.frm***, ***check,frm*** y ***options.frm***. El formulario principal se muestra en la Figura 3.1. Los formularios secundarios aparecen al clicar en cada uno de los botones del formulario principal o, alternativamente, al elegir la opción correspondiente en el menú ***Options*** (Habrá pues dos formas de hacer lo mismo: con los botones o con el menú).

Los restantes formularios aparecen en las Figuras 3.2, 3.3, 3.4 y 3.5.

Figura 3.1. Formulario principal.

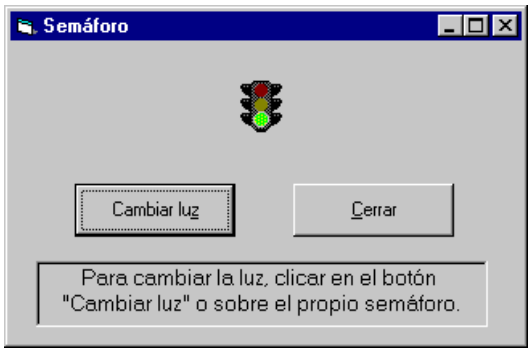


Figura 3.2. Formulario ***semaforo.frm***.

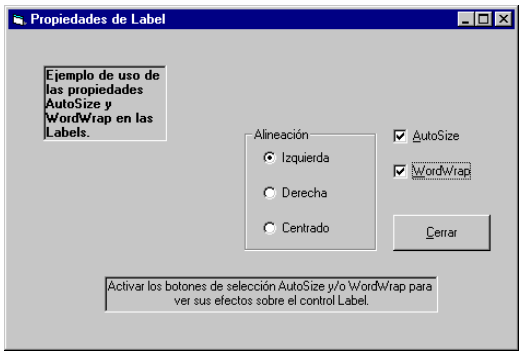


Figura 3.3. Formulario ***label.frm***.

Figura 3.4. Formulario ***check.frm***.

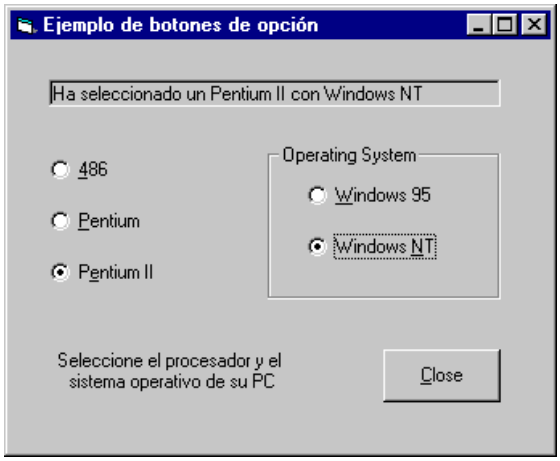


Figura 3.5. Formulario ***options.frm***.

Como este ejemplo es un poco más complejo que los hechos anteriormente, es muy importante ver cómo funciona el ejecutable de este programa, que se llama ***Controles.exe*** y está en el directorio ***Q:\Infor1\Prac03\Ejer1***.

Conviene ir haciendo este ejercicio poco a poco: primero se hace el formulario principal y luego se irán creando los otros cuatro formularios, pero de modo que no se pasa al siguiente hasta que no haya funcionado correctamente el anterior. Se sugiere ir haciendo los formularios en el siguiente orden (de más fácil a más difícil).

1. El formulario ***frmCheck*** (fichero ***check.frm***) contiene una caja de texto con la propiedad ***Text*** a *"Unas palabras tan sólo..."*. Con dos botones de selección se trata de cambiar las propiedades ***FontBold*** y ***FontItalic*** de dicha caja de texto. Se utilizará el evento ***Click*** de los botones de selección.
2. El formulario ***frmOptions*** (fichero ***options.frm***) permite escribir una frase en un control ***Label*** (propiedad ***Caption***) en la que aparece el tipo de PC seleccionado y el sistema operativo. Para esto se puede utilizar el operador de concatenación (&) o (+). Obsérvese que los botones de opción del sistema operativo están dentro de un ***frame*** (para formar un *grupo*), mientras que los del tipo de PC están directamente en el formulario, con lo que también forman *grupo*.
3. El formulario ***frmLabel*** (fichero ***label.frm***) se controlan las propiedades ***AutoSize*** y ***WordWrap*** de la ***Label***. La primera hace que el tamaño de la ***Label*** se adapte al de la propiedad ***Caption***, mientras que ***WordWrap*** permite que el texto de ***Caption*** ocupe varias líneas, si es preciso.
4. El formulario ***frmSemaforo*** (fichero ***semaforo.frm***) permite cambiar el color de la luz. Para ello se crearán tres controles ***Image*** en los que se cargarán los iconos ***imgGreen.ico***, ***imgYellow.ico*** y ***imgRed.ico***, que se entregan también en el directorio ***Q:\Infor1\Prac03***. Estos tres controles tendrán las mismas propiedades ***Left***, ***Top***, ***Height*** y ***Width***, de modo que estarán superpuestos. Al clicar sobre el botón ***Cambiar Luz*** o sobre el propio semáforo se hará invisible el control que se esté viendo y se hará visible el siguiente.

Los formularios de este proyecto tienen también algunos controles ***Label*** puramente explicativos de la función del propio formulario. Los ficheros de este proyecto pueden ser guardados por ejemplo en el directorio ***G:\Infor1\Prac03\Ejer1***.

3.2 EJERCICIO 2: MOVIMIENTO OSCILATORIO SINUSOIDAL.

Este ejercicio consiste en crear un programa en ***Visual Basic 6.0*** que permita mover un objeto dentro de una caja de dibujo con un movimiento oscilatorio sinusoidal.

Para realizar correctamente este ejercicio de la práctica es conveniente estudiar muy bien el ejecutable ***Oscila.exe*** que se encuentra en el directorio ***Q:\Infor1\Prac03\Ejer2*** y seguir los pasos que se indican a continuación; de lo contrario el alumno puede sentirse desorientado y desbordado por la cantidad de comandos a programar.

Se debe observar el funcionamiento del programa ejecutable, viendo la función que realiza cada uno de los botones, y ***pensar*** en cada caso el cometido que está realizando cada uno de ellos. Cuanto mejor se entienda el funcionamiento de los botones, menos tiempo se requerirá para la generación del código.

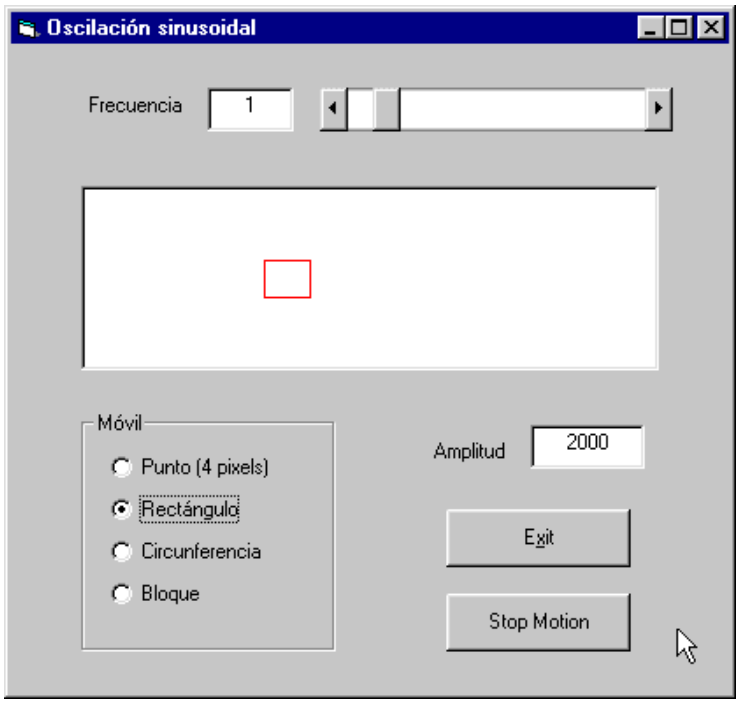


Figura 3.6. Formulario del programa ***Oscila***.

Los pasos a dar para la realización de este ejercicio podrían ser los siguientes:

1. Poner el nombre ***frmOscila*** al formulario, y cambiar la propiedad ***Caption*** adecuadamente. Clicar en el botón ***Save*** y guardar los ficheros ***Oscila.vbp*** y ***Oscila.frm*** en el directorio ***G:\Infor1\Prac03\Ejer2***.

2. Añadir al formulario un objeto ***Timer,*** de nombre ***Timer1*** (es el nombre por defecto). Un objeto de este tipo no es visible en tiempo de ejecución, pero permite definir ***eventos*** con una periodicidad determinada. Es decir, un ***Timer*** es un objeto que entra en una función cada cierto número de ***milisegundos***, según está definido en una de sus propiedades (***Interval***). La función que se ejecuta periódicamente se llama ***Timer1_timer***. De momento no es necesario programarla. Para que un control ***Timer*** esté activo, su propiedad ***Enabled*** debe estar a ***True***.

3. Crear los botones de empezar/parar el movimiento y salir del programa, llamados ***cmdStart*** y ***cmdExit***, con el código correspondiente. El botón ***cmdStart*** debe cambiar su propiedad ***Caption*** en función del texto que tenga en cada caso. Es decir si la propiedad ***Caption*** es ***Start Motion***, y se pulsa el botón, ésta debe cambiarse a ***Stop Motion***, y viceversa. Además en cada caso se debe modificar la propiedad ***Enabled*** del ***Timer***.

4. Crear una caja de dibujo de nombre ***pctBox***, y cambiarle el color de fondo a blanco.

5. Crear el bloque mediante una etiqueta de nombre ***lblBloque*** y con la propiedad ***Caption*** vacía.

6. Crear la etiqueta de la amplitud del movimiento y la caja de texto con los nombres ***lblAmplitud*** y ***txtAmplitud*** respectivamente. La amplitud del movimiento se debe guardar en una variable global (***Xa***), de forma que se pueda acceder a ella desde cualquier función. Para crear una variable global, basta con definirla fuera de todas las funciones, en la sección de ***general***.

7. Otra variable global ***Ind*** indicará cuál es el botón de opción elegido para representar el móvil (un punto, un rectángulo, una circunferencia o un control *label*). Los botones de opción correspondientes forman un array, es decir todos tienen el mismo nombre (por ejemplo, ***optMovil***) y se diferencian en la propiedad ***Index***, que varia de 0 a 3. El evento ***click*** en estos botones dará valor a la variable ***Ind***, y el programa de dibujo (en el evento ***Timer*** del control ***Timer1***) dibujará una cosa u otra según el valor de esta variable.

8. Se deben crear también otras variables globales de interés para el ***Timer***: ***Frec*** (determina la frecuencia del movimiento, es decir las oscilaciones por segundo). ***Pi*** (es la constante *p*). ***Ti*** (el intervalo en milisegundos entre dos eventos del ***Timer***) y ***t*** (el tiempo total transcurrido). La amplitud puede obedecer a la siguiente expresión:

$$X = Xa * \sin(2p * frec * t)$$

9. Programar las opciones y la inicialización de variables en la carga del formulario:

```
Private Sub Form_Load()
    Xa = 2000
    txtAmplitud.Text = Xa
    cmdStart.Caption = "Stop Motion"
    frec = 1
    txtFrec.Text = 1
    Pi = 3.141592654 ' Pi=4*Atn(1)
    Ti = 40 ' Son milisegundos, es decir 25 imagenes por segundo
    Timer1.Interval = Ti
    pctBox.DrawWidth = 4
    lblBloque.Visible = False
    ' fijar las coordenadas en la picture box
    pctBox.Scale (-2500, 100)-(2500, -100)
End Sub
```

10. Programar la función ***Timer1_timer()*** del control ***Timer*** para que dibuje un móvil desplazándose a lo largo de la caja de dibujo.

```
Private Sub Timer1_Timer()
    X = Xa * Sin(2 * Pi * frec * t)
    pctBox.Cls
    Select Case Ind
        Case 0
            pctBox.PSet (X, 0), vbBlue
        Case 1
            pctBox.Line (X - 200, 20)-(X + 200, -20), vbRed, B
        Case 2
            pctBox.Circle (X, 0), 200, vbGreen
        Case 3
            lblBloque.Left = X - lblBloque.Width / 2
    End Select
    t = t + Ti / 1000    ' incrementar el tiempo
End Sub
```

11. Crear el frame (***fraMovil***) de los botones de opción que definen el tipo de objeto que se mueve, y los cuatro botones de selección. Estos últimos se deben crear con un array (***optMovil***). Programar la función de ***click***.

Los ficheros (***.vbp*** y ***.frm***) de este proyecto se pueden guardar en el directorio ***G:\Infor1\Prac03\Ejer2***.

3.3 EJERCICIO 3: LANZAMIENTO PARABÓLICO CON OBSTÁCULO.

En este ejercicio se pide realizar un programa que sea capaz de representar gráficamente el lanzamiento de una pelota de tenis en un plano vertical, intentando acertar con dicho lanzamiento a un círculo que se encuentra en el otro extremo. Se trata de un ***tiro parabólico*** en un plano**.**

Es muy poco probable que te dé tiempo a hacer este ejercicio en el tiempo disponible para la práctica: no dejes sin embargo de intentarlo en cuanto te sea posible.

El ejecutable modelo de este ejercicio se llama ***Lanzador.exe*** y está en el directorio ***Q:\Infor1\Prac03\Ejer3***. Ejecútalo y obsérvalo con detenimiento.

