PROGRAMMING LOGIC AND TECHNIQUES

Algorithms: Arrays

Introduction

Imagine an algorithm that would allow us to display the average of 5 temperatures entered by the user, while keeping all the entered values saved for future use.

Introduction

Analysis

Input:
- Temperature 1
- Temperature 2
- Temperature 3
- Temperature 4
- Temperature 5

Output:
- The average of the temperatures entered

Constants:
- None

Procedure:
- Ask the user to enter the 5 temperatures, and then display the average.

Introduction

- Pseudocode

```
VARIABLES
  Real : temperature1, temperature2, temperature3,
      temperature4, temperature5
  Real : averageTemperature
START
  WRITE "Enter the 5 temperatures: "
  READ temperature1, temperature2, temperature3, temperature4,
      temperature5
  averageTemperature = (temperature1 + temperature2 +
      temperature3 + temperature4 + temperature5) / 5
  WRITE "The average is ", averageTemperature
END
```

Introduction

- This program behaves as it should, and provides the correct result without issue. However, certain problem may arise in the future:
  - What if we want to increase or decrease the number of temperatures entered?
  - What if we have 1000 temperature values to enter? Will we create 1000 variables one by one?
  - Etc…

It is for these reasons that **arrays** were developed.

Arrays: declaration

Syntax:

Array <arrayType> **:** <arrayName>[<sizeOfArray>]

- The name of the array follows the syntax conventions of the language used
- The size of the array is **always** an integer value that is greater than or equal to 1 (never 0 or negative)
- The array type represents the type of elements that can be stored in the array.

Arrays: accessing an element

Syntax:

<arrayName>[<elementIndex>]

- The indices that allow access to the values of an array are, in the vast majority of programming languages, values ranging from **0** (first index) to the **size of the array – 1** (last index).

- So, for an array declared as follows:

  Array Integer : arrayName[10]

  the first element is accessed with **arrayName[0]**, and the last element is accessed with **arrayName[9]**.

Example of pseudo code

- Pseudocode

```
VARIABLES
  Array Integer : arrayValues[10]
START
  WRITE “Enter 10 values: ”
  FOR i = 0 TO 9
      WRITE “Enter a value: ”
      READ arrayValues[i]
  ENDFOR
  FOR i = 0 TO 9
      WRITE “Value ”,  i + 1 ,  “ = ”,  arrayValues[i]
  ENDFOR
END
```

Caution

- When using arrays, it is important to take care to not make any logical errors whereby you try to access array elements that do not exist.

- For example, given an array declared as follows:

  Array Integer : arrayName[10]

  the following syntax is invalid: **arrayName[10]**, because it in fact indicates that we are trying to access the 11th element of the array!

Caution

- It is also important to make sure to use arrays to only store values of the type specified in the array declaration. If a certain array can hold **integer** values, that same array cannot, for example, be used to store **character** values!

Caution

- When using arrays, it is important to not make any logical errors whereby you try to store values of the wrong type.

- For example, given an array declared as follows:

    Array Integer : arrayName[10]

  the following syntax in invalid: **arrayName[0] = 'A'**, because elements in this array must be integers, but 'A' is a character.

Dynamic arrays

It is also possible to declare dynamic arrays:

- Pseudocode

```
VARIABLES
   Array Integer : arrayValues[] // No size is initially defined
   Integer : numberOfValues
START
   WRITE "How many values are there to enter: "
   READ numberOfValues
   REDIM arrayValues[numberOfValues]
   FOR i = 0 TO numberOfValues – 1
         WRITE "Enter a value: "
         READ arrayValues[i]
   ENDFOR
   FOR i = 0 TO numberOfValues – 1
         WRITE "Value ", i, " = ", arrayValues[i]
   ENDFOR
END
```

Exercises

Create the following:

1. An algorithm that initializes an array of real numbers using an entered value.
2. An algorithm that initializes every element of an array of integers with the cube of the element's index.
3. An algorithm that multiplies each element of an array of real numbers by an entered real number.
4. An algorithm that returns the average value of an array of real numbers that are entered by the user.
5. An algorithm that receives an array of integers and requests the input of an integer **n**, and then returns **1** if **n** is in the array, and **0** if it is not.

Exercises

6. An algorithm that receives an array of integers and requests the input of an integer **n**. It returns the number of times that the integer **n** occurs in the array.
7. An algorithm that receives an array of integers and requests the input of an integer **n**. It returns the number of elements in the array that are greater than or equal to **n**.
8. An algorithm that receives an array of integers and an integer **n**. The algorithm performs **n** shifts to the right of the elements in the array. For each shift, the last element in the array is moved into the first position, at index 0, while all the other elements' indices increase by 1.

   For example, if the array provided contains: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, and **n** = 4, then after the algorithm is run the array will contain: 7, 8, 9, 0, 1, 2, 3, 4, 5, 6