

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-застосунків на
платформі .NET"
Тема: "Модульне тестування. Ознайомлення з засобами та
практиками модульного тестування"
Варіант: №8 Кільцевий список

Виконав:

студент групи ІП-93

Домінський Валентин

Олексійович

Перевірила:

Крамар Юлія Михайлівна

Київ 2022

Зміст:

Мета:	3
Вихідний код	3
Результат роботи:	26
Контрольні питання:	26
Висновки:	28

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("العَرَبِيَّة")]
    [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[] { "Паляница" },
        new object[] { "Русский военный корабль" },
        new object[] { "European Union" },
        new object[] { "汉字" },
        new object[] { "العَرَبِيَّة" },
        new object[] { "🇷🇺🇺🇦🇷🇺🇺🇦" },
        new object[] { "🇷🇺🇺🇦🇷🇺🇺🇦" },
    };

    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[] { "Паляница", "Полуница" },
        new object[] { "Русский военный корабль", "Иди" },
        new object[] { "European Union", "NATO" },
        new object[] { "汉字", "漢字" },
        new object[] { "العَرَبِيَّة", "الخُرُوف" },
        new object[] { "🇷🇺🇺🇦🇷🇺🇺🇦", "🇷🇺🇺🇦🇷🇺🇺🇦" },
        new object[] { "🇷🇺🇺🇦🇷🇺🇺🇦", "🇷🇺🇺🇦🇷🇺🇺🇦" },
    };

    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[] { "Паляница", "Полуница", "ОлЕні, Олені" },
    new object[] { "Русский военный корабль", "Иди", "далеко" },
    new object[] { "European Union", "NATO", "IAEA" },
    new object[] { "汉字", "漢字", "ツミツテツ" },
    new object[] { "العَرَبِيَّة", "الخُزُوف", "هَجَائِي" },
    new object[] { "𐄂𐄃𐄄𐄅", "𐄆𐄇𐄈𐄉", "𐄊𐄋𐄌𐄍𐄎𐄏" },
    new object[] { "A𐄐𐄑", "0βf≡%£", "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;

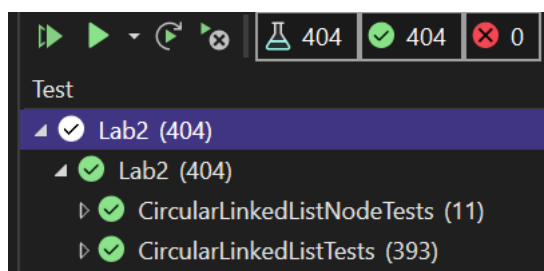
    // Act
    var actual = circularLinkedList.ToString();

    // Assert
    Assert.Equal(expected, actual);
}

#endregion ToString
}

```

Результат роботи:



lab1.dll	15	6,58%	213	93,42%
{ } Lab1.CircularLinkedList	15	6,64%	211	93,36%
{ } Lab1.CircularLinkedListNode	0	0,00%	2	100,00%

Контрольні питання:

1) Що таке модульне тестування?

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

- a. Mocks vs Stubs = Поведінкове тестування проти тестування стану
- b. Mocks і stubs чекають відповідь на питання: який результат?
- c. Mocks також цікавить: Як досягнуто результату?

Висновки:

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