La trayectoria del móvil está definida por su posición y velocidad iniciales, tal y como se expresa en las siguientes ecuaciones:

$$\begin{cases} x = x_0 + v_{xo}t + \dfrac{1}{2}a_x t^2 \\ y = y_0 + v_{yo}t + \dfrac{1}{2}a_y t^2 \end{cases} \qquad (1)$$

donde x_o e y_o determinan la situación inicial del móvil, v_{xo} y v_{yo} son las componentes iniciales de la velocidad en los ejes cartesianos y ***t*** el tiempo. En el caso que se presenta, no existe aceleración según el ***eje x*** por lo que $a_x = 0$. La aceleración según ***y*** es la de la gravedad $a_y = -g$. Si se toma como ***origen de coordenadas el extremo desde donde parte el tiro*** *($x_o = 0, y_o = 0$)* se tiene:

$$\begin{cases} x(t) = v_{xo}t \\ y(t) = v_{yo}t - \dfrac{1}{2}gt^2 \end{cases} \qquad \text{con g} = 9.81 \qquad (2)$$

El movimiento de la pelota queda determinado según las fórmulas (2). Por lo tanto con conocer los valores iniciales de v_{xo} y de v_{yo}, o lo que es lo mismo, a partir del módulo de la velocidad ***v*** y su ***ángulo*** de salida (entre 0 y 90 grados) es suficiente para calcular ***x(t)*** e ***y(t)***. En este caso se deben introducir los valores del módulo ***v*** y el ***ángulo*** (en grados) en sendas cajas de texto. Se recuerda que los valores que se deben utilizar como argumentos en las funciones ***sin()*** y ***cos()***deben estar en ***radianes***.

Entre el punto desde donde se realiza el lanzamiento y el objetivo existe un obstáculo de anchura ***w*** y altura ***h*** situado a una distancia ***xPos*** del lugar de lanzamiento tal y como se aprecia en la Figura 3.7. Estos tres valores deben poder ser modificados mediante ***barras de desplazamiento***, tal como se muestra en el ejecutable. Si la pelota choca con el obstáculo, deberá aparecer un mensaje comunicando dicho "percance" y se deberá detener la trayectoria. A su vez si la pelota impacta sobre el círculo se deberá mostrar otro mensaje indicando que se ha acertado el tiro. Si el tiro no acierta o sale de los límites de la ventana no se emite mensaje ni se para la ejecución.

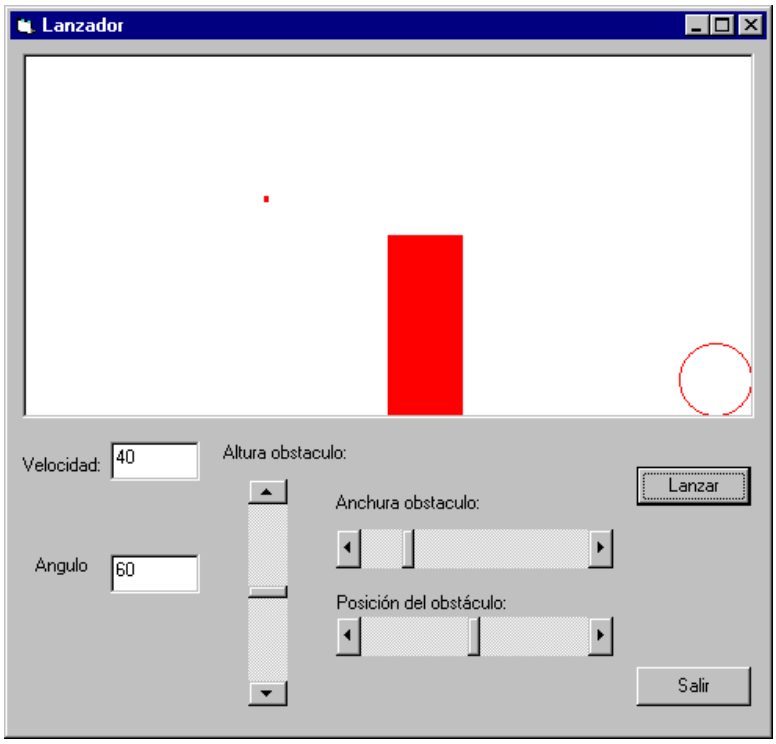


Figura 3.7. Tiro parabólico sobre un obstáculo.

Al ejecutar el programa aparecen dos botones, uno (***Lanzar***) que realiza el lanzamiento, y otro (***Salir***) que finaliza el programa. En cualquier momento que se pulse el botón ***Lanzar***, comenzará un nuevo lanzamiento, deteniéndose el anterior si aún no hubiera terminado. Durante un lanzamiento no es necesario cambiar los valores de la velocidad aunque se varíen los valores en las cajas de texto. Los nuevos valores serán asignados como velocidad inicial cuando se pulse el botón ***Lanzar***. La posición, altura y anchura del obstáculo sí se deberán poder variar durante el lanzamiento.

Metodología de programación:

Se van a enumerar aquí las etapas en las que se puede realizar este ejercicio para que su crecimiento sea sencillo, lógico y progresivo. Es importante darse cuenta de que ***es inútil pasar a una etapa posterior si no se han resuelto satisfactoriamente las anteriores***.

1. Entender completamente el funcionamiento del programa ejecutable ***Lanzador.exe***.
2. Pensar y decidir cómo se va a resolver el problema.
3. Escribir en un papel, aunque sea esquemáticamente, cómo se van a resolver las operaciones clave (en este caso la función asociada con el botón ***Lanzar***, el tamaño y posición del obstáculo, etc.).
4. Colocar el ***recinto del dibujo***, los dos ***botones***, las ***cajas de texto*** con sus ***etiquetas*** y el ***Timer*** en el formulario principal. Para evitar distorsiones al dibujar el círculo, es importante que el recinto se cree en la etapa de diseño con la anchura el doble que la altura. Se recomienda utilizar ***Height = 3500*** y ***Width = 7000***. Al ejecutar el programa se deberá realizar un cambio de escala colocando el origen en la esquina inferior izquierda, con el eje de ordenadas hacia arriba, y quedando unas dimensiones para el recinto de: anchura ***200m*** y altura ***100m***. Se aconseja un intervalo de ***20 milisegundos*** para el ***Timer***.
5. ***Guardar muy cuidadosamente*** el proyecto y el formulario. Se guardarán en el directorio ***G:\Infor1\Prac03\Ejer3***. El proyecto llamará ***Lanzador.vbp*** y el formulario ***Lanzador.frm***.

6. Establecer las condiciones iniciales del movimiento, es decir hacer ***x = 0*** e ***y = 0*** y dar un valor por defecto a la velocidad y al ángulo. Se recuerda que cada vez que se pulse ***Lanzar*** se deberán asignar nuevamente estos valores.
7. Programar los ***botones*** y comprobar que funcionan correctamente. La ***circunferencia*** (la diana) debe dibujarse con un ***grosor de 1 pixel*** mientras que la ***pelota*** será dibujada como un punto con un ***grosor de 3 pixels***. El obstáculo será un rectángulo construido a partir de ***line*** con la opción adecuada para que dibuje realmente un rectángulo.

Una vez que se consiga lo anterior, se puede pasar a concretar la programación de la variación de dimensiones del obstáculo introduciendo las ***barras de desplazamiento***. En la realización de las barras de desplazamiento se aconseja lo siguiente:

1. ***Altura*** del obstáculo: Máximo = 100, Mínimo = 1 e incrementos de 1-10.
2. ***Anchura*** del obstáculo: Máximo = 100, Mínimo = 1 e incrementos de 1-10.
3. ***Posición*** del obstáculo: Máximo = 150, Mínimo = 50 e incrementos de 1-10.
4. Se recuerda que al variar cualquiera de las barras, el dibujo debe variar en ese momento, y el cálculo deberá tener el cuenta las nuevas dimensiones para detectar posibles colisiones.

Para calcular la colisión con el obstáculo basta comprobar si ***x*** está entre ***xpos*** y ***xpos+w*** al mismo tiempo que ***y*** está entre ***0*** y ***h***. Para calcular el acierto en el blanco hay que comprobar si la distancia entre la pelota y el centro de la circunferencia es menor que el radio.

4 CUARTA PRÁCTICA

4.1 EJERCICIO 1: OPERACIONES DIVERSAS SOBRE LOS ELEMENTOS DE UNA LISTA.

En este ejercicio se pide realizar un programa que sea capaz de realizar distintas operaciones sobre una lista de palabras.

El ejecutable ***ListaSelect2000.exe*** indica cómo debe funcionar exactamente la aplicación a desarrollar. Dicho programa se encuentra en el directorio ***Q:\Infor1\Prac04\Ejer1***. Se debe copiar dicho ejecutable al directorio ***G:\Infor1\Prac04\Ejer1*** y ejecutar desde el propio directorio, observando bien su funcionamiento.

La Figura 4.1 muestra el aspecto general del programa que hay que realizar. En la columna de la izquierda aparecen:

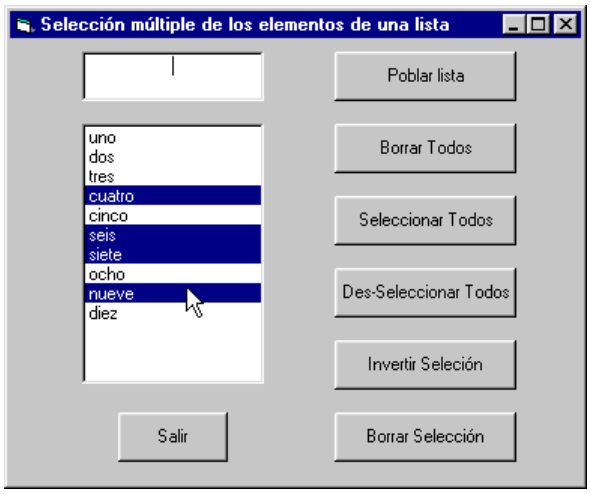


Figura 4.1: Ejecución del programa ListaSelect2000.exe.

1. Una ***caja de texto*** por medio de la cual se pueden introducir las palabras en la lista. Dicha caja de texto deberá tener el ***focus*** en todo momento, de modo que esté siempre preparada para empezar a teclear sobre ella. La palabra se introducirá en la lista al pulsar la tecla ***Intro***.
2. Una ***lista*** que contenga las palabras introducidas hasta ese momento. Esta lista es el elemento central de la aplicación. Deberá permitir ***selecciones múltiples***, tal como se ve en la Figura 4.1. Más adelante se proporciona una ayuda especial sobre este punto.
3. Un botón de comandos ***Salir*** que permite terminar la ejecución de la aplicación.

Por otra parte, en la columna de la derecha hay seis botones de comando que dan acceso a las operaciones principales que se pueden realizar sobre la lista. A continuación se describe la operación de cada uno de estos botones:

1. Botón **"Poblar Lista"**. Con objeto de no tener que estar tecleando continuamente elementos para introducir en la lista mientras se programa la aplicación, se ha dispuesto este botón que simplemente introduce en la lista los números del 1 al 10, escritos con letras (ver Figura 4.1).
2. Botón **"Borrar Todos"**. Este botón debe borrar todos los elementos de la lista, dejándola en blanco.
3. Botón "**Seleccionar Todos**". Este botón selecciona todos los elementos de la lista, sin tener en cuenta si anteriormente estaban o no seleccionados.
4. Botón "**Des-Seleccionar Todos**". Este botón des-selecciona (hace que no haya ningún elemento seleccionado y perdón por el anglicismo) todos los elementos de la lista, también sin tener en cuenta si anteriormente estaban o no seleccionados.
5. Botón "**Invertir Selección**". Este botón hace que los elementos seleccionados pases a no estarlo, y los que no estaban seleccionados pasen a estarlo.

6. Botón "**Borrar Selección**". Este último botón elimina de la lista los elementos seleccionados. Es quizás el más difícil de programar, y por eso se recomienda dejarlo para el final del ejercicio.

Nota sobre la selección múltiple en el control ListBox

En los Apuntes de "*Aprenda Visual Basic 6.0 como si estuviera en Primero*" sólo se habla de la ***selección simple*** y tampoco se han realizado previamente ejercicios de ***selección múltiple***. Por eso conviene prestar atención a lo que se expone a continuación sobre este tema, que es realmente bastante sencillo de entender y programar:

El control ***ListBox*** de Visual Basic 6.0 tiene una propiedad llamada ***MultiSelect*** con tres posibles valores:

0 – none Con este valor no es posible la selección múltiple y por tanto al clicar sobre un elemento para seleccionarlo, se pierde la selección en cualquier elemento seleccionado anteriormente. Ésta es la *opción por defecto*.

1 – simple Este valor permite seleccionar múltiples elementos simplemente clicando sobre ellos. Si se clica sobre un elemento seleccionado, pierde la selección.

2 – extended Ésta es la forma habitual de trabajar en ***Windows*** con las selecciones múltiples, por ejemplo con las listas de ficheros y directorios en el ***Explorer***. Para seleccionar varios elementos separados hay que clicar sobre ellos manteniendo pulsada la tecla ***Ctrl***. Por su parte la tecla ***Shift*** permite seleccionar rangos: por ejemplo clicando en el segundo elemento y luego clicando en el sexto mientras se mantiene ***Shift*** pulsada hace que se selecciones todos los elementos entre el segundo y el sexto. Para eliminar la selección de un elemento sin eliminar la de los demás basta clicar sobre él manteniendo pulsada la tecla ***Ctrl***. Ésta es la opción que hay que utilizar y con la que se ha programado el ejecutable modelo ***ListaSelect.exe***.

Con la propiedad ***MultiSelect*** activada (en **2 – Extended**) el control ***ListBox*** dispone de algunas propiedades interesantes y muy útiles para la realización de este ejercicio. Estas propiedades sólo están disponibles en tiempo de ejecución:

1. Propiedad ***SelCount***, que indica el número de elementos de la lista que están seleccionados en un determinado momento de la ejecución del programa. En realidad no es necesario utilizar esta propiedad en este ejercicio.

2. Propiedad ***Selected()***. Esta propiedad es un ***array*** de valores ***boolean*** que indican si un elemento está seleccionado o no (***True*** si lo está y ***False*** si no lo está). Los valores de la propiedad ***Selected*** se corresponden uno a uno con los de la propiedad ***List***, que es también un ***array*** con las cadenas de caracteres que están introducidas en la lista. La Tabla 6 muestra los valores de las propiedades ***List(i)*** y ***Selected(i)*** para el ejemplo mostrado en la Figura 4.1, en la cual aparecen seleccionados los elementos "cuatro", "seis", "siete" y "nueve" (índices 3, 5, 6 y 8).

