Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

**про виконання лабораторних та практичних робіт блоку № 6**

На тему:  «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

***з дисципліни:*** «Мови та парадигми програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт № 6

студент групи ШІ-14

Шевченко Данііл Вікторович

# **Тема роботи:**

Первинне ознайомлення з динамічними структурами даних, такими як стек, черга, списки, бінарні дерева. Спроби реалізації різних видів списків(однозв’язний лінійний та двозв’язний лінійний), ознайомлення з одно- та двозв’язними циклічними списками. Ознайомлення з елементами об’єктно-орієнтованого програмування(класи, об’єкти, методи, конструктори, деструктори і т.д)

# **Мета роботи:**

Ознайомитись з різними видами динамічних структур, зокрема стеком, чергою, різними видами списків(linked lists), бінарними деревами. Спробувати реалізувати однозв’язний та двозв’язний лінійний список, елементи бінарного дерева. Отримати початкові уявлення про елементи ООП: класи, об’єкти, методи. Ознайомитись з конструкторами та деструкторами, отримати поняття про специфікатори доступу, зрозуміти різницю між класами та структурами.

Завдання:

* Daniil Shevchenko - Epic 6 Task 1 - Theory Education Activities
* Daniil Shevchenko - Epic 6 Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)
* Daniil Shevchenko - Epic 6 Task 3 - Lab# programming: VNS Lab 10
* Daniil Shevchenko - Epic 6 Task 4 - Lab# programming: Algotester Lab 5
* Daniil Shevchenko - Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8
* Daniil Shevchenko - Epic 6 Task 6 - Practice# programming: Class Practice Task
* Daniil Shevchenko - Epic 6 Task 7  - Practice# programming:  Self Practice Task
* Daniil Shevchenko - Epic 6 Task 8  - Result Documentation Report and Outcomes Placement Activities (Docs and Programs on GitHub)
* Daniil Shevchenko - Epic 6 Task 9 - Results Evaluation and Release

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічні структури даних(черга, стек, хіп)
* Тема №2: Linked Lists(1/2-зв’язні, лінійні/циркулярні)
* Тема №3: Ознайомлення з елементами ООП
* Тема №4: Специфікатори доступа public і private
* Тема №5: Методи класів, геттери та сеттери
* Тема №6: Конструктори та деструктори
* Тема №7: Введення в перевантаження операторів

1. Індивідуальний план опрацювання теорії:

* Тема №1: Динамічні структури даних(черга, стек, хіп)
  + Джерела Інформації
    - Практика в НУ “ЛП”
    - Курси https://acode.com.ua/
  + Що опрацьовано:
    - Ознайомлення зі структурою стеку
    - Ознайомлення з чергою
    - Ознайомлення з хіпом(купою)
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.23
  + Звершення опрацювання теми: 11.12.23
* Тема №2: Зв’язні списки(Linked Lists)
  + Джерела Інформації
    - Практика в НУ “ЛП”
    - Курси <https://acode.com.ua/>
    - Відео https://www.youtube.com/watch?v=3mRfWluedHo
  + Що опрацьовано:
    - Ознайомлено та реалізовано однозв’язний лінійний список
    - Ознайомлено та реалізовано двозв'язний лінійний список
    - Ознайомлено з однозв’язним циклічним списком
    - Ознайомлено з двозв’язним циклічним списком
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.23
  + Звершення опрацювання теми: 11.12.23

Тема №3: Ознайомлення з елементами ООП

* Джерела Інформації
  + - Практика в НУ “ЛП”
    - Курси https://acode.com.ua/
  + Що опрацьовано:
    - Ознайомлення з ООП: класами, об’єктами, методами
    - Поняття про класи та їх створення
    - Відмінність класів від структур
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.23
* Звершення опрацювання теми: 11.12.23
* Тема №4: Специфікатори доступа public і private
* Джерела Інформації

Практика в НУ “ЛП”

* + - Курси https://acode.com.ua/
  + Що опрацьовано:
    - Ознайомлено зі специфікаторами public і private, що відсутні в структурах
    - Поняття про правильне використання специфікаторів для змінних та методів класу
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.23
* Звершення опрацювання теми: 11.12.23
* Тема №5: Методи класів, геттери та сеттери
* Джерела Інформації
  + - Практика в НУ “ЛП”
    - Курси https://acode.com.ua/
  + Що опрацьовано:
    - Поняття про методи класів
    - Сеттери на прикладі push\_back, push\_front, insert, set і т.д
    - Геттери на прикладі pop\_back, pop\_front, erase, get, [] і т.д
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.23
* Звершення опрацювання теми: 11.12.23
* Тема №6: Конструктори та деструктори

Джерела Інформації

* + - Практика в НУ “ЛП”
    - Курси https://acode.com.ua/
  + Що опрацьовано:
    - Конструктори для створення та ініціалізації об’єкту класу за замовчуванням
    - Деструктори для знищення об’єкту класу
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.23
* Звершення опрацювання теми: 11.12.23
* Тема №7: Введення в перевантаження операторів

Джерела Інформації

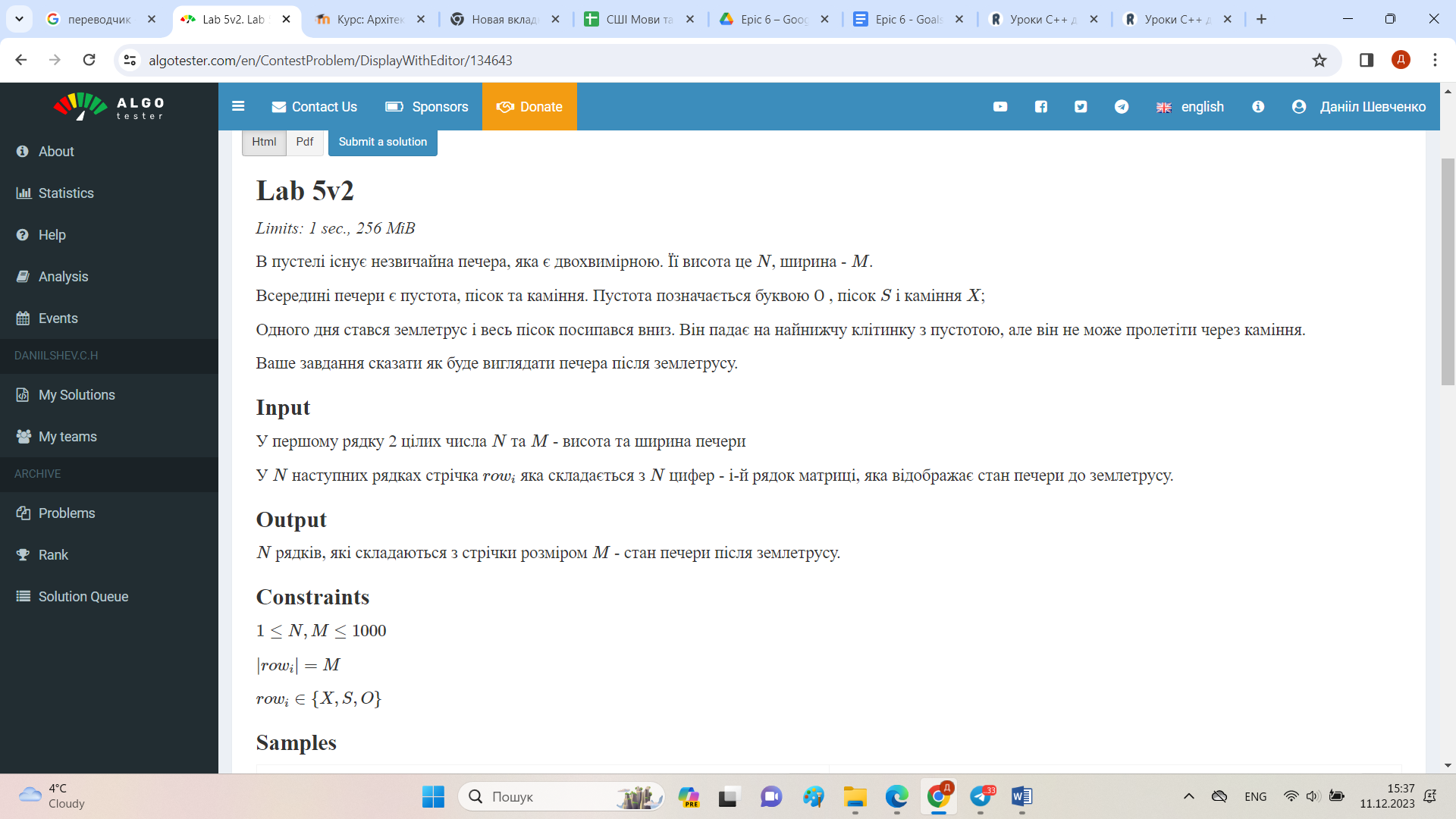
* + - Практика в НУ “ЛП”
    - Курси https://acode.com.ua/
  + Що опрацьовано:
    - Поняття перевантаження оператора
    - Перевантаження оператора []
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.23
* Звершення опрацювання теми: 11.12.23

