

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

Кафедра ІІІ

Звіт

з лабораторної роботи № 7 з дисципліни
«Основи програмування. Частина 2. Методології програмування»

“Побудова та використання структур даних”

Виконав

ІП-41 Дикий Дмитро Миколайович

(шифр, прізвище, ім'я, по батькові)

Перевірила

Вітковська Ірина Іванівна

(прізвище, ім'я, по батькові)

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

Написати програму мовою C#, де описати власну структуру даних згідно з варіантом (Табл. 1), створити 2 проекти, в одному – має бути функціонал списку, в другому - його використання. Виведення на консоль даних зі списку вважати використанням списку, а не його функціоналом! В списку потрібно передбачити крім функціональності, заданої варіантом, можливість отримати з нього значення, наприклад, стандартним оператором `foreach`. Додати до списку функцію індексації (елементи списку мають бути доступні на читання за індексом). Додати до списку операцію видалення елемента за номером (номер елемента задається параметром). Єдиний стиль найменувань обов'язковий. Застосовувати в коді лише XML-коментарі.

Продемонструвати функціональність розробленої структури шляхом застосування всіх її операцій (створити декілька об'єктів структур, додати в них певну кількість значень, зчитати значення зі списку, викликати операції над значеннями відповідно варіанту).

В завданні, де вказується на літерал цілого типу, дійсного, символьного та ін. (наприклад, «!»: знайти перше входження символу «!»), значення літералу не “зашивати” у код операції, а передавати як параметр (тобто передбачити можливість виконання операції з різними значеннями):

Варіант завдання

№	Тип даних елементів	Тип списку	Спосіб додавання елементів до списку	Операції зі списком
6	short	Односпрямований	Включення в кінець списку	1. Знайти елемент, кратний заданому значенню. 2. Замінити елементи, що розташовані на парних позиціях, на «0» (нумерація починається з голови списку). 3. Отримати новий список зі значень елементів, більших за задане значення. 4. Видалити елементи, що розташовані на непарних позиціях (нумерація починається з голови списку).

Код програми

Class1.cs

```

using System.Collections;

namespace MyLinkedList
{
    /// <summary>
    /// Class which contains the data of type short and reference to next node.
    /// </summary>
    public class Node
    {
        public short Data;
        public Node? Next;
        public Node(short data)
        {
            Data = data;
            Next = null;
        }
    }
}

```

```

}
/// <summary>
/// Class which contains the attributes and methods for LinkedList.
/// </summary>
public class TheLinkedList : IEnumerable<short>
{
    private int _numOfElements;
    private Node? _head;
    private Node? _tail;
    /// <summary>
    /// Constructor for creating a new empty list.
    /// </summary>
    public TheLinkedList()
    {
        _numOfElements = 0;
        _head = null;
        _tail = null;
    }
    /// <summary>
    /// Adds one number at the end of list.
    /// </summary>
    /// <param name="value">The number which will be added to list.</param>
    public void Add(short value)
    {
        Node node = new Node(value);
        if (this.IsEmpty())
            _head = _tail = node;
        else
        {
            _tail.Next = node;
            _tail = node;
        }
        _numOfElements++;
    }
    /// <summary>
    /// Adds few numbers at the end of list.
    /// </summary>
    /// <param name="value">Array of numbers which will be added to list.</param>
    public void AddSomeElements(params short[] value)
    {
        foreach (short element in value)
            Add(element);
    }
    /// <summary>
    /// The number of elements in list.
    /// </summary>
    public int Size => _numOfElements;
    /// <summary>
    /// Checks whether list is empty or not.

