Escuela Universitaria de Ingenieria Vitoria-Gasteiz
Ingeniaritzako Unibertsitate Eskola Vitoria-Gasteiz

Fundamentals of Computer Science
2010-2011
Laboratory 10
Functions and procedures (3)

eman ta zabal zazu

Objectives:

- To go deeply into the use of **subprograms** and passing parameters **by reference**
- To go deeply into the manipulation of **strings**
- **`Form_Load`** event
- **`Locked`** and **`MaxLength`** properties of text boxes
- Data formatting using **`Format`**

Complete program to solve a 2nd degree equation

Interface

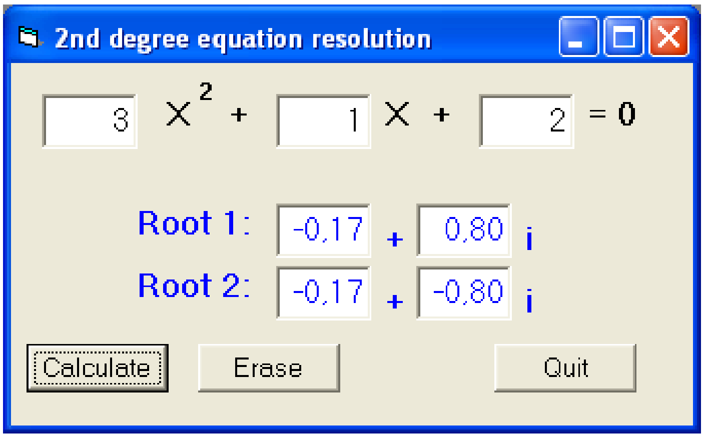

Figure 10.1 Objects present in the interface of the equation calculator.

Operation

This program is similar to that of Laboratory 4 to solve 1st and 2nd degree equations, apart from considering as well **complex solutions**. Additionally, instead of using the **`Enabled`** property to prevent the modification it illustrates the use of the **`Locked`** property.

It is recommended to reuse the original simplified version as the solution will be algorithmically similar.

Steps

1. Create the objects of type and aspect as the ones shown in Figure 10.1. We shall only give a particular name to the ones that are going to be used for any reason in the program, both for reading or for modifying their properties at some point in the program. In Figure 10.2 we show the names used in the proposed resolution.

2. Add the code to the events, in our case the form loading and the click on the command buttons:

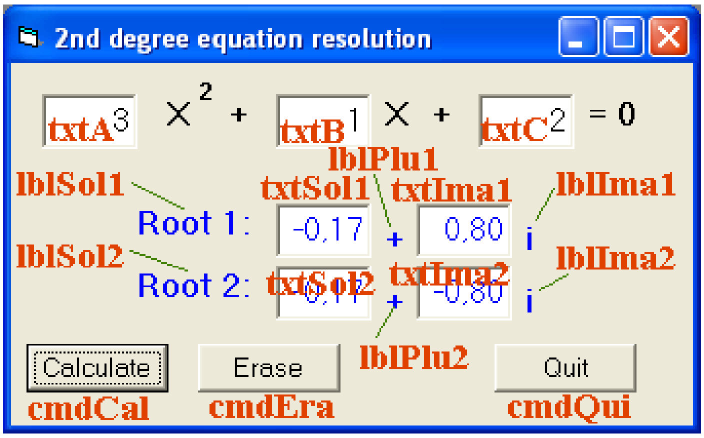

Figure 10.2 Name of the objects in the equation calculator.

- **Code for the form load**: to specify the code to be executed when the form is loaded we double click on the form during the design. In our particular program what we need to do is to block all the text boxes for the solution (once for ever) using the **`Locked`** property (note that in Lab 3 we used the property **`Enabled`** which has the **`True`-`False`** logic inverted). We shall also set all the objects associated with the solution to invisible (they will be made visible when appropriate):

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

It can be observed that at the beginning of this procedure we have put a call to the **`setSolutionInvisible`** procedure, which is a subprogram that we shall write just after this. In this subprogram we specify one by one that the objects related with the solution (**`lblSol1`**, **`lblSol2`**, **`txtSol1`**, **`txtSol2`**, **`lblMas1`**, **`lablMas2`**, **`txtIma1`**, **`txtIma2`**, **`lblIma1`** y **`lblIma2`**) are going to be invisible, that is, their **`Visible`** will be set to **`False`**. After, depending on the type of solution, we shall make visible some of them and we shall even change the labels as necessary.

