KHARKIV NATIONAL UNIVERSITY OF RADIO ELECTRONICS

INFORMATICS

PRACTICE #3

SIMPLE MANIPULATIONS WITH ARRAYS

Associate Professor A.S. Eremenko

1. Calculation of average of a fixed array of numbers using a `for` loop.

```
// FixedArrayAverage -- Average a fixed array of numbers using a loop.
namespace FixedArrayAverage
{
using System;
public class Program
    {
        public static void Main(string[] args)
        {
        double[] doublesArray = {5, 2, 7, 3.5, 6.5, 8, 1, 9, 1, 3};
        // Accumulate the values in the array into the variable sum.
        double sum = 0;
        for (int i = 0; i < 10; i++)
            {
            sum = sum + doublesArray[i];
            }
        // Now calculate the average.
        double average = sum / 10;
        Console.WriteLine("Average = {0}", average);
        Console.WriteLine("Press Enter to terminate...");
        Console.Read();
        }
    }
}
```

2. The updated program enables the user to specify the number of values to enter. (`N` has to come from somewhere.) Because the program retains the values entered, not only does it calculate the average, but it also displays the results in a pleasant format:

```
// VariableArrayAverage -- Average an array whose size is
// determined by the user at runtime, accumulating the values
// in an array. Allows them to be referenced as often as
// desired. In this case, the array creates an attractive output.
namespace VariableArrayAverage
{
using System;
public class Program
    {
        public static void Main(string[] args)
        {
        // First read in the number of doubles the user intends to enter.
        Console.Write("Enter the number of values to average: ");
        string numElementsInput = Console.ReadLine();
        int numElements = Convert.ToInt32(numElementsInput);
        Console.WriteLine();
        // Now declare an array of that size.
        double[] doublesArray = new double[numElements]; // Here's the 'N'.
        // Accumulate the values into an array.
        for (int i = 0; i < numElements; i++)
            {
            // Prompt the user for another double.
            Console.Write("enter double #" + (i + 1) + ": ");
            string val = Console.ReadLine();
            double value = Convert.ToDouble(val);
            // Add this to the array using bracket notation.
            doublesArray[i] = value;
            }
        // Accumulate 'numElements' values from
        // the array in the variable sum.
        double sum = 0;
        for (int i = 0; i < numElements; i++)
            {
            sum = sum + doublesArray[i];
            }
        // Now calculate the average.
        double average = sum / numElements;
        // Output the results in an attractive format.
        Console.WriteLine();
        Console.Write(average + " is the average of (" + doublesArray[0]);
        for (int i = 1; i < numElements; i++)
            {
            Console.Write(" + " + doublesArray[i]);
            }
        Console.WriteLine(") / " + numElements);
        // Wait for user to acknowledge the results.
        Console.WriteLine("Press Enter to terminate...");
        Console.Read();
        }
    }
}
```

3. **Sorting arrays of data**.

A common programming challenge is the need to sort the elements within an array. Just because an array cannot grow or shrink doesn't mean that the elements within it cannot be moved, removed, or added. The following program demonstrates how to use the ability to manipulate elements within an array as part of a sort. This particular sorting algorithm is the *bubble sort.* Though it's not so great on large arrays with thousands of elements, it's simple and effective on small arrays:

```
// BubbleSortArray -- Given a list of planets, sort their
// names: first, in alphabetical order.
// Second, by the length of their names, shortest to longest.
// Third, from longest to shortest.
// This demonstrates using and sorting arrays, working with
// them by array index. Two sort algorithms are used:
// 1. The Sort algorithm used by class Array’s Sort() method.
// 2. The classic Bubble Sort algorithm.
using System;
namespace BubbleSortArray
{
    class Program
    {
        static void Main(string[] args)
        {
        Console.WriteLine("The 5 planets closest to the sun, in order: ");
        string[] planets =
        new string[] { "Mercury", "Venus", "Earth", "Mars", "Jupiter" };
        foreach (string planet in planets)
            {
            // Use the special char \t to insert a tab in the printed line.
            Console.WriteLine("\t" + planet);
            }
        Console.WriteLine("\nNow listed alphabetically: ");
        // Array.Sort() is a method on the Array class.
        // Array.Sort() does its work in-place in the planets array,
        // which leaves you without a copy of the original array. The
        // solution is to copy the old array to a new one and sort it.
        string[] sortedNames = planets;
        Array.Sort(sortedNames);
        // This demonstrates that (a) sortedNames contains the same
        // strings as planets and (b) that they’re now sorted.
        foreach (string planet in sortedNames)
            {
            Console.WriteLine("\t" + planet);
            }
        Console.WriteLine("\nList by name length - shortest first: ");
        // This algorithm is called “Bubble Sort”: It’s the simplest
        // but worst-performing sort. The Array.Sort() method is much
        // more efficient, but I couldn’t use it directly to sort the
        // planets in order of name length because it sorts strings,
        // not their lengths.
        int outer; // Index of the outer loop
        int inner; // Index of the inner loop
        // Loop DOWN from last index to first: planets[4] to planets[0].
        for (outer = planets.Length - 1; outer >= 0; outer--)
            {
            // On each outer loop, loop through all elements BEYOND the
            // current outer element. This loop goes up, from planets[1]
            // to planets[4]. Using the for loop, you can traverse the
            // array in either direction.
            for (inner = 1; inner <= outer; inner++)
                {
                // Compare adjacent elements. If the earlier one is longer
                // than the later one, swap them. This shows how you can
                // swap one array element with another when they’re out of order.
                if (planets[inner - 1].Length > planets[inner].Length)
                    {
                    // Temporarily store one planet.
                    string temp = planets[inner - 1];
                    // Now overwrite that planet with the other one.
                    planets[inner - 1] = planets[inner];
                    // Finally, reclaim the planet stored in temp and put
                    // it in place of the other.
                    planets[inner] = temp;
                    }
                }
            }
        foreach (string planet in planets)
            {
            Console.WriteLine("\t" + planet);
            }
        Console.WriteLine("\nNow listed longest first: ");
        // That is, just loop down through the sorted planets.
        for(int i = planets.Length - 1; i >= 0; i--)
            {
            Console.WriteLine("\t" + planets[i]);
            }
        Console.WriteLine("\nPress Enter to terminate...");
        Console.Read();
        }
    }
}
```

Bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.