

Objetivos:

- ❖ Profundizar en el uso de **subprogramas** y el paso de **parámetros por referencia**
- ❖ Profundizar en la manipulación de **cadenas de caracteres**
- ❖ Evento de **carga de un formulario**
- ❖ Propiedades **Locked** y **MaxLength** de los cuadros de texto
- ❖ **Dar formato** a los datos mediante la función **Format**

Programa completo de resolución de ecuaciones de 2º grado

Interfaz

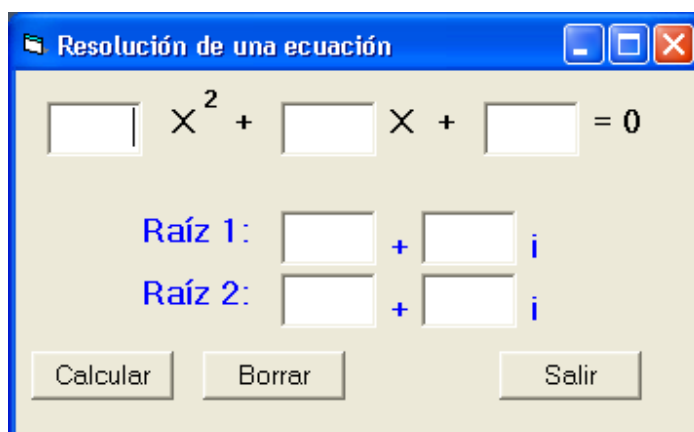


Figura 10.1 Objetos presentes en la interfaz de la calculadora de ecuaciones.

Funcionamiento

El funcionamiento de esta calculadora para resolver ecuaciones de primero y segundo grado es similar al visto en el Laboratorio 4 salvo que ahora vamos a considerar también las **soluciones complejas**. Es recomendable partir de la solución ya vista tanto para el diseño gráfico como para la codificación, ya que la solución será algorítmicamente similar.

Pasos a seguir

1. Crearemos los objetos del tipo y forma mostrados en la Figura 10.1. Sólo daremos un nombre particular a aquéllos que nos interesa leer o modificar sus propiedades en algún momento del programa. En la Figura 10.2 se muestran en rojo los nombres propuestos para estos objetos.
2. Añadir el código a los eventos, es decir, la carga del formulario y el clic sobre los botones:

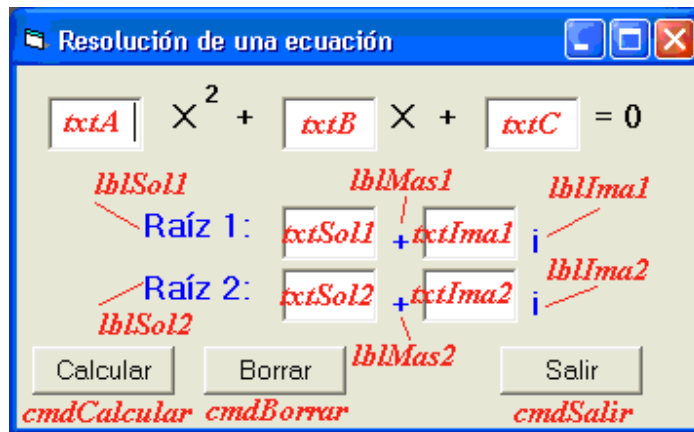


Figura 10.2 Nombres de los objetos en la interfaz de la calculadora de ecuaciones.

- a) **Código de carga del formulario:** para especificar el código de carga del formulario haremos clic dos veces sobre el formulario. Lo que conviene hacer es bloquear las cajas de texto de la solución (de una vez por todas) mediante la propiedad **Locked** (nótese que en el Laboratorio 3 utilizábamos la propiedad **Enabled** que tiene la lógica **True-False** inversa). También pondremos todos los objetos asociados a la solución invisibles (que haremos visibles cuando sea preciso):

```
Sub Form_Load()
    Call ponerSolucionInvisible
    txtSol1.Locked = True
    txtSol2.Locked = True
    txtIma1.Locked = True
    txtIma2.Locked = True
End Sub
```

Se observa que al principio de este procedimiento hemos puesto una llamada (**Call**) al procedimiento **ponerSolucionInvisible**, que es un subprograma que escribiremos inmediatamente a continuación.