The skeleton for this procedure follows (it needs completion):

```
Sub setSolutionInvisible()
   ...
End Sub
```

- **Code for Calculate button**: we control the validity of the coefficients (they must be numeric) and we call the `CalEquation` procedure which has the header shown in Figure 10.3.

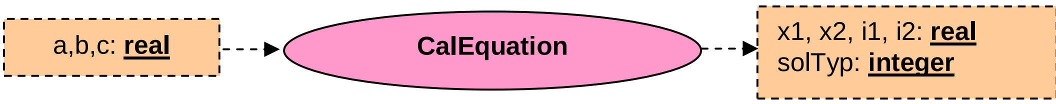

Figure 10.3 Header of the `CalEquation` procedure.

The input parameters are the **`a`**, **`b`** and **`c`** coefficients of the equation.

The **`solTyp`** output parameter determines the type of equation among the possible ones shown in Table 10.1.

solTyp	Description
1	**Real** equation of 2nd degree
2	**Complex** equation of 2nd degree
3	1st degree equation
4	Not an equation

Table 10.1 Equation types.

Although in Lab 3 we didn't take into account the complex solutions now we are going to do it.

The rest of the parameters contain the solution, depending on solTyp:

- When the solution is a **real** equation of **2nd degree** **`x1`** and **`x2`** will get these solutions.
- When the solution is a **complex** equation of **2nd degree** **`x1`** and **`x2`** will get the **real** part, while **`i1`** and **`i2`** will get the **imaginary** parts.
- When the solution is a 1st degree equation **`x1`** will get the solution.
- When it is not an equation, none of the output parameters **`x1`**, **`x2`**, **`i1`** or **`i2`** will get a significant value.

- To give the appropriate format to the data we shall use the VB **`Format`** function by means of the "`0.00`" mode, which supposes converting to a string with two decimal values. After this we shall make visible the corresponding graphical objects.

```
Sub cmdCal_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 solTyp As Integer
   ' Obtain the values of the coefficients
  If IsNumeric(txtA.Text) And _
     IsNumeric(txtB.Text) And _
     IsNumeric(txtC.Text) Then
      ' The values must be numeric
    a = CDbl(txtA.Text)
    b = CDbl (txtB.Text)
    c = CDbl (txtC.Text)
    Call CalEquation(a, b, c, x1, x2, i1, i2, solTyp)
    If solType = 1
      txtSol1.Text = Format(x1, "0.00")
      txtSol2.Text = Format(x2, "0.00")
      Call set2ndDegreeReal
    ElseIf SolType = 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 set2ndDegreeImag
    ElseIf SolType = 3
      txtSol1.Text = Format(x1, "0.00")
      Call set1stDegree
    ElseIf SolType = = 4
      MsgBox "Error: Not an equation"
    Else
      MsgBox "Program error: incorrect equation type"
    End If
  Else
    MsgBox "Error: non-numeric coefficients"
  End If
End Sub
```

The code associated with the **set2ndDegreeReal** subprogram makes visible the objects necessary to show the 2nd degree real roots and looks as follows (it needs completion):

```
Sub set2ndDegreeReal()
  Call blockABC
  lblSol1.Caption = "Root 1:"
  lblSol2.Caption = "Root 2:"
  ...
End Sub
```

Whenever we show a solution we block the input coefficients objects so that they are always consistent with the solutions shown. To do so we use the **blockABC** subprogram with the code (to be completed):

```
Sub blockABC ()
  txtA.Locked = True
   ...
End Sub
```

When we are dealing with a 2nd degree equation with imaginary solutions we call the **set2ndDegreeImag** subprogram. We first make visible the same objects as in the previous case by calling the **set2ndDegreeReal** and on top of them we make visible the rest of objects, as follows (to be completed):

```
Sub set2ndDegreeImag ()
  Call set2ndDegreeReal
  lblMas1.Visible = True
   ...
End Sub
```

Finally, when the **a** coefficient is null we will have a lineal solution, with only one root. We add a subprogram to make visible only one solution. With this we finish the code associated with the cmdCal button. The code for **set1stDegree** is simpler than the previous ones:

```
Sub set1stDegree ()
  Call blockABC
  lblSol1.Caption = "Root:"
   ...
End Sub
```

- **Code for button cmdEra**: Before we have created a subprogram to erase all the objects for the solution while loading the form, called **setSolutionInvisible**. On top of this we unblock the fields for the coefficients **a**, **b** and **c**, setting them empty:

```
Sub cmdEra_Click ()
  Call setSolutionInvisible
  Call unBlockABC
End Sub
```

The subprogram to unblock the coefficients can be easily guessed:

```
Sub unBlockABC ()
  txtA.Locked = False
  txtA.Text = ""
  ...
End Sub
```

Exercise 10.1: NIC letter control

Interface

Figure 10.4 Check the Spanish NIC.

