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

Практикум №2

з курсу «Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

на тему: «Модульне тестування. Ознайомлення з засобами та практиками модульного тестування»

Викладач:
Бардін В.

Виконав:
Хільчук А.В.
студент 3 курсу
групи ІП-14 ФІОТ

Київ-2023

Практична робота №2

Тема: Модульне тестування. Ознайомлення з засобами та практиками модульного тестування

Завдання:

1. Додати до проекту власної узагальненої колекції (застосувативиконану лабораторну роботу No1) проект модульних тестів, використовуючи певний фреймворк (Nunit, Xunit, тощо).
2. Розробити модульні тести для функціоналу колекції.
3. Дослідити ступінь покриття модульними тестами вихідного коду колекції, використовуючи, наприклад, засіб AxoCover.

Виконання:

Код модульних тестів:

AdditionTest.cs:

```
using System;

using System.Collections.Generic;

using System.Text;

using Xunit;

using MyList;

using System.Linq;


namespace MyList.Tests

{

   public class AdditionTest

   {

      [Theory]

      [MemberData(nameof(GetAddValidTestData))]

      public void Add_WhenCollectionIsNotNull_MustSucceed(List<int> toAdd, CustomList<int> coll)

      {

         int initCount = coll.Count;
```

```
            foreach(var value in toAdd)
            {
                coll.Add(value);

                Assert.Equal(value, coll.Last());
            }

            Assert.Equal(initCount + toAdd.Count, coll.Count);
        }

        [Theory]
        [MemberData(nameof(GetInsertInRangeTestData))]
        public void
Insert_WhenIndexIsInRangeOfList_MustSucceed(List<Tuple<int, int>>
valueIndexes, CustomList<int> coll)
        {
            foreach(var valueIndex in valueIndexes)
            {
                int value = valueIndex.Item1;
                int index = valueIndex.Item2;
                int oldCount = coll.Count;
                int oldValueAtIndex = coll[index];

                coll.Insert(index, value);

                Assert.Equal(value, coll[index]);
                Assert.Equal(oldCount + 1, coll.Count);
                Assert.Equal(oldValueAtIndex, coll[index + 1]);
```

```
        }
    }

    [Theory]
    [MemberData(nameof(GetInsertOneAfterTestData))]
    public void Insert_WhenIndexIsOneAfterTheEndOfList_MustSucceed(List<Tuple<int, int>> valueIndexes, CustomList<int> coll)
    {
        foreach (var valueIndex in valueIndexes)
        {
            int value = valueIndex.Item1;
            int index = valueIndex.Item2;
            int oldCount = coll.Count;

            coll.Insert(index, value);

            Assert.Equal(value, coll[index]);
            Assert.Equal(oldCount + 1, coll.Count);
        }
    }

    [Theory]
    [MemberData(nameof(GetInsertInvalidTestData))]
    public void Insert_WhenIndexIsOutside_MustThrow(Tuple<int, int> valueIndex, CustomList<int> coll)
    {
        int value = valueIndex.Item1;
```

```
            int index = valueIndex.Item2;


            Action wrongInsert = () => coll.Insert(index, value);


            var exception =
Assert.Throws<ArgumentOutOfRangeException>(wrongInsert);
            Assert.Equal("Index out of range", exception.ParamName);
        }


        public static IEnumerable<object[]> GetAddValidTestData()
        {
            //returns list of elements to add and initial CustomList
            yield return new object[] { new List<int> { 6 }, new CustomList<int> {
1, 2, 3, 4, 5 } };
            yield return new object[] { new List<int> { 6, 7, 8, 9 }, new
CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { new List<int> { 1 }, new
CustomList<int>()};
            yield return new object[] { new List<int> { 1, 2, 3 }, new
CustomList<int>() };
        }


        public static IEnumerable<object[]> GetInsertInRangeTestData()
        {
            //returns list of elements to insert in format: Tuple< VALUE, INDEX >
and initial CustomList
            yield return new object[] { new List<Tuple<int, int>> {
Tuple.Create(0,0), Tuple.Create(3, 3), Tuple.Create(6, 6), }, new
CustomList<int> { 1, 2, 3, 4, 5 } };
```

```
            yield return new object[] { new List<Tuple<int, int>> { Tuple.Create(0, 0), Tuple.Create(0, 0), Tuple.Create(0, 0), }, new CustomList<int> { 1, 2, 3, 4, 5 } };

            yield return new object[] { new List<Tuple<int, int>> { Tuple.Create(3, 3), Tuple.Create(3, 3), Tuple.Create(3, 3), }, new CustomList<int> { 1, 2, 3, 4, 5 } };

        }

        public static IEnumerable<object[]> GetInsertOneAfterTestData()

        {

            //returns list of elements to insert in format: Tuple< VALUE, INDEX > and initial CustomList

            yield return new object[] { new List<Tuple<int, int>> { Tuple.Create(6, 5), Tuple.Create(7, 6), Tuple.Create(8, 7), }, new CustomList<int> { 1, 2, 3, 4, 5 } };

            yield return new object[] { new List<Tuple<int, int>> { Tuple.Create(1, 0), Tuple.Create(3, 1), Tuple.Create(2, 1), }, new CustomList<int>() };

        }

        public static IEnumerable<object[]> GetInsertInvalidTestData()

        {

            //returns element to insert in format: Tuple< VALUE, INDEX > and initial CustomList

            yield return new object[] { Tuple.Create(0, -1) , new CustomList<int> { 1, 2, 3, 4, 5 } };

            yield return new object[] { Tuple.Create(0, 6) , new CustomList<int> { 1, 2, 3, 4, 5 } };

            yield return new object[] { Tuple.Create(0, 1), new CustomList<int>() };

        }

    }

}
```