Lo que tendremos que hacer es especificar uno a uno que los objetos relacionados con la solución (**lblSol1**, **lblSol2**, **txtSol1**, **txtSol2**, **lblMas1**, **lblMas2**, **txtIma1**, **txtIma2**, **lblIma1** y **lblIma2**) van a estar invisibles, es decir, van a tener la propiedad **visible** a **False**. Luego, dependiendo del tipo de solución, haremos visibles de manera selectiva algunos de ellos, e incluso cambiaremos las etiquetas según convenga.

El esqueleto de este procedimiento se da a continuación (habrá que completarlo):

```
Sub ponerSolucionInvisible()
    ...
End Sub
```

- b) **Código del botón Calcular:** se **controlará la validez de los coeficientes** (han de ser numéricos) y se llamará al procedimiento **CalcularEcuacion** cuya cabecera se muestra en la Figura 10.3.

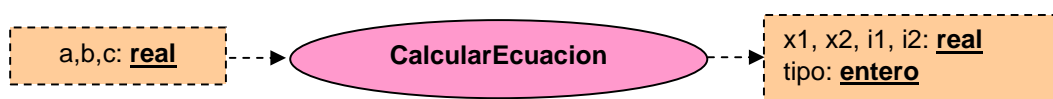


Figura 10.3. Cabecera del procedimiento **CalcularEcuacion**.

Los parámetros de entrada son los coeficientes **a**, **b** y **c** de la ecuación.

El parámetro de salida **tipo** determinará el tipo de ecuación de acuerdo con los valores posibles de la Tabla 10.1.

Tipo	Descripción
1	Ecuación Real de Segundo Grado
2	Ecuación Compleja de Segundo Grado
3	Ecuación de Primer Grado
4	No es una ecuación

Tabla 10.1. Tipos de ecuación.

Aunque en el Laboratorio 3 no tuvimos en cuenta las soluciones complejas, ahora sí lo vamos a hacer.

El resto de los parámetros de salida contienen la solución, dependiendo del tipo:

- Cuando la solución sea una ecuación **real** de **segundo grado** **x1** y **x2** albergarán estas soluciones.
 - Cuando la solución sea **compleja** de **segundo grado** **x1** y **x2** albergarán la parte **real**, obteniéndose en **i1** e **i2** las partes **imaginarias**.
 - Cuando la solución sea de **primer grado** **x1** albergará la solución.
 - Cuando **no sea una ecuación**, ninguno de los cuatro valores de salida **x1**, **x2**, **i1** o **i2** tendrán un valor significativo.
- c) Se dará formato a los resultados mediante la función **Format** que es similar a **CStr** permitiendo especificar el número de decimales. Por ejemplo, mediante el modo "0.00" mostraremos dos valores decimales. A continuación se harán visibles los objetos gráficos que corresponda.

```

Sub cmdCalcular_Click()
    Dim a As Double, b As Double, c As Double
    Dim x1 As Double, x2 As Double
    Dim i1 As Double, i2 As Double
    Dim tipo As Integer
    ' Obtener los valores de los coeficientes
    If IsNumeric(txtA.Text) And _
        IsNumeric(txtB.Text) And _
        IsNumeric(txtC.Text) Then
        ' Los valores son numéricos
        a = Cdbl (txtA.Text)
        b = Cdbl (txtB.Text)
        c = Cdbl (txtC.Text)
        Call CalcularEcuacion(a, b, c, x1, x2, i1, i2, tipo)
        If tipo = 1
            txtSol1.Text = Format(x1, "0.00")
            txtSol2.Text = Format(x2, "0.00")
            Call ponerEcuacionRealSegundoGrado
        ElseIf tipo = 2
            txtSol1.Text = Format(x1, "0.00")
            txtSol2.Text = Format(x2, "0.00")
            txtIma1.Text = Format(i1, "0.00")

