.NET PROGRAMMING LAB 7

Name: CH Bala Gowtham

Id: 2000032067

Section: S-13

Task1: To Create generic type CustomArray – one dimensional array with random index range.

CustomArray is a collection – array of random type values with fixed length and with original index that is specified by the user.

Example1: array of 20 elements length, array values- symbols, index starts with 18.
Example2: array of 10 elements length, array values- objects of class Animals, index starts with -5.

Values of random type can be located in array, custom first index and the number of elements in array should be specified while creating. The length and range of indexes cannot be changed after creating. Values of array elements can be set while creating the array and later with the help of indexer.

Initial and finite index, array length, array elements in the form of standard array Array that starts with 0 can be obtained from array.

CustomArray should be able to use operator foreach and other constructions that are oriented to the presence of numerator in class.

The task has two levels of Complexity: Low and Advanced.

Low level tasks require implementation of the following functionality:

- Creating of empty user array and the one based on standard existing
- Receiving first, last indexes, length, values in form of standard array with 0
- Access to writing and reading element based on predetermined correct index

Advanced level tasks require implementation of the following functionality:

- All completed tasks of Lowlevel
- Creating of array based on values params
- Generating exceptions, specified in xml-comments to class methods
- Receiving numerator from array for operator foreach

Code:

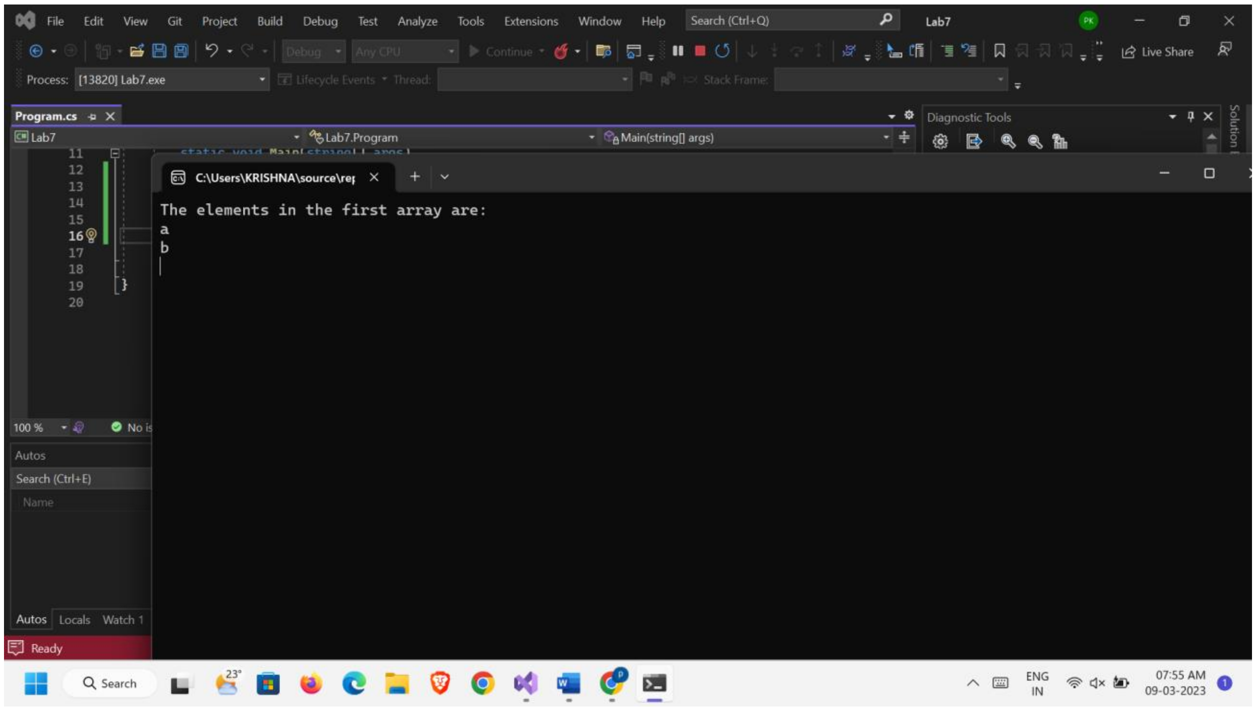
```
using System;
using System.Collections;
using System.Collections.Generic;

namespace CustomArrayDemo
{
    // Define the event argument class for the OnChangeElement and
OnChangeEqualElement events
    public class ArrayEventArgs<T> : EventArgs
    {
        public int Id { get; set; }
        public T Value { get; set; }
        public string Message { get; set; }

        public ArrayEventArgs(int id, T value, string message)
        {
            Id = id;
            Value = value;
            Message = message;
        }
    }

    // Define the delegate type for the OnChangeElement and
OnChangeEqualElement events
    public delegate void ArrayHandler<T>(object sender, ArrayEventArgs<T> e);
    // Define the CustomArray class
    public class CustomArray<T> : IEnumerable<T>
    {
        private T[] _array;
        private readonly int _startIndex;

        public event ArrayHandler<T> OnChangeElement;
        public event ArrayHandler<T> OnChangeEqualElement;

        // Constructor that creates an empty array with a specified start index and
length
        public CustomArray(int startIndex, int length)
        {
            _array = new T[length];
            _startIndex = startIndex;
        }

        // Constructor that creates an array with specified values and start index
        public CustomArray(int startIndex, params T[] values)
        {
            _array = values;
            _startIndex = startIndex;
        }

        // Property to get the length of the array
```

```
public int Length => _array.Length;

// Indexer to access elements of the array based on their index
public T this[int index]
{
    get
    {
        if (index < _startIndex || index >= _startIndex + _array.Length)
        {
            throw new IndexOutOfRangeException("Index is out of range.");
        }

        return _array[index - _startIndex];
    }
    set
    {
        if (index < _startIndex || index >= _startIndex + _array.Length)
        {
            throw new IndexOutOfRangeException("Index is out of range.");
        }

        T oldValue = _array[index - _startIndex];
        if (!EqualityComparer<T>.Default.Equals(oldValue, value))
        {
            _array[index - _startIndex] = value;
```

```
                    OnChangeElement?.Invoke(this, new ArrayEventArgs<T>(index,
value, "Element value changed."));
                }
                else
                {
                    OnChangeEqualElement?.Invoke(this, new
ArrayEventArgs<T>(index, value, "Equal element value not changed."));
                }
            }
        }

        // Method to get the first index of the array
        public int GetFirstIndex()
        {
            return _startIndex;
        }

        // Method to get the last index of the array
        public int GetLastIndex()
        {
            return _startIndex + _array.Length - 1;
        }

        // Method to get the array as a standard array starting with 0
        public T[] ToArray()
        {
            T[] result = new T[_array.Length];
            Array.Copy(_array, result, _array.Length);
            return result;
        }

        // Implementation of IEnumerable<T> interface for foreach loop
        public IEnumerator<T> GetEnumerator()
        {
            return ((IEnumerable<T>)_array).GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return _array.GetEnumerator();
        }
    }
}
```

Output:

Advance Level Task:



```
The elements in the first array are:
a
b
```