Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №2**

з дисципліни "Сучасні технології розробки WEB-застосувань на платформі .NET"

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

Варіант: №8 Кільцевий список

Виконав: Перевірила:

студент групи ІП-93 Крамар Юлія Михайлівна

Домінський Валентин

Олексійович

Київ 2022

Зміст:

[Мета: 3](#_Toc99827371)

[Вихідний код 3](#_Toc99827372)

[Результат роботи: 26](#_Toc99827373)

[Контрольні питання: 26](#_Toc99827374)

[Висновки: 28](#_Toc99827375)

## Мета:

Навчитися створювати модульні тести для вихідного коду розроблювального програмного забезпечення

## Вихідний код

CircularLinkedListNodeTests:

using Lab1.CircularLinkedListNode;

using Xunit;

namespace Lab2;

public class CircularLinkedListNodeTests

{

[Theory]

[InlineData(0)]

[InlineData(-1)]

[InlineData(1)]

[InlineData(int.MinValue)]

[InlineData(int.MaxValue)]

public void Constructor\_Int\_ReturnsCorrectValues(int expected)

{

// Arrange

var node = new CircularLinkedListNode<int>(expected);

// Act

var actualData = node.Data;

var actualNext = node.Next;

// Assert

Assert.Equal(expected, actualData);

Assert.Null(actualNext);

}

[Theory]

[InlineData("Паляниця")]

[InlineData("Русский военный корабль")]

[InlineData("European Union")]

[InlineData("汉字 and 漢字")]

[InlineData("الْعَرَبِيَّة")]

[InlineData("👾🤓😎🥸🤩🥳")]

public void Constructor\_String\_ReturnsCorrectValues(string expected)

{

// Arrange

var node = new CircularLinkedListNode<string>(expected);

// Act

var actualData = node.Data;

var actualNext = node.Next;

// Assert

Assert.Equal(expected, actualData);

Assert.Null(actualNext);

}

}

CircularLinkedListTests:

using System;

using System.Collections.Generic;

using Lab1.CircularLinkedList;

using Xunit;

namespace Lab2;

public class CircularLinkedListTests

{

#region PremadeData

public static IEnumerable<object[]> IntTestData => new List<object[]>

{

new object[] { 0 },

new object[] { 1 },

new object[] { -1 },

new object[] { int.MaxValue },

new object[] { int.MinValue },

};

public static IEnumerable<object[]> StringTestData => new List<object[]>

{

new object[] { "Паляниця" },

new object[] { "Русский военный корабль" },

new object[] { "European Union" },

new object[] { "汉字" },

new object[] { "الْعَرَبِيَّة" },

new object[] { "👾🤓😎🥸🤩🥳" },

new object[] { "█Ã░╬ðØ" },

};

public static IEnumerable<object[]> IntTwoElementsArrayTestData => new List<object[]>

{

new object[] { 0, 10 },

new object[] { 1, 11 },

new object[] { -1, -11 },

new object[] { int.MaxValue, int.MaxValue - 10 },

new object[] { int.MinValue, int.MinValue + 10 },

};

public static IEnumerable<object[]> StringTwoElementsArrayTestData => new List<object[]>

{

new object[] { "Паляниця", "Полуниця" },

new object[] { "Русский военный корабль", "Иди" },

new object[] { "European Union", "NATO" },

new object[] { "汉字", "漢字" },

new object[] { "الْعَرَبِيَّة", "الْحُرُوف" },

new object[] { "👾🤓😎🥸🤩🥳", "🧳🌂☂️🧵🧶👓" },

new object[] { "█Ã░╬ðØ", "®ßƒ≡¾Æ" },

};

public static IEnumerable<object[]> IntMultipleElementsArrayTestData => new List<object[]>

{

new object[] { 0, 10, 100 },

new object[] { 1, 11, 111 },

new object[] { -1, -11, -111 },

new object[] { int.MaxValue, int.MaxValue - 10, int.MaxValue - 100 },

new object[] { int.MinValue, int.MinValue + 10, int.MinValue + 100 },

};

public static IEnumerable<object[]> StringMultipleElementsArrayTestData => new List<object[]>