```

```

/// </summary>
/// <returns>True if list is empty, otherwise false.</returns>
public bool IsEmpty() => _head == null;
/// <summary>
/// Searches for node at specified index in list.
/// </summary>
/// <param name="index">Index of node which is wanted to be found.</param>
/// <returns>Node at the specified index.</returns>
/// <exception cref="IndexOutOfRangeException">Index is less than zero or greater than max
possible one.</exception>
/// <exception cref="NullReferenceException">List is empty.</exception>
private Node GetNodeAtIndex(int index)
{
    if (index < 0 || index > _numOfElements - 1)
        throw new IndexOutOfRangeException("Oops, index is out of range");
    if (this.IsEmpty())
        throw new NullReferenceException("Sorry, but list doesn't contain any element");
    Node current = _head;
    for (int i = 0; i < index; i++)
        current = current.Next;
    return current;
}
/// <summary>
/// Indexator to reach element in list at specified index.
/// </summary>
/// <param name="index">Index of element in list.</param>
/// <returns>The element at specifield index.</returns>
public short this[int index]
{
    get => GetNodeAtIndex(index).Data;
    set => GetNodeAtIndex(index).Data = value;
}
/// <summary>
/// Sets the size of list to 0. Sets both "head" and "tail" fields to null.
/// </summary>
public void Clear()
{
    _numOfElements = 0;
    _head = _tail = null;
}
/// <summary>
/// Removes element from list at specified index.
/// </summary>
/// <param name="index">Index of element to be removed.</param>
/// <exception cref="IndexOutOfRangeException">Index is less than zero or greater than max
possible one.</exception>
/// <exception cref="NullReferenceException">List is empty, so there are no elements to
delete.</exception>
public void DeleteTheElementAtIndex(int index)

```

```

{
    if (this.IsEmpty())
        throw new NullReferenceException("Sorry, but list doesn't contain any element to delete");
    if (index < 0 || index > _numOfElements - 1)
        throw new IndexOutOfRangeException("Oops, index is out of range");
    if (index == 0)
    {
        _head = _head.Next;
        if (_head == null)
            _tail = null;
    }
    else
    {
        Node temp = GetNodeAtIndex(index - 1);
        temp.Next = temp.Next.Next;
        if (index == _numOfElements - 1)
            _tail = temp;
    }
    _numOfElements--;
}

/// <summary>
/// Finds the number which is multiple to element.
/// </summary>
/// <param name="element">The divisor.</param>
/// <param name="multiple">The multiple to element.</param>
/// <returns>True if multiple is found, otherwise false.</returns>
public bool FindTheMultipleNumber(short element, ref short multiple)
{
    if (element == 0)
        return false;
    Node temp = _head;
    bool isFound = false;
    while (temp != null && !isFound)
    {
        if (temp.Data % element == 0)
        {
            multiple = temp.Data;
            isFound = true;
        }
        temp = temp.Next;
    }
    return isFound;
}

/// <summary>
/// Changes elements with even indexes to 0.
/// </summary>
public void ChangeElementsWithEvenIndex()
{

```

```

        if (this.IsEmpty())
            return;
        Node temp = _head;
        int index = 0;
        while (temp != null)
        {
            if (index % 2 == 0)
                temp.Data = 0;
            temp = temp.Next;
            index++;
        }
    }
    /// <summary>
    /// Creates a new list of elements which greater than specified value.
    /// </summary>
    /// <param name="number">Specified value which is less than every element in new
list.</param>
    /// <returns>New list, which also may be empty if no elements greater than specified
value.</returns>
    public TheLinkedList CreateListWithGreaterNumbers(short number)
    {
        TheLinkedList newList = new TheLinkedList();
        Node temp = _head;
        while (temp != null)
        {
            if (temp.Data > number)
                newList.Add(temp.Data);
            temp = temp.Next;
        }
        return newList;
    }
    /// <summary>
    /// Removes elements with odd indexes from the list.
    /// </summary>
    public void DeleteElementsWithOddIndex()
    {
        if (_head == null || _head.Next == null)
            return;
        Node temp = _head;
        while (temp != null && temp.Next != null)
        {
            temp.Next = temp.Next.Next;
            _numOfElements--;
            temp = temp.Next;
        }
        _tail = temp;
    }
    /// <summary>

```

/// Returns an iterator that iterates through all the elements of the list in the order they were added.

/// </summary>

/// <returns>Iterator for traversing list elements of type <see cref="short">.</returns>

public IEnumerator<short> GetEnumerator()

{

 Node? current = _head;

 while (current != null)

 {

 yield return current.Data;

 current = current.Next;

 }

}

/// </summary>

/// Returns a non-generic iterator for iterating over the elements of a list.