The Spanish NIC (National Identifying Card) has a redundant letter to check if it has been correctly introduced. It is obtained by calculating the rest of the division of the number by 23, and the resulting number will be the position (starting from 0) in the string: TRWAGMYFPDXBNJZSQVHLCKE.

This way, for a NIC 12345678 we would calculate:

- n = 12345678 **Mod** 23

We get 15, so we check position 16 (starting from 1) in the given string, obtaining letter Z.

For the exercise it is asked:

- Verify that 8 digits have been input
- Verify that all input characters are digits from "0" to "9"
- Verify that a letter has been introduced (lower case or upper case)
- Calculate the NIC letter
- Say if the input letter is correct (the calculated one) or incorrect
- Check someone's Spanish NIC

We ensure that in the corresponding field no more than the required characters are introduced (8 and 1 respectively) we set the `MaxLength` property of the text fields.

Exercise 10.2: Bank account number check

Interface

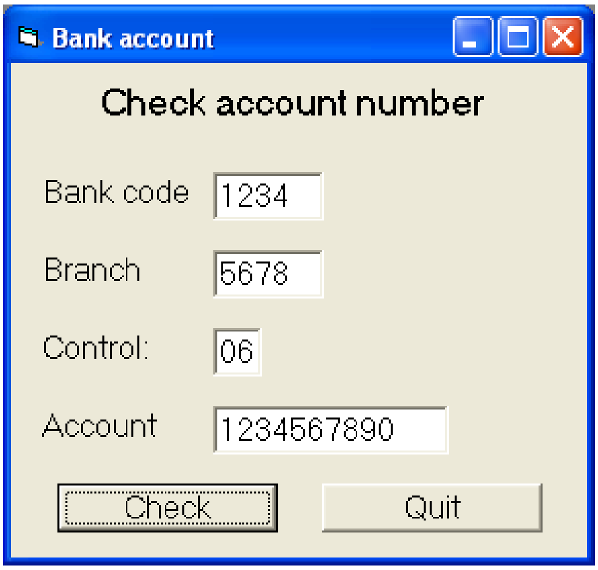

Figure 10.5 Bank account number checker.

A complete bank account number is composed of 20 digits (d_{19} to d_0) which correspond to the concepts expressed in Table 10.2.

Bank				Branch				Control		Account number									
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

Table 10.1 Digits for an account number.

This way, the first four digits of an account number are the bank code, for example, "Caja Vital"; the following four digits refer to a unique code for each branch of that bank; the two following digits are called the "control digits" and are useful to check the correction of the complete account number; the ten trailing digits are the account number, unique for a given branch of a bank.

The two control digits are calculated after the other 18 digits and only one of the 100 possible combinations (from "00" to "99") is valid. More precisely, the first control digit (d_{11}) corresponds to the first eight digits (all accounts in an office share the same digit) and the second control digit (d_{10}) corresponds to the account number.

The calculation method for the control digits is not infallible but enables the detection of two different consecutive digits that are exchanged. It consists of adding the digit numbers weighted up by a coefficient and after obtaining a single digit after the resulting number. Table 10.3 shows the weights corresponding to each digit position.

Bank				Branch				Control		Account number									
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

Table 10.3 Coefficient weights for the control digit calculation.

For control digits d_{11} and d_{10} we shall add up respectively:

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

The calculation to be carried out then for d_{11} is:

- $d_{11} = 11 - (s_{11}$ **Mod** $11)$

If the result is greater than 9 we obtain:

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

Similarly, for d_{10} we obtain:

- $d_{10} = 11 - (s_{10}$ **Mod** $11)$

If the result is greater than 9 we also get:

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

Other details

We must control that all characters are digits and that all the necessary digits are input. To ensure that we don't exceed the limit we limit tha corresponding fields to the maximum values (4, 4, 2 and 10 respectively) by means of the **`MaxLength`** property.

`Mid (ByVal cad As String, ByVal ini As Long, [ByVal len As Long]) As String`	Substring from `ini` with `len` length
`Len (ByVal str As String}) As Integer`	Length of `str` (it can also be used with other types)
`Ucase (ByVal exp As String) As String` `Lcase (ByVal exp As String) As String`	Convert `exp` to Uppercase or Lowercase.
`Format (ByVal num As Double, ByVal fmt As String) As String`	Give format to `num` using style `fmt` (it may be used with other types, for example dates)

Table 10.4 List of relevant functions in Visual Basic