3. La propiedad ***ListIndex*** indica el índice del elemento seleccionado (si sólo hay uno) y vale –1 en el caso de que no haya ningún elemento seleccionado. En el caso de las listas con selección múltiple esta propiedad indica sólo el índice del último elemento seleccionado, por lo que no es de especial utilidad. Es mejor basar los programas en la propiedad ***Selected()***.

Índice i	Valor de List(i)	Valor de Selected(i)
0	uno	False
1	dos	False
2	tres	False
3	cuatro	True
4	cinco	False
5	seis	True
6	siete	True
7	ocho	False
8	nueve	True
9	diez	False

Tabla 6. Valores de las propiedades ***List*** y ***Selected*** para los elementos de la lista en la Figura 4.1.

Metodología de programación:

Se van a enumerar aquí las etapas en las que se puede realizar este ejercicio para que su crecimiento sea sencillo, lógico y progresivo. Es importantísimo darse cuenta de que ***es inútil pasar a una etapa posterior si no se han resuelto satisfactoriamente todas las anteriores***.

1. Entender completamente el funcionamiento del programa modelo que se entrega
2. Pensar y decidir a grandes rasgos cómo se va a resolver el problema.
3. Describir sobre un papel, aunque sea esquemáticamente, cómo se van a resolver las operaciones clave (en este caso los botones que indican las operaciones a realizar sobre la lista). Cuanto más en detalle se estudie el problema en esta etapa menos tiempo se necesitará luego para programarlo.
4. Crear en el directorio ***Prac04*** un sub-directorio llamado ***Ejer1***.
5. Crear en ***Visual Basic 6.0*** un nuevo proyecto y guardarlo cuanto antes en el directorio ***G:\Infor1\Prac04\Ejer1*** con el nombre ***ListaSelect2000.vbp***. Guardar en ese mismo directorio el formulario con el nombre ***ListaSelect2000.frm***. Asegurarse con ***Windows Explorer*** de que los ficheros han sido guardados en el directorio indicado.
6. Colocar en el formulario la caja de texto (llamada ***txtBox***) y la lista (llamada ***lstBox***). Programar la introducción de palabras en la lista desde la caja de texto de modo que funcione como en el modelo. Añadir el botón "***Salir***" y comprobar que funciona correctamente.
7. Añadir el botón "***Poblar Lista***". Comprobar que añade los números del uno al diez (con letras), borrando cualquier cosa que hubiera previamente en la lista. Añadir el botón "***Borrar Todos***" y programarlo de modo que deje la lista en blanco, sin ningún elemento.
8. Añadir los botones, "***Seleccionar Todos***", "***Des-Seleccionar Todos***" e "***Invertir Selección***". Programarlos haciendo uso de la propiedad ***Selected()*** descrita previamente.
9. Por último, añadir el botón "***Borrar Selección***" que es el menos fácil de programar. Para programar este botón conviene tener en cuenta que cada vez que se borra un elemento cambia tanto el número de elementos de la lista (propiedad ***ListCount***) como la posición de los elementos que estaban detrás del elemento eliminado. Conviene además tener en cuenta una limitación del bucle ***for*** de ***Visual Basic***. Si por ejemplo se tiene un bucle;

```
for i=0 to lstBox.Listcount-1
   ...
next
```

y dentro de dicho bucle se elimina un elemento de la lista, aunque la propiedad ***ListCount*** cambie el bucle ***for*** no se ejecutará menos veces. Es como si el valor que tiene ***ListCount*** cuando empieza el bucle fuera copiado como valor límite y luego fuera ya inalterable. Esta dificultad se puede subsanar utilizando en su lugar un bucle ***do while … loop***.

10. Para terminar, asegurarse de que después de clicar cualquiera de los botones de la derecha, el ***focus*** queda siempre en la caja de texto, de modo que en cualquier momento se puede teclear un nuevo dato.

Recordar que está disponible la ***Ayuda*** de ***Visual Basic***. Seleccionando un control y pulsando la tecla ***F1*** aparece ayuda sobre el control seleccionado. En esa ayuda se pueden ver sus ***propiedades***, ***métodos***, etc.

4.2 EJERCICIO 2. AYUDA INFORMÁTICA PARA LA LIGA DE LAS ESTRELLAS.

Con el fin del milenio y la entrada de lleno de las cadenas de televisión en el negocio del fútbol no se deben ahorrar medios para que el espectáculo no defraude. No basta que los técnicos sepan tomar apuntes: es necesario que utilicen la informática al límite de sus posibilidades (de la informática, no de los técnicos).

Este ejercicio se propone como ayuda a los entrenadores de fútbol. Su finalidad es ayudarles a confeccionar las alineaciones, avisándoles de ciertos errores tontos que se cometen a veces y que en unos minutos pueden echar a perder el trabajo de toda una semana e incluso de una temporada. El ejecutable de este ejercicio se llama ***VanGaal2000.exe***, en honor al entrenador cuyo equipo ganó la liga pasada. Puedes encontrarlo en el directorio ***Q:\Infor1\Prac04\Ejer2***.

La Figura 4.2 muestra la pantalla principal de este ejercicio, en el momento de arrancar la aplicación. A la izquierda aparecen cuatro botones de opción que permiten elegir la lista de jugadores por líneas (*Porteros*, *Defensas*, *Medios* y *Delanteros*). En el centro aparece una lista de ***Jugadores*** y a la derecha otra lista con la ***Alineación*** (inicialmente vacía). Al arrancar el programa la lista de ***Jugadores*** muestra los ***Porteros***. Clicando en uno de los nombres en la lista de ***Jugadores*** pasa a la ***Alineación***. En la parte inferior aparecen dos botones que dan acceso a dos formularios con sendas fotografías de equipo (para acordarse de los jugadores de que dispone) y del propio técnico.

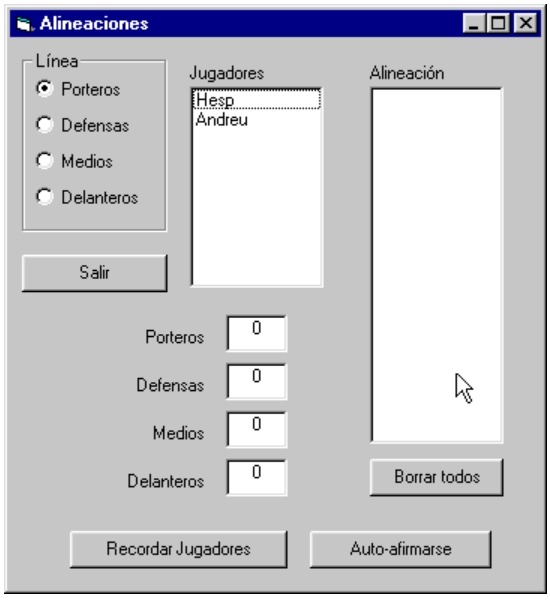


Figura 4.2. Pantalla inicial del programa.

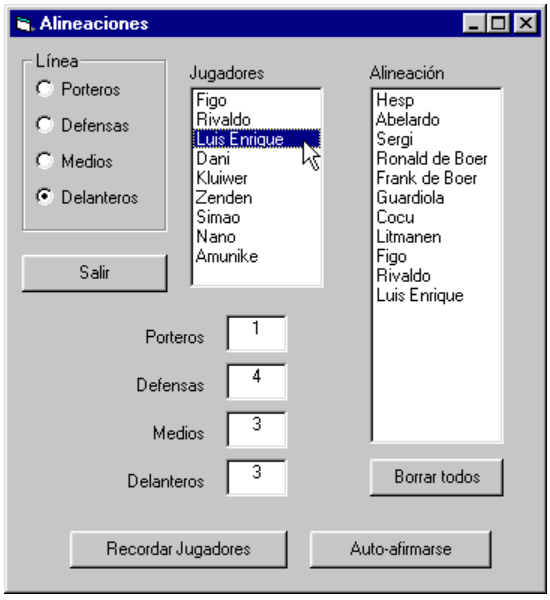


Figura 4.3. Pantalla en otro momento.

La Figura 4.3 muestra el estado de la aplicación cuando ya se está a punto de completar la ***Alineación***. Debajo de la lista de ***Jugadores*** aparecen cuatro ***labels con fondo gris*** (*Porteros*, *Defensas*, *Medios* y *Delanteros*) y otras cuatro ***labels con fondo blanco*** que indican el ***número de jugadores de cada línea*** introducidos en la alineación. Al parecer estos números son muy importantes para las tácticas.

El funcionamiento de este programa es muy sencillo:

1. Al seleccionar una línea con los botones de opción, los jugadores de la plantilla correspondientes a esa línea aparecen en la lista de ***Jugadores***.
2. Al clicar sobre uno de los nombres en la lista de ***Jugadores***, ese nombre se añade a la ***Alineación***, incrementándose en una unidad el número de Jugadores de la línea correspondiente en las ***labels***.
3. Al clicar sobre uno de los nombres en la lista de ***Alineación***, ese nombre se borra de dicha lista y el número de ***Jugadores*** de esa línea se reduce en una unidad en los controles ***label***.
4. El botón ***Salir*** termina la aplicación, mientras que el botón ***Borrar Todos*** limpia la lista de ***Alineación*** dejándola en blanco, y actualiza los contadores y labels. Por ejemplo, este botón puede ser de utilidad al acabar los partidos, y también cuando hay cambio de entrenador.

Además del funcionamiento básico descrito, este programa permite también evitar errores tontos, de despiste, más frecuentes en personas con responsabilidades grandes y complejas. En este ejercicio, que en realidad corresponde a una ***versión beta*** del programa, se han hecho sólo tres comprobaciones:

1. ***Sólo conviene jugar con un portero***. Por eso, si se intenta alinear más de un portero el programa da un aviso (aunque no lo impide).
2. Como lo que gusta a los aficionados son los goles y el fútbol ofensivo, el programa da otro aviso si se alinean ***más de cinco defensas*** (aunque tampoco lo impide).
3. Finalmente, el programa avisa también si en la alineación hay ***más de 11 jugadores***.

Para realizar este ejercicio crea un directorio llamado ***G:\Infor1\Prac04\Ejer2***. En ese directorio guarda un proyecto llamado ***VanGaal2000.vbp*** y un formulario llamado ***VanGaal2000.frm***. Con objeto de no perder tiempo tecleando durante el examen la amplia plantilla del ***Barça***, en ***Q:\Infor1\Prac04\Ejer2*** hay un fichero de texto llamado ***Barsa.txt*** que puede utilizarse para inicializar los vectores ***String*** correspondientes en el procedimiento ***Form_Load(),*** haciendo ***Copy*** y ***Paste***.

Dado que en Informática la táctica y la estrategia no son menos importantes que en el deporte (aunque están mucho peor pagadas, desgraciadamente), a continuación se incluyen unas sencillas recomendaciones para realizar con éxito este ejercicio.

Metodología de programación aconsejada:

Se van a enumerar aquí las etapas en las que se puede realizar este ejercicio para que su crecimiento sea sencillo, lógico y progresivo. Es importantísimo darse cuenta de que ***es inútil pasar a una etapa posterior si no se han resuelto satisfactoriamente todas las anteriores***.

1. Entender completamente el funcionamiento del programa ejecutable ***VanGaal2000.exe***.
2. Se sugiere que los nombres de los jugadores de cada línea se guarden en unos vectores llamados ***Por()***, ***Def()***, ***Med()*** y ***Del()***, según se muestra en el fichero ***Barsa.txt***, que puede ser utilizado para dar valor a esos vectores en el evento ***Load***. Otras variables que se pueden utilizar son ***nTotalPor***, ***nTotalDef***, ***nTotalMed*** y ***nTotalDel***, que representan el número total de juga-

dores de la plantilla en cada línea, y ***nPor***, ***nDef***, ***nMed*** y ***nDel***, que representan el número de jugadores de cada línea que se han incluido en la ***Alineación*** hasta el momento (son los números que aparecen en los controles ***label*** de fondo blanco).

3. Empezar a realizar el ejercicio siguiendo los pasos ***a)***, ***b)***, ***c)***, ***d)*** y ***e)*** del punto 4. Los elementos gráficos se irán añadiendo a medida que se vayan necesitando. Se sugieren los siguientes nombres para los controles: ***optPor***, ***optDef***, ***optMed***, ***optDel***, ***lstJuga***, ***lstAlin***, ***lblPor***, ***lblDef***, ***lblMed***, ***lblDel***, ***frmBarsa*** y ***frmVanGaal***.

4. Escribir en un papel, aunque sea esquemáticamente, cuáles son las ***operaciones clave*** y cómo se van a resolver. En este caso las operaciones clave podrían ser las siguientes:
   a) La función asociada con los ***botones de opción***, que al elegir uno de ellos hace que la lista de jugadores de esa línea aparezca en la lista ***Jugadores***. Para esta etapa se necesitan sólo los ***botones de opción*** y la lista ***Jugadores***.
   b) El paso de nombres de ***Jugadores*** a ***Alineación***, que se realiza clicando sobre uno de los nombres de la lista. La propiedad ***ListIndex*** indica el número del elemento sobre el que se ha clicado (contando desde cero). Se debe evitar que se pueda pasar dos veces el mismo jugador. Para esta etapa hay que comenzar añadiendo la lista ***Alineación***. El siguiente paso puede ser simplemente pasar el nombre. Cuando esto funcione, se debe modificar el código para que antes de añadir el nuevo nombre compruebe que no está ya en la ***Alineación***, comparándolo con los nombres que estén ya en ***Alineación***.
   c) El borrado individual de nombres de la lista ***Alineación***. Borrar el nombre clicado por el usuario es fácil, y de nuevo es conveniente empezar sólo por esto. Cuando se borren los nombres correctamente se pasará a la etapa siguiente.
   d) El objetivo de esta etapa es mantener actualizados los ***contadores*** en las operaciones anteriores. Pasar un nombre de la lista de ***Jugadores*** a la ***Alineación*** es más fácil, porque se puede saber de que línea procede viendo qué botón de opción está activado. Es más difícil el actualizar los contadores cuando se elimina un nombre de la ***Alineación***, ya que entonces no hay más remedio que irlo comparando con las listas de jugadores de cada línea y ver en cuál se encuentra.
   e) Chequear los ***tres tipos de errores*** tontos y dar el aviso correspondiente, según el modelo.