ClearTest.cs:

```
using System;
```

```
using System.Collections.Generic;
using System.Text;
using Xunit;
using MyList;

namespace MyList.Tests
{
    public class ClearTest
    {
        [Fact]
        public void Clear_WhenListIsNotNull_MustSucceed()
        {
            var coll = new CustomList<int>() { 1, 2, 3, 4, 5 };

            coll.Clear();

            Assert.Empty(coll);
        }
    }
}
```

ContainsTest.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using Xunit;
namespace MyList.Tests
{
```

```
    public class ContainsTest
    {
        [Theory]
        [MemberData(nameof(GetContainsTestData))]
        public void Contains_WhenCollectionIsNotNull_MustSucceed(int searchedValue, bool expectedResult, CustomList<int> coll)
        {
            var result = coll.Contains(searchedValue);


            Assert.Equal(expectedResult, result);
        }


        public static IEnumerable<object[]> GetContainsTestData()
        {
            //returns value to search index for, expected result and initial collection
            yield return new object[] { 1, true, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 3, true, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 5, true, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 6, false, new CustomList<int> { 1, 2, 3, 4, 5 } };
        }
    }
}
```

CopyToTest.cs:

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

```
using System.Text;
using Xunit;
using MyList;
using System.Linq;

namespace MyList.Tests
{
    public class CopyToTest
    {
        [Theory]
        [MemberData(nameof(GetValidCopyToData))]
        public void CopyTo_WhenArrayCanFit_MustSucceed(int[] targetArray, int startIndex, CustomList<int> coll)
        {
            int[] arraySnapshot = (int[])targetArray.Clone();

            coll.CopyTo(targetArray, startIndex);

            Assert.Equal(arraySnapshot.Take(startIndex), targetArray.Take(startIndex));
            Assert.Equal(coll, targetArray.Skip(startIndex).Take(coll.Count));
            Assert.Equal(arraySnapshot.Skip(startIndex + coll.Count), targetArray.Skip(startIndex + coll.Count));
        }

        [Theory]
        [MemberData(nameof(GetInvalidCopyToData))]
        public void CopyTo_WhenArrayCantFit_MustThrow(int[] targetArray, int startIndex, CustomList<int> coll)
```

