Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра ІПІ

Звіт

до лабораторної роботи № 7 з дисципліни

«Основи програмування. Частина 2. Методології програмування»

“ПОБУДОВА ТА ВИКОРИСТАННЯ СТРУКТУР ДАНИХ”

Виконав(ла) ІП-42 Книр Я. В.

Перевірив(ла) Куценко М. О.

Київ 2025

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

Варіант 9

9	double	Двоспря-мований	Включення до початку	1.Знайти перше входження елементу меншого за середнє значення. 2.Знайти суму елементів, які розташовані після максимального елементу. 3.Отримати новий список зі значень елементів більших за задане значення. 4.Видалити елементи, які розташовані до максимального елементу.

Код програми :

- Бібліотека класів

```
using System;
using System.Collections;
using System.Collections.Generic;


namespace DoubleLinkedListLibrary
{
    /// <summary>
    /// Представляє вузол у двозв'язному списку.
    /// </summary>
    public class Node
    {
        public double Value { get; set; }
        public Node Next { get; set; }
        public Node Previous { get; set; }

        public Node(double value)
        {
            Value = value;
            Next = null;
            Previous = null;
        }
    }

    /// <summary>
    /// Представляє двозв'язний список з елементами типу double.
    /// </summary>
    public class DoubleLinkedList : IEnumerable<double>
    {
        private Node head;

        /// <summary>
        /// Ініціалізує порожній список.
        /// </summary>
        public DoubleLinkedList()
        {
            head = null;
```

```
}

/// <summary>
/// Додає елемент на початок списку.
/// </summary>
/// <param name="value"></param>
public void AddToHead(double value)
{
    Node newNode = new Node(value);
    if (head == null)
    {
        head = newNode;
    }
    else
    {
        newNode.Next = head;
        head.Previous = newNode;
        head = newNode;
    }
}

/// <summary>
/// Видаляє елемент за його індексом.
/// </summary>
/// <param name="index"></param>
/// <exception cref="IndexOutOfRangeException"></exception>
public void Delete(int index)
{
    if (index<0)
    {
        throw new IndexOutOfRangeException("Index can`t be negative");
    }
    if( head == null)
    {
        throw new IndexOutOfRangeException("This list is empty");
    }
    if (index == 0)
    {
        head=head.Next;
        if(head!=null)
        {
            head.Previous = null;
        }
        return;
    }
    Node current= head;
    int currentIndex = 0;
    while (current != null)
    {
        if(currentIndex==index)
        {
            if(current.Previous != null)
            {
                current.Previous.Next = current.Next;
            }
            if(current.Next!=null)
            {
                current.Next.Previous = current.Previous;
            }
            return;
        }
        current=current.Next;
        currentIndex++;
    }
    throw new IndexOutOfRangeException("Index is out of range");
}

/// <summary>
```

```
/// Індексатор (отримує елемент за індексом).
/// </summary>
/// <param name="index"></param>
/// <returns></returns>
/// <exception cref="IndexOutOfRangeException"></exception>
public double this[int index]
{
    get
    {
        if (index<0)
        {
            throw new IndexOutOfRangeException("Index can`t be negative");
        }
        Node current = head;
        int currentIndex = 0;
        while (current != null)
        {
            if(currentIndex==index)
            {
                return current.Value;
            }
            current = current.Next;
            currentIndex++;
        }
        throw new IndexOutOfRangeException("Index is out of range");
    }
}

/// <summary>
/// Дозволяє перебирати елементи у списку.
/// </summary>
/// <returns></returns>
public IEnumerator<double> GetEnumerator()
{
    Node current = head;
    while (current!=null)
    {
        yield return current.Value;
        current= current.Next;
    }
}

/// <summary>
/// Знаходить перше входження елементу, менше за середнє значення.
/// </summary>
/// <returns></returns>
public double? LessThanAverage()
{
    if (head==null)
    {
        return null;
    }
    int count = 0;
    double sum = 0.0;
    Node current = head;
    while (current != null)
    {
        sum += current.Value;
        count++;
        current = current.Next;
    }
    double average = sum / count;
    current = head;
    while (current != null)
    {
        if(current.Value <average)
        {
            return current.Value;
```

