Objectives:
- Practice with typical algorithms using **vectors** (one-dimensional arrays)

Operations with vectors

Interface

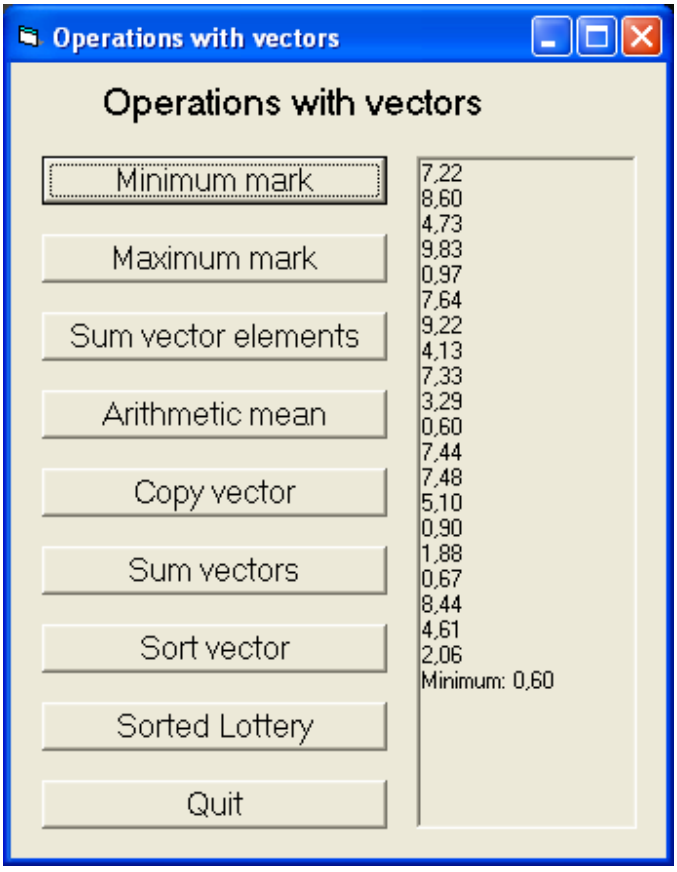

Figure 12.1 Interface of the program of operations with vectors.

Operation

In the previous laboratory some exercises to obtain a list of random marks were proposed. In this laboratory we retake part of the code to simplify the introduction of testing numbers from the keyboard. In general we shall use mark vectors (real numbers from 1 to 10) although in the last exercise we retake the generation of six different integer numbers from 1 to 49 to fill a lottery ticket in a sorted manner.

The basic operation of each button consists in generating one or two random lists from 0 to 10 and obtain minimum, maximum, sum and arithmetic mean of the elements, obtain a copy of the

vector, sum two vectors, sort a vector and the afore mentioned list of lottery numbers. Lets study them step by step.

"Minimum mark" button

We declare a vector **v()** with elements and we fill it with random real numbers from 0 to 10 using the o **IniMarksVector**, the same used in laboratory 11. We call the **iMin** function, which returns the index of the smallest of the **n** elements of a vector; we show the vector with **ShowDblVector**, which is similar to the one seen in the previous laboratory but this time we show the data on a picture box instead of using **MsgBox**; finally we show the smallest.

```
Sub cmd1_Click()
  Dim v(1 To 20) As Double
  Dim i As Integer
  Dim min As Double
  pctRes.Cls ' Clear the results picture box
  Randomize  ' Initialise the seed with the system clock
  Call IniMarksVector(v)
  i = iMin(20, v)
  min = v(i)
  Call ShowDblVector(v)
  pctRes.Print "Minimum: " & Format(min, "0.00")
End Sub
```

As a reminder the subprogram to initialise the vector of marks:

```
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 flow diagram for function **iMin** to obtain the index of the smallest element of the vector is shown in Figure 12.2. Given this index we can obtain the smallest number, **min**. Note that this number may be repeated although not probable as the random number generator gives series of different numbers. Even if we format these numbers they will be stored with a better precision.

The procedure to show in the picture box the vector formatted could be:

```
Sub ShowDblVector(ByRef v() As Double)
  Dim i As Integer
  For i = LBound(v) To UBound(v) Step 1
    pctRes.Print Format(v(i), "0.00")
  Next i
End Sub
```

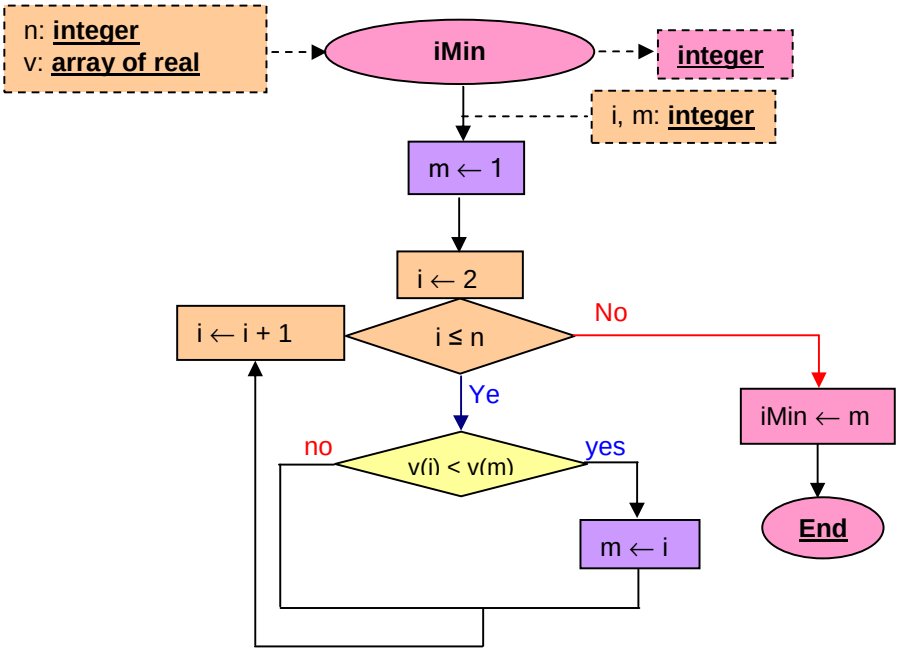

Figure 12.2 Flowchart **`iMin`** for the function.

"Maximum mark" button

The action is similar to the previous "Minimum mark".

"Sum vector" button

Obtains the sum of the elements of the vector:

```
Sub cmd3_Click()
  Dim v(1 To 20) As Double
  Dim sum As Double
  pctRes.Cls
  Randomize
  Call IniMarksVector(v)
  sum = SumVector(20, v)
  Call ShowDblVector(v)
  pctRes.Print "Sum: " & Format(sum, "0.00")
End Sub
```

"Arithmetic mean" button

It obtains the arithmetic mean of the elements of the vector. It may be obtained straight dividing the value obtained in the previous function **`SumVector`** by the number of elements.

```
Sub cmd4_Click()
  Dim v(1 To 20) As Double
  Dim amean As Double
  pctRes.Cls
  Randomize
  Call IniMarksVector(v)
  amean = ArithMean(10, v)
```

```
  Call ShowDblVector(v)
  pctRes.Print "Arithmetic mean: " & Format(amean, "0.00")
End Sub
```

"Copy vector" button

It obtains a copy of the vector. We define a vector **v1** and we initialise it using the **IniMarksVector** procedure, copying it to the destination vector **v2**; we then show both at the picture box.

