

**Міністерство освіти і науки України**  
**Національний технічний університет України «Київський політехнічний**  
**інститут імені Ігоря Сікорського" Факультет інформатики та**  
**обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 2 з дисципліни  
«Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

**“Модульне тестування. Ознайомлення з засобами та практиками  
модульного тестування”**

**Виконав(ла)**

ІП-15 Костін В.А.  
(шифр, прізвище, ім'я, по батькові)

**Перевірів**

Бардін В.  
(прізвище, ім'я, по батькові)

Київ 2023

## Лабораторна робота 2

### Модульне тестування. Ознайомлення з засобами та практиками модульного тестування

9	Динамічний масив з довільним діапазоном індексу	Див. List<T>	Збереження даних за допомогою вектору
---	---	--------------	---------------------------------------

Код тестів:

```
public class CustomArrayTests : CollectionTestsBase
{
    #region Constructor
    [Fact]
    public void Constructor_NoParametres_EmptyCollectionCreated()
    {
        // Act
        var collection = new Collection();

        // Assert
        Assert.Empty(collection);
    }

    [Fact]
    public void Constructor_NegativeCapacity_ThrowsArgumentException()
    {
        // Arrange
        int capacity = -5;

        // Assert
        Assert.Throws<ArgumentException>(() =>
        {
            // Act
            var collection = new CustomArray<int>(capacity);
        });
    }

    [Theory]
    [InlineData(5)]
    [InlineData(0)]
    public void Constructor_NotNegative_SuccessfullyCreation(int capacity)
    {
        // Act
        var collection = new CustomArray<int>(capacity);

        // Assert
        Assert.Empty(collection);
    }

    [Fact]
    public void Custructor_NullArray_ThrowsArgumentNullException()
    {
        // Arrange
        IEnumerable<int> arr = null;

        // Assert
        Assert.Throws<ArgumentNullException>(() =>
        {
```

```

        // Act
        var collection = new CustomArray<int>(arr);
    });
}

[Theory]
[InlineData(new int[] { 1, 2, 3, 4, 5 })]
[InlineData(new int[] { -1, 0, 1 })]
[InlineData(new int[] { 7, 6, 9, 11, 24 })]
public void
Constructor_CollectionPassed_SuccessfullCreation(IEnumerable<int> arr)
{
    // Act
    var collection = new CustomArray<int>(arr);

    // Assert
    Assert.Equal(arr, collection);
}
#endregion

#region Index
[Fact]
public void
Index_CallToEmptyCollection_ThrowsArgumentOutOfRangeException()
{
    // Arrange
    var collection = new CustomArray<int>();

    // Assert
    Assert.Throws<ArgumentOutOfRangeException>(() =>
    {
        // Act
        int value = collection[0];
    });
}

[Fact]
public void Index_ProperIndex_ReturnsProperElement()
{
    // Arrange
    var testElements = new List<int>() { 1, 2, 3 };
    var collection = new CustomArray<int>(testElements);

    // Act
    int first = collection[0];
    int second = collection[1];
    int third = collection[2];

    // Assert
    Assert.Equal(testElements[0], first);
    Assert.Equal(testElements[1], second);
    Assert.Equal(testElements[2], third);
}

[Theory]
[InlineData(new int[] { 1, 2, 3 }, 2, 4)]
[InlineData(new int[] { -1, 0, 1 }, 1, 2)]
[InlineData(new int[] { 10, 20, 30, 40, 50 }, 4, 60)]
public void Index_SetValueByIndex_ElementChanges(int[] arr, int index, int
newValue)
{
    // Arrange
    var collection = new CustomArray<int>(arr);
    var oldValue = collection[index];

```

```

        // Act
        collection[index] = newValue;

        // Assert
        Assert.Contains(newValue, collection);
        Assert.DoesNotContain(oldValue, collection);
        Assert.Equal(index, collection.IndexOf(newValue));
    }

    [Fact]
    public void Index_IndexOutOfRange_ReturnsProperElement()
    {
        // Arrange
        int count = 5;
        var collection = new
CustomArray<int>(GetDefaultSetForCollection(count));

        int actualPositiveIndex = 7;
        int expectedPositiveIndex = actualPositiveIndex % count;

        int actualNegativeIndex = -2;
        int expectedNegativeIndex = (count + actualNegativeIndex % count) %
count;

        // Act
        var expectedElementPositive = collection[expectedPositiveIndex];
        var actualElementPositive = collection[actualPositiveIndex];

        var expectedElementNegative = collection[expectedNegativeIndex];
        var actualElementNegative = collection[actualNegativeIndex];

        // Assert
        Assert.Equal(expectedElementPositive, actualElementPositive);
        Assert.Equal(expectedElementNegative, actualElementNegative);
    }

#endregion

#region Enumerator
[Fact]
public void Enumerator_LastElement_MoveNextReturnsFalse()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2 };
    var enumerator = collection.GetEnumerator();
    enumerator.MoveNext();
    enumerator.MoveNext();

    // Act
    var result = enumerator.MoveNext();

    // Assert
    Assert.False(result);
}

[Fact]
public void Enumerator_Reset_CurrentMovesToTheFirstElement()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3, 4, 5 };
    var enumerator = collection.GetEnumerator();

    enumerator.MoveNext();