5. El programa hay que desarrollarlo de modo incremental, paso a paso, comprobando que cada paso está bien terminado antes de pasar al siguiente. Se empieza abriendo ***Visual Basic*** y haciendo que el formulario tenga un tamaño similar al del modelo.

6. Cuando una operación deba ser realizada varias veces en varios lugares, pensar en la posibilidad de definir una ***función*** o un ***procedimiento***. También se puede definir una ***función*** o un ***procedimiento*** para aquellas tareas que tengan un sentido claro en sí mismas, aunque sólo haya que hacerlas una vez. Las funciones hacen el código más fácil de entender y de chequear.

5 QUINTA PRÁCTICA

5.1 EJERCICIO 1: SIMULACIÓN DEL MOVIMIENTO DE UN PEZ DENTRO DE UNA PECERA

En este ejercicio se pide realizar un programa que sea capaz de representar gráficamente la ***trayectoria recta*** de un pez en una pecera. Una costumbre extendida entre los ingenieros es realizar simplificaciones o aproximaciones del problema real para crear un modelo más fácil de estudiar. En este caso se va a aproximar la pecera o aquarium por un ***recinto plano***. Como es lógico, el pez siempre se mantiene en el ***interior del recinto***, es decir que no puede sobrepasar los límites del mismo.

El ejecutable ***Aquarium.exe*** indica cómo debe funcionar exactamente la aplicación a desarrollar. Dicho programa se encuentra en el directorio ***Q:\Infor1\Prac05\Ejer1***. Se debe copiar al directorio ***G:\Infor1\Prac05\Ejer1*** y observar su funcionamiento.

El elemento móvil, es decir el pez, se representará por un círculo (con un control ***Shape***) tal y como aparece en el programa de muestra. Su nombré será ***shpPez***. La pecera se representará utilizando un ***PictureBox*** que se llamará ***pctPecera***. La Figura 5.1 muestra el esquema general del resultado de esta Ejercicio.

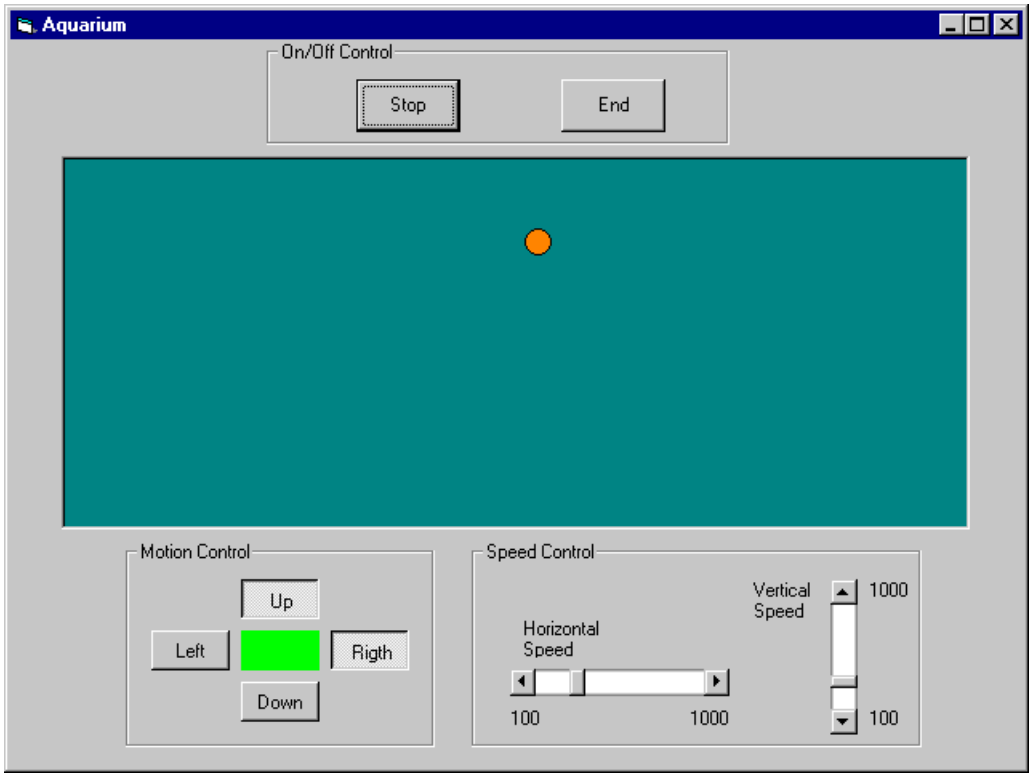


Figura 5.1: Ejecución del programa Aquarium.exe.

Se va a considerar en cada instante de tiempo que la posición del pez es la del instante anterior más el desplazamiento correspondiente al incremento de tiempo. Las expresiones para calcular la posición vertical y horizontal son las siguientes:

$$\begin{cases} x_k = x_{k-1} + v_x \Delta t \\ y_k = y_{k-1} + v_y \Delta t \end{cases}$$

donde v_x y v_y son las componentes de la velocidad en los ejes cartesianos y Δt el incremento del tiempo.

Al ejecutar el programa aparecen tres paneles de control (en tres *frames*): ***On/Off Control***, ***Motion Control*** y ***Speed Control***. En el primero de ellos tiene dos botones, uno (***Start*** o ***Stop***) que permite parar cuando el móvil está en marcha y reiniciar el movimiento cuando está parado, y otro (***End***) que finaliza la ejecución cuando es pulsado. El botón ***Start/Stop*** debe cambiar su texto cada vez que se pulsa de manera que informe de la acción que va a realizar al ser pulsado.

El panel ***Motion Control*** contiene 4 controles ***CheckBox*** (***Up***, ***Down***, ***Left*** y ***Rigth***) y un ***label*** central (***lblChoque***). Debido a su semejanza con un botón (***conseguida modificando la propiedad Style de cada control CheckBox al valor "1-Graphical"***) nos referiremos a ellos como ***botones***. Estos botones indican la dirección del movimiento del pez. Los botones ***Up*** y ***Down*** activan la componente vertical (v_y) de la velocidad, con signo positivo el botón ***Up*** y negativo el botón ***Down***. Análogamente los botones ***Left*** y ***Rigth*** activan o desactivan la componente horizontal de la velocidad (v_x). Lógicamente, no es posible mantener activados simultáneamente los botones ***Up*** y ***Down*** ni los botones ***Left*** y ***Rigth***. Sin embargo sí que se permitirá mantener pulsado uno de componente horizontal y uno de componente vertical (*Up* y *Left, Up* y *Rigth, Down* y *Left* y *Down* y *Rigth*).

Un choque con un límite lateral a izquierda y derecha consiste en anular el término de la velocidad en el eje horizontal, manteniendo la del eje vertical. En un choque con un límite superior o inferior se mantiene la velocidad en el eje horizontal y se anula el término de incremento en el eje vertical. En ambos casos hay que cambiar a ***rojo*** el color del recuadro ***lblChoque*** que aparece entre los botones ***Up***, ***Down***, ***Left*** y ***Rigth***. Cuando deje de estar junto a un límite, el color de ***lblChoque*** será ***verde***.

La ***velocidad del móvil*** se debe poder modificar en tiempo de ejecución, en las dos direcciones independientemente. Para ello se deben actualizar las velocidades cada vez que se modifiquen las ***barras de desplazamiento*** incluidas en el panel ***Speed Control***.

Metodología de programación:

Se van a enumerar aquí las etapas en las que se puede realizar este ejercicio para que su crecimiento sea sencillo, lógico y progresivo. Es importante darse cuenta de que ***es inútil pasar a una etapa posterior si no se han resuelto satisfactoriamente todas las anteriores***.

1. Entender completamente el funcionamiento del programa.
2. Pensar y decidir a grandes rasgos cómo se va a resolver el problema.
3. Escribir en un papel, aunque sea esquemáticamente, cómo se van a resolver las operaciones clave (en este caso los ***choques con los límites*** y la ***modificación de las velocidades***). Cuanto más en detalle se estudie el problema en esta etapa menos tiempo se necesitará luego para programarlo.
4. Colocar el ***recinto***, el ***pez***, el panel de ***On/Off Control***, y por último los dos ***botones Start/Stop*** y ***End*** en el formulario principal. Se recomienda utilizar ***Height = 3500*** y ***Width = 8500*** para el tamaño del recinto y ***Height = 255*** y ***Width = 255*** para el tamaño del pez.
5. ***Guardar muy cuidadosamente*** el proyecto y el formulario en el directorio ***G:\Infor1\Prac05\Ejer1***. Al proyecto se le pondrá el nombre de ***Aquarium.vbp*** y al formulario el de ***Aquarium.frm***.
6. Cambiar el origen de coordenadas de la ***pecera*** de modo que éste se encuentre en el centro del recinto. Establecer el punto inicial del pez y las condiciones iniciales del movimiento (posición inicial del pez y componentes v_x y v_y de la velocidad inicial).

7. Incluir en el formulario un control ***Timer***. Se utilizará un intervalo de ***100*** milisegundos. Programar el código correspondiente a los botones ***Start/Stop*** y ***End***.

8. Crear el panel ***Motion Control*** con los 4 botones (de tipo ***CheckBox***: ***Up***, ***Down***, ***Left*** y ***Rigth***) y el ***label*** central (***lblChoque***). Reproducir el funcionamiento de los 4 botones (activación y desactivación) sin necesidad de mover el pez. Una vez conseguido el funcionamiento deseado permitir que el móvil se desplace manteniendo constantes sus velocidades horizontal y vertical.

9. Introducir el panel de control de la velocidad (***Speed Control***) para permitir la modificación de la velocidad del móvil. La ***barra de desplazamiento horizontal*** tendrá las siguientes características: Máximo = ***1000***, Mínimo = ***100*** e incrementos de ***20*** el mínimo (***Small Change***) y ***100*** el máximo (***LargeChange***). La características de la barra ***vertical*** son: Máximo = ***100***, Mínimo = ***1000*** e incrementos de ***20*** el mínimo y ***100*** el máximo. Se recuerda que al variar cualquiera de las barras, la posición del pez deberá calcularse teniendo en cuenta las nuevas componentes de la velocidad.

10. Por último se pide introducir el código necesario para comprobar si el pez alcanza alguno de los límites de la pecera. Modificar el color del control ***lblChoque*** : ***rojo*** si está en uno de los límites y ***verde*** en el resto de los casos.

5.2 EJERCICIO 2: UTILIZACIÓN DEL DEBUGGER: DIBUJO INTERACTIVO DE POLÍGONOS

Este ejercicio consiste en corregir un programa ya hecho en el cual se han introducido deliberadamente al menos **3 ERRORES.**

Este ejercicio consta de tres archivos: el proyecto ***debugPol.vbp***, el formulario ***debugPol.frm*** y el ejecutable correcto ***debugPol.exe***, que se encuentran en el directorio ***Q:\Infor1\Prac05\Ejer2***. ***Se han de copiar todos estos archivos a la cuenta propia del alumno antes de usarlos***. Se guardarán en el directorio ***G:\Infor1\Prac05\Ejer2***. Se ***corregirán los errores*** que se encuentren introduciendo una nueva línea y ***se mantendrá la sentencia que se ha cambiado comentada*** con el carácter (‘) y ***marcada con la palabra “ERROR” y el número ordinal del error encontrado (ejemplo: ERROR 1, ERROR 2, ERROR 3, …)***.

Cuando arranca el programa tiene el aspecto mostrado en la Figura 5.2. En la parte superior aparece una caja de dibujo (***PictureBox***) cuyas coordenadas varían entre –100 y +100 en el eje de abscisas y entre –50 y +50 en el de ordenadas. Debajo aparecen dos listas cuyas etiquetas son “X” e “Y” y dos botones, uno para salir del programa y otro para volver a la situación inicial. Obsérvese que el cursor tiene la forma de flecha.

Si se empieza a clicar sobre la caja de dibujo empiezan a parecer los puntos en los que se ha clicado unidos por segmentos de recta, según puede verse en la Figura 5.3. Al mismo tiempo, las coordenadas de los puntos en que se ha clicado van añadiéndose a las dos listas.

Para ***cerrar el polígono*** se sigue el siguiente procedimiento: mientras se van introduciendo puntos el cursor sigue con la forma de ***flecha*** de la Figura 5.2, pero si el cursor se acerca suficientemente al primer punto introducido su forma cambia a la de una ***cruz***, tal como aparece en Figura 5.3. Si en ese momento se clica el polígono se cierra y ya no se pueden introducir más puntos, a no ser que se pulse el botón ***Reset***, que devuelve el programa a la situación inicial.

Figura 5.2: Pantalla inicial del programa.

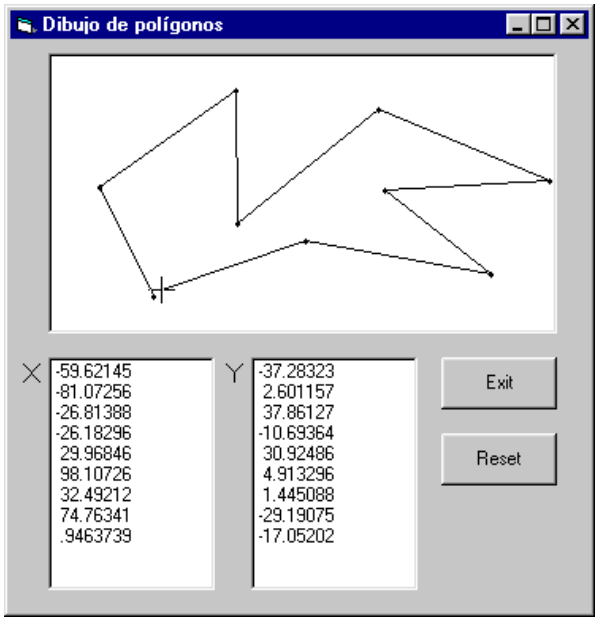


Figura 5.3: Definición interactiva de polígonos.