# **Виконання роботи:**

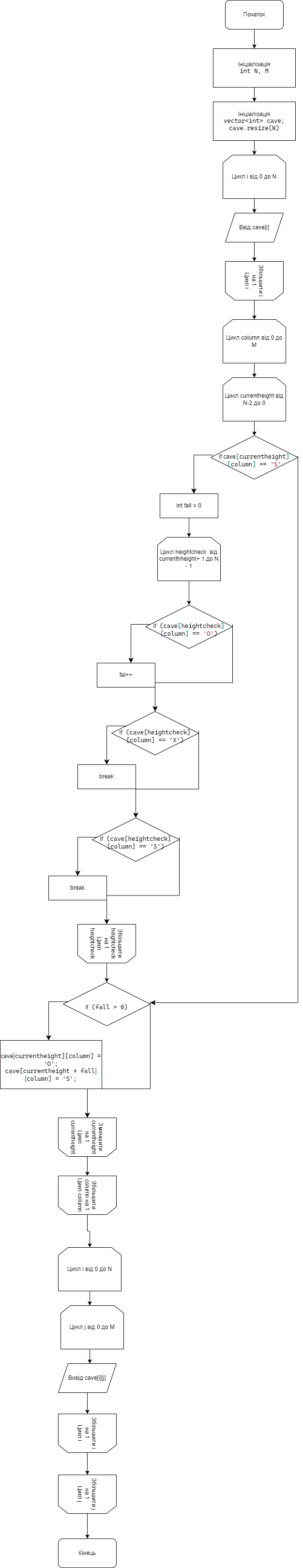
**Algotester Lab5v2**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №1 Algotester Lab5v2



## **2. Код програм з посиланням на зовнішні ресурси:**



#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main()

{

int N, M;

cin >> N >> M;

vector<string>cave;

cave.resize(N);

cin.ignore(30000, '\n');

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

{

getline(cin, cave[i]);

}

for (int column = 0; column < M; column++)

{

for (int currentheight = N - 2; currentheight >= 0; currentheight--)

{

if (cave[currentheight][column] == 'S')

{

int fall = 0;

for (int heightcheck = currentheight+1; heightcheck <= N - 1; heightcheck++)

{

if (cave[heightcheck][column] == 'O')

{

fall++;

}

if (cave[heightcheck][column] == 'X')

{

break;

}

if (cave[heightcheck][column] == 'S')

{

break;

}

}

if (fall > 0)

{

cave[currentheight][column] = 'O';

cave[currentheight + fall][column] = 'S';

}

}

}

}

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

{

for (int j = 0; j < M; j++)

{

cout << cave[i][j];

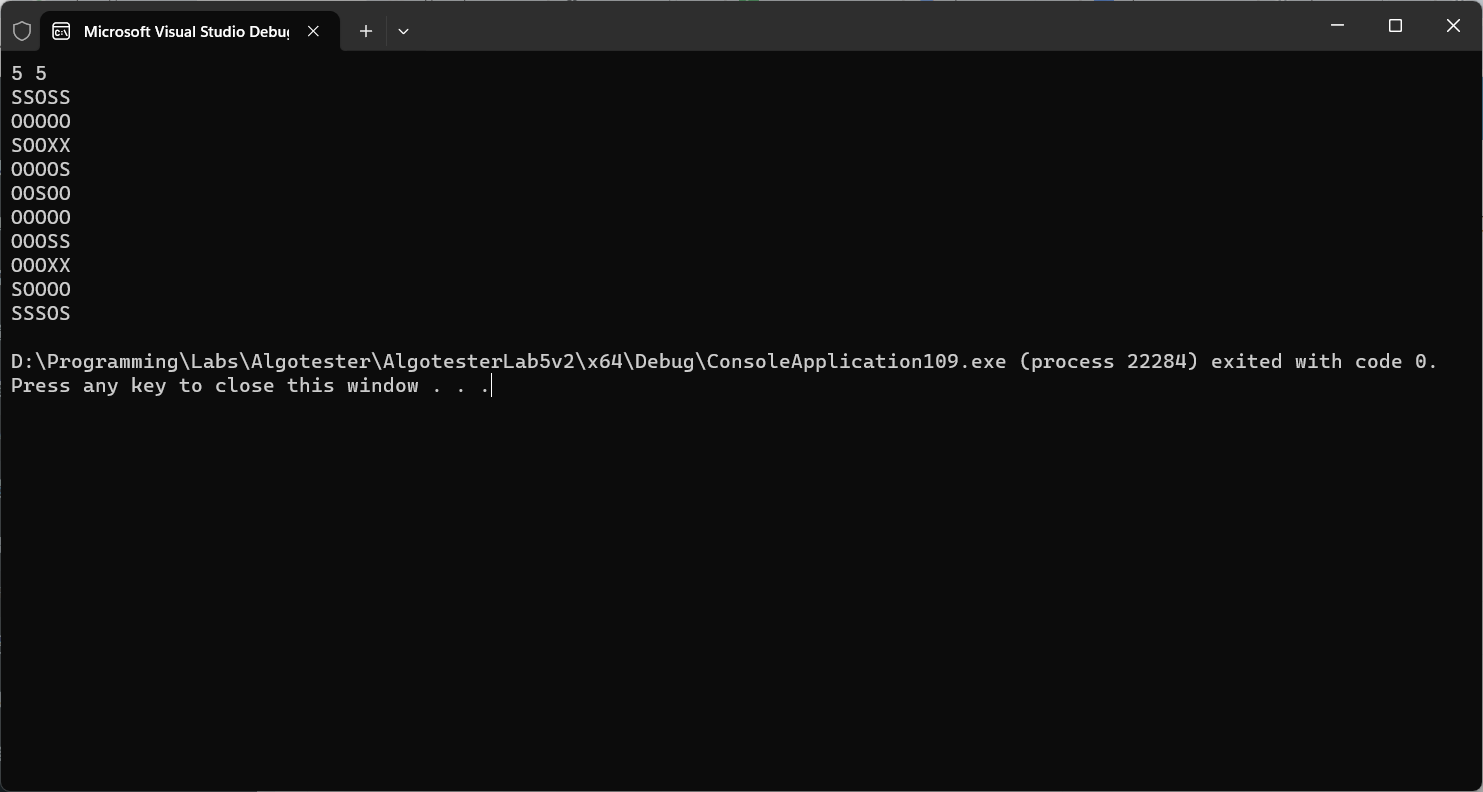
}

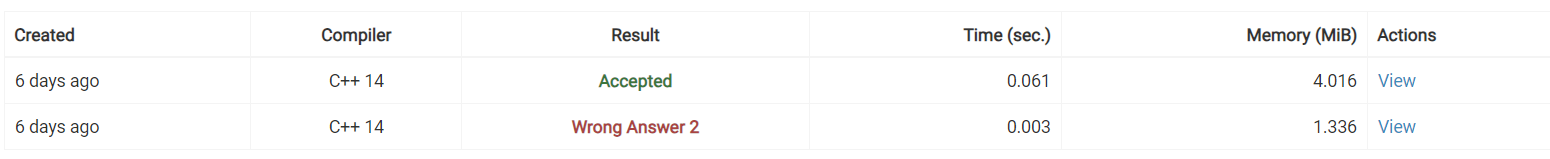
cout << endl;