```
Sub cmd5_Click()
  Dim v1(1 To 10) As Double, v2(1 To 10) As Double
  pctRes.Cls
  Randomize
  Call IniMarksVector(v1)
  Call CopyVector(10, v1, v2)
  pctRes.Print "Original vector:"
  Call ShowDblVector(v1)
  pctRes.Print "Copied vector:"
  Call ShowDblVector(v2)
End Sub
```

The header for the copying procedure is given in Figure 12.3.

Figure 12.3 Header for the procedure to copy a vector on another.

"Sum vectors" button

It obtains the sum of two vectors. We define two vectors **v1** and **v2**; we initialise them by means of procedure **IniMarksVector** and we add them to a result vector **v**, showing the contents of the three in the picture box.

```
Sub cmd6_Click()
  Dim v1(1 To 7) As Double, v2(1 To 7) As Double
  Dim v(1 To 7) As Double
  pctRes.Cls
  Randomize
  Call IniMarksVector(v1)
  Call IniMarksVector(v2)
  Call SumVectors(7, v1, v2, v)
  pctRes.Print "Vector 1:"
  Call ShowDblVector(v1)
  pctRes.Print "Vector 2:"
  Call ShowDblVector(v2)
  pctRes.Print "Sum vector:"
  Call ShowDblVector(v)
End Sub
```

The summing procedure will have the header shown in Figure 12.4.

Figure 12.4 Header for the procedure to sum vectors.

"Sort vector" button

It fills a vector, it obtains a copy and it sorts the copy, showing both vectors as they result.

```
Sub cmd7_Click()
  Dim v1(1 To 10) As Double, v2(1 To 10) As Double
  pctRes.Cls
  Randomize
  Call IniMarksVector(v1)
  Call CopiaVector(10, v1, v2)
  Call OrdenaVector(10, v2)
  pctRes.Print "Original vector:"
  Call ShowDblVector(v1)
  pctRes.Print "Sorted vector:"
  Call ShowDblVector(v2)
End Sub
```

The header for the sorting procedure is shown in Figure 12.5.

Figure 12.5 Header for the procedure to sort a vector.

Note that **v** is an input/output parameter.

To sort the vector we use an easy algorithm whose diagram is shown in Figure 12.6. Described in words, we search for the index of the smallest and exchange it with the first one, after the second and successively.

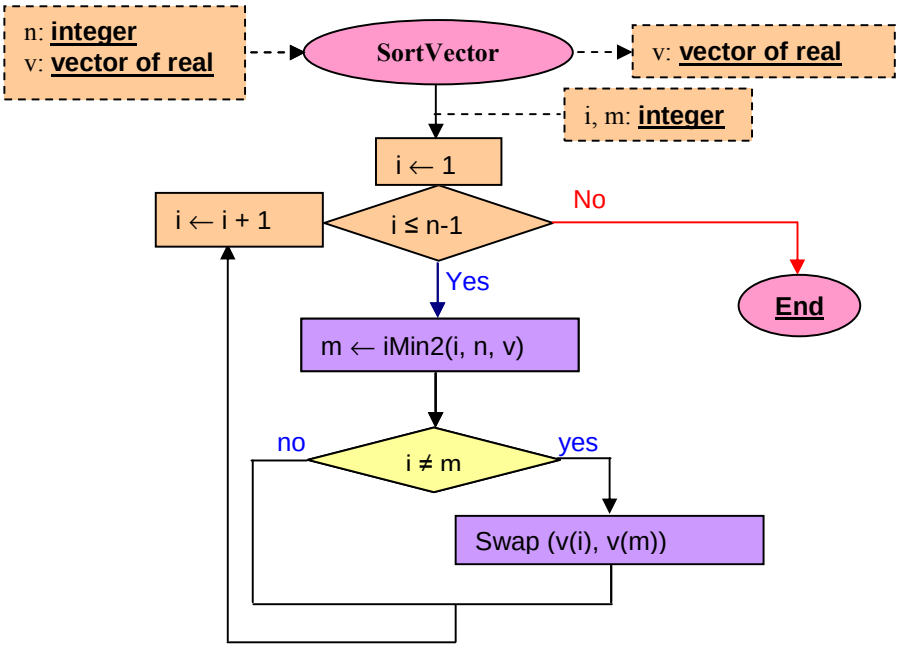

Figure 12.6 Flowchart of the sorting procedure

We shall use the **`iMin2`** function that returns the index of the smallest element of a vector in between to indices, **`i`** and **`n`**. This function if similar to **`iMin`**, used in the "Minimum mark" button, whose flowchart is shown in Figure 12.2, with the difference that instead of starting from 1 it receives the index of the position to start as an input parameter.

We also use the **`Swap`** function that, as its own name says, swaps the contents of its arguments. Figure 12.7 shows its header, with **`x`** and **`y`** as both input and output parameters.

Figure 12.7 Flowchart of the header of procedure **`Swap`**.

"Sorted lottery" button

It generates a set of six random numbers in between 1 and 49 sorted in a vector. It uses for so the sorted insertion algorithm, that is, for each new generated number we calculate the position where it should go. If it already exists it will not be inserted; if it doesn't exist we will shift right all the rest of the numbers at the right of the position to make room for the new number.

The code associated with the proposed button follows. The **`PosOrd`** inserts the new number **`num`** at the end of the list (called *sentry*) and returns the position of the first element greater than or equal to **`num`**. This way we know that we will always find at least one number greater than or equal to the sentry. If it exists it will give us the position (and we needn't do anything). If it

needs to be added at the end we only need to increment the counter of inserted numbers expressing that it is already inserted.

