Fundamentals of Computer Science
2010-2011
Laboratory 11
Vectors (1)

Objectives:
- Acquire abilities in the use of **vectors** (one-dimensional arrays)
- Get to know the generation of pseudo-random numbers
- Lower and Upper bound of a VB vector: **LBound** and **UBound**
- Initialise a vector using the **Array** instruction

Random numbers program

Interface

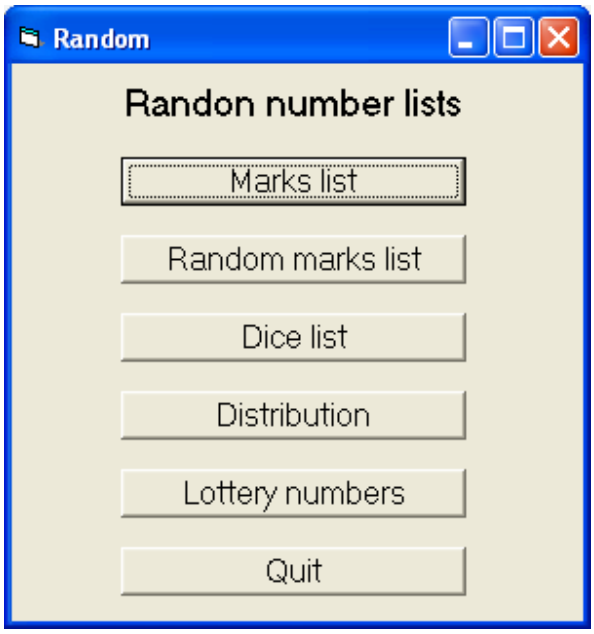

Figure 11.1 Objects present in the random number generating program.

Operation

We are going to solve part of this exercise to demonstrate the use of pseudo-random variables in Visual Basic.

The **Rnd** function returns a pseudo-random real number with a uniform distribution in the [0, 1) interval (from 0 to 1, the latter excluded). With uniform distribution we mean that all numbers have the same possibility, as it happens when we toss a coin: the number of heads tends to be the same as the number of tails.

We call them pseudo-random numbers because they are not really random: within the series a given number always has the same number after it. If we start from the same number we always obtain the same series. This first number of the series is called seed.

"Marks list" button

The first button of this program shows the scenario of an arbitrary lecturer that uses a program to generate random marks from 0 to 100 (although never gives a mark of 10 to anyone). If you check carefully you'll realise that clicking on this button at the beginning of the program we always get the same marks, the ones shown in Figure 11.2.

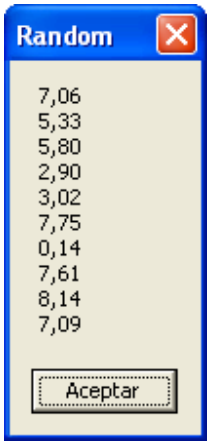

Figure 11.2 List of notes just after executing the program.

This fact of always obtaining the same series of numbers (we shall see the implementation a little further) is due to the fact that, if we do not indicate the contrary, Visual Basic always initialises the seed with the same value. If we use a subprogram `IniMarksVector` to initialise the vector to a list of random numbers and the subprogram `ShowDblVector` to show the whole vector using **`MsgBox`** the action associated to the click on the button is quite simple:

```
Sub cmd1_Click()
  Dim v(1 To 10) As Double
  Call IniMarksVector(v)
  Call ShowDblVector(v)
End Sub
```

The `IniMarksVector` procedure will make use of the lower bound (**`LBound`**) and upper bound (**`UBound`**) to fill all the vector with values in the interval [0, 10), multiplying by 10 the values obtained in the [0, 1) interval. Observe how we pass the vector parameter by reference. We could have omitted the **`ByRef`** key word, as it is the default value.

```
Sub IniMarksVector(ByRef v() As Double)
  Dim i As Integer
  For i = LBound(v) To UBound(v) Step 1
    v(i) = Rnd * 10
  Next i
End Sub
```

The `ShowDblVector` procedure obtains a string the list of values of the vector and after shows them using **`MsgBox`**.

To align all values we use the **`Format`** function seen in the previous lab.

```
Sub ShowDblVector(ByRef v() As Double)
  Dim i As Integer
  Dim s As String
  s = ""
  For i = LBound(v) To UBound(v) Step 1
    s = s & Format(v(i), "0.00") & vbCrLf
  Next i
  MsgBox s
End Sub
```

"Random marks list" button

The only difference with the result of executing the action associated with this button is that we are not going to always show the same list of random numbers as we shall use the **`Randomize`** instruction that modifies the seed (first number of the series) each time with the system clock.

We may reuse everything seen for the previous button (it is not necessary to redefine the subprograms `IniMarksVector` y `ShowDblVector`.

The code (in grey the elements that don't change) follows:

```
Sub cmd2_Click()
  Dim v(1 To 10) As Double
  Randomize
  Call IniMarksVector(v)
  Call ShowDblVector(v)
End Sub
```

We could have called directly the previous subprogram but it is not recommended because it is easier to change the actions associated with a button without checking the impacts (who uses it):

```
Sub cmd2_Click()
  Randomize
  Call cmd1_Click
End Sub
```

"Dice list" button

This button shows a list of numbers from 1 to 6 simulating a die. To do so we need a function to generate them maintaining the uniform distribution.[1].

[1] To illustrate this problem imagine that we want to obtain numbers from 1 to 4 using numbers from 1 to 6 by throwing a die. If we calculate the number just by dividing the number by 4 and adding 1 we only obtain numbers from 1 to 4 but numbers 1 and 2 will be twice as frequent as numbers 3 and 4 which would receive the occurrences of 5 and 6 respectively.

This function if given as a recipe:

```
Function RandInterv(ByVal min As Long, ByVal max As Long) As Long
  RandInterv = Int((max - min + 1) * Rnd) + min
End Function
```

With this the code for the button to show the list of dice follows:

```
Sub cmd3_Click()
  Dim v(1 To 10) As Integer
  Randomize
  Call IniDiceVector(v)
  Call ShowIntVector(v)
End Sub
```

We have used two new procedures: `IniDiceVector` y `ShowIntVector`. The first is similar to that to fill the marks but now we fill it with numbers from 1 to 6 calling function `RandInterv`.

```
Sub IniDiceVector(ByRef v() As Integer)
  Dim i As Integer
  For i = LBound(v) To UBound(v) Step 1
    v(i) = RandInterv(1, 6)
  Next i
End Sub
```

The `ShowIntVector` procedure is similar al to the one seen for real numbers.

"Distribution" button

This button measures the "quality" of the random number generator. It declares a vector with 600 integer numbers and fills it with random numbers from 1 to 6 like in the previous exercise ("throws" 600 times the die). Following this it will count occurrences of number 1, number 2 and son on. With a pure uniform distribution each of the number should appear exactly the same number of times, 100 in our case.

"Lottery numbers" button

This button generates 6 different numbers from 1 to 49 to fill in the lottery ticket.

To do so it uses the function `PosInVector` which return the position of a number `num` in vector `v` assuming that there are `n` valid elements and that the first valid element in the vector is 1. If `num` is not found in the vector it returns a 0.

The header of the function is as follows:

```
Function PosInVector(ByVal num As Integer, ByRef v() As Integer, _
                     ByVal n As Integer)_
                     As Integer
```

It is noted that in this function the vector is passed by reference but it is an input parameter, that is, it is not modified inside the function.

Checking an account number with a coefficient vector (resolved)

Interface

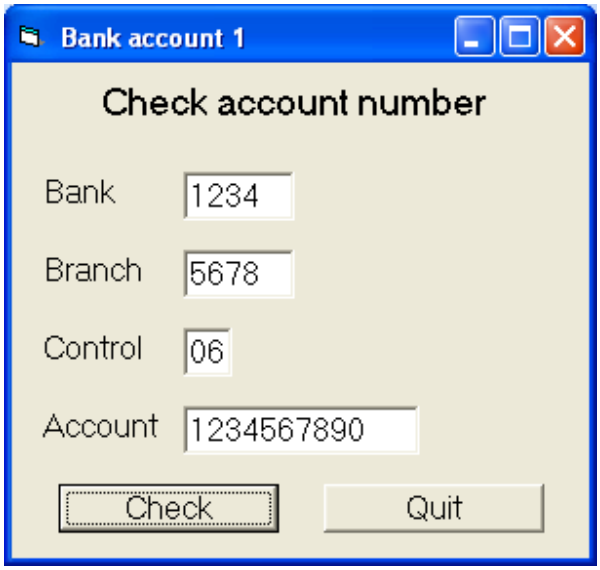

Figure 11.3 Bank account checking

Operation

In the previous laboratory an exercise to check a bank account was proposed (exercise 10.2). In this laboratory we propose, as a demonstration, a solution for the calculation of the digits using a coefficient vector (see Table 10.3 in the previous laboratory).

Proposed code

1. The code associated with the button will be similar but we now define a vector of coefficients `k()`.
2. The definition of the variable will be with type **`Variant`**.

```
Dim k() As Variant
```

3. If we want to initialise vector **`k`** with a set of values we do it through the VB **`Array`** instruction.

```
k = Array(4, 8, 5, 10, 9, 7, 3, 6)
```

4. Defined the coefficients vector we may carry out the calculation of the control digit using the `CalCtrlDig` function. Using this function the code for the **Check** button could be:

```
Sub cmdComprobar_Click()
  Dim d1 As Integer, d2 As Integer
  Dim k() As Variant
  ...
    k = Array(4, 8, 5, 10, 9, 7, 3, 6)
    d1 = CalCtrlDig(txtEnt.Text & txtOfi.Text, k)
    k = Array(1, 2, 4, 8, 5, 10, 9, 7, 3, 6)
    d2 = CalCtrlDig(txtCta.Text, k)
    If txtCtr.Text = d1 & d2 Then
      MsgBox "Correct bank account"
    ...
End Sub
```

5. The `CalCtrlDig` calculation function will make use of the lower bound (**LBound**) and upper bound (**UBound**) functions to obtain the limits of the vector. Note that this is done to simplify the parameter passing to the function.

```
Function CalCtrlDig(ByVal str As String, ByRef k() As Variant) _
                    As Integer
  Dim d As Integer, i As Integer
  d = 0
  For i = LBound(k) To UBound(k) Step 1
    d = d + CInt(Mid(str, i + 1, 1)) * k(i)
  Next i
  d = 11 - d Mod 11
  If d > 9 Then
    d = 11 – d
  End If
  CalCtrlDig = d
End Function
```

Rnd	Seudo-random numbers with uniform distribution in [0, 1)
Randomize	Initialise the random number seed with the system clock
LBound (v()) As Long	Obtain the lower bound of a vector
UBound (v()) As Long	Obtain the upper bound of a vector

Table 11.1 List of relevant functions in Visual Basic