/// </summary>

/// <returns>Iterator <see cref="System.Collections.IEnumerator"/>.</returns>

IEnumerator IEnumerable.GetEnumerator()

{

 return GetEnumerator();

}

}

}

Program.cs

using MyLinkedList;

namespace LaboratoryWork7

{

 internal class Program

 {

 /// <summary>

 /// Creates an array with randomly generated short values.

 /// </summary>

 /// <param name="n">Number of elements.</param>

 /// <returns>Array containing randomly generated values of type short.</returns>

 static short[] GenerateRandomValues(int n)

 {

 short[] values = new short[n];

 Random rand = new Random();

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

 values[i] = (short)rand.Next(-300, 300);

 return values;

 }

 /// <summary>

 /// Shows all elements in list.

 /// </summary>

 /// <param name="list">The list whose elements should be shown.</param>


```

static void ShowList(TheLinkedList list)
{
    int counter = 0;
    foreach (var item in list)
    {
        if (counter % 15 == 0 && counter > 0)
            Console.WriteLine();
        Console.Write("{0, -6}", item);
        counter++;
    }
    Console.WriteLine();
}
/// <summary>
/// Used for assigning correct value to a variable of type short.
/// </summary>
/// <returns>Value of type short.</returns>
static short ValidateInput(string message = "", short min = short.MinValue, short max =
    short.MaxValue)
{
    bool isCorrect = true;
    short value = 0;
    do
    {
        Console.Write(message);
        try
        {
            isCorrect = true;
            value = short.Parse(Console.ReadLine());
        }
        catch (FormatException ex)
        {
            Console.Write("Enter ONLY integer, not symbols.\nRepeat the input: ");
            isCorrect = false;
            continue;
        }
        catch (OverflowException ex)
        {
            Console.Write($"You number should be from {short.MinValue} to
{short.MaxValue}.\nRepeat the input: ");
            isCorrect = false;
            continue;
        }
        if (value < min || value > max)
        {
            Console.Write("Your input value is not allowed.\nRepeat the input: ");
            isCorrect = false;
        }
    } while (!isCorrect);
    return value;
}

```

```

}
static void Main(string[] args)
{
    TheLinkedList myList = new TheLinkedList();
    Console.WriteLine($"Enter the number which you want to add (from {short.MinValue} to {short.MaxValue}): ");
    short number = ValidateInput();
    myList.Add(number);
    int size1 = myList.Size;
    Console.WriteLine("The size of list: {0}", size1);
    Console.WriteLine("The element" + (size1 > 1 ? "s" : "") + " in list: ");
    ShowList(myList);
    Console.ReadKey();
    Console.WriteLine("Enter the number of elements you want to add to the list [1-100]: ");
    number = ValidateInput("", 1, 100);
    short[] numbers = new short[number];
    Console.WriteLine("Do you want to input values or use randomly generated ones?");
    Console.WriteLine("1. Input.");
    Console.WriteLine("2. Fill randomly.");
    short choice = ValidateInput("", 1, 2);
    switch (choice)
    {
        case 1:
            Console.WriteLine($"Now enter your values (from {short.MinValue} to {short.MaxValue}).");
            for (int i = 0; i < number; i++)
                numbers[i] = ValidateInput($" {i + 1} -> ");
            break;
        case 2:
            numbers = GenerateRandomValues(number);
            break;
        default:
            Console.WriteLine("Something strange happened...");
            break;
    }
    myList.AddSomeElements(numbers);
    size1 = myList.Size;
    Console.WriteLine($"List after adding {number} elements:");
    ShowList(myList);
    Console.WriteLine("The size of list: {0}", size1);
    Console.WriteLine("Now list is " + (myList.IsEmpty() ? "" : "not ") + "empty.");
    Console.WriteLine($"Enter the number, which will be less than elements of new list (from {short.MinValue} to {short.MaxValue}): ");
    short compNumber = ValidateInput();
    TheLinkedList newList = myList.CreateListWithGreaterNumbers(compNumber);
    int size2 = newList.Size;
    if (newList.IsEmpty())
        Console.WriteLine("New list is empty.");
    else