```
    {
        Action copyToSmall = () => coll.CopyTo(targetArray, startIndex);

        var exception = Assert.Throws<Exception>(copyToSmall);
        Assert.Equal("Array doesn't have enough space", exception.Message);
    }

    [Fact]
    public void CopyTo_WhenArrayIsNull_MustThrow()
    {
        int[] arr = null;
        CustomList<int> coll = new CustomList<int>() { 1, 2, 3, 4, 5 };

        Action copyToNull = () => coll.CopyTo(arr, 0);

        var exception = Assert.Throws<Exception>(copyToNull);
        Assert.Equal("Array is null",exception.Message);
    }

    public static IEnumerable<object[]> GetValidCopyToData()
    {
        //returns array to copy data to, starting index, and collection itself
        yield return new object[] { new int[] { -1, -1, -1, -1, -1, }, 0, new
CustomList<int>() { 1, 2, 3, 4, 5 } };
        yield return new object[] { new int[] { -1, -1, -1, -1, -1, }, 2, new
CustomList<int>() { 3, 4, 5 } };
        yield return new object[] { new int[] { -1, -1, -1, -1, -1, }, 2, new
CustomList<int>() };
```

```
            yield return new object[] { new int[] { -1, -1, -1, -1, -1, }, 2, new CustomList<int>() { 3} };
        }

        public static IEnumerable<object[]> GetInvalidCopyToData()
        {
            //returns array to copy data to, starting index, and collection itself
            yield return new object[] { new int[] { -1, -1, -1, -1,}, 0, new CustomList<int>() { 1, 2, 3, 4, 5 } };
            yield return new object[] { new int[] { -1, -1, -1, -1, -1 }, 5, new CustomList<int>() { 1, 2, 3, 4, 5 } };
            yield return new object[] { new int[] { -1, -1, -1, -1, -1 }, 4, new CustomList<int>() { 1, 2} };
        }
    }
}
```

EnumeratorTest.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using Xunit;
using MyList;

namespace MyList.Tests
{
    public class EnumeratorTest
    {
        [Theory]
        [MemberData(nameof(GetMoveNextValidTestData))]
```

```
    public void
MoveNext_WhenCanAdvance_MustAdvance(CustomList<int> coll)
    {
        var enumerator = coll.GetEnumerator();

        var next = enumerator.MoveNext();

        Assert.True(next);
        Assert.Equal(1,enumerator.Current);
    }

    [Fact]
    public void MoveNext_WhenUsedBeyondEnd_MustReturnFalse()
    {
        CustomList<int> coll = new CustomList<int>() { 1 };
        var enumerator = coll.GetEnumerator();
        enumerator.MoveNext();

        var next = enumerator.MoveNext();

        Assert.False(next);
    }

    [Fact]
    public void MoveNext_WhenUsedOnEmpty_MustPointAtDefault()
    {
        CustomList<int> coll = new CustomList<int>();
        var enumerator = coll.GetEnumerator();
```

```
        var next = enumerator.MoveNext();

        Assert.False(next);
        Assert.Equal(default(int), enumerator.Current);
    }

    [Fact]
    public void MoveNext_AfterReachingTheEnd_MustResetEnumerator()
    {
        CustomList<int> coll = new CustomList<int>() { 1, 2, 3, 4 };
        var enumerator = coll.GetEnumerator();

        foreach (var a in coll) { }

        Assert.Equal(1, enumerator.Current);
    }

    [Fact]
    public void
Enumerator_WhenTraversingCollection_MustEnumerateProperly()
    {
        CustomList<int> coll = new CustomList<int>() { 1, 2, 3, 4 };
        List<int> expected = new List<int>() { 1, 2, 3, 4 };
        List<int> traverseSeuqence = new List<int>();

        foreach (var a in coll)
        {
```

```
        traverseSeuqence.Add(a);
    }

    Assert.Equal(expected, traverseSeuqence);
}

[Fact]
public void Enumerator_WhenCollectionIsEmpty_MustPointAtDefault()
{
    CustomList<int> coll = new CustomList<int>();

    var enumerator = coll.GetEnumerator();

    Assert.Equal(default(int), enumerator.Current);
}

[Fact]
public void Enumerator_WhenVersionChanged_MustThrow()
{
    CustomList<int> coll = new CustomList<int>() { 1, 2, 3, 4 };
    var enumerator = coll.GetEnumerator();
    coll.Add(5);

    Action moveNextAfterChange = () => enumerator.MoveNext();

    var exception = Assert.Throws<Exception>(moveNextAfterChange);
    Assert.Equal("The collection has been modified", exception.Message);
```

```
    }

    [Fact]
    public void Reset_WhenUsed_MustPointAtBeginning()
    {
        CustomList<int> coll = new CustomList<int>() { 1, 2, 3, 4 };
        var enumerator = coll.GetEnumerator();
        enumerator.MoveNext();
        enumerator.MoveNext();

        enumerator.Reset();

        Assert.Equal(1, enumerator.Current);
    }

    [Fact]
    public void
Reset_WhenCollectionIsEmpty_EnumeratorMustPointAtDefault()
    {
        CustomList<int> coll = new CustomList<int>();
        var enumerator = coll.GetEnumerator();

        enumerator.Reset();

        Assert.Equal(default(int), enumerator.Current);
    }

    public static IEnumerable<object[]> GetMoveNextValidTestData()
```

```
        {
            yield return new object[] { new CustomList<int>() { 1, 2, 3, 4, 5 } };
            yield return new object[] { new CustomList<int>() { 1 } };
        }
    }
}
```