{

new object[] { "Паляниця", "Полуниця", "ОлЕні, Олені" },

new object[] { "Русский военный корабль", "Иди", "далеко" },

new object[] { "European Union", "NATO", "IAEA" },

new object[] { "汉字", "漢字", "ッミツテヅ" },

new object[] { "الْعَرَبِيَّة", "الْحُرُوف", "هِجَائِي" },

new object[] { "👾🤓😎🥸🤩🥳", "🧳🌂☂️🧵🧶👓", "🐶🐱🐭🐹🐰🦊" },

new object[] { "█Ã░╬ðØ", "®ßƒ≡¾Æ", "123456" },

};

#endregion PremadeData

#region Constructors

[Fact]

public void Constructor\_NoParameters\_IntType\_ReturnsCorrectValues()

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

var expectedCount = 0;

// Act

var actualTail = circularLinkedList.Tail;

var actualHead = circularLinkedList.Head;

var actualCount = circularLinkedList.Count;

var actualIsReadOnly = circularLinkedList.IsReadOnly;

// Assert

Assert.Null(actualTail);

Assert.Null(actualHead);

Assert.Equal(expectedCount, actualCount);

Assert.False(actualIsReadOnly);

}

[Fact]

public void Constructor\_NoParameters\_StringType\_ReturnsCorrectValues()

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

var expectedCount = 0;

// Act

var actualTail = circularLinkedList.Tail;

var actualHead = circularLinkedList.Head;

var actualCount = circularLinkedList.Count;

var actualIsReadOnly = circularLinkedList.IsReadOnly;

// Assert

Assert.Null(actualTail);

Assert.Null(actualHead);

Assert.Equal(expectedCount, actualCount);

Assert.False(actualIsReadOnly);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void Constructor\_WithParameter\_IntType\_ReturnsCorrectValues(int expectedData)

{

// Arrange

var expectedCount = 1;

var circularLinkedList = new CircularLinkedList<int>(expectedData);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualHeadData);

Assert.Equal(expectedData, actualTailData);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Constructor\_WithParameter\_StringType\_ReturnsCorrectValues(string expectedData)

{

// Arrange

var expectedCount = 1;

var circularLinkedList = new CircularLinkedList<string>(expectedData);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualHeadData);

Assert.Equal(expectedData, actualTailData);

}

#endregion Constructors

#region Indexer

#region IndexerGet

[Theory]

[MemberData(nameof(IntTestData))]

public void IndexerGet\_OneElement\_IntType\_ReturnsCorrectValues

(int expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>(expectedData);

// Act

var actualIndexData = circularLinkedList[0];

// Assert

Assert.Equal(expectedData, actualIndexData);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void IndexerGet\_OneElement\_StringType\_ReturnsCorrectValues

(string expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(expectedData);

// Act

var actualIndexData = circularLinkedList[0];

// Assert

Assert.Equal(expectedData, actualIndexData);

}

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void IndexerGet\_DifferentElements\_IntType\_ReturnsCorrectValues

(int expectedHead, int expectedTail)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.Add(expectedHead);

circularLinkedList.Add(expectedTail);

// Act

var actualHeadIndexData = circularLinkedList[0];

var actualTailIndexData = circularLinkedList[1];

// Assert

Assert.Equal(expectedHead, actualHeadIndexData);

Assert.Equal(expectedTail, actualTailIndexData);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void IndexerGet\_DifferentElements\_StringType\_ReturnsCorrectValues

(string expectedHead, string expectedTail)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.Add(expectedHead);

circularLinkedList.Add(expectedTail);

// Act

var actualHeadIndexData = circularLinkedList[0];

var actualTailIndexData = circularLinkedList[1];

// Assert

Assert.Equal(expectedHead, actualHeadIndexData);

Assert.Equal(expectedTail, actualTailIndexData);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void IndexerGet\_OneElement\_IntType\_ReturnsException

(int data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>(data);

// Assert

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[1]);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void IndexerGet\_OneElement\_StringType\_ReturnsException