El programa tiene ***tres estados diferentes***:

1. Al arrancar (y después de pulsar ***Reset***). Este estado se caracteriza porque no hay nada dibujado en la caja de dibujo y porque al mover el cursor sobre dicha caja no se realiza ninguna acción. Este estado termina cuando se clica por primera vez. En este estado la caja de dibujo responde al evento ***MouseDown*** (que se produce al clicar[2]), pero no al evento ***MouseMove*** (que se produce al mover el ratón con el cursor sobre la caja de dibujo).
2. Este segundo estado dura desde que se clica por primera vez hasta que se cierra el polígono. En este estado se dibuja el polígono con los puntos clicados hasta el momento y aparece una línea que sigue al cursor desde el último punto clicado hasta la posición del cursor. De este estado se sale clicando cerca del primer punto para cerrar el polígono. En este estado, la caja de dibujo responde a los dos eventos ***MouseDown*** y ***MouseMove***.
3. El tercer estado se desarrolla desde que se cierra el polígono hasta que se pulsa uno de los dos botones (***Exit*** para terminar o ***Reset*** para volver al estado 1). Este estado se caracteriza porque la caja de dibujo no responde a ninguno de los eventos del ratón (ni ***MouseDown*** ni ***MouseMove***, es decir ni clicar ni mover).

Para distinguir entre estos tres estados se han creado dos variables ***boolean*** llamadas ***started*** y ***finished***. La Tabla 7 muestra los estados del programa en relación con el valor de dichas variables.

Estado	started	finished
1	False	False
2	True	False
3	False	True

Tabla 7. Estados del programa ***DebugSep99.exe***.

[2] Se utiliza el evento ***MouseDown*** y no el evento ***Click*** de la caja de dibujo porque el procedimiento que gestiona los eventos de este último no recibe información sobre las coordenadas del punto en el que se ha clicado, aspecto que es esencial para poder construir el polígono.

Para realizar este ejercicio se pueden hacer las siguientes recomendaciones:

1. ***Probar el ejecutable correcto DebugPol.exe*** con mucha atención, entendiendo bien cómo funciona.
2. Abrir el proyecto correspondiente a los ficheros que se entregan ***DebugPol9.vbp*** y ***DebugPol.frm***, e intentar ejecutarlo.
3. Usar el ***debugger*** para examinar el programa ***paso a paso*** y mirar los valores que adquieren las variables importantes en cada momento, en particular justo antes de producirse el fallo del programa. Para parar la ejecución en una línea concreta del código es aconsejable usar ***breakpoints*** o el comando ***Run/Step to Cursor***.

Recuérdese que para este ejercicio se dispone de un tiempo en exclusiva de **30 minutos**.

Listado del programa entregado

Para facilitar la búsqueda de los errores se incluye con este enunciado un listado de los procedimientos y funciones del programa entregado.

```
Option Explicit
Dim xv As Single, yv As Single
Dim started As Boolean, finished As Boolean


Private Sub dibujarPoligono()
    Dim i As Integer
    Dim x1 As Single, y1 As Single, x2 As Single, y2 As Single
    ' se borra el contenido de la picture box
    pctBox.Cls
    ' se dibujan los lados del polígono
    For i = 1 To lstX.ListCount - 2
        x1 = Val(lstX.List(i))
        y1 = Val(lstY.List(i))
        x2 = Val(lstX.List(i + 1))
        y2 = Val(lstY.List(i + 1))
        pctBox.Line (x1, y1)-(x2, y2)
    Next
    ' se dibujan los vértices del polígono
    pctBox.DrawWidth = 3
    For i = 0 To lstX.ListCount - 1
        x1 = Val(lstX.List(i))
        y1 = Val(lstY.List(i))
        pctBox.PSet (x1, y1)
    Next
    pctBox.DrawWidth = 1
End Sub


Private Sub cmdExit_Click()
    End
End Sub


Private Sub cmdReset_Click()
    pctBox.Cls
    lstX.Clear
    lstY.Clear
    started = False
    finished = False
End Sub
```

```
Private Sub Form_Load()
    ' se define la escala de la picture box
    pctBox.Scale (-100, 50)-(100, -50)
    ' se inicializan los indicadores de comienzo y fin
    started = True
    finished = False
End Sub


Private Sub pctBox_MouseDown(Button As Integer, Shift As Integer,
                             X As Single, Y As Single)
    ' si el programa ha terminado, no se responde a este evento
    If finished = True Then Exit Sub
    ' la primera vez que se clica se comienza la introducción de puntos
    started = True
    ' se guardan las coordenadas del último punto clicado
    xv = X
    xv = Y
    ' se chequea si se está cerca del primer punto (cambia el cursor)
    If pctBox.MousePointer = vbCrosshair Then
        ' si se está cerca al clicar se termina
        started = False
        lstX.AddItem (lstX.List(0))
        lstY.AddItem (lstY.List(0))
        Call dibujarPoligono
        pctBox.MousePointer = vbArrow
        finished = True
        Exit Sub
    Else
        ' si no se está cerca se añade un nuevo punto
        lstX.AddItem (Str(X))
        lstY.AddItem (Str(Y))
    End If
End Sub




Private Sub pctBox_MouseMove(Button As Integer, Shift As Integer,
                             X As Single, Y As Single)
    Dim i As Integer
    Dim x0 As Single, y0 As Single
    ' si no se ha clicado todavía ninguna vez no se hace nada
    If started = False Then Exit Sub
    ' se recuperan las coordenadas del primer punto del polígono
    x0 = Val(lstX.List(0))
    y0 = Val(lstY.List(0))
    ' se chequea si el cursor está cerca del primer punto
    If ((X - x0) ^ 2 + (Y - y0) ^ 2) < 25 Then
        ' si se está cerca el cursor cambia a una cruz
        pctBox.MousePointer = vbCrosshair
    Else
        ' si no se está cerca el cursor es la flecha
        pctBox.MousePointer = vbArrow
    End If
    ' se dibuja el polígono hasta el último punto clicado
    Call dibujarPoligono
    ' se dibuja una línea desde el último punto clicado
    '    hasta la posición del cursor
    pctBox.Line (xv, yv)-(X, Y)
End Sub
```

5.3 EJERCICIO 3: DEFINIR UN POLÍGONO Y AVERIGUAR SI UNA SERIE DE PUNTOS ESTÁN DENTRO O FUERA.

El ejecutable-modelo de este ejercicio se llama ***Poligono.exe***, y puede encontrarse en el directorio ***Q:\Infor1\Prac05\Ejer3***. Antes de abrirlo para ver cómo funciona cópiese al propio directorio ***G:\Infor1\Prac05\Ejer3***. Créese para ello el directorio ***Ejer3*** si no se ha hecho ya.

Este ejercicio tiene una gran similitud con el ***Ejercicio 2***, por lo que será de gran ayuda el haber entendido dicho ejercicio perfectamente (incluso aunque no se hayan encontrado todos los errores). El objetivo de este ejercicio es primero definir un polígono y luego ir clicando en distintos puntos de la caja de dibujo de modo que el programa nos diga con un nuevo mensaje si el punto clicado está dentro o fuera del polígono.

La Figura 5.4 muestra la pantalla inicial del programa: junto al formulario principal aparece una caja de mensajes que indica que ya se pueden empezar a introducir los puntos con el ratón. Pulsando ***O.K.*** se ve ya el formulario del programa listo para introducir los vértices del polígono.

Figura 5.4. Pantalla inicial del programa.

Figura 5.5. Pantalla en otro momento.

Si se empieza a clicar sobre la caja de dibujo empiezan a parecer los puntos en los que se ha clicado unidos por segmentos de recta, mientras el cursor aparece con forma de cruz en todo momento. Al mismo tiempo, las coordenadas de los puntos en que se ha clicado se irán guardando en dos vectores.

Para ***cerrar el polígono*** hay que hacer doble clic, uniéndose en ese caso el punto en el que se ha clicado con el primero de todos. Practíquese un poco con el ejecutable ***Poligono.exe*** para entender bien cómo funciona.

Una vez que se ha cerrado el polígono el programa da un nuevo mensaje con el texto "***Clique sobre la caja de dibujo para definir puntos***". A partir de este momento comienza la parte más importante del programa: el usuario va clicando en distintos puntos de la caja de dibujo, dichos puntos se representan en azul y con una anchura de 5 pixels y aparece un mensaje diciendo si el punto en el que se ha clicado está ***dentro*** o ***fuera*** del polígono. La Figura 5.5 muestra un ejemplo con un polígono y un punto que resulta estar dentro del mismo. Se puede chequear la posición de tantos puntos como se desee respecto al mismo polígono. Con el botón ***Reset*** se vuelve a la situación inicial del programa.

Algoritmo para saber si un punto está dentro o fuera de un polígono.

Un punto fundamental de este ejercicio es disponer de un algoritmo sencillo que permita saber si un punto está dentro o fuera de un polígono. Dicho algoritmo se explica a continuación. Es importante entenderlo bien, porque luego hay que programarlo y es muy difícil programar bien lo que se ha entendido mal. La Figura 5.6 muestra distintas posiciones de puntos respecto a un polígono arbitrario. Esta figura es la base para el algoritmo que se va a explicar a continuación.

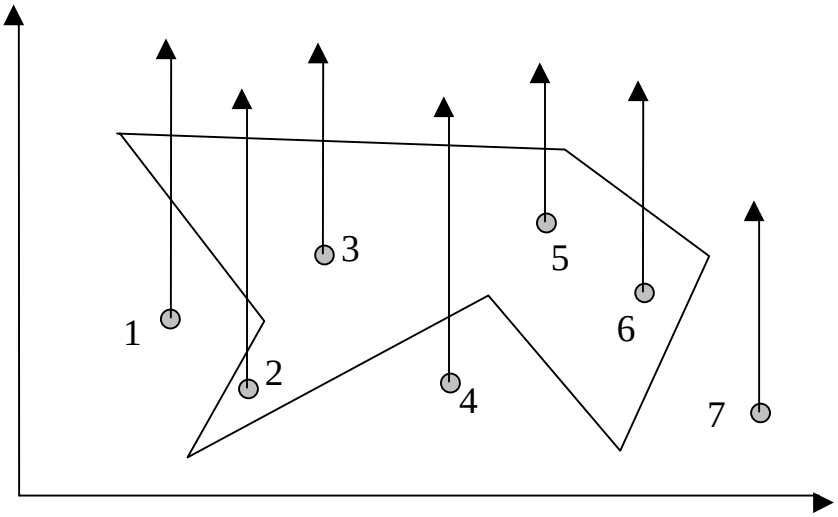


Figura 5.6. Distintas posiciones de puntos respecto a un polígono.

La idea es colocar en cada punto un semieje vertical ascendente y estudiar para cada uno de dichos semiejes el número de intersecciones con los lados del polígono:

- Si el ***número de intersecciones*** es ***impar*** el punto está ***dentro*** del polígono (puntos 2, 3, 5 y 6 en la Figura 5.6).
- Si el ***número de intersecciones*** es ***par*** el punto está ***fuera*** del polígono (puntos 1, 4 y 7 de la Figura 5.6).
- NOTA: Para este ejercicio no es necesario considerar los casos en que el punto esté exactamente sobre un lado o sobre un vértice.

Con un poco más de detalle, el algoritmo resultante podría ser como sigue:

1. Las coordenadas de los vértices del polígono se tienen guardadas en unos vectores **x1(20)** e **y1(20)**. No hace falta considerar polígonos de más de 20 lados. Aunque Visual Basic empieza a contar desde cero, el primer punto es el (**x1(1),y1(1)**) y el último el (**x1(n),y1(n)**).
2. Conocidas las coordenadas (**x,y**) del punto a chequear es conveniente trasladar el origen de coordenadas a dicho punto, de modo que el semieje a estudiar sea el semieje **y positivo**. De este modo las expresiones serán más sencillas. Para trasladar el origen a dicho punto hay que restar a las coordenadas de todos los vértices del polígono las coordenadas del punto (**x,y**). Con esto se obtendrán unos nuevos vectores de coordenadas de los vértices **x2()** e **y2()** que serán diferentes para cada punto (**x,y**) cuya posición se quiera chequear.
3. La Figura 5.7 muestra el caso concreto del **punto 3**, con los ejes de coordenadas ya trasladados a dicho punto. Por conveniencia se muestra también la numeración de los lados del polígono. Ahora hay que estudiar la intersección del eje **y positivo** con cada uno de los lados del polígono. Se pueden presentar varios casos:

- Los lados que tienen las coordenadas **y** de sus dos extremos negativas están por debajo del eje **x**, y por tanto es imposible que corten al semieje **y positivo**. Tal sucede con los lados 4, 5, 6 y 7 de la Figura 5.7. Un **If** que compruebe esta situación evita ya hacer un análisis detallado de la intersección.
- Los lados cuyos extremos tienen coordenada **x** del mismo signo están a uno u otro lado del semieje **y positivo**, pero en cualquier caso es imposible que lo corten. Tal sucede por ejemplo con los lados 1, 6 y 7, cuyos extremos tienen ambos coordenada **x** positiva, y con los lados 3 y 4, cuyos extremos tienen ambos coordenada **x** negativa.

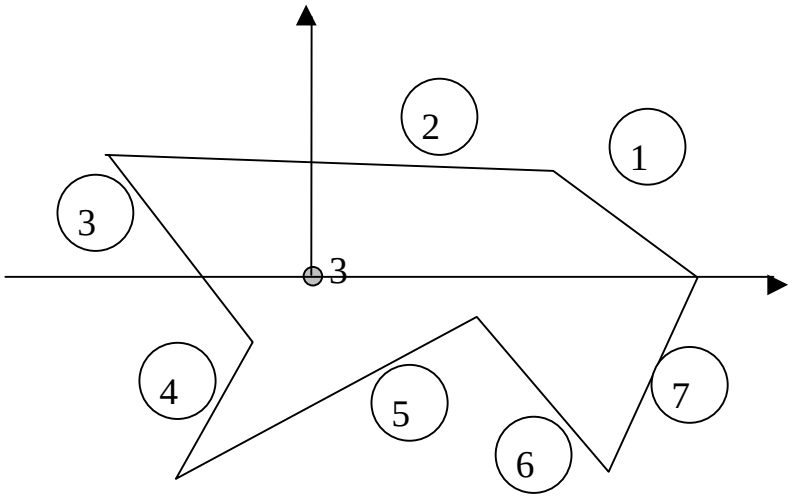


Figura 5.7. Análisis de la posición de un punto concreto.

