# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

# Лабораторна робота №2

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

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

Виконав: студент гр. IC-11 Іваніцький М. К. Викладач: Бардін В. **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.

## Лістинг програмного коду:

### MyLinkedListTests.cs

```
namespace MyCollectionTesting
{
   public abstract class Tests
       public MyLinkedList<int> list;
        [SetUp]
       public void Setup()
           list = new MyLinkedList<int>();
        }
    }
    [TestFixture]
   public class ConstructorTests
        [Test]
       public void
ConstructorWithCollection NormalCollection ResultListIsEqualToInitial()
            List<int> oldList = new List<int>() { 2, 3, 4 };
            var newList = new MyLinkedList<int>(oldList);
```

```
Assert.Multiple(() =>
            Assert.That(newList.Count, Is.EqualTo(3));
            Assert.That(newList.First!.item, Is.EqualTo(2));
            Assert.That(newList.Last!.item, Is.EqualTo(4));
            Assert.That(newList.Last.prev!.item, Is.EqualTo(3));
        });
    }
}
[TestFixture]
public class AddLastTests : Tests
{
    [Test]
    public void Add_NewValue_EmptyList_ResultListWithOneNode()
        var value = 5;
        list.Add(value);
        Assert.Multiple(() =>
            Assert.That( list.Last!.item, Is.EqualTo(value));
            Assert. That (list.Count, Is. Equal To (1));
        }
        );
    }
    [Test]
    public void Add NewValue ListWithOneNode ResultLastValueIsNewValue()
    {
        var value = 5;
        list.Add(value);
        var value1 = 7;
        list.Add(value1);
        Assert.Multiple(() =>
        {
            Assert.That( list.Last!.item, Is.EqualTo(value1));
            Assert.That( list.Count, Is.EqualTo(2));
        });
```

```
}
}
[TestFixture]
public class AddFirstTests : Tests
{
    [Test]
    public void AddFirst_NewValue_EmptyList_ResultListWithOneNode()
        var value = 5;
        _list.AddFirst(value);
        Assert.Multiple(() =>
            Assert.That( list.First!.item, Is.EqualTo(value));
            Assert.That(_list.Count, Is.EqualTo(1));
        });
    }
    [Test]
    public void AddFirst_NewValue_ListWithOneNode_ResultFirstIsNewValue()
    {
        var value = 5;
        list.AddFirst(value);
        var value1 = 7;
        _list.AddFirst(value1);
        Assert.Multiple(() =>
        {
            Assert.That( list.First!.item, Is.EqualTo(value1));
            Assert.That( list.Count, Is.EqualTo(2));
        });
    }
}
[TestFixture]
public class AddAfterTests : Tests
{
    [Test]
```

```
public void
AddAfter NewValue BetweenTwoValues ResultNewValueInItsPlace()
            var value = 5;
            list.Add(value);
            var value1 = 7;
            list.Add(value1);
            var value2 = 6;
            list.AddAfter( list.First!, value2);
            var node = list.Find(value2);
            Assert.Multiple(() =>
                Assert.That( list.First!.next, Is.EqualTo(node));
                Assert.That(node!.prev, Is.EqualTo( list.First));
                Assert. That ( list.Last!.prev, Is.EqualTo(node));
                Assert.That(node!.next, Is.EqualTo( list.Last));
                Assert.That(_list.Count, Is.EqualTo(3));
            });
        }
        [Test]
        public void AddAfter NewValue AfterLast ResultTheLastIsNewValue()
            var value = 5;
            list.Add(value);
            var value1 = 7;
            list.AddAfter( list.Last!, value1);
            var node = list.Find(7);
            Assert.Multiple(() =>
            {
                Assert.That( list.First!.next, Is.EqualTo(node));
                Assert.That(node!.prev, Is.EqualTo( list.First));
                Assert.That(node, Is.EqualTo( list.Last));
                Assert.That(node!.next, Is.EqualTo(null));
                Assert.That( list.Count, Is.EqualTo(2));
            });
        }
```

```
[Test]
        public void AddAfter_NewValue_AfterNull_ResultExeption()
            var node = list.Last;
            var value = 5;
            Assert.Throws<ArgumentNullException>(() => list.AddAfter(node,
value));
        }
        [Test]
        public void AddAfter NewValue FakeNode ResultExeption()
            var node = new Node<int>(5);
            var value = 7;
            Assert.Throws<InvalidOperationException>(() =>
list.AddAfter(node, value));
    }
    [TestFixture]
    public class AddBeforeTests : Tests
    {
        [Test]
        public void
AddBefore_NewValue_BetweenTwoValues_ResultNewValueInItsPlace()
        {
           var value = 5;
            list.Add(value);
            var value1 = 7;
            list.Add(value1);
            var value2 = 6;
            list.AddBefore( list.Last, value2);
            var node = _list.Find(value2);
            Assert.Multiple(() =>
                Assert.That( list.First!.next, Is.EqualTo(node));
                Assert.That(node!.prev, Is.EqualTo( list.First));
                Assert.That( list.Last!.prev, Is.EqualTo(node));
                Assert.That(node!.next, Is.EqualTo(list.Last));
```

```
});
        }
        [Test]
        public void AddBefore NewValue BeforeFirst ResultTheFirstIsNewValue()
            var value = 5;
            list.Add(value);
            var value1 = 7;
            list.AddBefore( list.First, value1);
            var node = _list.Find(7);
            Assert.Multiple(() =>
                Assert.That(_list.Last!.prev, Is.EqualTo(node));
                Assert.That(node!.next, Is.EqualTo( list.Last));
                Assert.That(node, Is.EqualTo(list.First));
                Assert.That(node!.prev, Is.EqualTo(null));
                Assert.That( list.Count, Is.EqualTo(2));
            });
        }
        [Test]
        public void AddBefore NewValue AfterNull ResultExeption()
            var node = _list.Last;
            var value = 5;
            Assert.Throws<ArgumentNullException>(() => list.AddBefore(node,
value));
        }
        [Test]
        public void AddBefore NewValue FakeNode ResultExeption()
            var node = new Node<int>(5);
            var value = 7;
            Assert.Throws<InvalidOperationException>(() =>
list.AddBefore(node, value));
```

