Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 2

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

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

Виконала:
Жеглова Євгенія
гр. ІК-13

Перевірив:
Викладач кафедри ІСТ
ФІОТ
Бардін В.

Київ, 2023

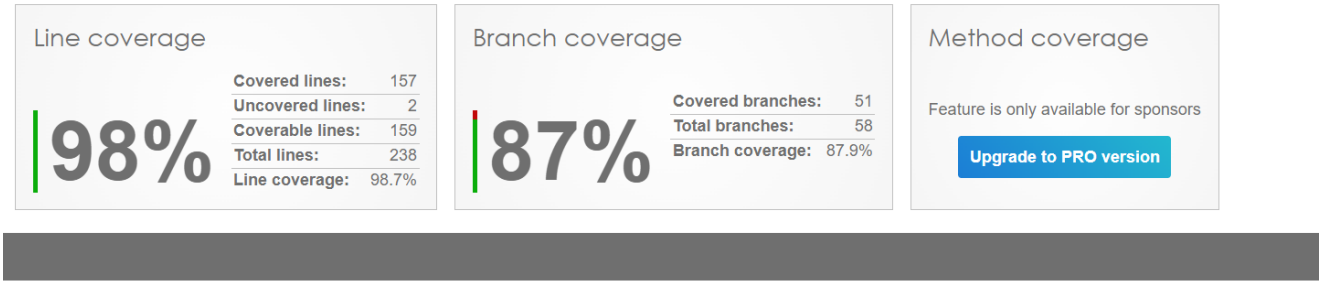
Мета лабораторної роботи – навчитися створювати модульні тести для вихідного коду розроблювального програмного забезпечення.

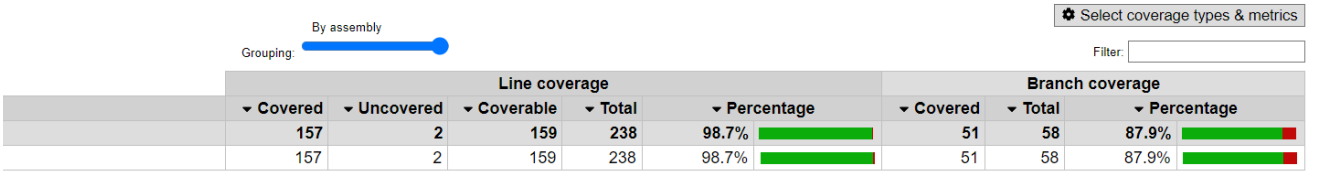
Завдання:

1. Додати до проекту власної узагальненої колекції (застосувати виконану 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.

