**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**

(шифр, прізвище, ім'я, по батькові)

*ІП-15, Тонконог В.В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Бардін В.*

Київ 2023

Варіант 8

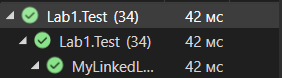
Завдання



Код виконання:

public class MyLinkedListTests  
{  
 [Fact]  
 public void CreateMyLinkedList\_WithoutParams\_EmpltyList()  
 {  
 *//Act* var actialResult = new MyLinkedList<object>();  
  
 *//Assert* actialResult.Count.Should().Be(0);  
 actialResult.Head.Should().BeNull();  
 actialResult.Tail.Should().BeNull();  
 }  
  
 [Fact]  
 public void CreateMyLinkedList\_WithNullList\_ShouldThrowArgumentNullException()  
 {  
 *//Act* var action = () => new MyLinkedList<object>(null);  
  
 *//Assert* action.Should().Throw<ArgumentNullException>();  
 }  
  
 [Fact]  
 public void Add\_NullParam\_ThrowArhumentNullException()  
 {  
 *//Arrange* var list = new MyLinkedList<object>();  
  
 *//Act* var action = () => list.Add(null);  
  
 *//Assert* action.Should().Throw<ArgumentNullException>();  
 }  
  
 [Fact]  
 public void Add\_FirstValueToEmtyList\_HeadEqualTailCount1()  
 {  
 *//Arrange* var list = new MyLinkedList<object>();  
  
 *//Act* list.Add("dd");  
  
 *//Assert* list.Head.Should().Be(list.Tail);  
 list.Head.Value.Should().Be("dd");  
 list.Count.Should().Be(1);  
 }  
  
 [Fact]  
 public void AddToNonEmptyList\_SomeValue\_CorrectTail()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new List<int> { 1, 2, 3 });  
  
 *//Act* list.Add(4);  
  
 *//Assert* list.Head.Previous.Value.Should().Be(4);  
 list.Tail.Value.Should().Be(4);  
 list.Count.Should().Be(4);  
 }  
  
 [Fact]  
 public void AddFirst\_FirstValueToEmtyList\_HeadEqualTailCount1()  
 {  
 *//Arrange* var list = new MyLinkedList<object>();  
  
 *//Act* list.AddFirst("dd");  
  
 *//Assert* list.Head.Should().Be(list.Tail);  
 list.Head.Value.Should().Be("dd");  
 list.Count.Should().Be(1);  
 }  
  
 [Fact]  
 public void AddFirstToNonEmptyList\_SomeValue\_CorrectHead()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new List<int> { 1, 2, 3 });  
  
 *//Act* list.AddFirst(4);  
  
 *//Assert* list.Head.Value.Should().Be(4);  
 list.Tail.Next.Value.Should().Be(4);  
 list.Count.Should().Be(4);  
 }  
  
 [Fact]  
 public void AddFirst\_NullParam\_ThrowArhumentNullException()  
 {  
 *//Arrange* var list = new MyLinkedList<object>();  
  
 *//Act* var action = () => list.Add(null);  
  
 *//Assert* action.Should().Throw<ArgumentNullException>();  
 }  
  
 [Fact]  
 public void Clear\_ShouldBeEmtyList()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* list.Clear();  
  
 *//Assert* list.Head.Should().BeNull();  
 list.Tail.Should().BeNull();  
 list.Count.Should().Be(0);   
 }  
  
 [Fact]  
 public void Find\_NonContains\_Null()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Find(4);  
  
 *//Assert* result.Should().BeNull();  
 }  
  
 [Fact]  
 public void Find\_Contains\_CorrectValue()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Find(3);  
  
 *//Assert* result.Value.Should().Be(3);  
 }  
  
 [Fact]  
 public void Remove\_NonContainsElement\_False()  
 {  
 *//Arrange* var array = new int[] { 1, 2, 3, };  
 var list = new MyLinkedList<int>(array);  
  
 *//Act* var result = list.Remove(4);  
  
  
 *//Assert* result.Should().Be(false);  
 list.ToList().Should().BeEquivalentTo(array);  
 }  
  
 [Fact]  
 public void Remove\_ContainsElement\_True()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Remove(4);  
  
 *//Assert* result.Should().Be(false);  
 list.Count.Should().Be(3);  
 }  
   
 [Fact]  
 public void RemoveNode\_RemoveHead\_True()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
 var newHead = list.Head.Next;  
  
 *//Act* var result = list.Remove(list.Head);  
  
 *//Assert* result.Should().Be(true);  
 list.Head.Should().Be(newHead);  
 list.Count.Should().Be(2);  
 list.ToArray().Should().BeEquivalentTo(new int[] { 2, 3 });  
 }  
   
 [Fact]  
 public void RemoveNode\_RemoveTail\_True()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
 var newTail = list.Tail.Previous;  
  
 *//Act* var result = list.Remove(list.Tail);  
  
 *//Assert* result.Should().Be(true);  
 list.Tail.Should().Be(newTail);  
 list.Count.Should().Be(2);  
 list.ToArray().Should().BeEquivalentTo(new int[] { 1, 2 });  
 }  
   
 [Fact]  
 public void RemoveNode\_RemoveAloneElement\_True()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1 });  
  
 *//Act* var result = list.Remove(list.Tail);  
  
 *//Assert* result.Should().Be(true);  
 list.Tail.Should().BeNull();  
 list.Head.Should().BeNull();  
 list.Count.Should().Be(0);  
 list.ToArray().Should().BeEquivalentTo(new int[] {});  
 }  
   
 [Fact]  
 public void RemoveNode\_RemoveNonContailsNode\_False()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3 });  
 var oldHead = list.Head;  
 var oldTail = list.Tail;  
 var nodeToRemove = new MyLinkedListNode<int>(1);  
  
