**КПІ ім. Ігоря Сікорського**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт до комп’ютерного практикуму з курсу**

**“Основи програмування. Частина 2”**

Прийняв Виконав

Доцент кафедри ІПІ Студент групи ІП-41

Крамар Ю.М. Чередніченко А. О.

“23” квітня 2025 р.

**Київ 2024**

**Комп’ютерний практикум №7**

***Тема:*** *Побудова та використання структур даних*

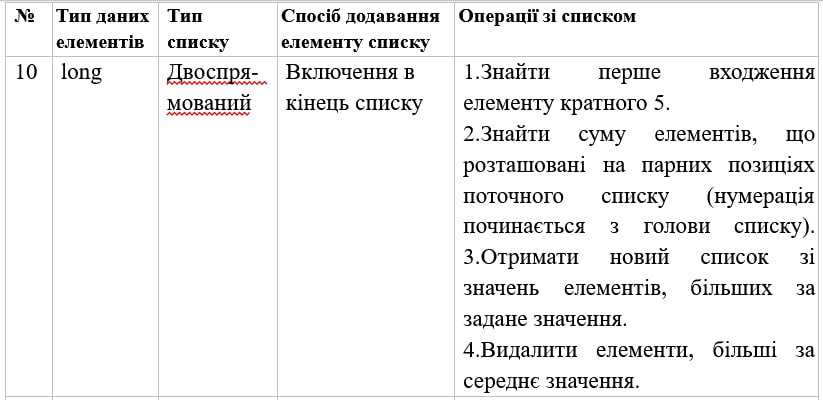
**Завдання:**

Написати програму мовою С#, 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). 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 10:



**Текст програмної реалізації класу Node(вузол)**

public class Node

{

public Node(long value)

{

Value = value;

Next = null;

Previous = null;

}

public long Value { get; set; }

public Node Next { get; set; }

public Node Previous { get; set; }

}

**Текст програмної реалізації класу myLinkedList(двоспрямований список)**

public class MyLinkedList : IEnumerable<long>

{

private Node head;

private Node tail;

private uint count;

public void Add(long value)

{

Node newNode = new Node(value);

if (head == null)

{

head = newNode;

tail = newNode;

}

else

{

tail.Next = newNode;

newNode.Previous = tail;

tail = newNode;

}

count++;

}

public uint Size => count;

public bool isEmpty => count == 0;

public void Clear()

{

if (isEmpty)

throw new InvalidOperationException("Список уже очищений");

head = null;

tail = null;

count = 0;

}

public bool Contains(long value)

{

Node current = head;

while (current != null)

{

if (current.Value == value)

{

return true;

}

current = current.Next;

}

return false;

}

public long? firstElementDivisible(long number)

{

Node current = head;

while (current != null)

{

if (current.Value % number == 0)

{

return current.Value;

}

current = current.Next;

}

return null;

}

public long sumOnEvenPositions()

{

long sum = 0;

int i = 0;

Node current = head;

while (current != null)

{

if (i % 2 == 0)

{

sum += current.Value;

}

current = current.Next;

i++;

}

return sum;

}

public MyLinkedList filterGreaterThan(long value)

{

MyLinkedList result = new MyLinkedList();

Node current = head;

while (current != null)

{

if (current.Value > value)

{

result.Add(current.Value);

}

current = current.Next;

}

return result;

}

public void RemoveNode(Node node)

{

if (node.Previous != null)

node.Previous.Next = node.Next;

else

head = node.Next;

if (node.Next != null)

node.Next.Previous = node.Previous;

else

tail = node.Previous;

count--;

}

public void removeGreaterThanAverage()

{

if (count == 0)

return;

long totalNumbers = 0;

Node current = head;

while (current != null)

{

totalNumbers += current.Value;

current = current.Next;

}

double averageNumber = (double)totalNumbers / count;

current = head;

while (current != null)

{

Node next = current.Next;

if (current.Value > averageNumber)

{

RemoveNode(current);

}

current = next;

}

}

public long this[int index]

{

get

{

if (isEmpty)

{

throw new InvalidOperationException("Список порожній.");

}

if (index < 0 || index >= count)

{

throw new IndexOutOfRangeException("Індекс поза межами діапазону.");

}

Node current = head;

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

{

current = current.Next;

}

return current.Value;

}

set

{

if (isEmpty)

{

throw new InvalidOperationException("Список порожній.");

}

if (index < 0 || index >= count)

{

throw new IndexOutOfRangeException("Індекс поза межами діапазону.");

}

Node current = head;

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

{

current = current.Next;

}

current.Value = value;

}

}

public void removeAtIndex(int index)

{

if (index < 0 || index >= count)

throw new IndexOutOfRangeException();

Node current = head;

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

{

current = current.Next;

}

RemoveNode(current);

}

public IEnumerator<long> GetEnumerator()

{

Node current = head;

while (current != null)

{

yield return current.Value;

current = current.Next;

}

}

IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

}

**Текст програми, що використана для демонстрації можливостей описаних класів**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static void Main(string[] args)

{

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

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

MyLinkedList list = new MyLinkedList();

Console.WriteLine("Порожній список створено.");

Console.WriteLine($"Розмір: {list.Size}");

Console.WriteLine(list.isEmpty ? "Список порожній" : "Список не порожній");

for (long i = 1; i <= 10; i++)

{

list.Add(i);

}

list.Add(25);

list.Add(40);

Console.WriteLine("\nЕлементи додано.");

Console.WriteLine($"Розмір: {list.Size}");

Console.WriteLine("\nСписок:");

foreach (var item in list)

{

Console.Write($"{item} ");

}

long divisor = 5;

var firstDivisible = list.firstElementDivisible(divisor);

Console.WriteLine($"\n\nПерше кратне {divisor}: {(firstDivisible.HasValue ? firstDivisible.ToString() : "не знайдено")}");

long sumEven = list.sumOnEvenPositions();

Console.WriteLine($"Сума елементів на парних позиціях: {sumEven}");

long threshold = 8;

var filtered = list.filterGreaterThan(threshold);

Console.WriteLine($"\nЕлементи більші за {threshold}:");

foreach (var item in filtered)

{

Console.Write($"{item} ");

}

list.removeGreaterThanAverage();

Console.WriteLine("\n\nПісля видалення елементів, більших за середнє:");

foreach (var item in list)

{

Console.Write($"{item} ");

}

Console.WriteLine(list.isEmpty ? "Список порожній" : "Список не порожній");

Console.WriteLine($"\n\nВидалення елементу за індексом 0:");

list.removeAtIndex(0);

foreach (var item in list)

{

Console.Write($"{item} ");

}

list.Clear();

Console.WriteLine("\n\nСписок очищено.");

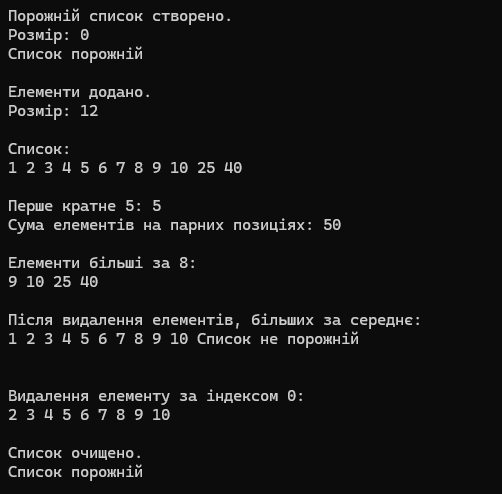
Console.WriteLine(list.isEmpty ? "Список порожній" : "Список не порожній");

}

}

}

**Введені та одерджані результати:**

****