}

}

**3. Результати виконання завдань, тестування та фактично затрачений час:**



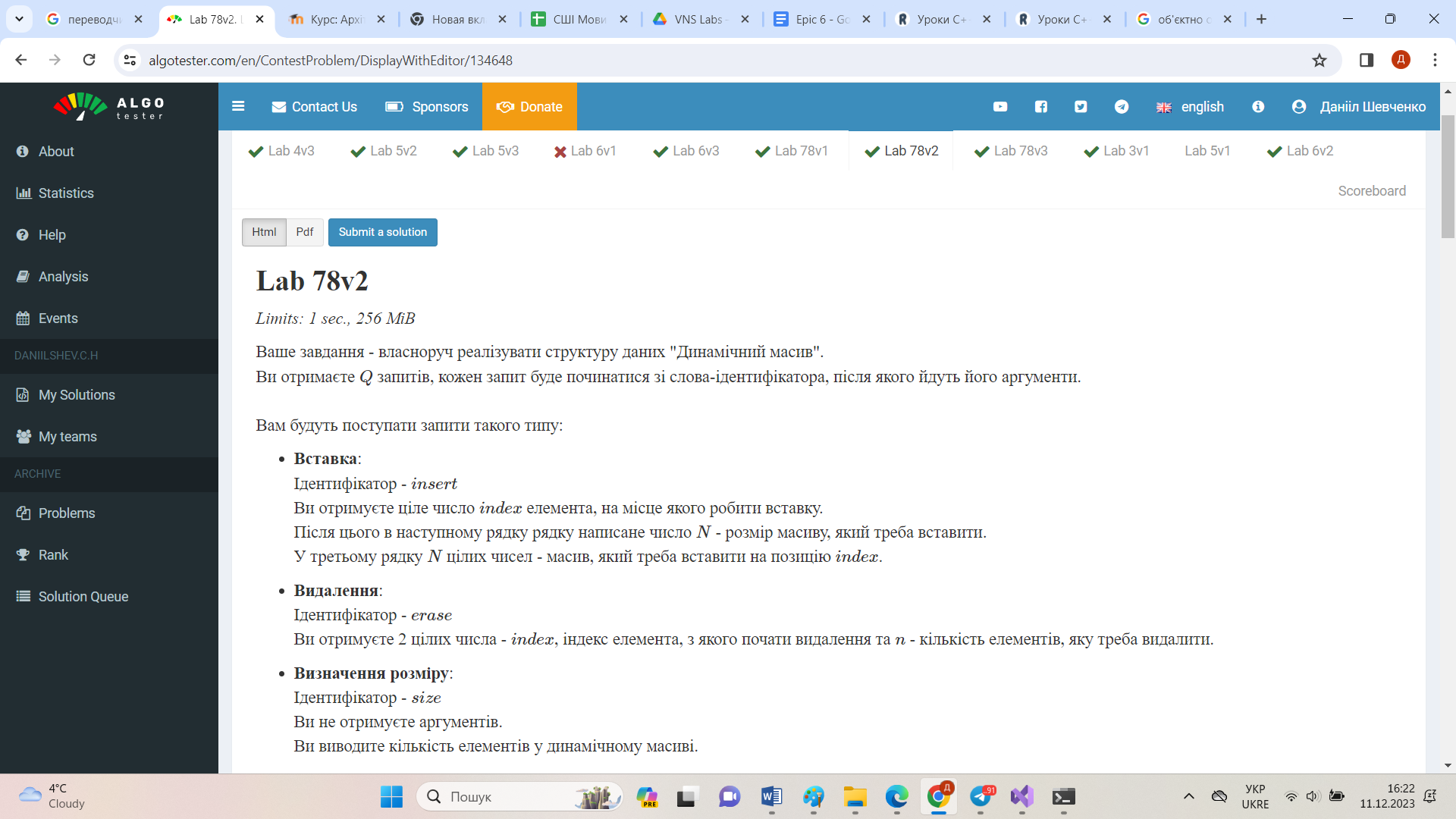


Час затрачений на виконання завдання – 60 хв

**Algotester Lab 78v2**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №2 Algotester Lab78v2



## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

#include <vector>

#include <string>

#include <cmath>

using namespace std;

class dynamicarray

{

private:

vector<int> array;

int capacity = 1;

public:

void insert(int index, int N, vector<int> insertarray)

{

array.insert(array.begin() + index, insertarray.begin(), insertarray.end());

if (array.size() >= capacity)

{

while (capacity <= array.size())

{

capacity = capacity \* 2;

}

}

}

int get(int index)

{

return array[index];

}

int size()

{

return array.size();

}

void set(int index, int number)

{

array[index] = number;

}

void erase(int index, int amount)

{

array.erase(array.begin() + index, array.begin() + index + amount);

}

void print()

{

for (int i = 0; i < array.size(); i++)

{

cout << array[i] << " ";

}

}

int capacity\_()

{

return capacity;

}

};

int main()

{

dynamicarray test;

string identificator;

int Q;

cin >> Q;

cin.ignore(30000, '\n');

for (int m = 0; m < Q; m++)

{

cin >> identificator;

if (identificator == "insert")

{

int index;

cin >> index;

int N;

cin >> N;

cin.ignore(30000, '\n');

int j;

vector<int> input;

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

{

cin >> j;

input.push\_back(j);

}

cin.ignore(30000, '\n');

test.insert(index, N, input);

cout << endl;

}

if (identificator == "get")

{

int index;

cin >> index;

cin.ignore(30000, '\n');

cout << test.get(index);

cout << endl;

}

if (identificator == "size")

{

cout << test.size();

cout << endl;

}

if (identificator == "set")

{

int x;

cin >> x;

int y;

cin >> y;

test.set(x, y);

cout << endl;

}

if (identificator == "erase")

{

int x;

cin >> x;

int y;

cin >> y;

test.erase(x, y);

cout << endl;

}

if (identificator == "print")

{

test.print();

cout << endl;

}

if (identificator == "capacity")

{

cout << test.capacity\_();

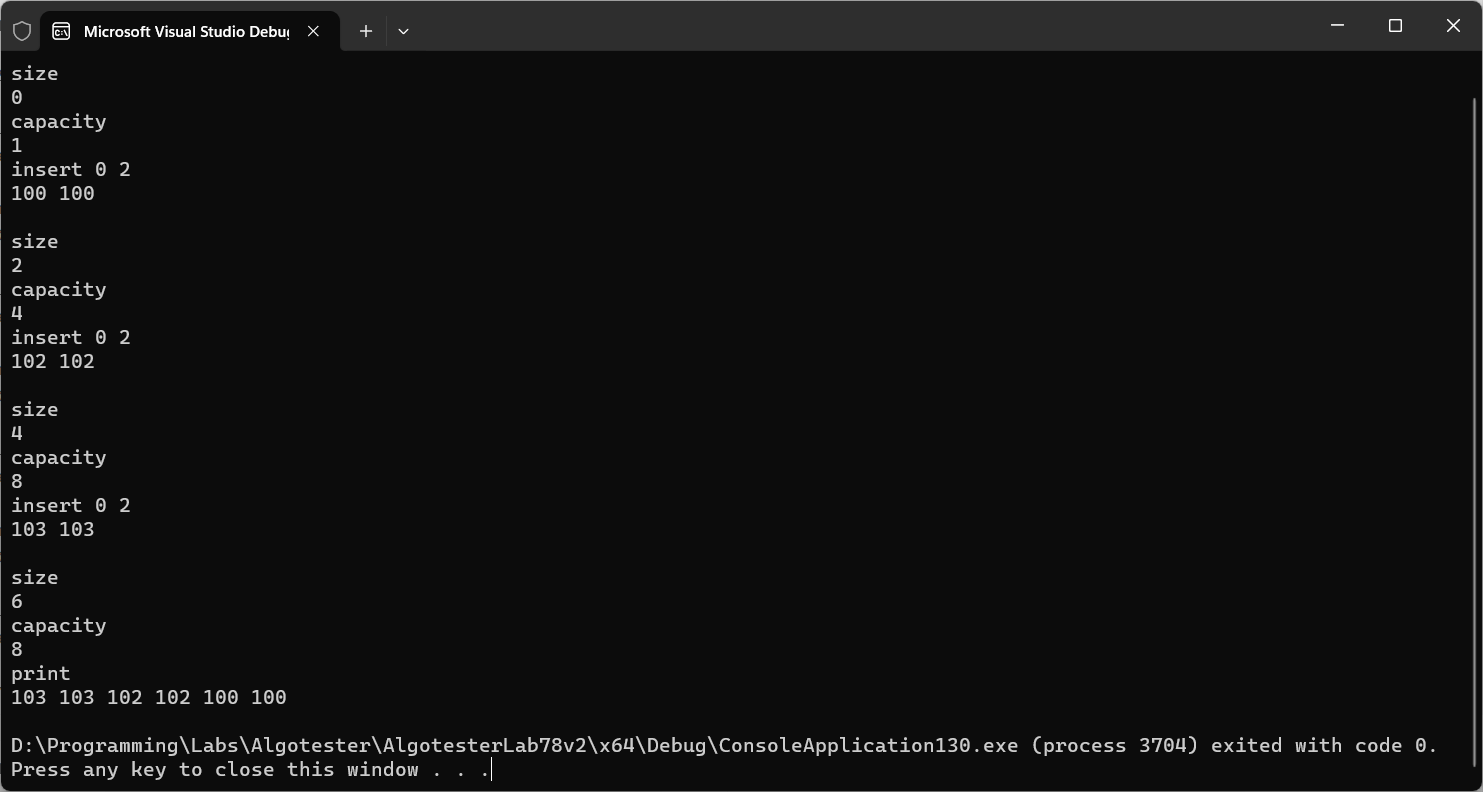
cout << endl;

}

}

}

## **3. Результати виконання завдань, тестування та фактично затрачений час:**





Час затрачений на виконання завдання – 60 хвилин

**VNS Lab 10**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №3 VNS Lab 10

23.Запису в лінійному списку містять ключове поле типу \*char (рядок

символів). Сформувати двонаправлений список. Знищити елемент із заданим

ключем. Додати К елементів після елемента із заданим ключем.

## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Node

{

public:

string data;

Node\* prev;

Node\* next;

public:

Node(string input)

{

this->data = input;

this->prev = nullptr;

this->next = nullptr;

}

};

class DoubleLinkedList

{

public:

Node\* head;

Node\* tail;

DoubleLinkedList()

{

head = nullptr;

tail = nullptr;

}

~DoubleLinkedList()

{

while (head != nullptr)

{

pop\_front();

}

}

void push\_front(string input)

{

Node\* node = new Node(input);

node->next = head;

if (head != nullptr)

{

head->prev = node;

}

if (tail == nullptr)

{

tail = node;

}

head = node;

}

void push\_back(string input)

{

Node\* node = new Node(input);

node->prev = tail;

if (tail != nullptr)

{

tail->next = node;

}

if (head == nullptr)

{

head = node;

}

tail = node;

}

void pop\_front()

{

if (head == nullptr)

{

return;

}

Node\* node = head->next;

if (node != nullptr)

{

node->prev = nullptr;

}

else

{

tail = node;

}

delete head;

head = node;

}

void pop\_back()

{

if (tail == nullptr)

{

return;

}

Node\* node = tail->prev;

if (node != nullptr)

{

node->next = nullptr;

}

else

{

head = node;

}

delete tail;

tail = node;

}

Node\* get(int index)

{

if (index < 0)

{

return nullptr;

}

Node\* node = head;

int N = 0;

while (node != nullptr && N != index && node->next != nullptr)

{

node = node->next;

N++;

}

if (N == index)

{

return node;

}

else

{

return nullptr;

}

}

string getdata(int index)

{

if (get(index) == nullptr)

{

return "NONE";

}

return get(index)->data;

}

string operator [](int index)

{

if (get(index) == nullptr)

{

return "NONE";

}

return get(index)->data;

}

void insert(int index, string input)

{

Node\* left = get(index);

if (left == nullptr)

{

push\_front(input);

}

Node\* right = left->next;

Node\* node = new Node(input);

left->next = node;

node->next = right;

node->prev = left;

node->next = right;

if (right == nullptr)

{

tail = node;

}

}

void erase(int index)

{

Node\* node = get(index);

if (node == nullptr)

{

return;

}

if (node->prev == nullptr)

{

pop\_front();

return;

}

if (node->next == nullptr)

{

pop\_back();

return;

}

Node\* left = node->prev;

Node\* right = node->next;

left->next = right;

right->prev = left;

delete node;

node = nullptr;

}

void print()

{

for (Node\* node = head; node != nullptr; node = node->next)

{

cout << node->data << " ";

}

if (head == nullptr)

{

cout << "\033[31m" << "The list is empty" << endl << "\033[0m";

}

}

};

void writeinto(DoubleLinkedList test, int size)

{

ofstream out("Lab10.txt");

if (!out)

{

cout << "Can't create file for writing" << endl;

return;

}

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

{

out << test[i] << endl;

}

out.close();

}

void readfrom(DoubleLinkedList& mylist)

{

ifstream input("Lab10.txt");

if (!input)

{

cout << "Can't open file for reading" << endl;

return;

}

string str;

while (getline(input, str))

{

mylist.push\_back(str);

}

input.close();

}

int main()

{

DoubleLinkedList test;

int Q;

cout << "\033[32m" << "Enter the amount of elements: " << "\033[0m";

cin >> Q;

if (Q > 0)

{

cout << "\033[32m" << "Enter the words:" << endl << "\033[0m";

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

{

string x;

cin >> x;

test.push\_back(x);

}

}

cout << "\033[32m" << "The list is now: " << "\033[0m" << endl;

test.print();

cout << endl;

int index1;

tryagain:

cout << "\033[36m" << "Enter the element to erase: " << "\033[0m";

cin >> index1;

if (index1 >= Q)

{

cin.clear();

goto tryagain;

}

test.erase(index1);

cout << "\033[32m" << "The list is now: " << "\033[0m" << endl;

test.print();

cout << endl;

int amount;

cout << "\033[1m\033[32m" << "Enter the amount of elements to insert: " << "\033[0m";

cin >> amount;

cout << "\033[1m\033[32m" << "Enter the elements to insert: " << "\033[0m";

string\* input = new string[amount];

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

{

cin >> input[i];

}

int index2;

tryagain2:

cout << "\033[1m\033[32m" << "Enter the element index to insert after: " << "\033[0m";

cin >> index2;

if (index2 >= Q - 1)

{

goto tryagain2;

}

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

{

test.insert(index2 + i, input[i]);

}

cout << "\033[32m" << "The list is now: " << "\033[0m" << endl;

test.print();

cout << endl;

writeinto(test, Q - 1 + amount);

DoubleLinkedList test2;

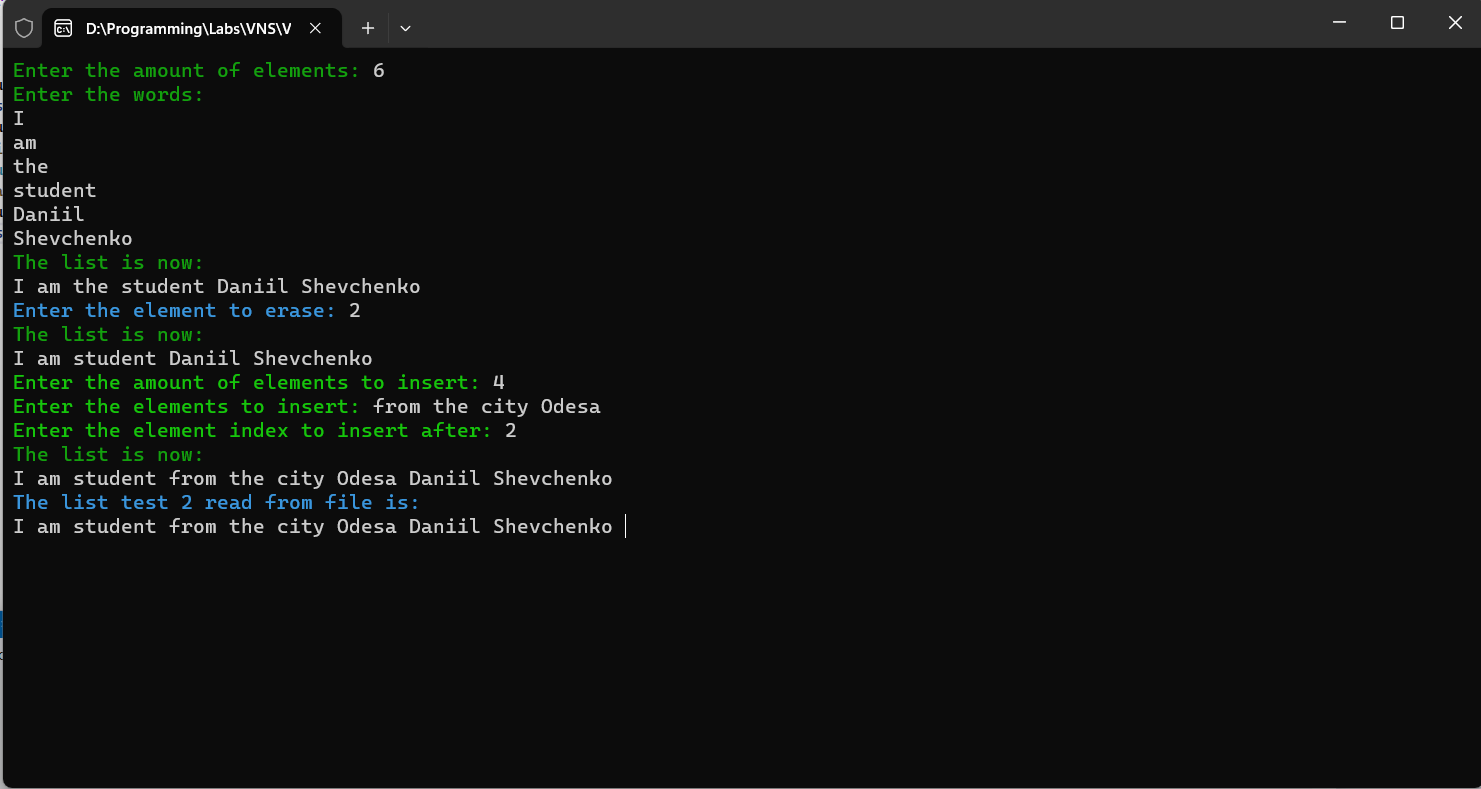
readfrom(test2);

cout << "\033[36m" << "The list test 2 read from file is: " << "\033[0m" << endl;

test2.print();

}

## **3. Результати виконання завдань, тестування та фактично затрачений час:**

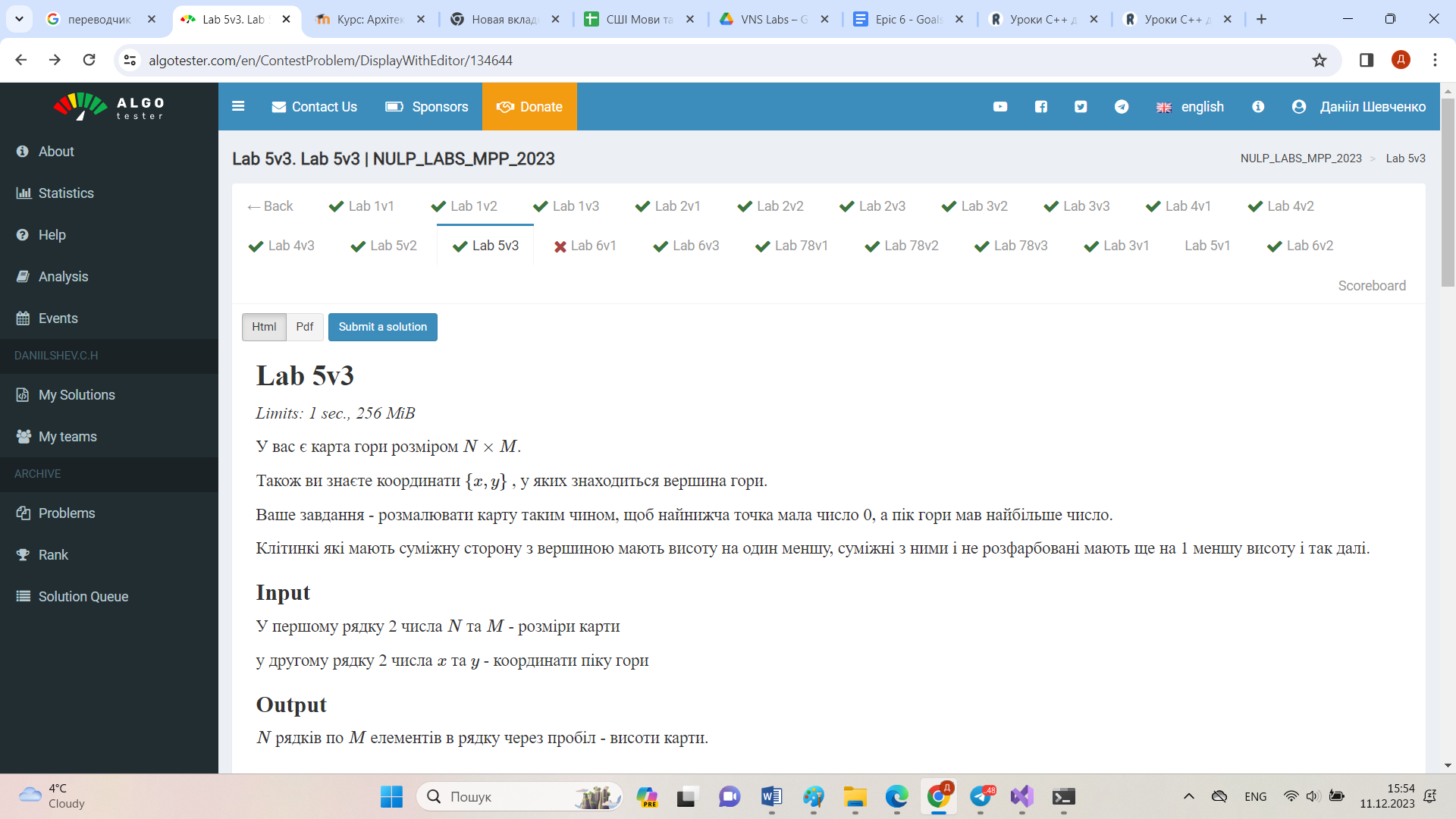


Час затрачений на виконання завдання – 180 хвилин

**Algotester Lab5 V3(Self-Practice)**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання 4 Algotester Lab5 V3



## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

#include <vector>

using namespace std;

int minfounder(vector<vector<int>> array, int N, int M)

{

int min = array[0][0];

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

{

for (int j = 0; j < M; j++)

{

if (array[i][j] < min)

{

min = array[i][j];

}

}

}

return min;

}

using namespace std;

int main()

{

int N, M;

cin >> N >> M;

int x, y;

cin >> x >> y;

int xmod = x - 1;

int ymod = y - 1;

vector<vector<int>> mountain;

mountain.resize(N);

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

{

mountain[i].resize(M);

}

int testpoint = 1002;

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

{

for (int j = 0; j < M; j++)

{

mountain[i][j] = -1;

}

}

mountain[xmod][ymod] = testpoint;

if (N > 1)

{

for (int i = xmod; i > 0; i--)

{

for (int j = ymod; j > 0; j--)

{

if (mountain[i][j - 1] == -1)

{

mountain[i][j - 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j >= 0; j--)

{

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

}

for (int j = ymod; j < M - 1; j++)

{

if (mountain[i][j + 1] == -1)

{

mountain[i][j + 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j <= M - 1; j++)

{

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

}

}

for (int i = xmod; i < N - 1; i++)

{

for (int j = ymod; j > 0; j--)

{

if (mountain[i][j - 1] == -1)

{

mountain[i][j - 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j >= 0; j--)

{

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

}

for (int j = ymod; j < M - 1; j++)

{

if (mountain[i][j + 1] == -1)

{

mountain[i][j + 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j <= M - 1; j++)

{

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

}

}

int min = minfounder(mountain, N, M);

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

{

for (int j = 0; j < M; j++)

{

mountain[i][j] = -1;

}

}

mountain[xmod][ymod] = testpoint - min;

for (int i = xmod; i > 0; i--)

{

for (int j = ymod; j > 0; j--)

{

if (mountain[i][j - 1] == -1)

{

mountain[i][j - 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j >= 0; j--)

{

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

}

for (int j = ymod; j < M - 1; j++)

{

if (mountain[i][j + 1] == -1)

{

mountain[i][j + 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j <= M - 1; j++)

{

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

}

}

for (int i = xmod; i < N - 1; i++)

{

for (int j = ymod; j > 0; j--)

{

if (mountain[i][j - 1] == -1)

{

mountain[i][j - 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j >= 0; j--)

{

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

}

for (int j = ymod; j < M - 1; j++)

{

if (mountain[i][j + 1] == -1)

{

mountain[i][j + 1] = mountain[i][j] - 1;

}

}

for (int j = ymod; j <= M - 1; j++)

{

if ((i < N - 1) && mountain[i + 1][j] == -1)

{

mountain[i + 1][j] = mountain[i][j] - 1;

}

if (i > 0 && mountain[i - 1][j] == -1)

{

mountain[i - 1][j] = mountain[i][j] - 1;

}

}

}

}

else if (N == 1)

{

for (int j = ymod; j > 0; j--)

{

mountain[0][j - 1] = mountain[0][j] - 1;

}

for (int j = ymod; j < M - 1; j++)

{

mountain[0][j + 1] = mountain[0][j] - 1;

}

int min = minfounder(mountain, N, M);

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

{

for (int j = 0; j < M; j++)

{

mountain[i][j] = -1;

}

}

mountain[xmod][ymod] = testpoint - min;

for (int j = ymod; j > 0; j--)

{

mountain[0][j - 1] = mountain[0][j] - 1;

}

for (int j = ymod; j < M - 1; j++)

{

mountain[0][j + 1] = mountain[0][j] - 1;

}

}

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

{

for (int j = 0; j < M; j++)

{

cout << mountain[i][j] << " ";

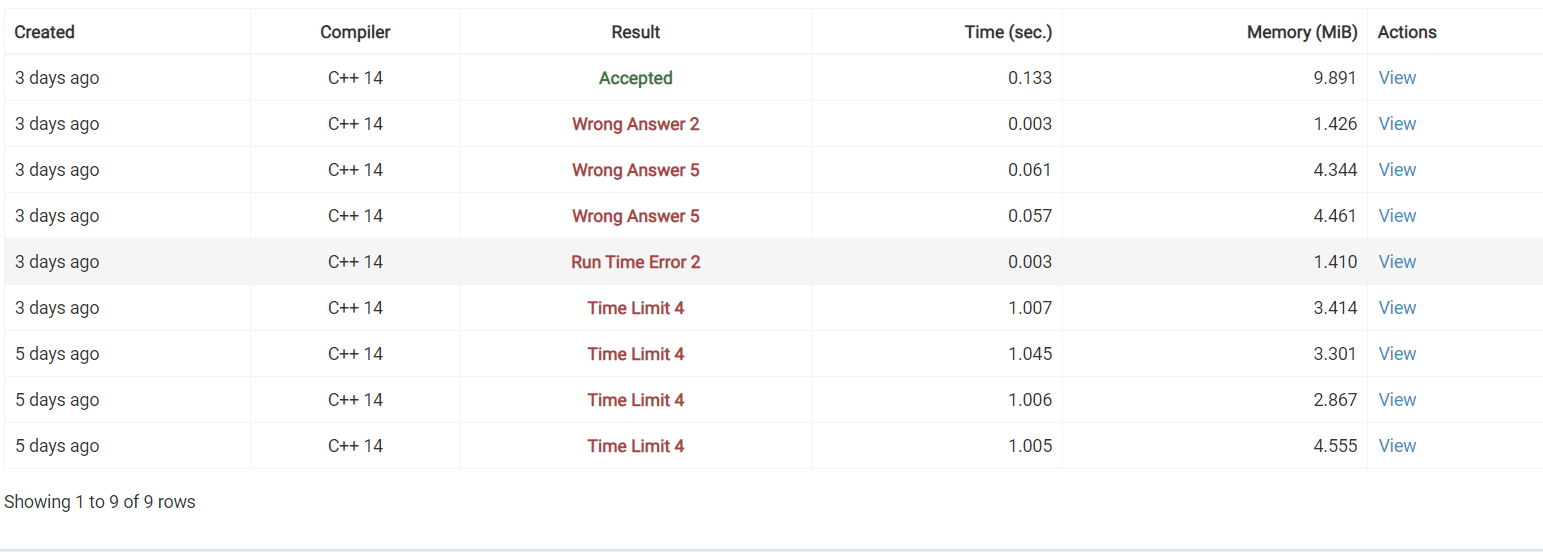
}

cout << endl;

}

## }

## **3. Результати виконання завдань, тестування та фактично затрачений час:**

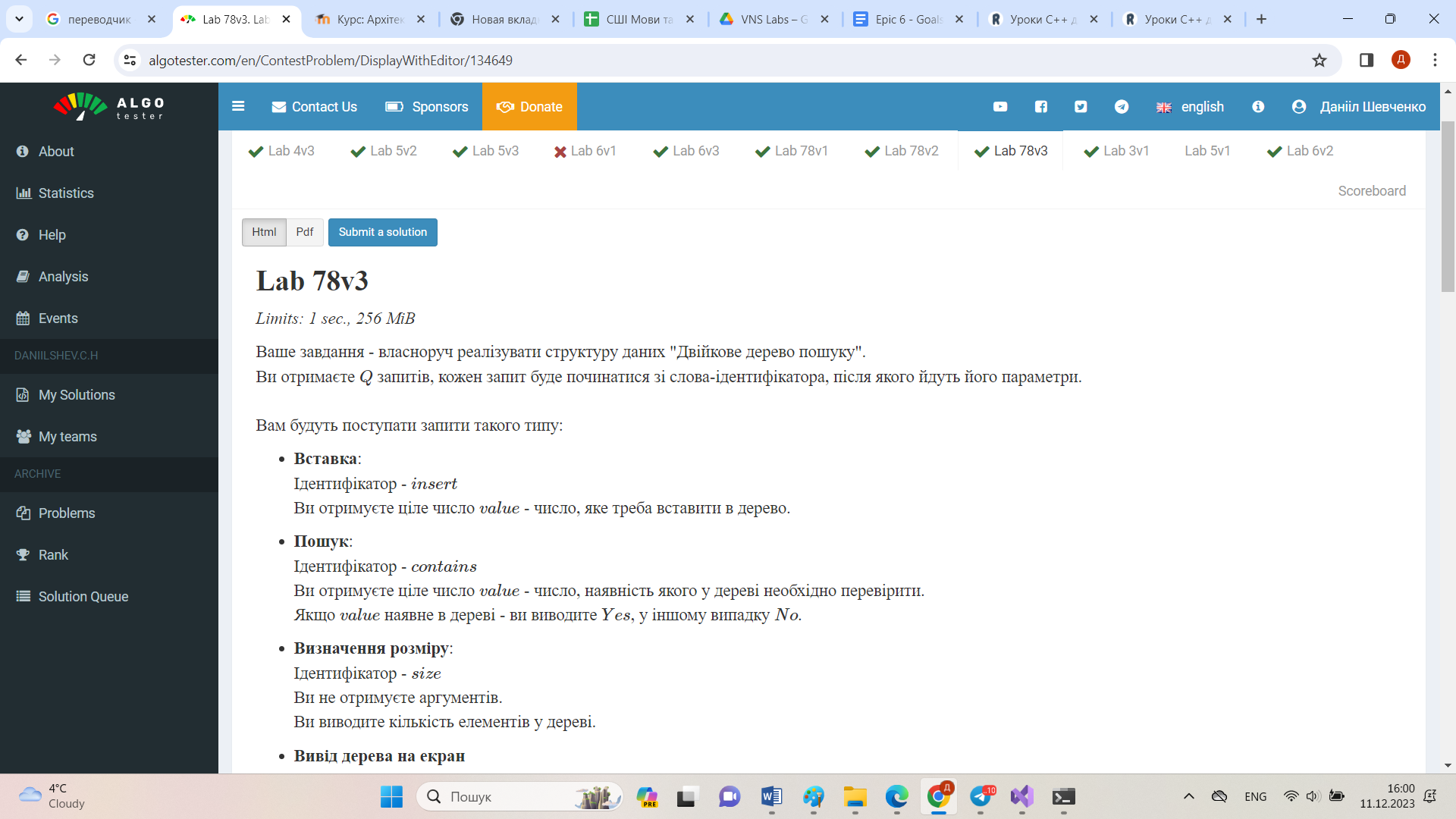


Час затрачений на виконання завдання – 2 години

**Algotester Lab78v3(Self-Practice)**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №5 Algotester Lab78v3(Self-Practice)



## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

void upgradesort(vector<int>& array, int length)

{

for (int startindex = 0; startindex < length; startindex++)

{

int minindex = startindex;

for (int currentindex = startindex; currentindex < length; currentindex++)

{

if (array[currentindex] < array[minindex])

{

minindex = currentindex;

}

}

swap(array[startindex], array[minindex]);

}

}

class binarytree

{

private:

vector<int> tree;

public:

void insert(int input)

{

bool presence = false;

for (int i = 0; i < tree.size(); i++)

{

if (tree[i] == input)

{

presence = true;

}

}

if (!presence)

{

tree.insert(tree.begin(), input);

upgradesort(tree, tree.size());

}

}

string contains(int input)

{

bool presence = false;

for (int i = 0; i < tree.size(); i++)

{

if (tree[i] == input)

{

presence = true;

}

}

if (presence)

{

return "Yes";

}

else

{

return "No";

}

}

int size()

{

return tree.size();

}

void print()

{

for (int i = 0; i < tree.size(); i++)

{

cout << tree[i] << " ";

}

}

};

int main()

{

binarytree tree;

string identificator;

int Q;

cin >> Q;

cin.ignore(30000, '\n');

for (int m = 0; m < Q; m++)

{

cin >> identificator;

if (identificator == "insert")

{

int input;

cin >> input;

cin.ignore(30000, '\n');

tree.insert(input);

}

if (identificator == "contains")

{

int input;

cin >> input;

cin.ignore(30000, '\n');

cout << tree.contains(input);

cout << endl;

}

if (identificator == "size")

{

cout << tree.size();

cout << endl;

}

if (identificator == "print")

{

tree.print();

cout << endl;

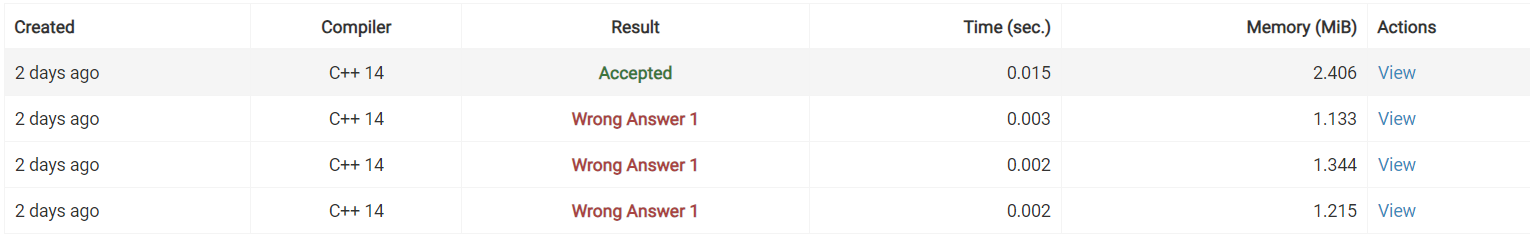
}

}

}

## }

## **3 . Результати виконання завдань, тестування та фактично затрачений час:**



Час затрачений на виконання завдання – 30 хвилин

**Practice Task # 6**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №6 Practice-Task # 6

# Зв’язаний список

## Задача №1 - Реверс списку (Reverse list)

***Реалізувати метод реверсу списку:*** Node\* reverse(Node \*head);

*Умови задачі:*

-       використовувати цілочисельні значення в списку;

-       реалізувати метод реверсу;

-       реалізувати допоміжний метод виведення вхідного і обернутого списків;

### Мета задачі

***Розуміння структур даних:*** Реалізація методу реверсу для зв’язаних списків є чудовим способом для поглиблення розуміння зв’язаних списків як фундаментальної структури даних. Він заохочує практичний підхід до вивчення того, як структуруються пов’язані списки та як ними маніпулювати.