 *//Act* var result = list.Remove(nodeToRemove);  
  
 *//Assert* result.Should().Be(false);  
 list.Tail.Should().Be(oldTail);  
 list.Head.Should().Be(oldHead);  
 list.Count.Should().Be(3);  
 list.ToArray().Should().BeEquivalentTo(new int[] { 1, 2, 3 });  
 }  
   
 [Fact]  
 public void RemoveNode\_Null\_ShouldThrowArgumentNullException()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1 });  
 var oldHead = list.Head;  
 var oldTail = list.Tail;  
  
 *//Act* var action = () => list.Remove(null);  
  
 *//Assert* action.Should().Throw<ArgumentNullException>();  
 list.Tail.Should().Be(oldTail);  
 list.Head.Should().Be(oldHead);  
 list.Count.Should().Be(1);  
 list.ToArray().Should().BeEquivalentTo(new int[] { 1 });  
 }  
  
 [Fact]  
 public void Contains\_NonContains\_False()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Contains(4);  
  
 *//Assert* result.Should().BeFalse();  
 }  
  
 [Fact]  
 public void Contains\_ContainsElement\_True()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Contains(3);  
  
 *//Assert* result.Should().BeTrue();  
 }  
  
 [Fact]  
 public void ContainsNode\_ContainsNode\_True()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Contains(list.Head);  
  
 *//Assert* result.Should().BeTrue();  
 }  
  
 [Fact]  
 public void ContainsNode\_NonContainsNode\_Flase()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Contains(new MyLinkedListNode<int>(1));  
  
 *//Assert* result.Should().BeFalse();  
 }  
  
 [Fact]  
 public void ContainsNode\_NullNode\_Flase()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* var result = list.Contains(null);  
  
 *//Assert* result.Should().BeFalse();  
 }  
  
 [Fact]  
 public void CopyTo\_ArrayIndexLessThan0\_ShouldThrowArgumentOutOfRangeException()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
 var arr = new int[10];  
  
 *//Act* var action = () => list.CopyTo(arr, -1);  
  
 *//Assert* action.Should().Throw<ArgumentOutOfRangeException>();  
 }  
  
 [Fact]  
 public void CopyTo\_ArrayIndexMoreThenArrayLength\_ShouldThrowArgumentOutOfRangeException()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
 var arr = new int[10];  
  
 *//Act* var action = () => list.CopyTo(arr, 12);  
  
 *//Assert* action.Should().Throw<ArgumentOutOfRangeException>();  
 }  
  
 [Fact]  
 public void CopyTo\_ArrayIndexMoreThenArrayCapacity\_ShouldThrowArgumentException()  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
 var arr = new int[10];  
  
 *//Act* var action = () => list.CopyTo(arr, 9);  
  
 *//Assert* action.Should().Throw<ArgumentException>();  
 }  
  
 [Theory]  
 [InlineData(new int[] { 1, 2, 3, 4, 5 }, 2, new int[] { 1, 2, 1, 2, 3 })]  
 [InlineData(new int[] { 1, 2, 3, 4, 5 }, 0, new int[] { 1, 2, 3, 4, 5 })]  
 [InlineData(new int[] { 1, 2, 3, 4, 5 }, 1, new int[] { 1, 1, 2, 3, 5 })]  
 public void CopyTo\_CorrectCopy(int[] actualArray, int arayIndex, int[] expectedResult)  
 {  
 *//Arrange* var list = new MyLinkedList<int>(new int[] { 1, 2, 3, });  
  
 *//Act* list.CopyTo(actualArray, arayIndex);  
  
 *//Assert* actualArray.Should().BeEquivalentTo(expectedResult);  
 }  
  
 [Theory]  
 [InlineData(new int[] { 1, 2, 3, 4, 5 }, new int[] { 1, 2, 3, 4, 5 })]  
 [InlineData(new int[] { 1, 27, 35 }, new int[] { 1, 27, 35 })]  
 public void GetEnumerator\_EmptyList\_EmptyResult(IEnumerable<int> list, int[] expectedResult)  
 {  
 *//Arrange* var myLinkedList = new MyLinkedList<int>(list);  
  
 *//Act* var actualResult = myLinkedList.ToList();  
  
 *//Assert* actualResult.Should().BeEquivalentTo(expectedResult);  
 }  
   
 [Fact]  
 public void GetHashCode\_ValueIsNull\_ShouldBeZero()  
 {  
 *//Arrange* var list = new MyLinkedList<string>(new string[] { "ddd" });  
 list.Head.Value = null;  
  
 *//Act* var hashCode = list.Head.GetHashCode();  
  
 *//Assert* hashCode.Should().Be(0);  
 }  
   
 [Fact]  
 public void GetHashCode\_ForEqualLists\_ShouldBeEqual()  
 {  
 *//Arrange* var listFirst = new MyLinkedList<string>(new string[] { "ddd" });  
 var listSecond = new MyLinkedList<string>(new string[] { "ddd" });  
  
 *//Act* var hashCodeFirst = listFirst.Head.GetHashCode();  
 var hashCodeSecond = listSecond.Head.GetHashCode();  
  
 *//Assert* hashCodeFirst.Should().Be(hashCodeSecond);  
 }  
   
 [Fact]  
 public void IsReadOnly\_ShouldBeFalse()  
 {  
 *//Arrange* var list = new MyLinkedList<string>(new string[] { "ddd" });  
  
 *//Act* var isReadOnly = list.IsReadOnly;  
  
 *//Assert* isReadOnly.Should().BeFalse();  
 }  
}

Результат виконання тестів:



Ступінь покриття колекції тестами:



