**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 1

**З дисципліни:** *“Декларативне програмування”*

**На тему:** *“Рекурсія та декомпозиція (композиція) – базові принципи декларативного програмування”*

**Лектор:**

доцент каф. ПЗ

Левус Є.В.

**Виконав:**

ст. гр. ПЗ-45

Хруставчук М.Л.

**Прийняв:**

доцент каф. ПЗ

ТушницькийР.Б.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

∑= \_\_\_\_\_ .

Львів – 2025

**Тема роботи:** Рекурсія та декомпозиція (композиція) – базові принципи декларативного програмування.

**Мета роботи:** Навчитися використовувати декомпозицію/композицію та рекурсію у декларативному стилі програмування.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Принципи функціональної декомпозиції системи. Принципи об’єктної декомпозиції системи.

*Функціональна декомпозиція* (top-down): система розкладається на функції/процедури за потоками даних і трансформаціями: “що робимо з даними на кожному кроці”. Результат – пайплайни обробки, де легко застосувати композицію й чисті функції. Підходить для ETL-задач, математичних обчислень, трансформацій колекцій. Ризики: без дисципліни з’являються “довгі труби” і важко відстежувати побічні ефекти.

*Об’єктна декомпозиція* моделює сутності предметної області: “хто” взаємодіє і “які” мають інваріанти. Виходять класи з чіткими відповідальностями, стани інкапсульовано, поведінка поліморфна. Підходить для “багатих” доменів (DDD), складних правил і життєвих циклів. Ризики: надмірні ієрархії та слабка видимість потоків даних.

Практика: комбінувати. Домен моделювати об’єктно (агрегати, інваріанти), а обчислювальні конвеєри – функціонально (чисті трансформації). Межі IO – тонкі адаптери, що перетворюють ефекти у дані та навпаки. Це дає зрозумілу структуру, тестованість і масштабованість.

**КОД ПРОГРАМИ**

Файл MyList.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_01;

public class MyList

{

public object Head { get; set; }

public MyList Tail { get; private set; }

public MyList(object head, MyList tail)

{

Head = head;

Tail = tail;

}

public bool IsEmpty => Head == null && Tail == null;

public static MyList Empty => new MyList(null!, null!);

public override string ToString()

{

if (this.IsEmpty)

{

return "[]";

}

return "[" + ToStringElements(this) + "]";

}

private static string ToStringElements(MyList list)

{

if (list.IsEmpty)

{

return "";

}

if (list.Tail.IsEmpty)

{

return list.Head.ToString()!;

}

return list.Head.ToString() + ", " + ToStringElements(list.Tail);

}

public static MyList CreateList(string input)

{

var tokens = input.Split(' ', StringSplitOptions.RemoveEmptyEntries);

object[] parsed = tokens.Select(t =>

{

if (int.TryParse(t, out int num))

{

return (object)num;

}

if (t.Length == 1)

{

return (object)t[0];

}

return (object)t;

}).ToArray();

return FromArray(parsed);

}

private static MyList FromArray(object[] items) => FromArray(items, 0);

private static MyList FromArray(object[] items, int index)

{

if (items is null || index >= items.Length)

{

return Empty;

}

return new MyList(items[index], FromArray(items, index + 1));

}

}

Файл ListBaseFunctions.cs:

namespace Lab\_01;

static public partial class Program

{

static object SelectHead(MyList list)

{

if (list.IsEmpty)

{

throw new InvalidOperationException("Head is undefined for empty list.");

}

return list.Head;

}

static MyList SelectTail(MyList list)

{

if (list == null || list.IsEmpty)

{

throw new InvalidOperationException("Tail is undefined for empty list.");

}

return list.Tail;

}

static MyList Cons(object Head, MyList Tail)

{

if (Head is null)

{

throw new ArgumentNullException(nameof(Head));

}

return new MyList(Head, Tail ?? MyList.Empty);

}

}

Файл IndividualFunctions.cs

namespace Lab\_01;

static public partial class Program

{

static MyList ListModification(MyList list)

{

int zeros = CountZeros(list);

var transformed = MapTransform(list);

return AppendLast(transformed, zeros);

}

static int CountZeros(MyList list)

{

if (list == null || list.IsEmpty)

{

return 0;

}

var head = SelectHead(list);

var tail = SelectTail(list);

if (head is int && (int)head == 0)

{

return 1 + CountZeros(tail);

}

else

{

return CountZeros(tail);

}

}

static MyList MapTransform(MyList list)

{

if (list == null || list.IsEmpty)

{

return MyList.Empty;

}

var head = SelectHead(list);

var tail = SelectTail(list);

object newHead;

if (head is int value)

{

if (value < 0)

{

newHead = value + 1;

}

else if (value > 0)

{

newHead = "plus";

}

else

newHead = 0;

}

else

newHead = head;

return Cons(newHead, MapTransform(tail));

}

static MyList AppendLast(MyList list, object value)

{

if (list == null || list.IsEmpty)

return Cons(value, MyList.Empty);

var head = SelectHead(list);

return new MyList(head, AppendLast(SelectTail(list), value));

}

}

Файл Program.cs

namespace Lab\_01;

static public partial class Program

{

static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.WriteLine("Enter elements of list: ");

string input = Console.ReadLine();

var list = MyList.CreateList(input);

Console.WriteLine("Head of list: " + SelectHead(list));

Console.WriteLine("Tail of list: " + SelectTail(list));

Console.WriteLine("Added head 'a' to current list: " + Cons('a', list));

var modifiedList = ListModification(list);

Console.WriteLine("Modified list: " + modifiedList);