EventTests.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using Xunit;
using MyList;

namespace MyList.Tests
{
    public class EventTests
    {
        [Fact]
        public void Cleared_WhenListCleared_MustInvoke()
        {
            var list = new CustomList<int>() { 1, 2, 3, 4, 5 };
            var wasInvoked = false;
            EventHandler onClear = (sender, item) => wasInvoked = true;
            list.Cleared += onClear;

            list.Clear();
```

```
        Assert.True(wasInvoked);
    }

    [Fact]
    public void ItemAdded_WhenAdded_MustInvoke()
    {
        var list = new CustomList<int>() { 1, 2, 3, 4, 5 };
        var wasInvoked = false;
        int addedItem=-1;
        EventHandler<int> onAdded = (sender, item) => { wasInvoked = true; addedItem = item; };
        list.ItemAdded += onAdded;

        list.Add(6);

        Assert.True(wasInvoked);
        Assert.Equal(6,addedItem);
    }

    [Theory]
    [InlineData(0,0)]
    [InlineData(2,3)]
    [InlineData(5, 6)]
    public void ItemInserted_WhenInserted_MustInvoke( int index, int value)
    {
        var list = new CustomList<int>() { 1, 2, 3, 4, 5 };
        var wasInvoked = false;
        int insertedItem = -1;
```

```
        EventHandler<int> onInseted = (sender, item) => { wasInvoked = true; insertedItem = item; };
        list.ItemInserted += onInseted;

        list.Insert(index,value);

        Assert.True(wasInvoked);
        Assert.Equal(value, insertedItem);
    }

    [Fact]
    public void ItemInserted_WhenInsertedInEmptyAtStart_MustInvoke()
    {
        var list = new CustomList<int>();
        var wasInvoked = false;
        int insertedItem = -1;
        EventHandler<int> onInseted = (sender, item) => { wasInvoked = true; insertedItem = item; };
        list.ItemInserted += onInseted;

        list.Insert(0, 1);

