Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №7

з дисципліни

«Основи програмування. Частина 2. Методології програмування»

«ПОБУДОВА ТА ВИКОРИСТАННЯ СТРУКТУР ДАНИХ»

Варіант 15

Виконала ІП-44 Бідюк М. А.

Перевірила Вітковська І. І.

Київ 2025

Лабораторна робота 7
ПОБУДОВА ТА ВИКОРИСТАННЯ СТРУКТУР ДАНИХ

Мета лабораторної роботи – дослідити типи лінійних та нелінійних структур даних, навчитись користуватись бібліотечними реалізаціями структур даних та будувати власні.

Завдання:

№	Тип даних елементів	Тип списку	Спосіб додавання елементу списку	Операції зі списком
15	short	Односпрямований	Включення після першого елементу списку	1. Знайти перше входження елементу більше заданого значення. 2. Знайти суму елементів більших за середнє значення. 3. Отримати новий список із значень елементів менших за середнє значення. 4. Видалити елементи, які розташовані після максимального елементу.

Програмна реалізація:

1) Program.cs

```
class Program
{
   static void Main()
   {
       Console.OutputEncoding =
System.Text.Encoding.UTF8;

       CustomLinkedList list = new CustomLinkedList();
       list.AddAfterFirst(10);
       list.AddAfterFirst(20);
       list.AddAfterFirst(60);
       list.AddAfterFirst(40);
       list.AddAfterFirst(50);
```

```
        Console.WriteLine("Початковий список:");
        PrintList(list);

        short threshold = 25;
        short? firstGreater =
list.FirstGreaterThan(threshold);
        Console.WriteLine($"\n1. Перший елемент більший
за {threshold}: {(firstGreater.HasValue ?
firstGreater.ToString() : "не знайдено")}");

        short sumGreaterThanAvg =
list.SumGreaterThanAverage();
        Console.WriteLine($"2. Сума елементів більших за
середнє: {sumGreaterThanAvg}");

        CustomLinkedList lessThanAvgList =
list.GetLessThanAverage();
        Console.WriteLine("\n3. Новий список з елементів
менших за середнє:");
        PrintList(lessThanAvgList);

        list.RemoveAfterMax();
        Console.WriteLine("\n4. Список після видалення
елементів після максимального:");
        PrintList(list);

        Console.WriteLine("\nДемонстрація
індексатора:");
        for (int i = 0; i < list.Count; i++)
        {
```

```
            Console.WriteLine($"Елемент з індексом {i}:
{list[i]}");
        }

        int indexToRemove = 1;
        list.RemoveAt(indexToRemove);
        Console.WriteLine($"\nСписок після видалення
елементу з індексом {indexToRemove}:");
        PrintList(list);

        Console.WriteLine("\nДемонстрація foreach:");
        foreach (short item in list)
        {
            Console.WriteLine(item);
        }
    }

    private static void PrintList(CustomLinkedList list)
    {
        if (list.Count == 0)
        {
            Console.WriteLine("Список порожній");
            return;
        }

        foreach (short item in list)
        {
            Console.Write($"{item} ");
        }
        Console.WriteLine();
    }
}
```

2) CustomLinkedList.cs

```
public class CustomLinkedList : IEnumerable<short>
{
    private class Node
    {
        public short Value { get; set; }
        public Node Next { get; set; }
        public Node(short value)
        {
            Value = value;
            Next = null;
        }
    }

    private Node _head;
    private int _count;
    public int Count => _count;

    public CustomLinkedList()
    {
        _head = null;
        _count = 0;
    }

    public void AddAfterFirst(short value)
    {
        Node newNode = new Node(value);

        if (_head == null)
        {
```

```
            _head = newNode;
        }
        else
        {
            newNode.Next = _head.Next;
            _head.Next = newNode;
        }
        _count++;
    }

    public short? FirstGreaterThan(short value)
    {
        Node current = _head;
        while (current != null)
        {
            if (current.Value > value)
            {
                return current.Value;
            }
            current = current.Next;
        }
        return null;
    }

    private double CalculateAverage()
    {
        if (_count == 0) return 0;

        double sum = 0;
        Node current = _head;
        while (current != null)
        {
```

```
            sum += current.Value;
            current = current.Next;
        }
        return sum / _count;
    }

    public short SumGreaterThanAverage()
    {
        double average = CalculateAverage();
        short sum = 0;
        Node current = _head;
        while (current != null)
        {
            if (current.Value > average)
            {
                sum += current.Value;
            }
            current = current.Next;
        }
        return sum;
    }

    public CustomLinkedList GetLessThanAverage()
    {
        CustomLinkedList newList = new
CustomLinkedList();
        double average = CalculateAverage();
        Node current = _head;
        while (current != null)
        {
            if (current.Value < average)
            {
```

```
                newList.AddAfterFirst(current.Value);
            }
            current = current.Next;
        }
        return newList;
    }

    public void RemoveAfterMax()
    {
        if (_head == null || _head.Next == null) return;

        Node maxNode = _head;
        Node current = _head.Next;

        while (current != null)
        {
            if (current.Value > maxNode.Value)
            {
                maxNode = current;
            }
            current = current.Next;
        }

        maxNode.Next = null;
        _count = GetCountFromHead();
    }

    private int GetCountFromHead()
    {
        int count = 0;
        Node current = _head;
        while (current != null)
```

```
        {
            count++;
            current = current.Next;
        }
        return count;
    }

    public void RemoveAt(int index)
    {
        if (index < 0 || index >= _count)
        {
            throw new
ArgumentOutOfRangeException(nameof(index), "Index is
out of range");
        }

        if (index == 0)
        {
            _head = _head.Next;
        }
        else
        {
            Node previous = GetNodeAt(index - 1);
            previous.Next = previous.Next?.Next;
        }
        _count--;
    }

    private Node GetNodeAt(int index)
    {
        Node current = _head;
        for (int i = 0; i < index; i++)
```

```
        {
            current = current.Next;
        }
        return current;
    }

    public short this[int index]
    {
        get
        {
            if (index < 0 || index >= _count)
            {
                throw new
ArgumentOutOfRangeException(nameof(index), "Index is
out of range");
            }
            return GetNodeAt(index).Value;
        }
    }

    public IEnumerator<short> GetEnumerator()
    {
        Node current = _head;
        while (current != null)
        {
            yield return current.Value;
            current = current.Next;
        }
    }

    System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator()
```

```
    {
        return GetEnumerator();
    }
}
```

Висновок: У ході виконання лабораторної роботи було розроблено реалізацію однозв’язного списку для роботи з цілими числами типу short. Список підтримує всі необхідні операції, включаючи додавання елементів після першого вузла, пошук першого входження елемента більшого за задане значення, обчислення суми елементів більших за середнє значення, створення нового списку з елементів менших за середнє, видалення елементів після максимального, а також додатковий функціонал: індексацію, видалення за індексом та ітерацію через foreach. Реалізація була успішно протестована на різних наборах даних, що підтвердило коректність роботи всіх методів. Особливу увагу приділено ефективності операцій, які вимагають обходу списку, таких як пошук максимального елемента або обчислення середнього значення. Додатково було перевірено межі допустимих значень, включаючи роботу з порожнім списком та крайніми індексами. Робота демонструє глибоке розуміння принципів роботи однозв’язних списків та вміння реалізовувати їх у мові C# з дотриманням сучасних стандартів кодування. Отриманий результат може слугувати основою для подальшого вдосконалення, наприклад, шляхом додавання подвійного зв’язку або паралельної обробки елементів.