(string data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(data);

// Assert

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[1]);

}

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void IndexerGet\_DifferentElements\_IntType\_ReturnsException

(int head, int tail)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.Add(head);

circularLinkedList.Add(tail);

// Assert

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[2]);

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[3]);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void IndexerGet\_DifferentElements\_StringType\_ReturnsException

(string head, string tail)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.Add(head);

circularLinkedList.Add(tail);

// Assert

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[2]);

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[3]);

}

#endregion IndexerGet

#region IndexerSet

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void IndexerSet\_OneElement\_IntType\_ReturnsCorrectValues

(int dataToChange, int expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>(dataToChange);

// Act

circularLinkedList[0] = expectedData;

var actualIndexData = circularLinkedList[0];

// Assert

Assert.Equal(expectedData, actualIndexData);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void IndexerSet\_OneElement\_StringType\_ReturnsCorrectValues

(string dataToChange, string expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(dataToChange);

// Act

circularLinkedList[0] = expectedData;

var actualIndexData = circularLinkedList[0];

// Assert

Assert.Equal(expectedData, actualIndexData);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void IndexerSet\_OneElement\_IntType\_ReturnsOutOfRangeException

(int expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

// Assert

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[0] = expectedData);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void IndexerSet\_OneElement\_StringType\_ReturnsOutOfRangeException

(string expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

// Assert

Assert.Throws<ArgumentOutOfRangeException>(() => circularLinkedList[0] = expectedData);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void IndexerSet\_OneElement\_StringType\_ReturnsNullReferenceException

(string data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(data);

// Assert

Assert.Throws<ArgumentNullException>(() => circularLinkedList[0] = null);

}

#endregion IndexerSet

#endregion Indexer

#region Add

[Theory]

[MemberData(nameof(IntTestData))]

public void Add\_ConstructorNoParameters\_IntType\_ReturnsCorrectValues

(int expectedData)

{

// Arrange

var expectedCount = 1;

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.Add(expectedData);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualHeadData);

Assert.Equal(expectedData, actualTailData);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Add\_ConstructorNoParameters\_StringType\_ReturnsCorrectValues

(string expectedData)

{

// Arrange

var expectedCount = 1;

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.Add(expectedData);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualHeadData);

Assert.Equal(expectedData, actualTailData);

}

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void Add\_ConstructorWithParameter\_IntType\_ReturnsCorrectValues

(int expectedHead, int expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(expectedHead);

circularLinkedList.Add(expectedTail);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void Add\_ConstructorWithParameter\_StringType\_ReturnsCorrectValues

(string expectedHead, string expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(expectedHead);

circularLinkedList.Add(expectedTail);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

#endregion Add

#region AddFirst

[Theory]

[MemberData(nameof(IntTestData))]

public void AddFirst\_ConstructorNoParameters\_IntType\_ReturnsCorrectValues

(int expectedData)

{

// Arrange

var expectedCount = 1;

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.AddFirst(expectedData);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualHeadData);

Assert.Equal(expectedData, actualTailData);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void AddFirst\_ConstructorNoParameters\_StringType\_ReturnsCorrectValues

(string expectedData)

{

// Arrange

var expectedCount = 1;

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.AddFirst(expectedData);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualHeadData);

Assert.Equal(expectedData, actualTailData);

}

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void AddFirst\_ConstructorWithParameter\_IntType\_ReturnsCorrectValues

(int expectedHead, int expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(expectedTail);

circularLinkedList.AddFirst(expectedHead);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void AddFirst\_ConstructorWithParameter\_StringType\_ReturnsCorrectValues

(string expectedHead, string expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(expectedTail);

circularLinkedList.AddFirst(expectedHead);

// Act

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

#endregion AddFirst

#region AddAt

[Theory]

[MemberData(nameof(IntMultipleElementsArrayTestData))]

public void AddAt\_IntType\_ReturnsCorrectValues

(int head, int expectedData, int tail)

{

// Arrange

var expectedCount = 3;

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.Add(head);

circularLinkedList.Add(tail);