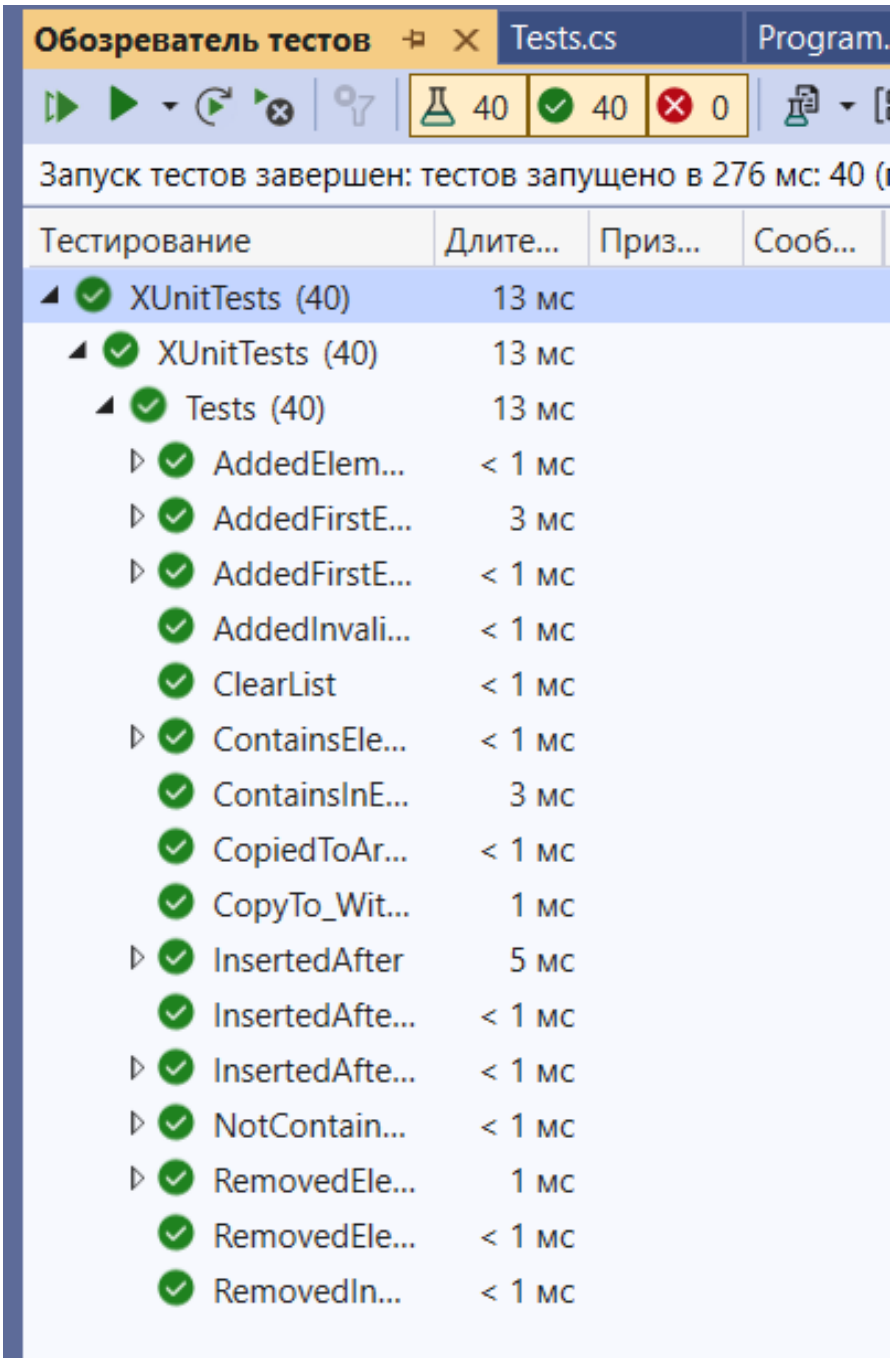
Результат роботи програми

Покриття 98,7% (використання програми Coverlet)

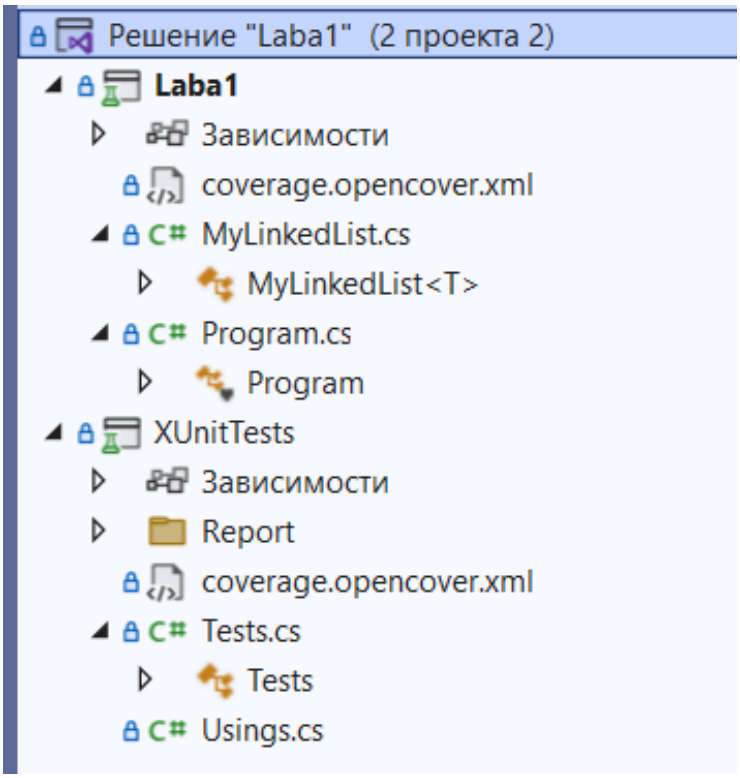




Усі тести працюють:



Код програми:




```
using System.Collections;

public class MyLinkedList<T> : ICollection<T>, IEnumerable<T>
{
    private Node<T> head;
    private Node<T> tail;
    private int count;

    public MyLinkedList()
    {
        head = null;
        tail = null;
        count = 0;
    }

    public int Count => count;
    public bool IsReadOnly => false;

    private class Node<TNode>
    {
        public TNode Data { get; }
        public Node<TNode> Next { get; set; }

        public Node(TNode data)
        {
            Data = data;
            Next = null;
        }
    }
    public void Add(T item)
    {
        if (item == null)
        {
            throw new ArgumentNullException($"{typeof(T)} {nameof(item)} is null");
        }

        Node<T> newNode = new Node<T>(item);
```

```
    if (head == null)
    {
        head = newNode;
        tail = newNode;
        tail.Next = head;
    }
    else
    {
        newNode.Next = head;
        tail.Next = newNode;
        tail = newNode;
    }
    count++;
    OnItemAdded(item);
}

public bool Remove(T item)
{
    if (item == null)
    {
        throw new ArgumentNullException($"{typeof(T)} {nameof(item)} is null");
    }

    if (head == null)
        return false;

    Node<T> current = head;
    Node<T> previous = null;

    do
    {
        if (current.Data.Equals(item))
        {
            if (previous != null)
            {
                previous.Next = current.Next;
                if (current == head)
                    head = current.Next;
                if (current == tail)
                    tail = previous;
            }
            else
            {
                head = head.Next;
                tail.Next = head;
            }
            count--;
            OnItemRemoved(item);
            return true;
        }

        previous = current;
        current = current.Next;
    } while (current != head);

    return false;
}

public bool Contains(T item)
{
    if (head == null)
        return false;

    Node<T> current = head;

    do
    {
```

```
            if (current.Data.Equals(item))
            {
                OnItemContain(item);
                return true;
            }

            current = current.Next;
        } while (current != head);

        return false;
    }

    public void Clear()
    {
        head = null;
        tail = null;
        count = 0;
    }

    public void CopyTo(T[] array, int arrayIndex)
    {
        if (array == null)
            throw new ArgumentNullException(nameof(array));

        if (arrayIndex < 0 || arrayIndex >= array.Length)
            throw new ArgumentOutOfRangeException(nameof(arrayIndex));

        if (count > array.Length - arrayIndex)
            throw new ArgumentException("The number of elements in the source collection is greater than the available space from the index to the end of the destination array.");

        Node<T> current = head;
        int currentIndex = 0;

        while (current != null && arrayIndex + currentIndex < array.Length)
        {
            array[arrayIndex + currentIndex] = current.Data;
            current = current.Next;
            currentIndex++;
        }
    }

    public bool InsertAfter(T existingItem, T newItem)
    {
        Node<T> newNode = new Node<T>(newItem);

        if (head == null)
            return false;

        Node<T> current = head;

        do
        {
            if (current.Data.Equals(existingItem))
            {
                newNode.Next = current.Next;
                current.Next = newNode;
                if (current == tail)
                    tail = newNode;
                count++;
                OnItemAdded(newItem);
                return true;
            }

            current = current.Next;
        } while (current != head);
```

```
    return false;
}

public void AddFirst(T item)
{
    Node<T> newNode = new Node<T>(item);
    if (head == null)
    {
        head = newNode;
        tail = newNode;
        tail.Next = head;
    }
    else
    {
        newNode.Next = head;
        tail.Next = newNode;
        head = newNode;
    }
    count++;
    OnItemAddedFirst(item);
}

public event Action<T> ItemAdded;

public event Action<T> ItemRemoved;

public event Action<T> Contain;

public event Action<T> ItemAddedFirst;

protected virtual void OnItemAdded(T item)
{
    ItemAdded?.Invoke(item);
}

protected virtual void OnItemRemoved(T item)
{
    ItemRemoved?.Invoke(item);
}

protected virtual void OnItemContain(T item)
{
    Contain?.Invoke(item);
}

protected virtual void OnItemAddedFirst(T item)
{
    ItemAddedFirst?.Invoke(item);
}

public IEnumerator<T> GetEnumerator()
{
    if (head != null)
    {
        Node<T> current = head;
        do
        {
            yield return current.Data;
            current = current.Next;
        } while (current != head);
    }
}

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

```
}


namespace XUnitTests
{
    public class Tests
    {
        [Theory]
        [InlineData(-11)]
        [InlineData(0)]
        [InlineData(2.1)]
        [InlineData(33)]
        [InlineData(1067 / 2)]
        public void AddedElement(int value)
        {
            MyLinkedList<int> list = new MyLinkedList<int>();
            list.Add(value);
            Assert.Contains(value, list);
            Assert.False(list.IsReadOnly);
        }

        [Fact]
        public void AddedInvalidElement()
        {
            MyLinkedList<string> list = new MyLinkedList<string>();
            Assert.Throws<ArgumentNullException>(() => list.Add(null));
        }