- Los únicos lados que pueden cortar al semieje **y positivo** son aquellos que tienen un extremo con coordenada **x** positiva y otro extremo con coordenada **x** negativa (caben dos posibles situaciones) y que además tienen al menos un extremo con coordenada **y** positiva. Para estos lados hay que hacer un análisis más detallado: es seguro que cortan al eje **y**, pero pueden cortarlo por debajo del origen.

  La Figura 5.8 muestra tres posibles casos de intersección del lado **i-j** del polígono con el eje de ordenadas trasladado. En el caso a) los dos extremos tienen coordenada **y** positiva y coordenada **x** de distinto signo, con lo cual es seguro que hay una intersección con el semieje **y positivo**. En los casos b) y c) los dos puntos tienen coordenada x de distinto signo pero uno está por encima del eje x y otro por debajo. En este caso no hay más remedio que hacer un análisis detallado de la intersección: se calcula la coordenada **yp** del punto de intersección; si es positiva como en el caso b) hay intersección con el semieje **y positivo**. Si dicha coordenada es negativa como en el caso c) no hay intersección con el semieje **y positivo**.

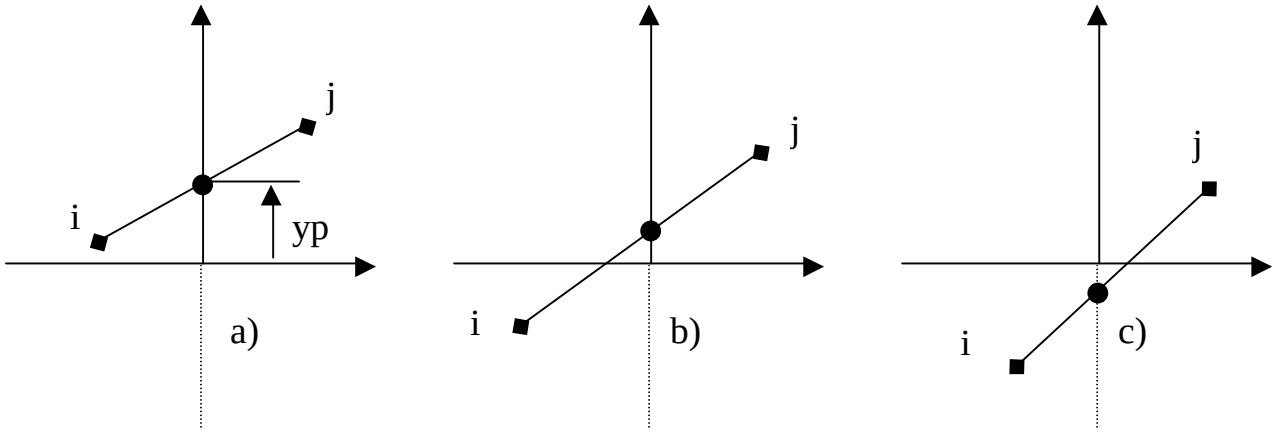


Figura 5.8. Distintos casos de intersección de un lado con el eje de ordenadas.

La expresión de la coordenada **yp** en función de las coordenadas de los puntos **i** y **j** es la siguiente (se puede deducir fácilmente aplicando el *Teorema de Tales*):

$$yp = y2(i) - \frac{y2(j) - y2(i)}{x2(j) - x2(i)} x2(i)$$

- Una vez que se haya calculado el número de lados que cortan al semieje **y positivo** hay que ver si este número es para o impar. Para ello se puede utilizar el operador ***Mod***, que calcula el ***resto de la división entre enteros***. Calculando el resto de dividir por 2 el número de cortes se puede saber si éste es par o impar, y en función de que lo sea o no sacar el mensaje correspondiente.

Metodología de programación aconsejada:

Se van a enumerar aquí las etapas en las que se puede realizar este ejercicio para que su crecimiento sea sencillo, lógico y progresivo. Es importante recordar que suele ***ser inútil pasar a una etapa posterior si no se han resuelto satisfactoriamente todas las anteriores***.

1. Entender completamente el funcionamiento del programa ejecutable ***Poligono.exe***.
2. Para realizar este ejercicio se creará un directorio llamado ***G:\Infor1\Prac05\Ejer3***. En ese directorio guarda un proyecto nuevo llamado ***Poligono.vbp*** y un formulario nuevo llamado ***Poligono.frm***. Conviene asegurarse con ***Windows Explorer*** que dichos ficheros se han guardado en el directorio adecuado.
3. Una vez creado y guardado el proyecto introducir los tres controles que se van a utilizar: la caja de dibujo (***pctBox***) y los dos botones (***cmdReset*** y ***cmdSalir***). Las coordenadas de la caja de dibujo oscilarán entre –100 y +100 en abscisas y entre –50 y +50 en ordenadas.
4. Escribir en un papel, aunque sea esquemáticamente, cuáles son los ***estados posibles del programa*** y cómo se van a distinguir unos de otros. Se sugiere utilizar ***variables booleanas***, al igual que se hizo en el ***Ejercicio 2***. Desde luego el programa se comporta diferente cuando se está introduciendo el polígono y cuando se están introduciendo los puntos para ver si están dentro o fuera. Al igual que en el ***Ejercicio 2***, los ***estados*** de pueden clasificar de acuerdo con la respuesta que debe tener el programa a los eventos ***MouseDown*** y ***MouseMove***. Puede ser conveniente definir una ***Tabla*** análoga a la presentada en el enunciado de dicho Ejercicio.
5. El siguiente paso puede ser definir el polígono:

a) Cuando se inicia la aplicación el ***cursor*** tiene ***forma de cruz*** y la caja de dibujo responde sólo al evento ***MouseDown***. Las variables ***boolean*** que definen este estado pueden definirse en el evento ***Load*** del formulario.
b) Cuando se clica por primera vez para introducir el primer punto la caja de dibujo empieza a responder también al evento ***MouseMove***, pues aparece una línea que va desde el último punto introducido hasta la posición actual del cursor. La imagen se ***refresca***, esto es, la línea que va hasta la posición del cursor se borra cuando se va a dibujar la siguiente. La forma más sencilla de refrescar la imagen es borrar todo y re-dibujar todo cada vez que sea necesario.
c) Para ***terminar de definir el polígono*** basta hacer ***doble clic*** en cualquier punto de la caja de dibujo. El punto en el que se ha hecho doble clic pasa a ser el último punto del polígono, que se une con el primero para cerrarlo. A partir de este momento la caja de dibujo deja de responder al evento ***MouseMove*** y response sólo al evento ***MouseDown***, con el que se introducen los puntos cuya posición se va a investigar.

6. En la situación de ***polígono introducido*** el usuario va introduciendo puntos con el ratón y el programa responde indicando si el punto está o no dentro del polígono. Los puntos anteriores no se borran, por lo que en este estado no hace falta refrescar la imagen de la caja de dibujo. Una vez conocidas las coordenadas del punto puede llamarse a una función o procedimiento que determine si el punto está dentro o fuera. Esta función o procedimiento es el centro del programa y podría llamarse por ejemplo ***isInside()***. No se debe empezar a programar esta función hasta que no se esté seguro de haber introducido bien el polígono.

7. En cualquier momento el usuario puede volver a la situación inicial pulsando sobre el botón ***Reset***. Para ello hay que borrar por completo el contenido de la caja de dibujo y volver al estado con el que arranca el programa.

8. Antes de dar por terminado el Ejercicio defínanse (si no se ha hecho ya) los mensajes que ayudan al usuario a saber que se ha pasado de un estado a otro o que el programa espera que realice determinada operación. Los mensajes deben ser los mismos que los del modelo ***Poligono.exe***.

Nota Final: Este Ejercicio no es difícil si se hace con orden y cuidado, incrementalmente y apoyándose en las partes análogas del ***Ejercicio 2***. Tampoco es especialmente largo.

6 SEXTA PRÁCTICA

6.1 EJERCICIO 1: SIMULACIÓN DEL LLENADO Y VACIADO DE UN DEPÓSITO

En este primer ejercicio se trata de realizar un programa en ***Visual Basic 6.0*** que simule el llenado y vaciado de un depósito, con un caudal de líquido (velocidad de llenado) variable. El usuario puede activar o detener con un ***botón*** tanto el llenado como el vaciado del depósito. Unos ***botones de opción*** determinan si se está llenando o vaciando. Es posible controlar la velocidad de llenado con una **barra de desplazamiento horizontal**. La altura alcanzada por el líquido varía entre 0 y 5000, y es indicada en una ***caja de texto*** al lado de la barra de desplazamiento. El botón ***Start*** se convierte en ***Stop*** cuando el fluido está entrando o saliendo del depósito.

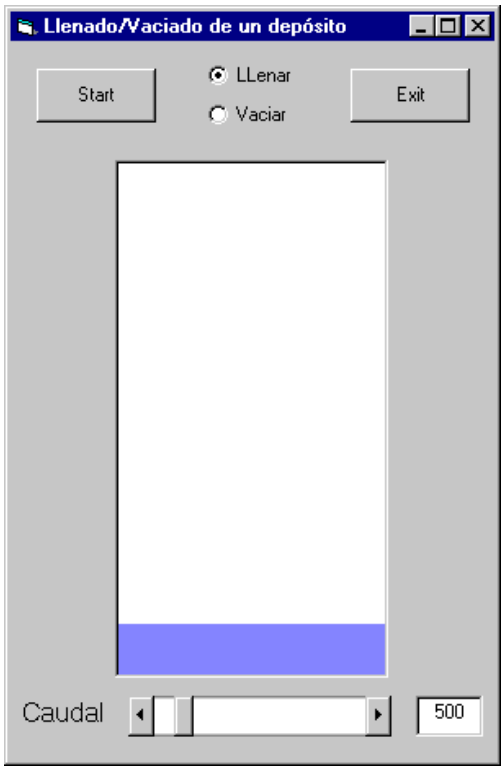


Figura 6.1. Llenado de un depósito

Se puede encontrar el fichero ejecutable de este ejercicio en ***Q:\Infor1\Prac06\Ejer1***\. Se debe crear en ***G:\Infor1\Prac06*** un directorio para este ejercicio llamado ***Ejer1***. Se deberá copiar en él el fichero ***depo99a.exe*** probar su funcionamiento. Los ficheros que realice el alumno se deberán llamar ***depo99a.vbp*** para el fichero del proyecto y ***depo99a.frm*** para el formulario.

Se recomienda observar con atención el funcionamiento del programa ejecutable anterior, viendo la función que realiza cada uno de los botones y controles , y ***pensar*** en cada caso la función que está realizando cada uno de ellos. Cuanto mejor se entienda el funcionamiento de los botones, menos tiempo se requerirá para la generación del código.

Nota: El depósito se simula mediante una ***Picture Box*** sobre la que se dibuja un cuadrado de color azul, cuya altura representa la altura alcanzada por el líquido y cuya anchura es igual a la de la ***Picture Box***. Este cuadrado se puede materializar de muchas formas, con un control ***Label*** (el sistema recomendado), con un cuadrado propiamente dicho dibujado con el método ***Line***, etc. El control ***Timer*** permitirá redibujar cada cierto número de milisegundos. Para ello este control dispone de un evento llamado también ***Timer*** que se genera de modo automático cada cierto número de milisegundos, determinado por una propiedad llamada ***Interval***. Por ejemplo, si ***Interval*** vale 50 milisegundos, se generarán 20 eventos en ***Timer*** cada segundo. Estos eventos se utilizarán para variar la altura del fluido, utilizando por ejemplo la propiedad ***Top*** del *label* azul que representa el fluido. El control ***Timer*** dispone también de la propiedad ***Enabled***; si está en ***true*** el reloj funciona y se producen los eventos. Si está en ***false*** el reloj se para.

El incremento de la altura del líquido depende del *caudal* o *velocidad de llenado*, que se establece con la barra de desplazamiento horizontal en la parte inferior del control. Se recomienda tener una variable (por ejemplo ***h*** o ***altura***) que defina la altura alcanzada por el fluido (entre 0 y 5000) y cuyo valor sea variado por el procedimiento ***Timer1_Timer()*** con un incremento o decremento que dependa del valor de la barra de desplazamiento horizontal.

Nota: En el directorio ***Ejer1b*** se incluye una versión muy similar de este ejercicio en la que se han mejorado los gráficos para evitar el parpadeo de la imagen.

6.2 EJERCICIO 2: CÁLCULO DE UNA RAÍZ DE UN POLINOMIO POR EL MÉTODO DE NEWTON.

En este primer ejercicio se trata de realizar un programa en ***Visual Basic 6.0*** que calcule una raíz de una función por el método de Newton. A continuación se explica dicho método:

El método de Newton permite hallar ***de modo iterativo*** (es decir, por aproximaciones sucesivas) raíces o ceros de funciones no lineales. Este método está basado en el desarrollo en ***serie de Taylor*** de la función *f(x)* en el punto x_i, siendo x_i una cierta aproximación de la solución *x* que se desea calcular. El desarrollo en serie de Taylor produce:

$$f(x) \approx f(x_i) + (x - x_i) f'(x_i) + ... = 0 \quad (1)$$

Utilizando sólo los dos primeros términos del desarrollo en serie (es decir, sustituyendo la función por la tangente en el punto x_i), se tiene la siguiente ecuación lineal:

$$f(x_i) + (x - x_i) f'(x_i) = 0 \quad (2)$$

de donde se puede despejar el valor de la *x* que anula esta expresión (la raíz de la ecuación linealizada), es decir, una nueva aproximación a la verdadera solución. A esta nueva aproximación le llamaremos x_{i+1}. Este valor se puede calcular a partir de (2) con la expresión:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (3)$$

Ahora volvemos a aplicar el mismo procedimiento, sustituyendo la función no lineal por los dos primeros términos (de nuevo la tangente) de su desarrollo en serie alrededor del punto x_{i+1}, y calculando una nueva y mejor aproximación x_{i+2}:

$$x_{i+2} = x_{i+1} - \frac{f(x_{i+1})}{f'(x_{i+1})} \quad (4)$$

expresión equivalente a la (3) con los subíndices actualizados. El proceso prosigue de la misma manera hasta que el valor absoluto de la diferencia relativa entre dos aproximaciones consecutivas sea menor que un determinado número *epsilon*,

$$\frac{|x_{i+1} - x_i|}{|x_{i+1}|} < epsilon \quad (5)$$

La Figura 6.2 ilustra el significado geométrico del método de Newton para hallar la raíz de una función no lineal: se parte de un primer punto en el que se linealiza la función (se sutituye la función por la recta tangente) y se halla la raiz de la función linealizada. En esa raíz se vuelve a linealizar la función original y se halla una nueva aproximación a la verdadera solución. El proceso prosigue hasta que la diferencia entre dos aproximaciones consecutivas es suficientemente pequeña.