// Act

circularLinkedList.AddAt(expectedData, 1);

var actualCount = circularLinkedList.Count;

var actualData = circularLinkedList[1];

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualData);

}

[Theory]

[MemberData(nameof(StringMultipleElementsArrayTestData))]

public void AddAt\_StringType\_ReturnsCorrectValues

(string head, string expectedData, string tail)

{

// Arrange

var expectedCount = 3;

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.Add(head);

circularLinkedList.Add(tail);

// Act

circularLinkedList.AddAt(expectedData, 1);

var actualCount = circularLinkedList.Count;

var actualData = circularLinkedList[1];

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedData, actualData);

}

#endregion AddAt

#region Clear

[Fact]

public void Clear\_ConstructorNoParameters\_IntType\_ReturnsCorrectValues()

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.Clear();

// Act

var actualCount = circularLinkedList.Count;

var actualHead = circularLinkedList.Head;

var actualTail = circularLinkedList.Tail;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Null(actualHead);

Assert.Null(actualTail);

}

[Fact]

public void Clear\_ConstructorNoParameters\_StringType\_ReturnsCorrectValues()

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.Clear();

// Act

var actualCount = circularLinkedList.Count;

var actualHead = circularLinkedList.Head;

var actualTail = circularLinkedList.Tail;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Null(actualHead);

Assert.Null(actualTail);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void Clear\_ConstructorNoParameters\_AddFirst\_IntType\_ReturnsCorrectValues

(int expectedData)

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.AddFirst(expectedData);

circularLinkedList.Clear();

// Act

var actualCount = circularLinkedList.Count;

var actualHead = circularLinkedList.Head;

var actualTail = circularLinkedList.Tail;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Null(actualHead);

Assert.Null(actualTail);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Clear\_ConstructorNoParameters\_AddFirst\_StringType\_ReturnsCorrectValues

(string expectedData)

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.AddFirst(expectedData);

circularLinkedList.Clear();

// Act

var actualCount = circularLinkedList.Count;

var actualHead = circularLinkedList.Head;

var actualTail = circularLinkedList.Tail;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Null(actualHead);

Assert.Null(actualTail);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void Clear\_ConstructorNoParameters\_Add\_IntType\_ReturnsCorrectValues

(int expectedData)

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.Add(expectedData);

circularLinkedList.Clear();

// Act

var actualCount = circularLinkedList.Count;

var actualHead = circularLinkedList.Head;

var actualTail = circularLinkedList.Tail;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Null(actualHead);

Assert.Null(actualTail);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Clear\_ConstructorNoParameters\_Add\_StringType\_ReturnsCorrectValues

(string expectedData)

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.Add(expectedData);

circularLinkedList.Clear();

// Act

var actualCount = circularLinkedList.Count;

var actualHead = circularLinkedList.Head;

var actualTail = circularLinkedList.Tail;

// Assert

Assert.Equal(expectedCount, actualCount);

Assert.Null(actualHead);

Assert.Null(actualTail);

}

#endregion Clear

#region Contains

[Theory]

[MemberData(nameof(IntTestData))]

public void Contains\_NoElements\_IntType\_ReturnsFalse

(int data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

// Act

var actual = circularLinkedList.Contains(data);

// Assert

Assert.False(actual);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Contains\_NoElements\_StringType\_ReturnsFalse

(string data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

// Act

var actual = circularLinkedList.Contains(data);

// Assert

Assert.False(actual);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void Contains\_DifferentElements\_AddFirst\_IntType\_ReturnsTrue

(int data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.AddFirst(data);

// Act

var actual = circularLinkedList.Contains(data);

// Assert

Assert.True(actual);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Contains\_DifferentElements\_AddFirst\_StringType\_ReturnsTrue

(string data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.AddFirst(data);

// Act

var actual = circularLinkedList.Contains(data);

// Assert

Assert.True(actual);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void Contains\_DifferentElements\_Add\_IntType\_ReturnsTrue

(int data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.Add(data);

// Act

var actual = circularLinkedList.Contains(data);