Assert. That (list.Count, Is. Equal To (3));

```
}
    }
    [TestFixture]
   public class RemoveTests : Tests
        [Test]
        public void Remove_EmptyList_ResultExeption()
            Assert.That( list.Remove(0), Is.EqualTo(false));
        }
        [Test]
        public void Remove BetweenTwoNodes ResultValueRemoved()
            var value = 5;
            var value1 = 6;
            var value2 = 7;
            list.Add(value);
            list.Add(value1);
            list.Add(value2);
            list.Remove(6);
            Assert.Multiple(() =>
                Assert.That(_list.First!.next, Is.EqualTo(_list.Last));
                Assert.That( list.Last!.prev, Is.EqualTo( list.First));
                Assert.That(_list.Count, Is.EqualTo(2));
            });
        }
        [Test]
        public void Remove FakeNode ResultExeption()
            var node = new Node<int>(5);
           Assert.Throws<InvalidOperationException>(() =>
list.Remove(node));
        }
```

```
[Test]
        public void Remove_OnlyNode_ResultEmptyList()
            var node = new Node<int>(5);
            list.AddLast(node);
            list.Remove(node);
           Assert.That(_list.Count, Is.EqualTo(0));
        }
   }
   [TestFixture]
   public class RemoveFirstTests : Tests
   {
        [Test]
       public void RemoveFirst_EmptyList_ResultExeption()
            Assert.Throws<InvalidOperationException>(() =>
_list.RemoveFirst());
        }
        [Test]
        public void RemoveFirst_TwoElements_ResultSecondBecomeFirst()
           var value = 5;
           var value1 = 7;
            list.AddLast(value);
            list.AddLast(value1);
            list.RemoveFirst();
            Assert.Multiple(() =>
            {
                Assert.That( list.First!.item, Is.EqualTo(value1));
               Assert.That( list.Count, Is.EqualTo(1));
            });
        }
   }
   [TestFixture]
   public class RemoveLastTests : Tests
```

```
{
        [Test]
        public void RemoveLast_EmptyList_ResultExeption()
            Assert.Throws<InvalidOperationException>(() =>
list.RemoveLast());
        [Test]
        public void RemoveLast TwoElements ResultFirstBecomeLast()
            var value = 5;
            var value1 = 7;
            list.AddLast(value);
            list.AddLast(value1);
           list.RemoveLast();
           Assert.Multiple(() =>
                Assert.That( list.Last!.item, Is.EqualTo(value));
                Assert.That( list.Count, Is.EqualTo(1));
            });
    }
    [TestFixture]
   public class ClearTests : Tests
        [Test]
        public void Clear ListWith4Elements ResultListIsEmpty()
            for(int i = 0; i < 4; i++)
            {
                list.AddLast(i);
            _list.Clear();
            Assert.Multiple(() =>
            {
                Assert.That( list.Count, Is.EqualTo(0));
```

```
Assert.That( list.Last, Is.Null);
                Assert.That(_list.First, Is.Null);
            });
        }
    }
    [TestFixture]
    public class FindTests : Tests
        [Test]
        public void Find_EmptyList_ResultIsNull()
        {
            var node = _list.Find(2);
            Assert.That(node, Is.Null);
        }
    }
    [TestFixture]
    public class ContainsTests : Tests
        [Test]
        public void Contains ResultIsTrue()
            var value = 5;
            _list.AddLast(value);
            Assert.IsTrue(_list.Contains(value));
        }
    }
    [TestFixture]
    public class CopyToTests : Tests
        [Test]
        public void
{\tt CopyTo\_ListWith3ElementsAndValidIndexParameter\_ResultArrayWith3Elements()}
            int[] arr = new int[3];
```