        Assert.True(wasInvoked);
        Assert.Equal(1, insertedItem);
    }

    [Theory]
    [InlineData(1)]
```

```
    [InlineData(3)]
    [InlineData(5)]
    public void ItemRemoved_WhenRemoved_MustInvoke(int value)
    {
        var list = new CustomList<int>() { 1, 2, 3, 4, 5 };
        var wasInvoked = false;
        int removedItem = -1;
        EventHandler<int> onRemoved = (sender, item) => { wasInvoked = true; removedItem = item; };
        list.ItemRemoved += onRemoved;


        list.Remove(value);


        Assert.True(wasInvoked);
        Assert.Equal(value, removedItem);
    }


    [Theory]
    [InlineData(0)]
    [InlineData(2)]
    [InlineData(4)]
    public void ItemRemoved_WhenRemovedAt_MustInvoke(int index)
    {
        var list = new CustomList<int>() { 1, 2, 3, 4, 5 };
        var wasInvoked = false;
        int removedItem = -1;
        int itemAtIndex = list[index];
```

```
        EventHandler<int> onRemoved = (sender, item) => { wasInvoked = true; removedItem = item; };
        list.ItemRemoved += onRemoved;

        list.RemoveAt(index);

        Assert.True(wasInvoked);
        Assert.Equal(itemAtIndex, removedItem);
    }

    [Theory]
    [InlineData(0,2)]
    [InlineData(2,4)]
    [InlineData(3,6)]
    public void ItemSet_WhenSet_MustInvoke(int index, int value)
    {
        var list = new CustomList<int>() { 1, 2, 3, 4, 5 };
        var wasInvoked = false;
        int indexToBeSet = -1;
        EventHandler<int> onSet = (sender, item) => { wasInvoked = true; indexToBeSet = index; };
        list.ItemSet += onSet;

        list[index] = value;

        Assert.True(wasInvoked);
        Assert.Equal(index, indexToBeSet);
    }
```

```
    }
}
```

IndexOfTest.cs:

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

namespace MyList.Tests
{
    public class IndexOfTest
    {
        [Theory]
        [MemberData(nameof(GetIndexOfData))]
        public void IndexOf_WhenListIsNotNull_MustSucceed(int
searchedValue,int expectedIndex, CustomList<int> coll)
        {
            var index = coll.IndexOf(searchedValue);

            Assert.Equal(expectedIndex, index);
        }

        public static IEnumerable<object[]> GetIndexOfData()
        {
            //returns value to search index for, expected index and initial collection
            yield return new object[] { 1, 0, new CustomList<int> { 1, 2, 3, 4, 5} };
            yield return new object[] { 3, 2, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 5, 4, new CustomList<int> { 1, 2, 3, 4, 5 } };
```

```
            yield return new object[] { 6, -1, new CustomList<int> { 1, 2, 3, 4, 5 } };
        }
    }
}
```

IndexerTest.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using Xunit;
using MyList;
using System.Linq;

namespace MyList.Tests
{
    public class IndexerTest
    {
        [Theory]
        [MemberData(nameof(GetSetterValidData))]
        public void Set_WhenIndexIsCorrect_MustSucced(int index, int value,
CustomList<int> coll)
        {
            coll[index] = value;

            int actualValue = coll.ElementAt(index);
            Assert.Equal(value, actualValue);
        }

        [Theory]
```

```
    [MemberData(nameof(GetInvalidData))]
    public void Set_WhenIndexIsOutside_MustThrow(int index,
CustomList<int> coll)
    {
        Action wrongSet = () => coll[index] = -1;

        var exception =
Assert.Throws<IndexOutOfRangeException>(wrongSet);
        Assert.Equal("Index was out of range", exception.Message);
    }

    [Theory]
    [MemberData(nameof(GetGetterValidData))]
    public void Get_WhenIndexIsCorrect_MustSucced(int index, int
expectedValue, CustomList<int> coll)
    {
        int actual = coll[index];

        Assert.Equal(expectedValue, actual);
    }

    [Theory]
    [MemberData(nameof(GetInvalidData))]
    public void Get_WhenIndexIsOutside_MustThrow(int index,
CustomList<int> coll)
    {
        Action wrongSet = () => { int a = coll[index]; };
```