// Assert

Assert.True(actual);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Contains\_DifferentElements\_Add\_StringType\_ReturnsTrue

(string data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.Add(data);

// Act

var actual = circularLinkedList.Contains(data);

// Assert

Assert.True(actual);

}

[Fact]

public void Contains\_NullElement\_StringType\_ReturnsException()

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

// Assert

Assert.Throws<ArgumentNullException>(() => circularLinkedList.Contains(null));

}

#endregion Contains

#region CopyTo

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void CopyTo\_DifferentElements\_IntType\_ReturnsCorrectValues

(int expectedFirstData, int expectedSecondData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>(expectedFirstData);

circularLinkedList.Add(expectedSecondData);

// Act

int[] actualArray = new int[2];

circularLinkedList.CopyTo(actualArray, 0);

// Assert

Assert.Equal(circularLinkedList[0], actualArray[0]);

Assert.Equal(circularLinkedList[1], actualArray[1]);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void CopyTo\_DifferentElements\_StringType\_ReturnsCorrectValues

(string expectedFirstData, string expectedSecondData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(expectedFirstData);

circularLinkedList.Add(expectedSecondData);

// Act

string[] actualArray = new string[2];

circularLinkedList.CopyTo(actualArray, 0);

// Assert

Assert.Equal(circularLinkedList[0], actualArray[0]);

Assert.Equal(circularLinkedList[1], actualArray[1]);

}

#endregion CopyTo

#region Clone

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void Clone\_DifferentElements\_IntType\_ReturnsCorrectValues

(int expectedFirstData, int expectedSecondData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>(expectedFirstData);

circularLinkedList.Add(expectedSecondData);

// Act

var actual = (CircularLinkedList<int>)circularLinkedList.Clone();

// Assert

Assert.Equal(circularLinkedList[0], actual[0]);

Assert.Equal(circularLinkedList[1], actual[1]);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void Clone\_DifferentElements\_StringType\_ReturnsCorrectValues

(string expectedFirstData, string expectedSecondData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(expectedFirstData);

circularLinkedList.Add(expectedSecondData);

// Act

var actual = (CircularLinkedList<string>)circularLinkedList.Clone();

// Assert

Assert.Equal(circularLinkedList[0], actual[0]);

Assert.Equal(circularLinkedList[1], actual[1]);

}

#endregion Clone

#region RemoveAt

[Theory]

[MemberData(nameof(IntMultipleElementsArrayTestData))]

public void RemoveAt\_RemoveHead\_IntType\_ReturnsCorrectValues

(int head, int expectedMiddleElement, int expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(head);

circularLinkedList.Add(expectedMiddleElement);

circularLinkedList.Add(expectedTail);

// Act

circularLinkedList.RemoveAt(0);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(head, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedMiddleElement, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

[Theory]

[MemberData(nameof(StringMultipleElementsArrayTestData))]

public void RemoveAt\_RemoveHead\_StringType\_ReturnsCorrectValues

(string head, string expectedMiddleElement, string expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(head);

circularLinkedList.Add(expectedMiddleElement);

circularLinkedList.Add(expectedTail);

// Act

circularLinkedList.RemoveAt(0);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(head, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedMiddleElement, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

[Theory]

[MemberData(nameof(IntMultipleElementsArrayTestData))]

public void RemoveAt\_RemoveMiddle\_IntType\_ReturnsCorrectValues

(int expectedHead, int middleElement, int expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(expectedHead);

circularLinkedList.Add(middleElement);

circularLinkedList.Add(expectedTail);

// Act

circularLinkedList.RemoveAt(1);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(middleElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

[Theory]

[MemberData(nameof(StringMultipleElementsArrayTestData))]

public void RemoveAt\_RemoveMiddle\_StringType\_ReturnsCorrectValues

(string expectedHead, string middleElement, string expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(expectedHead);

circularLinkedList.Add(middleElement);

circularLinkedList.Add(expectedTail);

// Act

circularLinkedList.RemoveAt(1);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(middleElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

}

[Theory]