```

```

txtIma2.Text = Format(i2, "0.00")
Call ponerEcuacionImagSegundoGrado
ElseIf tipo = 3
    txtSol1.Text = Format(x1, "0.00")
    Call ponerEcuacionPrimerGrado
ElseIf tipo = 4
    MsgBox "Error: No es una ecuación"
Else
    MsgBox "Error de programa: tipo de ecuación incorrecto"
End If
Else
    MsgBox "Error: Coeficientes no numéricos"
End If
End Sub

```

El código asociado al subprograma ponerEcuacionRealSegundoGrado se encarga de hacer visibles los objetos necesarios para mostrar la solución de las raíces reales, y es el siguiente (habrá que completarlo):

```

Sub ponerEcuacionRealSegundoGrado()
    Call bloquearABC
    lblSol1.Caption = "Raíz 1:"
    lblSol2.Caption = "Raíz 2:"
    ...
End Sub

```

Siempre que vayamos a mostrar la solución a una ecuación vamos a **bloquear** los objetos de **entrada de los coeficientes** de la ecuación, ya que si permitimos modificarlos no serán consistentes con las soluciones mostradas. Este bloqueo lo haremos con el subprograma bloquearABC. El código para esto será (a completar):

```

Sub bloquearABC ()
    txtA.Locked = True
    ...
End Sub

```

Cuando se trata de una ecuación de segundo grado con soluciones imaginarias tendremos que hacer visibles los mismos objetos que los de la solución anterior, lo cual haremos mediante un nuevo subprograma al que llamaremos ponerEcuacionImagSegundoGrado y luego llamaremos al subprograma anterior ponerEcuacionRealSegundoGrado, además de visualizar los objetos correspondientes a la parte imaginaria (a completar):

```

Sub ponerEcuacionImagSegundoGrado ()
    Call ponerEcuacionRealSegundoGrado
    lblMas1.Visible = True
    ...
End Sub

```

Finalmente, cuando el coeficiente **a** es nulo nos encontramos ante una ecuación lineal, que sólo tendrá una raíz. Con esto finalizaremos el código asociado al botón **Calcular** y añadiremos un subprograma para visualizar una única raíz. El código del subprograma ponerEcuacionPrimerGrado es más sencillo que los anteriores:

```

Sub ponerEcuacionPrimerGrado ()
    Call bloquearABC
    lblSol1.Caption = "Raíz:"
    ...
End Sub

```

Código del botón Borrar: antes hemos creado un subprograma para borrar todos los objetos de la solución al cargar el formulario llamada `ponerSolucionInvisible`. Además de eso hemos de desbloquear los campos de los coeficientes **a**, **b** y **c**, poniéndolos vacíos:

```

Sub cmdBorrar_Click ()
    Call ponerSolucionInvisible
    Call desbloquearABC
End Sub

```

El subprograma para desbloquear los coeficientes es fácilmente adivinable:

```

Sub desbloquearABC ()
    txtA.Locked = False
    txtA.Text = ""
    ...
End Sub

```

Ejercicio 10.1: Control de letra del DNI

Interfaz



Figura 10.4. Comprobador de la letra del DNI.

La letra del DNI sirve para comprobar que se ha introducido correctamente el número.

Se calcula obteniendo el resto de dividir el número de DNI por 23 y el número resultante será la posición de la letra (empezando por 0) en la cadena: TRWAGMYFPDXBNJZSQVHLCKE.

Así, para el DNI 12345678 haríamos:

- $n = 12345678 \bmod 23$

Nos da 15 y se comprobará que en la posición 16 (empezando por 1) en la cadena dada se encuentra la letra Z.

Se pide:

- Verificar que se introducen los 8 dígitos del DNI

- Verificar que todos los caracteres introducidos son dígitos del 0 al 9
- Verificar que se ha introducido la letra (en mayúsculas o minúsculas)
- Calcular la letra correspondiente al DNI
- Decir si la letra introducida es correcta (la calculada) o incorrecta
- Compruébese con el DNI de uno mismo

Para asegurarnos de que no se introducen en cada campo más dígitos de los necesarios se limitará el contenido del campo a los caracteres correspondientes (8 y 1 respectivamente) mediante la propiedad `MaxLength` de los cuadros de texto.