        [Theory]
        [InlineData(-22)]
        [InlineData(0)]
        [InlineData(4)]
        [InlineData(66)]
        public void RemovedElement(int value)
        {
            MyLinkedList<int> list = new MyLinkedList<int>() { -22, 0, 4, 66 };
            list.Remove(value);
            Assert.DoesNotContain(value, list);
            Assert.Equal(3, list.Count);
        }

        [Fact]
        public void RemovedInvalidElement()
        {
            MyLinkedList<string> list = new MyLinkedList<string>();
            Assert.Throws<ArgumentNullException>(() => list.Remove(null));
        }

        [Fact]
        public void RemovedElementFromEmptyList()
        {
            MyLinkedList<int> list = new MyLinkedList<int>() { };
            Assert.False(list.Remove(5));
        }

        [Theory]
        [InlineData(-22)]
        [InlineData(0)]
        [InlineData(4)]
        [InlineData(66)]
        public void ContainsElement(int value)
        {
            MyLinkedList<int> list = new MyLinkedList<int>() { -22, 0, 4, 66 };
            bool result = list.Contains(value);
            Assert.True(result);
        }
```

```
[Theory]
[InlineData(-11)]
[InlineData(2)]
[InlineData(5)]
[InlineData(7)]
public void NotContainsElement(int value)
{
    MyLinkedList<int> list = new MyLinkedList<int>() { -22, 0, 4, 66 };
    bool result = list.Contains(value);
    Assert.False(result);
}

[Fact]
public void ContainsInEmptyList()
{
    MyLinkedList<int> list = new MyLinkedList<int>() { };
    Assert.False(list.Contains(5));
}

[Fact]
public void ClearList()
{
    MyLinkedList<int> list = new MyLinkedList<int>() { -22, 0, 4, 66 };
    list.Clear();
    Assert.Empty(list);
}

[Fact]
public void CopiedToArray()
{
    MyLinkedList<int> list = new MyLinkedList<int>() { 1, 2, 3 };

    int[] array = new int[3];
    list.CopyTo(array, 0);

    Assert.Equal(1, array[0]);
    Assert.Equal(2, array[1]);
    Assert.Equal(3, array[2]);
}

[Fact]
public void CopyTo_WithInvalidArray()
{
    MyLinkedList<int> list = new MyLinkedList<int>();

    Assert.Throws<ArgumentNullException>(() => list.CopyTo(null, 0));

    int[] array = new int[3];
    Assert.Throws<ArgumentOutOfRangeException>(() => list.CopyTo(array, -1));

    list.Add(1);
    list.Add(2);
    list.Add(3);
    array = new int[2];
    Assert.Throws<ArgumentException>(() => list.CopyTo(array, 0));
}

[Theory]
[InlineData(-11, 0)]
[InlineData(0, 5)]
[InlineData(2, 1)]
[InlineData(33, 2)]
public void InsertedAfter(int value, int newvalue)
{
    MyLinkedList<int> list = new MyLinkedList<int>() { -11, 0, 2, 33 };
    list.InsertAfter(value, newvalue);
    Assert.Contains(newvalue, list);
```

```
        }

        [Theory]
        [InlineData(-5, 0)]
        [InlineData(1, 5)]
        [InlineData(6, 1)]
        [InlineData(105, 2)]
        public void InsertedAfterNotExistingItem(int value, int newvalue)
        {
            MyLinkedList<int> list = new MyLinkedList<int>() { -11, 0, 2, 33 };
            bool result = list.InsertAfter(value, newvalue);
            Assert.False(result);
        }

        [Fact]
        public void InsertedAfterInEmptyList()
        {
            MyLinkedList<int> list = new MyLinkedList<int>() { };
            Assert.False(list.InsertAfter(5, 2));
        }

        [Theory]
        [InlineData(-11)]
        [InlineData(0)]
        [InlineData(1)]
        [InlineData(33)]
        [InlineData(1067 / 2)]
        public void AddedFirstElement(int value)
        {
            MyLinkedList<int> list = new MyLinkedList<int>() { 2, 4 };
            list.AddFirst(value);
            Assert.Equal(value, list.First());
        }

        [Theory]
        [InlineData(0)]
        [InlineData(1067 / 2)]
        public void AddedFirstElementInEmptyList(int value)
        {
            MyLinkedList<int> list = new MyLinkedList<int>() { };
            list.AddFirst(value);
            Assert.Equal(value, list.First());
        }

    }
}
```

Висновок:

Виконуючи лабораторну роботу, я написала тести для своєї колекції за допомогою XUnit, та за допомогою Coverlet дослідила процент покриття тестами кода моєї колекції.