[MemberData(nameof(IntMultipleElementsArrayTestData))]

public void RemoveAt\_RemoveTail\_IntType\_ReturnsCorrectValues

(int expectedHead, int expectedMiddleElement, int tail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(expectedHead);

circularLinkedList.Add(expectedMiddleElement);

circularLinkedList.Add(tail);

// Act

circularLinkedList.RemoveAt(2);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(tail, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedMiddleElement, actualTailData);

}

[Theory]

[MemberData(nameof(StringMultipleElementsArrayTestData))]

public void RemoveAt\_RemoveTail\_StringType\_ReturnsCorrectValues

(string expectedHead, string expectedMiddleElement, string tail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(expectedHead);

circularLinkedList.Add(expectedMiddleElement);

circularLinkedList.Add(tail);

// Act

circularLinkedList.RemoveAt(2);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(tail, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedMiddleElement, actualTailData);

}

#endregion RemoveAt

#region Remove

[Theory]

[MemberData(nameof(IntMultipleElementsArrayTestData))]

public void Remove\_MultipleElements\_IntType\_ReturnsCorrectValues

(int expectedHead, int data, int expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(expectedHead);

circularLinkedList.Add(data);

circularLinkedList.Add(expectedTail);

// Act

circularLinkedList.Remove(data);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(data, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

Assert.False(circularLinkedList.Remove(data));

}

[Theory]

[MemberData(nameof(StringMultipleElementsArrayTestData))]

public void Remove\_MultipleElements\_StringType\_ReturnsCorrectValues

(string expectedHead, string data, string expectedTail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(expectedHead);

circularLinkedList.Add(data);

circularLinkedList.Add(expectedTail);

// Act

circularLinkedList.Remove(data);

var actualCount = circularLinkedList.Count;

var actualHeadData = circularLinkedList.Head.Data;

var actualTailData = circularLinkedList.Tail.Data;

// Assert

Assert.DoesNotContain(data, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

Assert.Equal(expectedHead, actualHeadData);

Assert.Equal(expectedTail, actualTailData);

Assert.False(circularLinkedList.Remove(data));

}

[Theory]

[MemberData(nameof(StringTestData))]

public void Remove\_MultipleElements\_StringType\_ReturnsArgumentNullException

(string data)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(data);

// Assert

Assert.Throws<ArgumentNullException>(() => circularLinkedList.Remove(null));

}

#endregion Remove

#region RemoveAll

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void RemoveAll\_RemoveMiddleElements\_IntType\_ReturnsCorrectValues

(int firstElement, int secondElement)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(firstElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(firstElement);

// Act

circularLinkedList.RemoveAll(secondElement);

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(secondElement, circularLinkedList);

Assert.Contains(firstElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void RemoveAll\_RemoveMiddleElements\_StringType\_ReturnsCorrectValues

(string firstElement, string secondElement)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(firstElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(firstElement);

// Act

circularLinkedList.RemoveAll(secondElement);

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(secondElement, circularLinkedList);

Assert.Contains(firstElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void RemoveAll\_RemoveEdgeElements\_IntType\_ReturnsCorrectValues

(int firstElement, int secondElement)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(firstElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(firstElement);

// Act

circularLinkedList.RemoveAll(firstElement);

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(firstElement, circularLinkedList);

Assert.Contains(secondElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void RemoveAll\_RemoveEdgeElements\_StringType\_ReturnsCorrectValues

(string firstElement, string secondElement)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(firstElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(secondElement);

circularLinkedList.Add(firstElement);

// Act

circularLinkedList.RemoveAll(firstElement);

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(firstElement, circularLinkedList);

Assert.Contains(secondElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

[Theory]

[MemberData(nameof(IntTestData))]

public void RemoveAll\_AllSameElements\_IntType\_ReturnsCorrectValues

(int firstElement)

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<int>(firstElement);

for (int i = 0; i < 4; i++)

{

circularLinkedList.Add(firstElement);

}

// Act

circularLinkedList.RemoveAll(firstElement);

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(firstElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

[Theory]

[MemberData(nameof(StringTestData))]

public void RemoveAll\_AllSameElements\_StringType\_ReturnsCorrectValues

(string firstElement)