```
Sub cmd8_Click()
  Dim n As Integer, i As Integer
  Dim p As Integer
  Dim num As Integer
  Dim v(1 To 6) As Integer

  pctRes.Cls
  n = 0
  Do
    num = RandInterv (1, 49)
    p = PosOrd(num, n, v)
    If p <= n Then
      If v(p) <> num Then ' if equal it was already in the vector
        For i = n To p Step -1 ' shift right starting at position p
          v(i + 1) = v(i)
        Next i
        n = n + 1
        v(p) = num
      End If
    Else
      n = n + 1
      ' PosOrd has already introduced num at this position
    End If
  Loop Until n = 6
  Call ShowVectorInt(v)
End Sub
```

Figure 12.8 represents the flowchart of function **`PosOrd`** with the previously described behaviour.

Note that we use a **`While`** loop as we don't know in advance how many comparisons need to be done.

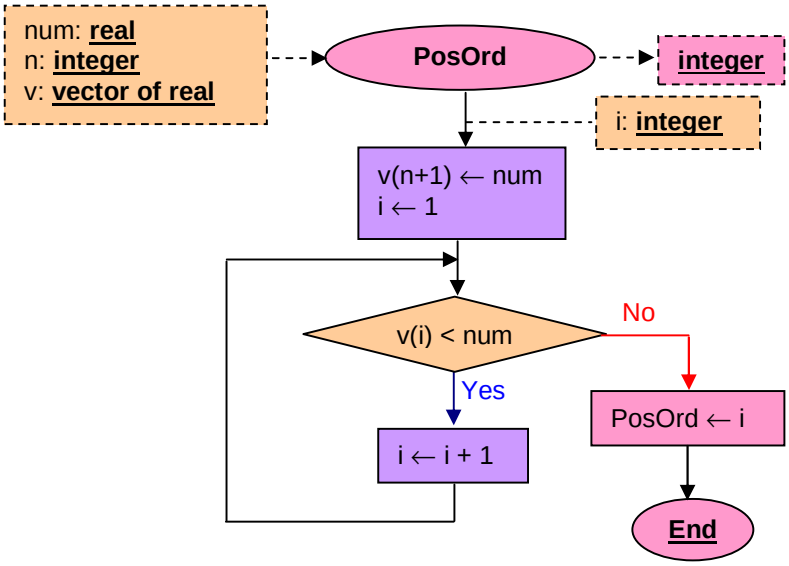

Figure 12.8 Flowchart of function **`PosOrd`**.