```
            var exception =
Assert.Throws<IndexOutOfRangeException>(wrongSet);
            Assert.Equal("Index was out of range", exception.Message);
        }
        public static IEnumerable<object[]> GetSetterValidData()
        {
            //returns index, value and collection
            yield return new object[] { 0, 6, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 4, 6, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 2, 6, new CustomList<int> { 1, 2, 3, 4, 5 } };
        }
        public static IEnumerable<object[]> GetGetterValidData()
        {
            //returns index, expected value and collection
            yield return new object[] { 0, 1, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 4, 5, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 2, 3, new CustomList<int> { 1, 2, 3, 4, 5 } };
        }
        public static IEnumerable<object[]> GetInvalidData()
        {
            //returns index and collection
            yield return new object[] { -1, new CustomList<int> { 1, 2, 3, 4, 5 } };
            yield return new object[] { 5, new CustomList<int> { 1, 2, 3, 4, 5 } };
        }
    }
}
```

RemoveTest.cs:

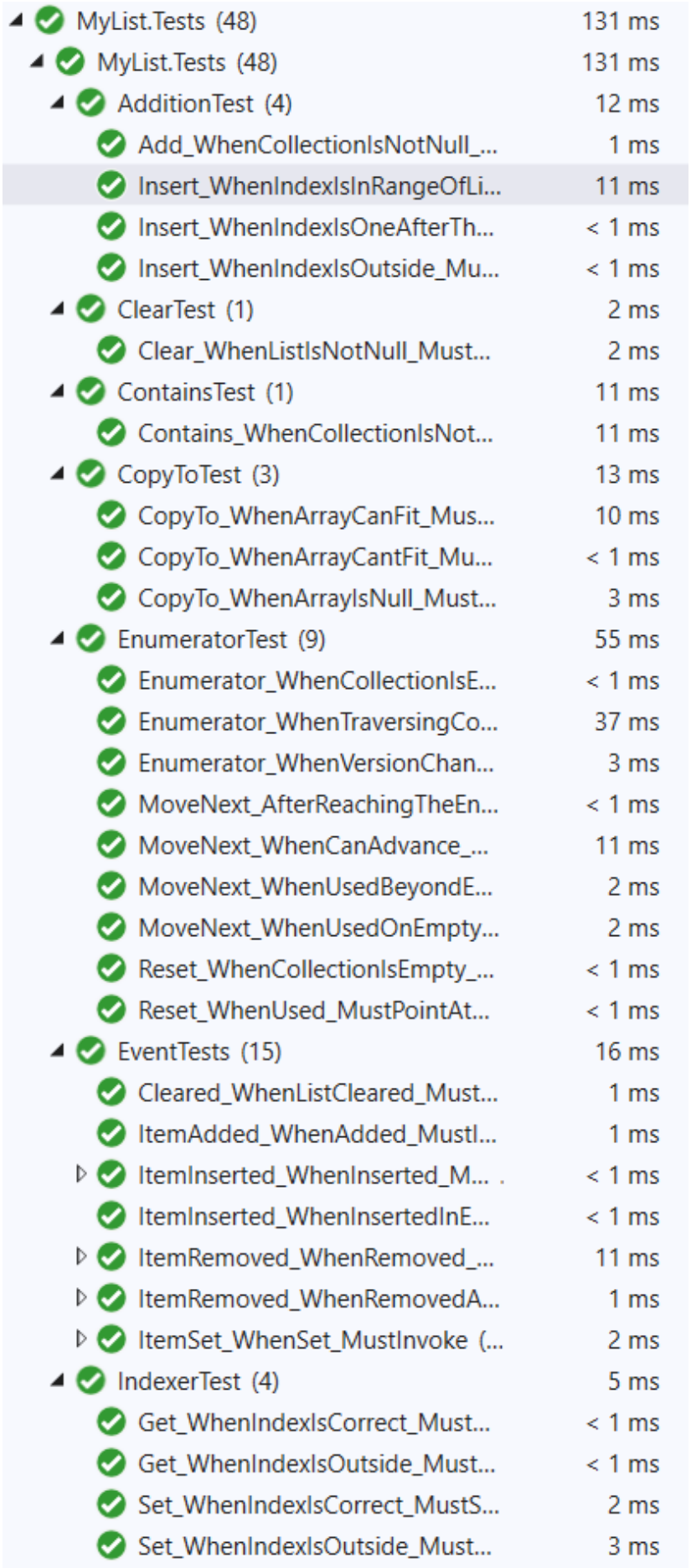
```
using System;
```

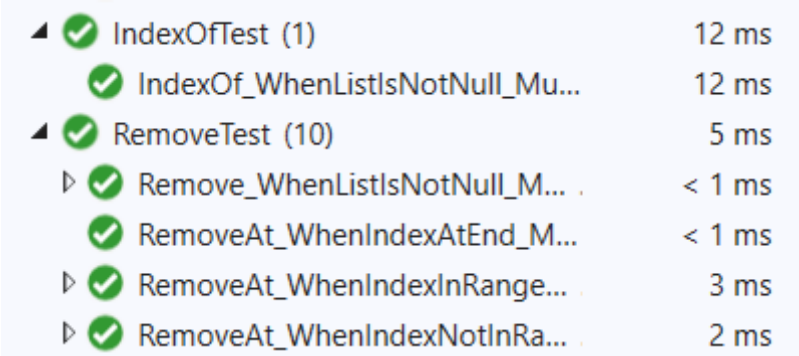
```
using System.Collections.Generic;
using System.Text;
using Xunit;
using MyList;
namespace MyList.Tests
{
    public class RemoveTest
    {
        [Theory]
        [InlineData(1, true)]
        [InlineData(3, true)]
        [InlineData(5, true)]
        [InlineData(6, false)]
        public void Remove_WhenListIsNotNull_MustSucceed(int
elementToRemove, bool expectedResult)
        {
            var coll = new CustomList<int>() { 1, 2, 3, 4, 5 };
            int initialCount = 5;