```

```

        var expectedItem = enumerator.Current;
        enumerator.MoveNext();

        // Act
        enumerator.Reset();

        // Assert
        Assert.Equal(expectedItem, enumerator.Current);
    }
#endregion

#region Add

[Fact]
public void Add_NewElement_AddedToTheEnd()
{
    // Arrange
    var collection = new CustomArray<int>(GetDefaultSetForCollection());
    int valueToAdd = 6;

    // Act
    collection.Add(valueToAdd);

    // Assert
    Assert.Equal(6, collection.Last());
}

[Fact]
public void Add_NewElement_CountIncrements()
{
    // Arrange
    var collection = new CustomArray<int>();
    int defaultCount = collection.Count;

    // Act
    collection.Add(0);

    // Assert
    Assert.Equal(1, collection.Count - defaultCount);
}

[Fact]
public void Add_NullElement_ThrowsArgumentNullException()
{
    // Arrange
    var collection = new CustomArray<TestItem>();

    // Assert
    Assert.Throws<ArgumentNullException>(() =>
    {
        // Act
        collection.Add(null);
    });
}

#endregion

#region Clear

[Fact]
public void Clear_EmptyCollection()
{
    // Arrange
    var collection = new CustomArray<int>(GetDefaultSetForCollection());

```

```

        // Act
        collection.Clear();

        // Assert
        Assert.Empty(collection);
    }

#endregion

#region Contains

[Fact]
public void Contains_ElementPassed_ReturnsResultOfExistence()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };

    // Act
    bool result1 = collection.Contains(2);
    bool result2 = collection.Contains(4);

    //Assert
    Assert.True(result1);
    Assert.False(result2);
}

[Fact]
public void Clear_NullElement_ThrowsArgumentNullException()
{
    // Arrange
    var collection = new CustomArray<TestItem>();

    // Assert
    Assert.Throws<ArgumentNullException>(() =>
    {
        // Act
        collection.Contains(null);
    });
}

#endregion

#region CopyTo

[Fact]
public void CopyTo_NullArray_ThrowsArgumentNullException()
{
    // Arrange
    var collection = new CustomArray<int>();
    int[] arrayCopyTo = null;
    int indexCopyTo = 0;

    // Assert
    Assert.Throws<ArgumentNullException>(() =>
    {
        // Act
        collection.CopyTo(arrayCopyTo, indexCopyTo);
    });
}

[Fact]
public void
CopyTo_ArrayDoesNotFitIntoTheRange_ThrowsArgumentOutOfRangeException()

```

```

{
    // Arrange
    var collection = new CustomArray<int>() { 4, 5, 6 };
    var arrayCopyTo = new int[] { 1, 2, 3 };
    int indexCopyTo = 2;

    // Assert
    Assert.Throws<ArgumentOutOfRangeException>(() =>
    {
        // Act
        collection.CopyTo(arrayCopyTo, indexCopyTo);
    });
}

[Fact]
public void CopyTo_CorrectArrayAndIndex_SuccessfullCopying()
{
    // Arrange
    var collection = new CustomArray<int>() { 4, 5, 6 };
    var arrayCopyTo = new int[6] { 1, 2, 3, 0, 0, 0 };
    var indexCopyTo = 3;

    // Act
    collection.CopyTo(arrayCopyTo, indexCopyTo);

    // Assert
    Assert.Equal(collection[0], arrayCopyTo[3]);
    Assert.Equal(collection[1], arrayCopyTo[4]);
    Assert.Equal(collection[2], arrayCopyTo[5]);
}

#endregion

#region IndexOf

[Fact]
public void IndexOf_NullElement_ThrowsArgumentNullException()
{
    // Arrange
    var collection = new CustomArray<TestItem>();

    // Assert
    Assert.Throws<ArgumentNullException>(() =>
    {
        // Act
        int index = collection.IndexOf(null);
    });
}

[Fact]
public void IndexOf_ElementDoesNotExist_ReturnsDefaultIndex()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };
    int element = 4;
    int defaultIndex = -1;

    // Act
    int actualIndex = collection.IndexOf(element);

    // Assert
    Assert.Equal(defaultIndex, actualIndex);
}