Ejercicio 10.2: Control de cuentas bancarias

Interfaz

Figura 10.5. Comprobador de cuenta bancaria.

Un número de cuenta bancaria completo se compone de 20 dígitos (d_{19} a d_0) que se corresponden con los conceptos expresados en la Tabla 10.2.

Entidad				Oficina				Control		Número de cuenta									
d_{19}	d_{18}	d_{17}	d_{16}	d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

Tabla 10.2. Dígitos de una cuenta bancaria.

Así, los cuatro primeros dígitos de una cuenta se corresponden con la **entidad**, por ejemplo, el código de “Caja Vital”; los cuatro dígitos siguientes hacen referencia al código único de una **oficina** o sucursal de esa entidad; los dos dígitos siguientes se llaman “**dígitos de control**” y

sirven para controlar la corrección de la cuenta completa; los diez últimos números se corresponden con el **número de cuenta**.

Los dos dígitos de control se calculan a partir de los otros 18 dígitos y sólo una de las 100 combinaciones posibles (de “00” a “99”) es válida. Más en concreto, el primer dígito de control (d_{11}) se corresponde con los ocho primeros dígitos (todas las cuentas de la misma oficina tienen el mismo dígito) y el segundo dígito de control (d_{10}) se corresponde con el número de cuenta.

El método de cálculo de los dígitos de control no es infalible pero permite detectar que se ha intercambiado el orden de dos dígitos consecutivos. Consiste en sumar los dígitos ponderados por un coeficiente y luego obtener un solo dígito a partir del número resultante. La Tabla 10.3 recoge los coeficientes correspondientes a cada dígito.

Entidad				Oficina				Control		Número de cuenta									
d_{19}	d_{18}	d_{17}	d_{16}	d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0
4	8	5	10	9	7	3	6	-	-	1	2	4	8	5	10	9	7	3	6

Tabla 10.3. Coeficientes para el cálculo del dígito de control.

Para los dígitos de control d_{11} y d_{10} sumaremos respectivamente:

- $s_{11} = 4 \cdot d_{19} + 8 \cdot d_{18} + 5 \cdot d_{17} + 10 \cdot d_{16} + 9 \cdot d_{15} + 7 \cdot d_{14} + 3 \cdot d_{13} + 6 \cdot d_{12}$
- $s_{10} = d_9 + 2 \cdot d_8 + 4 \cdot d_7 + 8 \cdot d_6 + 5 \cdot d_5 + 10 \cdot d_4 + 9 \cdot d_3 + 7 \cdot d_2 + 3 \cdot d_1 + 6 \cdot d_0$

El cálculo que se realizará a continuación será para d_{11} :

- $d_{11} = 11 - (s_{11} \bmod 11)$

Si el resultado es mayor que 9 se hará:

- $d_{11} = 11 - d_{11}$

De manera similar, para d_{10} haremos:

- $d_{10} = 11 - (s_{10} \bmod 11)$

Si el resultado es mayor que 9 se hará:

- $d_{10} = 11 - d_{10}$

Otros detalles

Se controlará que todos los caracteres son dígitos y que se introducen los necesarios. Para asegurarnos de que no se introducen en cada campo más dígitos de los necesarios se limitará el contenido del campo a los caracteres correspondientes (4, 4, 2 y 10 respectivamente) mediante la propiedad **MaxLength** de los respectivos cuadros de texto.

Mid (ByVal <i>cad</i> As String , ByVal <i>ini</i> As Long , [ByVal <i>lon</i> As Long]) As String	Subcadena desde <i>ini</i> hasta la longitud indicada
Len (ByVal <i>str</i> As String) As Integer	Longitud de <i>str</i> (puede utilizarse con otros tipos)
Ucase (ByVal <i>exp</i> As String) As String Lcase (ByVal <i>exp</i> As String) As String	Convierte la <i>exp</i> a Mayúsculas o Minúsculas.
Format (ByVal <i>num</i> As Double , ByVal <i>fmt</i> As String) As String	Da formato al número <i>num</i> utilizando el formato <i>fmt</i> (puede utilizarse con otros tipos, por ejemplo fechas)

Tabla 10.4. Lista de funciones relevantes de Visual Basic