            var remove = coll.Remove(elementToRemove);

            var resultingCount = expectedResult ? initialCount-1 : initialCount;
            Assert.Equal(expectedResult, remove);
            Assert.Equal(coll.Count, resultingCount);
            Assert.DoesNotContain(elementToRemove, coll);
        }

        [Theory]
```

```
    [InlineData(0)]
    [InlineData(2)]
    [InlineData(3)]
    public void RemoveAt_WhenIndexInRangeBeforeEnd_MustSucceedAndShiftNextOnes(int index)
    {
        var coll = new CustomList<int>() { 1, 2, 3, 4, 5 };
        int elementToRemove = coll[index];
        int initiallyNextOne = coll[index + 1];
        int initialCount = 5;

        coll.RemoveAt(index);

        Assert.Equal(coll.Count, initialCount - 1);
        Assert.Equal(initiallyNextOne, coll[index]);
        Assert.DoesNotContain(elementToRemove, coll);
    }

    [Theory]
    [MemberData(nameof(GetRemoveAtTheEndData))]
    public void RemoveAt_WhenIndexAtEnd_MustSucceed(CustomList<int> coll, int index)
    {
        int elementToRemove = coll[index];
        int initialCount = coll.Count;

        coll.RemoveAt(index);
```

```
            Assert.Equal(coll.Count, initialCount - 1);
            Assert.DoesNotContain(elementToRemove, coll);
        }

        [Theory]
        [InlineData(-1)]
        [InlineData(5)]
        public void RemoveAt_WhenIndexNotInRange_MustThrow(int index)
        {
            var coll = new CustomList<int>() { 1, 2, 3, 4, 5 };