***Розвиток алгоритмічне мислення:*** Це завдання розвиває алгоритмічне мислення. Перевертання пов’язаного списку вимагає логічного підходу до маніпулювання покажчиками, що є ключовим навиком у інформатиці.

***Засвоїти механізми маніпуляції з покажчиками:*** пов’язані списки значною мірою залежать від покажчиків. Це завдання покращить навички маніпулювання вказівниками, що є ключовим аспектом у таких мовах, як C++.

***Розвинути навички розв’язувати задачі:*** перевернути пов’язаний список непросто й вимагає творчого й логічного мислення, таким чином покращуючи свої навички розв’язування поставлених задач.

### Пояснення прикладу

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов’язаного списку.

Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.

***printList*** — допоміжна функція для відображення списку.

Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

## Задача №2 - Порівняння списків

bool compare(Node \*h1, Node \*h2);

*Умови задачі:*

-       використовувати цілочисельні значення в списку;

-       реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;

-       якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

### Мета задачі

***Розуміння рівності в структурах даних:*** це завдання допомагає зрозуміти, як визначається рівність у складних структурах даних, таких як зв’язані списки. На відміну від примітивних типів даних, рівність пов’язаного списку передбачає порівняння кожного елемента та їх порядку.

***Поглиблення розуміння зв’язаних списків:*** Порівнюючи зв’язані списки, дозволяють покращити своє розуміння обходу, фундаментальної операції в обробці зв’язаних списків.

***Розуміння ефективність алгоритму:*** це завдання також вводить поняття ефективності алгоритму. Студенти вчаться ефективно порівнювати елементи, що є навичкою, важливою для оптимізації та зменшення складності обчислень.

***Розвинути базові навики роботи з реальними програми:*** функції порівняння мають вирішальне значення в багатьох реальних програмах, таких як виявлення змін у даних, синхронізація структур даних або навіть у таких алгоритмах, як сортування та пошук.

***Розвинути навик вирішення проблем і увага до деталей:*** це завдання заохочує скрупульозний підхід до програмування, оскільки навіть найменша неуважність може призвести до неправильних результатів порівняння. Це покращує навички вирішення проблем і увагу до деталей.

### Пояснення прикладу

●      Для пов’язаного списку визначено структуру ***Node***.

●      Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.

●      Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

●      Основна функція ***main*** створює два списки та демонструє порівняння.

## Задача №3 – Додавання великих чисел

Node\* add(Node \*n1, Node \*n2);

*Умови задачі:*

-       використовувати цифри від 0 до 9 для значень у списку;

-       реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379  ⟹  9→7→3);

-       функція повертає новий список, передані в функцію списки не модифікуються.

### Мета задачі

***Розуміння операцій зі структурами даних:*** це завдання унаочнює практичне використання списка для обчислювальних потреб. Арифметичні операції з великими числами це окремий клас задач, для якого використання списків допомагає обійти обмеження у представленні цілого числа сучасними комп’ютерами.

***Поглиблення розуміння зв’язаних списків:*** Застосовування зв’язаних списків для арифметичних операції з великими числами дозволяє покращити розуміння операцій з обробки зв’язаних списків.

***Розуміння ефективність алгоритму:*** це завдання дозволяє порівняти швидкість алгоритму додавання з використанням списків зі швидкістю вбудованих арифметичних операцій. Студенти вчаться розрізняти позитивні та негативні ефекти при виборі структур даних для реалізації практичних програм.

***Розвинути базові навики роботи з реальними програми:*** арифметичні операції з великими числами використовуються у криптографії, теорії чисел, астрономії, та ін.

***Розвинути навик вирішення проблем і увага до деталей:*** завдання покращує розуміння обмежень у представленні цілого числа сучасними комп’ютерами та пропонує спосіб його вирішення.

# Бінарні дерева

## Задача №4 - Віддзеркалення дерева

TreeNode \*create\_mirror\_flip(TreeNode \*root);

*Умови задачі:*

-       використовувати цілі числа для значень у вузлах дерева

-       реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева

-       функція повертає нове дерево, передане в функцію дерево не модифікується

### Мета задачі

***Розуміння структур даних:*** Реалізація методу віддзеркалення бінарного дерева покращує розуміння структури бінарного дерева, виділення пам’яті для вузлів та зв’язування їх у єдине ціле. Це один з багатьох методів роботи з бінарними деревами.

***Розвиток алгоритмічне мислення:*** Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева продемонструє розгортання рекурсивного виклику.

## Задача №5 - Записати кожному батьківському вузлу суму підвузлів

void tree\_sum(TreeNode \*root);

*Умови задачі:*

-       використовувати цілочисельні значення у вузлах дерева;

-       реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів

-       вузол-листок не змінює значення

-       значення змінюються від листків до кореня дерева

### Мета задачі

***Розуміння структур даних:*** Реалізація методу підрахунку сум підвузлів бінарного дерева покращує розуміння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

***Розвиток алгоритмічне мислення:*** Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розгортання рекурсивного виклику.

## **2. Код програм з посиланням на зовнішні ресурси:**

TASK #1

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Node

{

public:

int data;

Node\* prev;

Node\* next;

public:

Node(int input)

{

this->data = input;

this->prev = nullptr;

this->next = nullptr;

}

};

class DoubleLinkedList

{

public:

Node\* head;

Node\* tail;

DoubleLinkedList()

{

head = nullptr;

tail = nullptr;

}

~DoubleLinkedList()

{

while (head != nullptr)

{

pop\_front();

}

}

void push\_front(int input)

{

Node\* node = new Node(input);

node->next = head;

if (head != nullptr)

{

head->prev = node;

}

if (tail == nullptr)

{

tail = node;

}

head = node;

}

void push\_back(int input)

{

Node\* node = new Node(input);

node->prev = tail;

if (tail != nullptr)

{

tail->next = node;

}

if (head == nullptr)

{

head = node;

}

tail = node;

}

void pop\_front()

{

if (head == nullptr)

{

return;

}

Node\* node = head->next;

if (node != nullptr)

{

node->prev = nullptr;

}

else

{

tail = node;

}

delete head;

head = node;

}

void pop\_back()

{

if (tail == nullptr)

{

return;

}

Node\* node = tail->prev;

if (node != nullptr)

{

node->next = nullptr;

}

else

{

head = node;

}

delete tail;

tail = node;

}

Node\* get(int index)

{

if (index < 0)

{

return nullptr;

}

Node\* node = head;

int N = 0;

while (node != nullptr && N != index && node->next != nullptr)

{

node = node->next;

N++;

}

if (N == index)

{

return node;

}

else

{

return nullptr;

}

}

int getdata(int index)

{

if (get(index) == nullptr)

{

return NAN;

}

return get(index)->data;

}

int operator [](int index)

{

if (get(index) == nullptr)

{

return NAN;

}

return get(index)->data;

}

void insert(int index, int input)

{

Node\* left = get(index);

if (left == nullptr)

{

push\_front(input);

}

Node\* right = left->next;

Node\* node = new Node(input);

left->next = node;

node->next = right;

node->prev = left;

node->next = right;

if (right == nullptr)

{

tail = node;

}

}

void erase(int index)

{

Node\* node = get(index);

if (node == nullptr)

{

return;

}

if (node->prev == nullptr)

{

pop\_front();

return;

}

if (node->next == nullptr)

{

pop\_back();

return;

}

Node\* left = node->prev;

Node\* right = node->next;

left->next = right;

right->prev = left;

delete node;

node = nullptr;

}

void print()

{

for (Node\* node = head; node != nullptr; node = node->next)

{

cout << node->data << " ";

}

if (head == nullptr)

{

cout << "\033[31m" << "The list is empty" << endl << "\033[0m";

}

}

int get\_front()

{

if (head == nullptr)

{

return NAN;

}

int x = head->data;

return x;