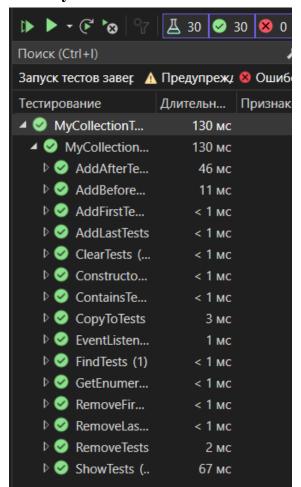
```
Node<int> node1 = new(2);
            Node<int> node2 = new(3);
            Node<int> node3 = new(4);
            list.AddLast(node1);
            list.AddLast(node2);
            _list.AddLast(node3);
            list.CopyTo(arr, 0);
            Assert.Multiple(() =>
                Assert. That (arr, Has. Length. Equal To (3));
                Assert.That(arr, Is.All.Not.Null);
            });
        }
        [Test]
        public void
CopyTo ListWith3ElementsAndWrongIndexInParams ThrowArgumentOutOfrangeExcep()
            int[] arr = new int[3];
            Node<int> node1 = new(2);
            Node<int> node2 = new(3);
            Node<int> node3 = new(4);
            list.AddLast(node1);
            list.AddLast(node2);
            list.AddLast(node3);
            TestDelegate res = () => list.CopyTo(arr, -1);
            Assert.Throws<ArgumentOutOfRangeException>(res);
        }
    }
    [TestFixture]
```

```
public class GetEnumeratorTests : Tests
        [Test]
        public void GetEnumerator ListWith3Elements AllAreCorrect()
            MyLinkedList<int>? list = new MyLinkedList<int>();
            Node<int> node1 = new(2);
            Node<int> node2 = new(3);
            Node<int> node3 = new(4);
            _list.AddLast(node1);
            list.AddLast(node2);
             list.AddLast(node3);
            foreach (var item in _list)
                list.Add(item);
            }
            Assert.That(list.First!.item, Is.EqualTo( list.First!.item));
            Assert.That(list.Last!.item, Is.EqualTo( list.Last!.item));
        }
    }
EventListenerTests.cs
[TestFixture]
    public class EventListenerTests
        private EventListener<int> _eventListener;
private MyLinkedList<int> _list;
        [SetUp]
        public void SetUp()
             eventListener = new EventListener<int>();
            _list = new MyLinkedList<int>();
        public void OnAddedHandler AddHandler TureReturn()
            var wasCalled = false;
```

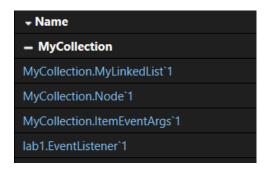
\_list.ItemAdded += \_eventListener.OnAdded!;
\_list.ItemRemoved += \_eventListener.OnRemoved!;

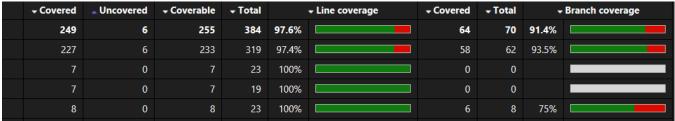
```
list.ItemAdded += (o, e) => wasCalled = true;
   Node<int> node = new(2);
   Node<int> node1 = new(3);
    _list.AddLast(node);
   list.AddLast(node1);
   Assert.That(wasCalled, Is.True);
}
[Test]
public void OnRemovedHandler_AddHandler TureReturn()
   var wasCalled = false;
    _list.ItemAdded += _eventListener.OnAdded!;
   _list.ItemRemoved += _eventListener.OnRemoved!;
   list.ItemAdded += (o, e) => wasCalled = true;
   Node<int> node = new(2);
   Node<int> node1 = new(3);
    _list.AddLast(node);
    list.AddLast(node1);
    list.RemoveLast();
   Assert. That (wasCalled, Is. True);
}
```

### Результат виконання тестів:



# Покриття тестами коду:





**Висновок:** у ході лабораторної роботи я познайомився з програмними компонентами та засобами тестування програмного коду, навчився писати юніт-тести та ознайомився з поняттям покриття коду