```

```

[Fact]
public void IndexOf_ElementExists_ReturnsElementsIndex()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };
    int element = 2;
    int expectedIndex = 1;

    // Act
    int actualIndex = collection.IndexOf(element);

    // Assert
    Assert.Equal(expectedIndex, actualIndex);
}

#endregion

#region Insert

[Fact]
public void Insert_NullElement_ThrowsArgumentNullException()
{
    // Arrange
    var collection = new CustomArray<TestItem>();

    // Assert
    Assert.Throws<ArgumentNullException>(() =>
    {
        // Act
        collection.Insert(0, null);
    });
}

[Fact]
public void Insert_ProperElement_SuccessfullInsertion()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 4, 5 };
    int elementToInsert = 3;
    int indexToInsert = 2;

    int defaultCount = collection.Count;

    // Act
    collection.Insert(indexToInsert, elementToInsert);

    // Assert
    Assert.Equal(elementToInsert, collection[indexToInsert]);
    Assert.Equal(1, collection.Count - defaultCount);
}

[Fact]
public void Insert_IndexOutOfRange_SuccessfullInsertion()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 4 };

    int expectedCount = collection.Count;

    int elementToInsert1 = 3;
    int indexToInsert1 = 6;
    int realIndex1 = indexToInsert1 % (expectedCount + 1);

    expectedCount++;

```



```

        int elementToInsert2 = 5;
        int indexToInsert2 = -1;
        int realIndex2 = (expectedCount + 1 + indexToInsert2 % (expectedCount
+ 1)) % (expectedCount + 1);

        expectedCount++;

        // Act
        collection.Insert(indexToInsert1, elementToInsert1);
        collection.Insert(indexToInsert2, elementToInsert2);

        // Assert
        Assert.Equal(expectedCount, collection.Count);
        Assert.Equal(elementToInsert1, collection[realIndex1]);
        Assert.Equal(elementToInsert2, collection[realIndex2]);
    }

#endregion

#region RemoveAt

[Fact]
public void RemoveAt_EmptyCollection_ThrowsArgumentOutOfRangeException()
{
    // Arrange
    var collection = new CustomArray<int>();

    // Assert
    Assert.Throws<ArgumentOutOfRangeException>(() =>
    {
        // Act
        collection.RemoveAt(0);
    });
}

[Fact]
public void RemoveAt_IndexPassed_SuccessfullRemoving()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };
    var indexToRemove = 1;
    var elementToRemove = collection[indexToRemove];
    var defaultCount = collection.Count;

    // Act
    collection.RemoveAt(indexToRemove);

    // Assert
    Assert.Equal(1, defaultCount - collection.Count);
    Assert.DoesNotContain(elementToRemove, collection);
}

[Fact]
public void RemoveAt_IndexOutOfRange_SuccessfullRemoving()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };

    var indexToRemove1 = 4;
    var elementToRemove1 = collection[indexToRemove1];

    var indexToRemove2 = -1;
    var elementToRemove2 = collection[indexToRemove2];

```

```

        int defaultCount = collection.Count;

        // Act
        collection.RemoveAt(indexToRemove1);
        collection.RemoveAt(indexToRemove2);

        // Assert
        Assert.Equal(2, defaultCount - collection.Count);
        Assert.DoesNotContain(elementToRemove1, collection);
        Assert.DoesNotContain(elementToRemove2, collection);
    }

#endregion

#region Remove

[Fact]
public void Remove_NullElement_ThrowsArgumentNullException()
{
    // Arrange
    var collection = new CustomArray<TestItem>();

    // Assert
    Assert.Throws<ArgumentNullException>(() =>
    {
        // Act
        var result = collection.Remove(null);
    });
}

[Fact]
public void Remove_ElementDoesNotExist_ReturnsFalse()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };
    var elementToRemove = 4;

    // Act
    var result = collection.Remove(elementToRemove);

    // Assert
    Assert.False(result);
}

[Fact]
public void Remove_ElementExists_SuccessfullRemoving()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };
    var elementToRemove = 3;
    int defaultCount = collection.Count;

    // Act
    var result = collection.Remove(elementToRemove);

    // Assert
    Assert.True(result);
    Assert.Equal(1, defaultCount - collection.Count);
    Assert.DoesNotContain(elementToRemove, collection);
}

#endregion

```

```

#region Events
[Fact]
public void ItemAdded_EventRaised()
{
    // Arrange
    var collection = new CustomArray<int>();
    var countEventRaised = 0;
    collection.ItemAdded += (sender, e) => countEventRaised++;

    // Act
    collection.Add(2);
    collection.Insert(0, 1);

    // Assert
    Assert.Equal(2, countEventRaised);
}

[Fact]
public void ItemRemoved_EventRaised()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };
    var countEventRaised = 0;
    collection.ItemRemoved += (sender, e) => countEventRaised++;

    // Act
    collection.Remove(2);
    collection.RemoveAt(0);

    // Assert
    Assert.Equal(2, countEventRaised);
}

[Fact]
public void ArrayCleared_EventRaised()
{
    // Arrange
    var collection = new CustomArray<int>() { 1, 2, 3 };
    var eventRaised = false;
    collection.ArrayCleared += (sender, e) => eventRaised = true;

    // Act
    collection.Clear();

    // Assert
    Assert.True(eventRaised);
}

[Fact]
public void ArrayResized_EventRaised()
{
    // Arrange
    var collection = new CustomArray<int>(4) { 1, 2, 3, 4 };
    var eventRaised = false;
    collection.ArrayResized += (sender, e) => eventRaised = true;

    // Act
    collection.Add(5);

    // Assert
    Assert.True(eventRaised);
}
}
#endregion
}

```

## Результат виконання тестів:

```
✓ Add_NewElement_AddedToTheEnd Success
✓ Add_NewElement_CountIncrements Success
✓ Add_NullElement_ThrowsArgumentNullException Success
✓ ArrayCleared_EventRaised Success
✓ ArrayResized_EventRaised Success
✓ Clear_EmptyCollection Success
✓ Clear_NullElement_ThrowsArgumentNullException Success
✓ Constructor_CollectionPassed_SuccessfullCreation (3 tests) Success
✓ Constructor_NegativeCapacity_ThrowsArgumentException Success
✓ Constructor_NoParametres_EmptyCollectionCreated Success
✓ Constructor_NotNegative_SuccessfullyCreation (2 tests) Success
✓ Contains_ElementPassed_ReturnsResultOfExistence Success
✓ CopyTo_ArrayDoesNotFitIntoTheRange_ThrowsArgumentOutOfRangeException Success
✓ CopyTo_CorrectArrayAndIndex_SuccessfullCopying Success
✓ CopyTo_NullArray_ThrowsArgumentNullException Success
✓ Custructor_NullArray_ThrowsArgumentNullException Success
✓ Enumerator_LastElement_MoveNextReturnsFalse Success
✓ Enumerator_Reset_CurrentMovesToTheFirstElement Success
✓ Index_CallToEmptyCollection_ThrowsArgumentOutOfRangeException Success
✓ Index_IndexOutOfRange_ReturnsProperElement Success
✓ Index_ProperIndex_ReturnsProperElement Success
✓ Index_SetValueByIndex_ElementChanges (3 tests) Success
✓ IndexOf_ElementDoesNotExist_ReturnsDefaultIndex Success
✓ IndexOf_ElementExists_ReturnsElementsIndex Success
✓ IndexOf_NullElement_ThrowsArgumentNullException Success
✓ Insert_IndexOutOfRange_SuccessfullInsertion Success
✓ Insert_NullElement_ThrowsArgumentNullException Success
✓ Insert_ProperElement_SuccessfullInsertion Success
✓ ItemAdded_EventRaised Success
✓ ItemRemoved_EventRaised Success
✓ Remove_ElementDoesNotExist_ReturnsFalse Success
✓ Remove_ElementExists_SuccessfullRemoving Success
✓ Remove_NullElement_ThrowsArgumentNullException Success
✓ RemoveAt_EmptyCollection_ThrowsArgumentOutOfRangeException Success
✓ RemoveAt_IndexOutOfRange_SuccessfullRemoving Success
✓ RemoveAt_IndexPassed_SuccessfullRemoving Success
```

## Покриття тестів (dotCover):

Symbol	Coverage (%)	Uncovered/Total ... ^
✓ Total	100%	0/210
✓ CustomCollections	100%	0/210
✓ CustomCollections	100%	0/210
> CustomEnumerator<T>	100%	0/26
> CustomArray<T>	100%	0/184