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

/// <summary>
/// Обчислює суму елементів, які розташовані після максимального.
/// </summary>
/// <returns></returns>
public double SumAfterMax()
{
    if(head==null)
    {
        return 0;
    }
    double max = double.MinValue;
    Node maxNode = null;
    Node current = head;
    while (current != null)
    {
        if (current.Value > max)
        {
            max= current.Value;
            maxNode = current;
        }
        current = current.Next;
    }
    if (maxNode == null)
    {
        return 0;
    }
    double sum = 0.0;
    current = maxNode.Next;
    while (current != null)
    {
        sum+= current.Value;
        current = current.Next;
    }
    return sum;
}

/// <summary>
/// Повертає новий список з елементів, які більші за задане значення.
/// </summary>
/// <param name="value"></param>
/// <returns></returns>
public DoubleLinkedList GetGreaterThan(double value)
{
    DoubleLinkedList newList = new DoubleLinkedList();
    Node current = head;
    while (current != null)
    {
        if(current.Value > value)
        {
            newList.AddToHead(current.Value);
        }
        current= current.Next;
    }
    return newList;
}

/// <summary>
/// Видаляє всі елементи, розташовані після максимального.
/// </summary>
public void DeleteBeforeMax()
{
    if(head == null)
```

```
            {
                return;
            }
            double max=double.MinValue;
            Node maxNode = null;
            Node current = head;
            while(current != null)
            {
                if(current.Value > max)
                {
                    max=current.Value;
                    maxNode= current;
                }
                current= current.Next;
            }
            if(maxNode!=null && maxNode!=head)
            {
                head = maxNode;
                head.Previous = null;
            }
        }

        /// <summary>
        /// Дозволяє перебирати елементи списку (foreach).
        /// </summary>
        /// <returns></returns>
        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }
}
```

- Використання списку

```
using System;
using DoubleLinkedListLibrary;

class Program
{
    static void Main()
    {
        var list1 = new DoubleLinkedList();
        var list2 = new DoubleLinkedList();

        list1.AddToHead(10.5);
        list1.AddToHead(5.3);
        list1.AddToHead(8.2);
        list1.AddToHead(12.7);
        list1.AddToHead(4.1);
        list1.AddToHead(6.9);
        list1.AddToHead(10.11);

        Console.WriteLine("Elements of list1: ");
        foreach (var value in list1)
        {
            Console.Write(value + "  ");
        }
        Console.WriteLine();

        Console.WriteLine($"\nElement with index 2: {list1[2]}");

        list1.Delete(2);
        Console.WriteLine("\nList1 after deleting third element");
        foreach (var value in list1)
        {
            Console.Write(value + "  ");
```

```
        }
        Console.WriteLine();

        var lessThanAverage = list1.LessThanAverage();
        Console.WriteLine($"\nFirst element that is less than average:
{lessThanAverage}");

        double sumAfterMax = list1.SumAfterMax();
        Console.WriteLine($"\nSum of elements after the biggest one:
{sumAfterMax}");

        double x = 8.0;
        list2 = list1.GetGreaterThan(x);
        Console.WriteLine($"\nElemenets of list1 > {x}:");
        foreach (var value in list2)
        {
            Console.Write(value + "  ");
        }
        Console.WriteLine();

        list1.DeleteBeforeMax();
        Console.WriteLine("\nList1 after deleting all elements before the gratest
one");
        foreach (var value in list1)
        {
            Console.Write(value + "  ");
        }

    }
}
```

Приклад роботи програми:

```
Elements of list1:
10,11  6,9  4,1  12,7  8,2  5,3  10,5

Element with index 2: 4,1

List1 after deleting third element
10,11  6,9  12,7  8,2  5,3  10,5

First element that is less than average: 6,9

Sum of elements after the biggest one: 24

Elemenets of list1 > 8:
10,5  8,2  12,7  10,11

List1 after deleting all elements before the gratest one
12,7  8,2  5,3  10,5
```

Висновок: В процесі виконання лабораторної роботи було досліджено типи лінійних та нелінійних структур даних та набуто навичок користування їх бібліотечними реалізаціями й побудови власних. Було розроблено програму мовою C#, в якій було описано двоспрямований список з елементів типу double із включенням до початку. Двоспрямований список є лінійною структурою даних, де кожен вузол зберігає посилання на наступний елемент та на

попередній. Було розроблено функціонал даної структури, який включає в себе індексацію, можливість отримати значення елемента за номером та видалити елемент за індексом. Також програма передбачає можливість знаходження першого входження елементу, меншого за середнє значення, суми елементів після максимального, отримання нового списку із елементів більших за задане значення та видалення елементів, які розташовані до максимального. Функціональність розробленої структури зберігається в одному проекті, а демонстрація її використання – в іншому.