{

// Arrange

var expectedCount = 0;

var circularLinkedList = new CircularLinkedList<string>(firstElement);

for (int i = 0; i < 4; i++)

{

circularLinkedList.Add(firstElement);

}

// Act

circularLinkedList.RemoveAll(firstElement);

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(firstElement, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

#endregion RemoveAll

#region RemoveHead

[Theory]

[MemberData(nameof(IntMultipleElementsArrayTestData))]

public void RemoveHead\_IntType\_ReturnsCorrectValues

(int head, int middle, int tail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(head);

circularLinkedList.Add(middle);

circularLinkedList.Add(tail);

// Act

circularLinkedList.RemoveHead();

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(head, circularLinkedList);

Assert.Contains(middle, circularLinkedList);

Assert.Contains(tail, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

[Theory]

[MemberData(nameof(StringMultipleElementsArrayTestData))]

public void RemoveHead\_StringType\_ReturnsCorrectValues

(string head, string middle, string tail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(head);

circularLinkedList.Add(middle);

circularLinkedList.Add(tail);

// Act

circularLinkedList.RemoveHead();

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(head, circularLinkedList);

Assert.Contains(middle, circularLinkedList);

Assert.Contains(tail, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

#endregion RemoveHead

#region RemoveTail

[Theory]

[MemberData(nameof(IntMultipleElementsArrayTestData))]

public void RemoveTail\_IntType\_ReturnsCorrectValues

(int head, int middle, int tail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<int>(head);

circularLinkedList.Add(middle);

circularLinkedList.Add(tail);

// Act

circularLinkedList.RemoveTail();

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(tail, circularLinkedList);

Assert.Contains(head, circularLinkedList);

Assert.Contains(middle, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

[Theory]

[MemberData(nameof(StringMultipleElementsArrayTestData))]

public void RemoveTail\_StringType\_ReturnsCorrectValues

(string head, string middle, string tail)

{

// Arrange

var expectedCount = 2;

var circularLinkedList = new CircularLinkedList<string>(head);

circularLinkedList.Add(middle);

circularLinkedList.Add(tail);

// Act

circularLinkedList.RemoveTail();

var actualCount = circularLinkedList.Count;

// Assert

Assert.DoesNotContain(tail, circularLinkedList);

Assert.Contains(head, circularLinkedList);

Assert.Contains(middle, circularLinkedList);

Assert.Equal(expectedCount, actualCount);

}

#endregion RemoveHead

#region GetEnumerator

[Theory]

[MemberData(nameof(IntTestData))]

public void GetEnumerator\_OneElement\_IntType\_ReturnsCorrectValues

(int expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>(expectedData);

// Assert

foreach (int actual in circularLinkedList)

{

Assert.Equal(expectedData, actual);

}

}

[Theory]

[MemberData(nameof(StringTestData))]

public void GetEnumerator\_OneElement\_StringType\_ReturnsCorrectValues

(string expectedData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(expectedData);

// Assert

foreach (string actual in circularLinkedList)

{

Assert.Equal(expectedData, actual);

}

}

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void GetEnumerator\_DifferentElements\_IntType\_ReturnsCorrectValues

(int expectedFirstData, int expectedSecondData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>(expectedFirstData);

circularLinkedList.Add(expectedSecondData);

var sequence = new int[] { expectedFirstData, expectedSecondData };

var counter = 0;

// Assert

foreach (int actual in circularLinkedList)

{

Assert.Equal(sequence[counter], actual);

counter++;

}

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void GetEnumerator\_DifferentElements\_StringType\_ReturnsCorrectValues

(string expectedFirstData, string expectedSecondData)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>(expectedFirstData);

circularLinkedList.Add(expectedSecondData);

var sequence = new string[] { expectedFirstData, expectedSecondData };

var counter = 0;

// Assert

foreach (string actual in circularLinkedList)

{

Assert.Equal(sequence[counter], actual);

counter++;

}

}

#endregion GetEnumerator

#region ToString

[Fact]

public void ToString\_NoElements\_IntType\_ReturnsCorrectValues()