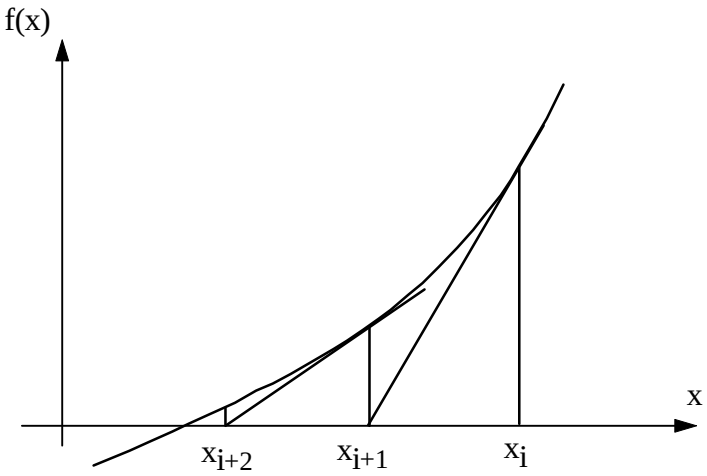


Figura 6.2. Convergencia del método de Newton.

Una vez expuesta la teoría, se pide realizar un programa que tenga las características que se explican a continuación. La Figura 6.3 muestra el aspecto del formulario al arrancar el programa y tras pulsar los botones ***Dibujar*** y ***Resolver***.

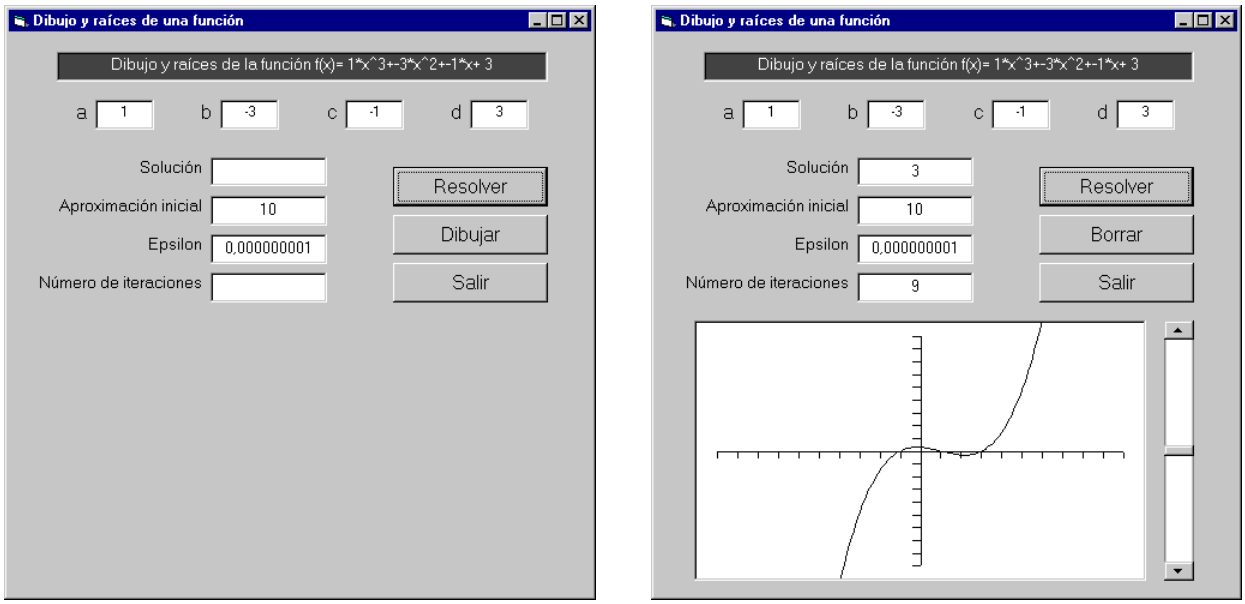


Figura 6.3. Formulario inicial y tras pulsar los botones ***Dibujar*** y ***Resolver***

Los elementos principales de este ejercicio son los siguientes:

1. Un ***label*** de fondo negro y letras blancas donde se representa la expresión del polinomio. Este label debe construirse en función de los valores de los coeficientes del polinomio.

2. Un ***array*** de cuatro ***text boxes*** (con sus cuatro ***labels*** correspondientes) para poder modificar los cuatro coeficientes del polinomio. Cuando se modifica el valor de uno de estos coeficientes y se pulsa ***Intro*** se debe hacer lo siguiente:

   Intro: Se comprueba que la cantidad introducida es numérica, y después se asigna al coeficiente correspondiente del polinomio (***a***, ***b***, ***c*** o ***d***) según el argumento ***Index*** del procedimiento que gestiona el evento ***KeyPress***. Tras aceptar el cambio en el coeficiente se deberá actualizar el ***Caption*** del control ***label*** que contiene la expresión matemática y se deberá también actualizar el dibujo.

3. Tres ***botones*** que corresponderán a:

   Resolver: Al clicar en este botón se calculará una raíz del polinomio a partir de la aproximación inicial dada. Si se desea podrán cambiarse los valores de ***Epsilon*** y de dicha aproximación inicial. Cuando terminen los cálculos deberán aparecer el número de iteraciones que han sido necesarias y la solución para la raíz.

   Dibujar: Al clicar en este botón deberá aparecer la gráfica de la función tal y como se ve en la Figura 6.3 derecha. La barra de desplazamiento vertical es un ***zoom*** para la *picture box* y el botón de ***Dibujar*** se convierte en ***Borrar***. Al clicar en ***Borrar***, desaparece la figura volviendo a la situación de la Figura 6.3 izquierda.

   Salir: Al clicar en este botón, finaliza la aplicación.

4. Cuatro *text boxes* que corresponderán a:

   Solución: Lugar en el que aparecerá la solución (una de las raíces de la función) cuando terminen los cálculos.

   Aproximación inicial: Valor inicial que introducimos para que dé comienzo la iteración.

   Epsilon: Tolerancia al error que introducimos. Se iterará hasta que el error sea menor que ***Epsilon***.

   Número de iteraciones: Número de iteraciones que se han tenido que realizar para obtener la solución con el error ***Epsilon*** introducido.

5. Una ***Picture Box*** en la que se realizarán los dibujos. Se dibujarán los ejes de ordenadas y de abscisas. La escala del eje de abscisas será fija e irá de -10 a +10, con una pequeña raya vertical para indicar cada unidad. La escala del eje de ordenadas se podrá controlar con una barra de desplazamiento vertical, dibujándose también unas pequeñas rayas horizontales que representen las decenas de unidades. Inicialmente la escala vertical variará entre -100 y +100.

6. Una ***barra de desplazamiento*** vertical que permitirá cambiar la escala en el eje de ordenadas

Inicialmente se aplica este método al polinomio $x^3-3x^2-x+3=0$, comenzando a iterar con un valor inicial x=10 y un Epsilon=10^{-9}. La función y su derivada deberán evaluarse a partir de los coeficientes introducidos en las cajas de texto por medio de sendas funciones ***f()*** y ***fd()*** introducidas en el propio formulario.

Crear un directorio llamado ***G:\Infor1\Prac06\Ejer2*** y en él crear un proyecto llamado también ***Newton99.vbp***. El formulario principal se llamará ***Newton99.frm***. Se deberá realizar un módulo llamado ***Dibujar99.bas***, que contendrá un procedimiento para dibujar la función.

Se puede encontrar el fichero ejecutable de este ejercicio en ***Q:\Infor1\Prac06\Ejer2***. Se deberá copiar el fichero ***newton99.exe*** al propio directorio y probar su funcionamiento. Se recomienda observar con atención el funcionamiento del programa ejecutable anterior, viendo la función que realiza cada uno de los botones y controles , y ***pensar*** en cada caso la función que está realizando cada uno de ellos. Cuanto mejor se entienda el funcionamiento de los botones, menos tiempo se requerirá para la generación del código.

6.3 EJERCICIO 3. SIMULACIÓN DE LLENADO DE UN DOBLE DEPÓSITO

En la Figura 6.4 se muestra la geometría de un ejemplo relacionado con el ejercicio 1 de esta práctica. Se trata de dos depósitos iguales y a distinta altura, unidos mediante una tubería a una determinada altura ***hc***, que permite cuando está abierta pasar fluido de un depósito al otro.

Los dos depósitos vienen definidos por las coordenadas de 4 puntos (A1, B1, A2, B2). La tubería de conexión está situada a una altura ***hc*** y tiene una sección de altura ***2e***. La diferencia de altura entre los dos depósitos viene dada por ***h12=A1y-A2y***. Las magnitudes ***h1*** y ***h2*** definen la altura del líquido en los dos depósitos.

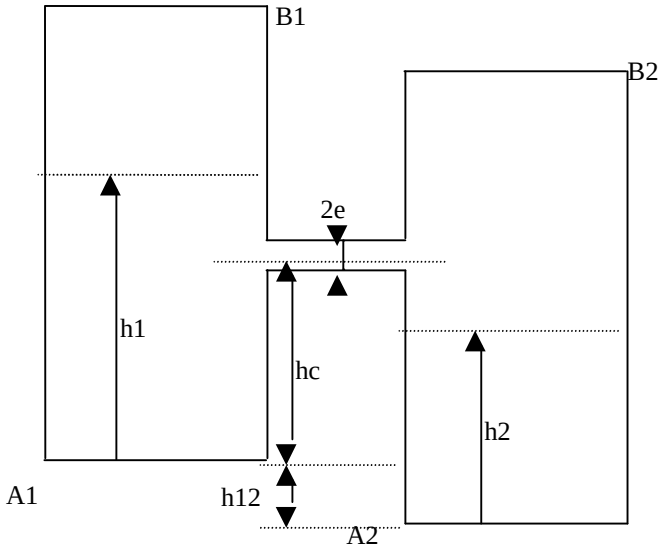


Figura 6.4. Llenado/vaciado de dos depósitos conectados.

Cada depósito tiene una entrada de fluido por la parte superior y un desagüe en la parte inferior. La apertura de las dos entradas, los dos desagües y la tubería de conexión están gobernados por barras de desplazamiento, según se ve en la Figura 6.5.

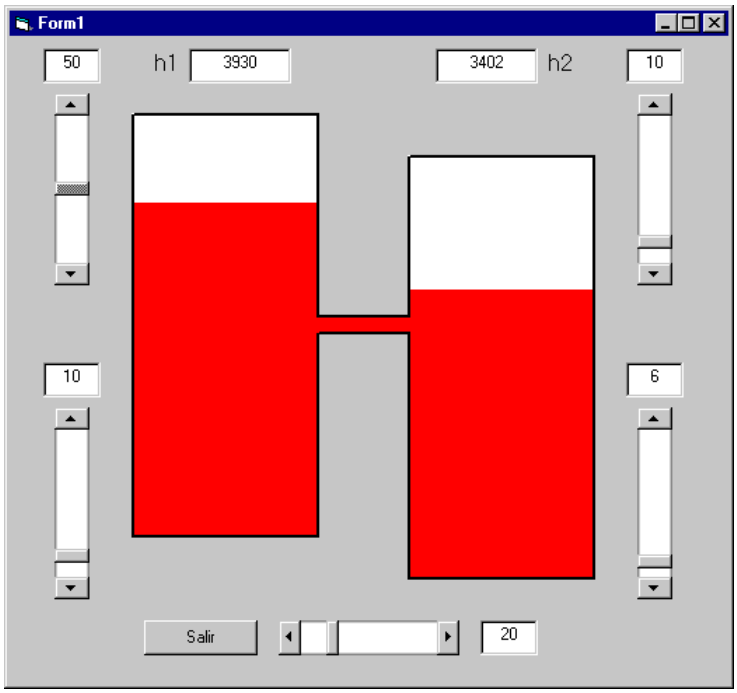


Figura 6.5. Formulario principal del doble depósito.

El caudal de las dos ***entradas superiores*** depende directamente de la ***apertura*** de las válvulas correspondientes, que varía de 0 a 100, con unos incrementos pequeño y grande de 1 y 10, respectivamente.

Sin embargo el caudal de los ***desagües*** depende de su ***apertura*** y de la ***presión del fluido***, que se supone igual a la ***altura*** alcanzada en ese depósito dividida por una constante que vale 1000. De esta forma, aunque la entrada esté más abierta que el desagüe, se alcanza un ***punto de equilibrio*** en el que entra tanto líquido como sale.