}

}

**Результати роботи програми**

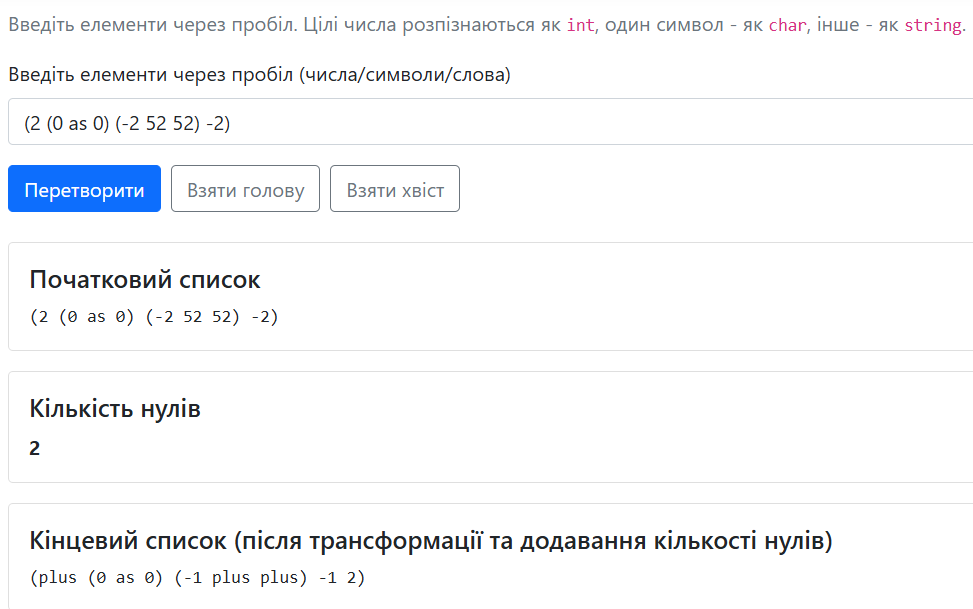


Рис. 1. Результат модифікації списку відповідно до індивідуального завдання

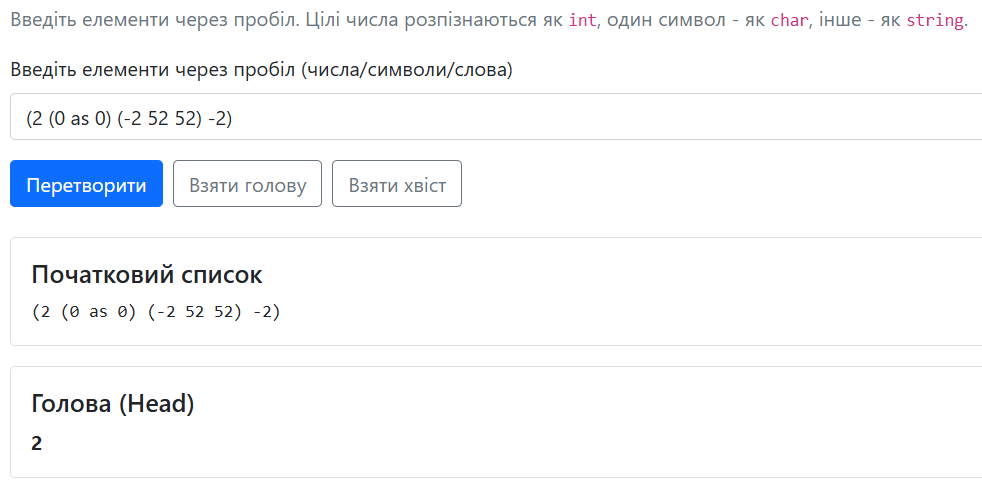


Рис. 2. Результат роботи функції CAR

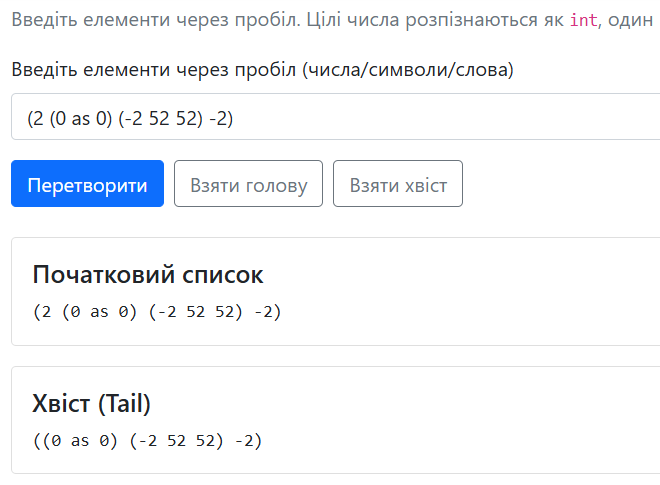


Рис. 3. Результат роботи функції CDR

**ВИСНОВКИ**

У результаті виконання лабораторної роботи завдання було виконано повністю: реалізовано базові операції над однозв’язними списками (селектори Head, Tail, конструктор Cons) та функцію опрацювання списку відповідно до індивідуального варіанту із застосуванням рекурсії.

У процесі роботи було вивчено основи декларативного стилю програмування, принципи рекурсивних обчислень, особливості побудови та маніпулювання однозв’язними структурами даних. Також отримано практичні навички декомпозиції задач на прості функції та їхньої композиції в загальну програму.

Найважче було реалізувати рекурсивну функцію обробки списку, оскільки потрібно було коректно визначити базовий випадок та забезпечити правильне поєднання результатів рекурсивних викликів без використання циклів та допоміжних колекцій.