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

var expected = "";

// Act

var actual = circularLinkedList.ToString();

// Assert

Assert.Equal(expected, actual);

}

[Fact]

public void ToString\_NoElements\_StringType\_ReturnsCorrectValues()

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

var expected = "";

// Act

var actual = circularLinkedList.ToString();

// Assert

Assert.Equal(expected, actual);

}

[Theory]

[MemberData(nameof(IntTwoElementsArrayTestData))]

public void ToString\_DifferentElements\_IntType\_ReturnsCorrectValues

(int firstElement, int secondElement)

{

// Arrange

var circularLinkedList = new CircularLinkedList<int>();

circularLinkedList.AddFirst(secondElement);

circularLinkedList.AddFirst(firstElement);

var firstPart = $"{ firstElement + Environment.NewLine + Environment.NewLine }";

var secondPart = $"{ secondElement + Environment.NewLine + Environment.NewLine }";

var expected = firstPart + secondPart;

// Act

var actual = circularLinkedList.ToString();

// Assert

Assert.Equal(expected, actual);

}

[Theory]

[MemberData(nameof(StringTwoElementsArrayTestData))]

public void ToString\_DifferentElements\_StringType\_ReturnsCorrectValues

(string firstElement, string secondElement)

{

// Arrange

var circularLinkedList = new CircularLinkedList<string>();

circularLinkedList.AddFirst(secondElement);

circularLinkedList.AddFirst(firstElement);

var firstPart = $"{ firstElement + Environment.NewLine + Environment.NewLine }";

var secondPart = $"{ secondElement + Environment.NewLine + Environment.NewLine }";

var expected = firstPart + secondPart;

// Act

var actual = circularLinkedList.ToString();

// Assert

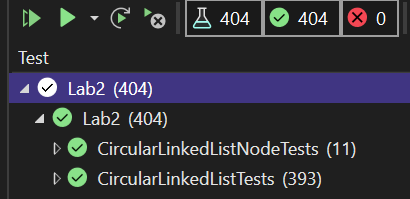
Assert.Equal(expected, actual);

}

#endregion ToString

}

## Результат роботи:





## Контрольні питання:

1. Що таке модульне тестування?
   1. Написання тестів, які тестують невеличку частину коду
2. Як використовуються модульні тести?
   1. Перевіряють функціонал ПЗ
3. Назвіть вимоги до юніт-тестів
   1. Бути достовірними
   2. Не залежати від оточення, на якому вони виконуються
   3. Легко підтримуватись
   4. Легко читатися та бути простими для розуміння (навіть новий розробник повинен зрозуміти що саме тестується)
   5. Дотримуватися єдиної конвенції іменування
   6. Запускатися регулярно в автоматичному режимі
4. Наведіть переваги використання юніт-тестів у розробці ПЗ
   1. При подальшому написанні/зміні коду Ми можемо розуміти, що нічого не зламалося
5. Що таке рефакторінг?
   1. Покращення коду без зміни його функціоналу
6. Які метрики та засоби використовуються для оцінювання ефективності застосування юніт-тестів у проекті?
   1. ступінь покриття модульними тестами вихідного коду
   2. AxoCover
7. Що таке TDD? Назвіть переваги застосування TDD.
   1. Test Driven Development – написання коду, аби той задовольняв тести
   2. Краще розуміння функціоналу, який потрібен
8. Що таке принцип «Triple A»? Поясніть сутність його використання.
   1. Arrange-Act-Assert
   2. Упорядковування коду, для кращого розуміння тестів
9. Як використовуються в юніт-тестах класи Assert?
   1. Для перевірки тверджень
10. Що таке Mock та Stub? З якою ціллю вони використовуються в юніттестах?
    1. Mocks vs Stubs = Поведінкове тестування проти тестування стану
    2. Mocks i stubs чекають відповідь на питання: який результат?
    3. Mocks також цікавить: Як досягнуто результату?

## Висновки:

Я дізнався більше інформації про тестування, його принципи, mock’и та stub’и