El caudal de la ***válvula que une ambos depósitos es*** un poco más complicado de calcular. Se supone que esta válvula está a una altura ***hc***, mientras que ***h1*** y ***h2*** son los niveles en los dos depósitos. El caudal en esta válvula es también proporcional a su ***apertura*** y a la ***diferencia de presiones*** entre ambos lados de la conexión, dividida por una constante que vale asimismo 1000. Este caudal ***q12*** puede ser positivo o negativo; *se supondrá positivo* si pasa del depósito de la izquierda al de la derecha, y *negativo* si circula en sentido contrario. Se pueden presentar los siguientes casos:

1. **h1<hc and h2<hc+h12**. El flujo ***q12*** es cero, porque el nivel en ambos depósitos está por debajo de la altura de la conexión.
2. **h1>=hc and h2<hc+h12**. En este caso puede pasar fluido del depósito de la izquierda al de la derecha. El caudal será positivo y valdrá: ***q12=A3*(h1-hc)/1000***, siendo ***A3*** la apertura de la válvula de paso.
3. **h1<hc and h2>=hc+h12**. En este caso puede pasar fluido del depósito de la derecha al de la izquierda. El caudal será negativo y valdrá: ***q12=A3*(hc+h12-h2)/1000***, siendo ***A3*** la apertura de la válvula de paso.
4. **h1>=hc and h2>=hc+h12**. En este caso puede pasar fluido del depósito de la derecha al de la izquierda o viceversa, dependiendo de las alturas ***h1*** y ***h2***. La altura ***hc*** no influye. El caudal por lo tanto podrá ser positivo o negativo y valdrá: ***q12=A3*(h1-h2)/1000***, siendo ***A3*** la apertura de la válvula de paso.

Las alturas de los dos depósitos son 5000 unidades. La altura ***hc*** es 2500 y ***e*** vale 100. Las coordenadas de los puntos que determinan las conexiones son las siguientes:

A1=(0, 500); B1=(400, 5500); A2=(600, 0); B2=(1000, 5000)

Inicialmente todas las válvulas están cerradas y las alturas iniciales son ***h1***=250 y ***h2***=500.

En el formulario de la Figura 6.5 se pueden ver los distintos controles de este ejercicio. Cada barra de desplazamiento tiene una caja de texto que indica su valor numérico. Se incluyen asimismo dos cajas de texto con dos *labels* ***h1*** y ***h2*** para expresar las alturas correspondientes. El botón Salir permite concluir la ejecución del programa.

Los depósitos están creados por medio de líneas trazadas sobre una *picture box* por medio del comando ***line***. También los fluidos se simulan mediante el comando ***line*** con la opción ***BF*** (*Box Fill*). Se sugiere concentra todas las sentencias de dibujo de los depósitos en un procedimiento llamado ***deposito()***, y todas las sentencias de dibujo de los fluidos en un procedimiento llamado ***fluido()***.

El ejecutable de este ejercicio está en ***Q:\Infor1\Prac06\Ejer3\depo99c.exe***. Crea el directorio ***G:\Infor1\Prac06\Ejer3*** y guarda en él los ficheros de este ejercicio, que se llamarán ***depo99c.vbp*** y ***depo99c.frm***.

7 SÉPTIMA PRÁCTICA

7.1 EJERCICIO 1: DESARROLLO DE UN EDITOR DE TEXTO: PROYECTO MINOTEPAD

El ejecutable de este ejercicio está en ***Q:\Infor1\Prac07\Ejer1\MiNotepad.exe***. Crea el directorio ***G:\Infor1\Prac07\Ejer1*** y guarda en él los ficheros de este ejercicio, que se llamarán ***MiNotepad.vbp*** y ***MiNotepad.frm***.

Se trata de crear un procesador de texto sencillo similar a ***Notepad***. Este procesador de texto se irá creando progresivamente a lo largo de la práctica. En este primer ejercicio se creará ***la versión más sencilla***, que luego se irá progresivamente ampliando.

Desde el punto de vista de los controles este ejemplo tiene solamente un ***formulario***, una ***caja de texto*** que ocupa todo el espacio disponible en el formulario, y unos ***menús***.

Es muy importante que la caja de texto ocupe siempre el máximo espacio disponible en el formulario. Para ello se puede utilizar el evento ***resize*** del formulario.

En este ejercicio se deben poder utilizar los comandos ***New***, ***Open***, ***Save***, ***Save As*** y ***Exit***, de acuerdo con las siguientes descripciones:

- ***New***. Este comando inicializa simplemente la caja de texto a la cadena vacía "". Hasta que el fichero se guarde por primera vez en la ***barra de título*** aparecerá el mensaje "***Untitled – MiNotepad***".
- ***Open***. Este comando abre una caja de diálogo –un ***Common Dialog Control***[3]- para seleccionar un fichero de texto, lo lee y muestra su contenido en al caja de texto. Se establecerá un ***filtro*** para seleccionar bien los ficheros de texto, bien todos los ficheros ("Texto (*.txt)|*.txt|All Files (*.*)|*.*"). Una vez abierto el fichero, en la barra de título aparecerá un mensaje con el nombre del fichero y el nombre de la aplicación, en la forma "***Fichero.txt – MiNotepad***".

- ***Save***. Actualiza el contenido del fichero de acuerdo con el texto actual de la caja de texto. Si el fichero existe, simplemente lo actualiza; si no existe, realiza la misma función que ***Save As***.
- ***Save As***. Utilizando el mismo ***Common Dialog Control*** se abre el cuadro de diálogo ***Save As*** típico de todas las aplicaciones de ***Windows***. La extensión por defecto (la que se utiliza si el usuario no pone otra) será ****.txt***. Después de guardar el fichero en el disco, en la barra de título aparecerá el mensaje "***Fichero.txt – MiNotepad***".
- ***Exit***. .Con este comando la aplicación se termina tal y como esté, es decir sin salvar y sin hacer ningún tipo de averiguación adicional.

Además se deberá poder cambiar el ***Font*** y el ***color de fondo***, utilizando también el ***Common Dialog Control***, con los comandos ***ShowFont*** y ***ShowColor*** en los eventos ***Click*** de los item ***Set Font*** y ***Set Bgnd Color***, que están en el menú ***Edit***. El programa deberá funcionar a este respecto igual que el modelo, incluyendo la posibilidad de efectos especiales (color del texto, tachado, y subrayado).

---

[3] Si el ***Common Dialog Control*** no aparece en el ***ToolBox*** de ***Visual Basic 6.0***, se puede hacer que aparezca por medio del comando ***Components*** en el menú ***Projects*** (se llama exactamente ***Microsoft Common Dialog Control 6.0***).

7.2 EJERCICIO 2: INTRODUCCIÓN DE MEJORAS: TENER EN CUENTA SI EL TEXTO SE HA MODIFICADO, Y NO CERRAR LA APLICACIÓN SIN AVISAR QUE SE PUEDE PERDER INFORMACIÓN (PROYECTO MINOTEPAD2)

Comienza este ejercicio creando en el directorio ***G:\Infor1\Prac07*** una carpeta llamada ***Ejer2*** y copia en ella los ficheros del ejercicio anterior. Después, cámbiales el nombre y haz que se llamen ***MiNotepad2.vbp*** y ***MiNotepad2.frm***. Cuando hayas hecho estos cambios ya podrás abrirlos y seguir trabajando sobre ellos.

El ejecutable de este ejercicio está en el directorio ***Q:\Infor1\Prac07\Ejer2*** y se llama ***MiNotepad2.exe***. Cópialo a tu directorio y observa detenidamente cómo funciona.

La principal diferencia de este ejercicio con el anterior es la programación de una serie de medidas de seguridad para no perder información involuntariamente. Se puede perder información siempre que el contenido de la caja de texto esté sin guardar en el disco o siempre que se haya introducido alguna modificación en el texto y no se haya actualizado el fichero de disco. Para establecer estas medidas de seguridad se hará lo siguiente:

- Se creará una variable de tipo ***boolean*** llamada ***Actualizado***. Siempre que el fichero contenga el mismo texto que la caja de texto esta variable valdrá ***True***. Cuando la caja de texto haya sufrido algún cambio y el fichero no esté actualizado, dicha variable valdrá ***False***. Para simplificar, aunque el cambio en la caja de texto la deje como estaba (por ejemplo, añadir una letra y luego borrarla), la variable ***Actualizado*** valdrá ***False***.
- Algunos editores de texto ***indican de un modo visual*** que el fichero no está actualizado (no es el caso de ***Notepad***). En este programa cuando el fichero esté sin actualizar aparecerá un ***asterisco*** (*) en última posición de la barra de títulos, justamente al final del nombre de la aplicación. Por ejemplo, con un fichero llamado ***Prueba1.txt*** la barra de títulos deberá mostrar "***Prueba1.txt – MiNotepad***" si el fichero está actualizado y "***Prueba1.txt – MiNotepad****" si no lo está. Para conseguir esto habrá que modificar la barra de títulos cada vez que se introduzca un cambio en el texto (poner el asterisco) y cada vez que se guarde el texto en el disco con ***Save*** o ***Save As*** (quitar el asterisco).
- La segunda medida de seguridad será pedir confirmación antes de cerrar el programa cuando está en la situación de ***Actualizado*** = ***False***. La información se puede perder en los supuestos siguientes:
  - Al ***cerrar la aplicación***. En este caso lo mejor es utilizar el evento ***QueryUnload*** de los formularios. El procedimiento para gestionar este evento tiene dos argumentos. Para este ejercicio sólo el primero, llamado ***Cancel***, tiene importancia[4]. Si ***Cancel*** recibe cualquier valor distinto de cero, no se prosigue con el proceso de descarga del formulario. El evento ***QueryUnload*** deberá sacar un mensaje de confirmación tal como el que se ve en la Figura 7.1. Si se pulsa ***Yes*** se actualiza el fichero y se sale de la aplicación; si se pulsa ***No*** se sale de la aplicación sin actualizar el fichero y si se pulsa ***Cancel*** se vuelve a la aplicación sin hacer nada (argumento ***Cancel*** del evento ***QueryUnload*** distinto de cero).

[4] El segundo argumento ***unloadMode*** sirve para saber de dónde viene la orden de descargar el formulario.

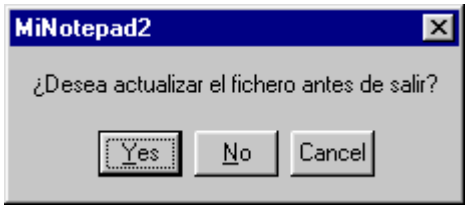


Figura 7.1.

Figura 7.2

- Teniendo un fichero abierto y sin actualizar, al abrir un nuevo fichero con ***File/Open*** o al crear un nuevo texto con ***File/New*** debe ofrecer la opción de guardar los cambios en el texto anterior. Esta posibilidad debe gestionarse directamente en el evento ***Click*** de los comandos de menú correspondientes. En este caso aparecerá una caja de mensajes ligeramente diferente, mostrada asimismo en la Figura 7.2.

7.3 EJERCICIO 3: INTRODUCCIÓN DE MEJORAS: BÚSQUEDA DE TEXTO (PROYECTO MINOTEPAD3)

Comienza este ejercicio creando en el directorio ***G:\Infor1\Prac07*** una carpeta llamada ***Ejer3*** y copia en ella los ficheros del ejercicio anterior. Después, cámbiales el nombre y haz que se llamen ***MiNotepad3.vbp*** y ***MiNotepad3.frm***. Cuando hayas hecho estos cambios ya podrás abrirlos y seguir trabajando sobre ellos.

El ejecutable de este ejercicio está en el directorio ***Q:\Infor1\Prac07\Ejer3*** y se llama ***MiNotepad3.exe***. Cópialo a tu directorio y observa detenidamente cómo funciona.

En primer lugar se trata de encontrar un texto (unas palabras o una frase) dentro del texto contenido en la caja de texto. Al elegir el comando ***Find*** en el menú ***Search*** se abre una caja de diálogo (que en realidad es un formulario) tal como la mostrada en la Figura 7.3 no hace falta buscar en ambas direcciones ni poder distinguir entre mayúsculas y minúsculas. Esas posibilidades aparecen en el formulario con fines únicamente "decorativos"). El usuario tecleará el texto a encontrar y pulsará el botón ***Find Next***. En ese momento el programa localiza la primera aparición del texto buscado y lo selecciona para que aparezca visible. Si no se encuentra el texto buscado aparece el mensaje de la Figura 7.4 Si se encuentra y el usuario vuelve a pulsar en ***Find Next*** busca la siguiente aparición y así hasta llegar al fin del fichero. Al llegar al final del fichero aparece un mensaje que lo indica y que pregunta si se quiere recomenzar la búsqueda desde el principio, como se muestra en la Figura 7.5

Figura 7.3.

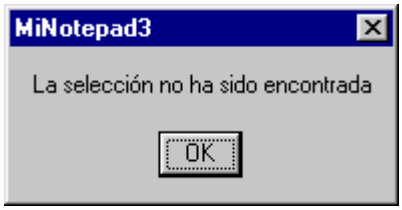


Figura 7.4.

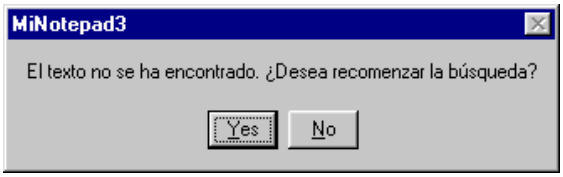


Figura 7.5.

Este ejercicio es un poco más complicado de lo habitual. Para seleccionar un texto en una caja de texto se debe dar valor a las propiedades ***SelStart*** y ***SelLength*** de dicha caja de texto. De todas formas esto no es suficiente: es necesario poner a ***False*** la propiedad ***HideSelection*** de la caja de texto, para que el texto aparezca seleccionado aunque el control de que se trate –la caja de texto- no tenga el ***focus***.

Conviene también que el formulario ***frmFind*** (suponiendo que se llame así el formulario de la Figura 7.3 de búsqueda de texto siempre esté visible por encima del formulario principal, aunque sea éste el formulario activo. Esto se consigue abriéndolo de una forma especial cuando el usuario ejecuta el comando ***Search/Find***. El código del procedimiento que se ejecuta es el siguiente (suponiendo que el formulario principal se llame ***Form1***):

```
Private Sub mnuSearchFind_Click()
    ' abre una ventana que no es modal, pero que está
    '   siempre delante de form1 (depende de form1)
    frmFind.Show vbModeless, Form1
End Sub
```

Por otra parte, es conveniente que el formulario ***frmFind*** no aparezca en la barra de tareas de ***Windows*** como si fuera una aplicación más. Ello se consigue modificando la propiedad ***ShowInTaskbar*** en la forma ***frmFind.ShowInTaskbar=False***.