```

```

{
    Console.WriteLine($"New list contains {size2} element" + (size2 > 1 ? "s." : "."));
    Console.WriteLine("Elements of new list:");
    ShowList(newList);
    Console.ReadKey();
    Console.WriteLine("Now we will find the multiple to specific number in new list.");
    Console.Write("Enter the divider: ");
    short divider = ValidateInput();
    short multiple = 0;
    bool isMultiple = newList.FindTheMultipleNumber(divider, ref multiple);
    if (!isMultiple)
        Console.WriteLine("There is no number which is multiple to {0}", divider);
    else
        Console.WriteLine($"{multiple} is multiple to {divider}");
    Console.ReadKey();
    Console.WriteLine("We will change last element in new list to our divider.");
    Console.WriteLine($"Element at index {size2 - 1} before changing: {newList[size2 - 1]}");
    newList[size2 - 1] = divider;
    Console.ReadKey();
    Console.WriteLine($"Element at index {size2 - 1} after changing: {newList[size2 - 1]}");
    Console.Write("Now input the index at which element should be removed: ");
    int index = ValidateInput("", 0, (short)(size2 - 1));
    try
    {
        newList.DeleteTheElementAtIndex(index);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    if (newList.IsEmpty())
        Console.WriteLine("List is empty now.");
    else
    {
        size2 = newList.Size;
        Console.WriteLine($"Now new list contains {size2} element" + (size2 > 1 ? "s." : "."));
        Console.WriteLine("All elements in new list:");
        ShowList(newList);
    }
}

Console.WriteLine("Now first list will be changed");
Console.ReadKey();
Console.WriteLine("First list before changing:");
ShowList(myList);
Console.ReadKey();
myList.ChangeElementsWithEvenIndex();
Console.WriteLine("First list after changing:");
ShowList(myList);

```

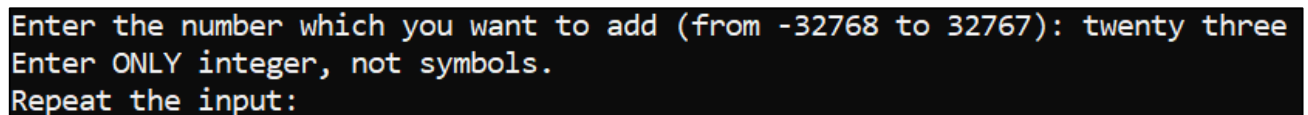
```

    Console.ReadKey();
    Console.WriteLine("We will also delete all elements at odd indexes from this list.");
    Console.WriteLine("The size before deleting: {0}", size1);
    Console.ReadKey();
    myList.DeleteElementsWithOddIndex();
    size1 = myList.Size;
    Console.WriteLine("The size after deleting: {0}", size1);
    Console.ReadKey();
    Console.WriteLine("The list looks like:");
    ShowList(myList);
    Console.WriteLine("And finally we will clear our lists.");
    Console.WriteLine("Size of first list before clearing: {0}", size1);
    myList.Clear();
    size1 = myList.Size;
    Console.WriteLine("Size of first list after clearing: {0}", size1);
    Console.ReadKey();
    Console.WriteLine("Size of second list before clearing: {0}", size2);
    newList.Clear();
    size2 = myList.Size;
    Console.WriteLine("Size of second list after clearing: {0}", size2);
    Console.ReadKey();
}
}
}

```

Приклади роботи програми

1) Обробка користувацького вводу (рис.1-2).

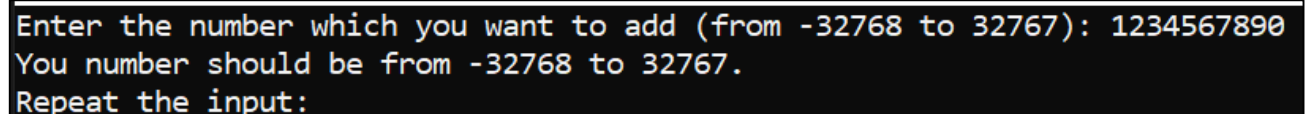


```

Enter the number which you want to add (from -32768 to 32767): twenty three
Enter ONLY integer, not symbols.
Repeat the input:

```

Рис. 1. Валідація введених даних (на вхід подано символи)



```

Enter the number which you want to add (from -32768 to 32767): 1234567890
You number should be from -32768 to 32767.
Repeat the input:

```

Рис. 2. Валідація введених даних (на вхід подано число, більше за максимально допустиме)

2) Демонстрація функціоналу програми (рис. 3-12).

```
Enter the number which you want to add (from -32768 to 32767): 1234
The size of list: 1
The element in list: 1234
```

Рис. 3. Додавання у список одного елемента

```
Enter the number of elements you want to add to the list [1-100]: 10
Do you want to input values or use randomly generated ones?
1. Input.
2. Fill randomly.
1
Now enter your values (from -32768 to 32767).
1-> 1
2-> 2
3-> 3
4-> 4
5-> 5
6-> 6
7-> 7
8-> 8
9-> 9
10-> 10
List after adding 10 elements:
1234 1 2 3 4 5 6 7 8 9 10
The size of list: 11
Now list is not empty.
```

Рис. 4. Додавання у список 10 елементів (ручне введення)

```
Enter the number of elements you want to add to the list [1-100]: 30
Do you want to input values or use randomly generated ones?
1. Input.
2. Fill randomly.
2
List after adding 30 elements:
1234 89 -24 35 -295 195 -296 160 109 -283 285 53 -68 -66 219
-93 259 64 -198 -180 87 -171 -140 267 -266 20 135 163 192 -120
201
The size of list: 31
Now list is not empty.
```

Рис. 5. Додавання у список 30 елементів (рандомізовано)

```

List after adding 30 elements:
1234 89 -24 35 -295 195 -296 160 109 -283 285 53 -68 -66 219
-93 259 64 -198 -180 87 -171 -140 267 -266 20 135 163 192 -120
201
The size of list: 31
Now list is not empty.
Enter the number, which will be less than elements of new list (from -32768 to 32767): 100
New list contains 12 elements.
Elements of new list:
1234 195 160 109 285 219 259 267 135 163 192 201

```

Рис. 6. Створення нового списку, елементи якого більші за задане значення
(задане значення – 100)

```

Elements of new list:
1234 195 160 109 285 219 259 267 135 163 192 201
Now we will find the multiple to specific number in new list.
Enter the divider: 123
There is no number which is multiple to 123

```

Рис. 7. Пошук першого числа, кратного до заданого значення (задане значення –
123, кратних йому не виявлено)

```

We will change last element in new list to our divider.
Element at index 11 before changing: 201
Element at index 11 after changing: 123

```

Рис. 8. Зміна значення елемента за індексом (індекс – 11, значення елемента до
зміни – 201, значення елемента після зміни - 123)

```

Now input the index at which element should be removed: 2
Now new list contains 11 elements.
All elements in new list:
1234 195 109 285 219 259 267 135 163 192 123

```

Рис. 9. Видалення елемента списку за індексом (індекс - 2)

```

Now first list will be changed
First list before changing:
1234 89 -24 35 -295 195 -296 160 109 -283 285 53 -68 -66 219
-93 259 64 -198 -180 87 -171 -140 267 -266 20 135 163 192 -120
201
First list after changing:
0 89 0 35 0 195 0 160 0 -283 0 53 0 -66 0
-93 0 64 0 -180 0 -171 0 267 0 20 0 163 0 -120
0

```

Рис. 10. Зміна значень елементів на парних позиціях на «0» (змінено початковий список)

```

We will also delete all elements at odd indexes from this list.
The size before deleting: 31
The size after deleting: 16
The list looks like:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

```

Рис. 11. Видалення елементів на непарних позиціях зі списку

```

And finally we will clear our lists.
Size of first list before clearing: 16
Size of first list after clearing: 0
Size of second list before clearing: 8
Size of second list after clearing: 0

```

Рис. 12. Очищення обох списків

Висновок: під час виконання цієї лабораторної роботи було досліджено типи лінійних та нелінійних структур даних, здобуто навички користування бібліотечними реалізаціями структур даних та побудови власних. Виконано завдання, що полягало у реалізації власної структури даних згідно варіанту та демонстрації її функціоналу. Розроблено програму, що коректно обробляє користувацький ввід та не виходить з ладу внаслідок введення некоректних даних.