            Action removeAt = () => coll.RemoveAt(index);

            var exception =
Assert.Throws<ArgumentOutOfRangeException>(removeAt);
            Assert.Equal("Argument was out of range", exception.ParamName);
        }
        public static IEnumerable<object[]> GetRemoveAtTheEndData()
        {
            yield return new object[] { new CustomList<int> { 1, 2, 3, 4, 5 }, 4 };
            yield return new object[] { new CustomList<int> { 1 }, 0 };
        }
    }
}
```

Результати тестування:


MyList.Tests (48) 131 ms
MyList.Tests (48) 131 ms
AdditionTest (4) 12 ms
Add_WhenCollectionIsNotNull_... 1 ms
Insert_WhenIndexIsInRangeOfLi... 11 ms
Insert_WhenIndexIsOneAfterTh... < 1 ms
Insert_WhenIndexIsOutside_Mu... < 1 ms
ClearTest (1) 2 ms
Clear_WhenListIsNotNull_Must... 2 ms
ContainsTest (1) 11 ms
Contains_WhenCollectionIsNot... 11 ms
CopyToTest (3) 13 ms
CopyTo_WhenArrayCanFit_Mus... 10 ms
CopyTo_WhenArrayCantFit_Mu... < 1 ms
CopyTo_WhenArrayIsNull_Must... 3 ms
EnumeratorTest (9) 55 ms
Enumerator_WhenCollectionIsE... < 1 ms
Enumerator_WhenTraversingCo... 37 ms
Enumerator_WhenVersionChan... 3 ms
MoveNext_AfterReachingTheEn... < 1 ms
MoveNext_WhenCanAdvance_... 11 ms
MoveNext_WhenUsedBeyondE... 2 ms
MoveNext_WhenUsedOnEmpty... 2 ms
Reset_WhenCollectionIsEmpty_... < 1 ms
Reset_WhenUsed_MustPointAt... < 1 ms
EventTests (15) 16 ms
Cleared_WhenListCleared_Must... 1 ms
ItemAdded_WhenAdded_MustI... 1 ms
ItemInserted_WhenInserted_M... < 1 ms
ItemInserted_WhenInsertedInE... < 1 ms
ItemRemoved_WhenRemoved_... 11 ms
ItemRemoved_WhenRemovedA... 1 ms
ItemSet_WhenSet_MustInvoke (... 2 ms
IndexerTest (4) 5 ms
Get_WhenIndexIsCorrect_Must... < 1 ms
Get_WhenIndexIsOutside_Must... < 1 ms
Set_WhenIndexIsCorrect_MustS... 2 ms
Set_WhenIndexIsOutside_Must... 3 ms



Покриття коду тестами:

```
Passed!  - Failed:     0, Passed:    75, Skipped:     0, Total:    75, Duration: 60 ms
)

Calculating coverage result...
  Generating report 'C:\Users\Artem\source\repos\MyList\coverage.json'
+--------+--------+--------+--------+
| Module | Line   | Branch | Method |
+--------+--------+--------+--------+
| MyList | 99.53% | 97.11% | 96.15% |
+--------+--------+--------+--------+

+---------+--------+--------+--------+
|         | Line   | Branch | Method |
+---------+--------+--------+--------+
| Total   | 99.53% | 97.11% | 96.15% |
+---------+--------+--------+--------+
| Average | 99.53% | 97.11% | 96.15% |
+---------+--------+--------+--------+
```

Висновок

Отож, у ході виконання лабораторної роботи було реалізовано набір модульних тестів для попередньо створеної колекції за допомогою фреймфорку xUnit. У рамках лабораторної роботи розроблено низку тестів: як фактичних, так параметризованих. Юніт-тестами було покрито методи додавання, віднімання елементів колекції, пошук елемента за індексом та перевірки на присутність в колекції, доступ за індексом, копіювання в масив та коректність виклику подій; протестовано енумератор даного класу. Урешті-решт, реалізовані тести було перевірено на покриття коду, розгалужень та методів класу колекції. Набуто практичних навичок написання модульних тестів.