}

int get\_back()

{

if (tail == nullptr)

{

return NAN;

}

int x = tail->data;

return x;

}

};

void reverselist(DoubleLinkedList first, DoubleLinkedList& second)

{

while (first.tail != nullptr)

{

second.push\_back(first.get\_back());

first.pop\_back();

}

}

int main()

{

DoubleLinkedList test;

int Q;

cout << "\033[32m" << "Enter the amount of elements: " << "\033[0m";

cin >> Q;

if (Q > 0)

{

cout << "\033[32m" << "Enter the numbers:" << endl << "\033[0m";

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

{

int x;

cin >> x;

test.push\_back(x);

}

}

DoubleLinkedList test2;

reverselist(test, test2);

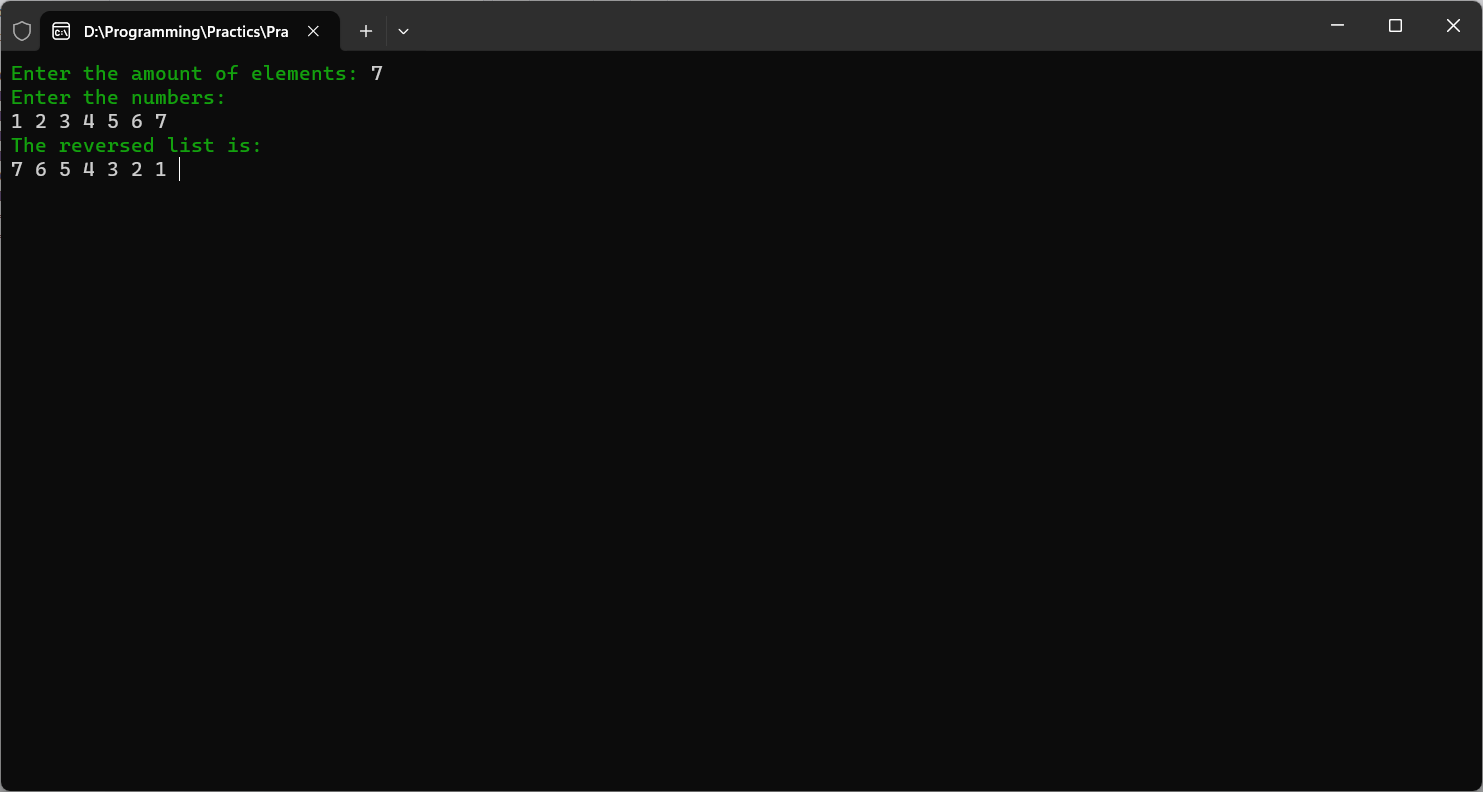
test = test2;

cout << "\033[32m" << "The reversed list is: " << "\033[0m" << endl;

test.print();

}

## **3. Результати виконання завдань, тестування та фактично затрачений час:**



Час затрачений на виконання завдання – 10 хвилин

TASK #2

## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Node

{

public:

int data;

Node\* prev;

Node\* next;

public:

Node(int input)

{

this->data = input;

this->prev = nullptr;

this->next = nullptr;

}

};

class DoubleLinkedList

{

public:

Node\* head;

Node\* tail;

DoubleLinkedList()

{

head = nullptr;

tail = nullptr;

}

~DoubleLinkedList()

{

while (head != nullptr)

{

pop\_front();

}

}

void push\_front(int input)

{

Node\* node = new Node(input);

node->next = head;

if (head != nullptr)

{

head->prev = node;

}

if (tail == nullptr)

{

tail = node;

}

head = node;

}

void push\_back(int input)

{

Node\* node = new Node(input);

node->prev = tail;

if (tail != nullptr)

{

tail->next = node;

}

if (head == nullptr)

{

head = node;

}

tail = node;

}

void pop\_front()

{

if (head == nullptr)

{

return;

}

Node\* node = head->next;

if (node != nullptr)

{

node->prev = nullptr;

}

else

{

tail = node;

}

delete head;

head = node;

}

void pop\_back()

{

if (tail == nullptr)

{

return;

}

Node\* node = tail->prev;

if (node != nullptr)

{

node->next = nullptr;

}

else

{

head = node;

}

delete tail;

tail = node;

}

Node\* get(int index)

{

if (index < 0)

{

return nullptr;

}

Node\* node = head;

int N = 0;

while (node != nullptr && N != index && node->next != nullptr)

{

node = node->next;

N++;

}

if (N == index)

{

return node;

}

else

{

return nullptr;

}

}

int getdata(int index)

{

if (get(index) == nullptr)

{

return NAN;

}

return get(index)->data;

}

int operator [](int index)

{

if (get(index) == nullptr)

{

return NAN;

}

return get(index)->data;

}

void insert(int index, int input)

{

Node\* left = get(index);

if (left == nullptr)

{

push\_front(input);

}

Node\* right = left->next;

Node\* node = new Node(input);

left->next = node;

node->next = right;

node->prev = left;

node->next = right;

if (right == nullptr)

{

tail = node;

}

}

void erase(int index)

{

Node\* node = get(index);

if (node == nullptr)

{

return;

}

if (node->prev == nullptr)

{

pop\_front();

return;

}

if (node->next == nullptr)

{

pop\_back();

return;

}

Node\* left = node->prev;

Node\* right = node->next;

left->next = right;

right->prev = left;

delete node;

node = nullptr;

}

void print()

{

for (Node\* node = head; node != nullptr; node = node->next)

{

cout << node->data << " ";

}

if (head == nullptr)

{

cout << "\033[31m" << "The list is empty" << endl << "\033[0m";

}

}

int get\_front()

{

if (head == nullptr)

{

return NAN;

}

int x = head->data;

return x;

}

int get\_back()

{

if (tail == nullptr)

{

return NAN;

}

int x = tail->data;

return x;

}

};

bool compare(DoubleLinkedList first, DoubleLinkedList second)

{

while (first.tail != nullptr)

{

if (first.get\_back() != second.get\_back())

{

return false;

}

if (first.get\_back() == second.get\_back())

{

first.pop\_back();

second.pop\_back();

}

}

if (first.tail == nullptr && second.tail == nullptr)

{

return true;

}

else

{

return false;

}

}

int main()

{

DoubleLinkedList test;

int Q;

cout << "\033[32m" << "Enter the amount of elements: " << "\033[0m";

cin >> Q;

if (Q > 0)

{

cout << "\033[32m" << "Enter the numbers:" << endl << "\033[0m";

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

{

int x;

cin >> x;

test.push\_back(x);

}

}

DoubleLinkedList test2;

int Q2;

cout << "\033[32m" << "Enter the amount of elements: " << "\033[0m";

cin >> Q2;

if (Q2 > 0)

{

cout << "\033[32m" << "Enter the numbers:" << endl << "\033[0m";

for (int i = 0; i < Q2; i++)

{

int x;

cin >> x;

test2.push\_back(x);

}

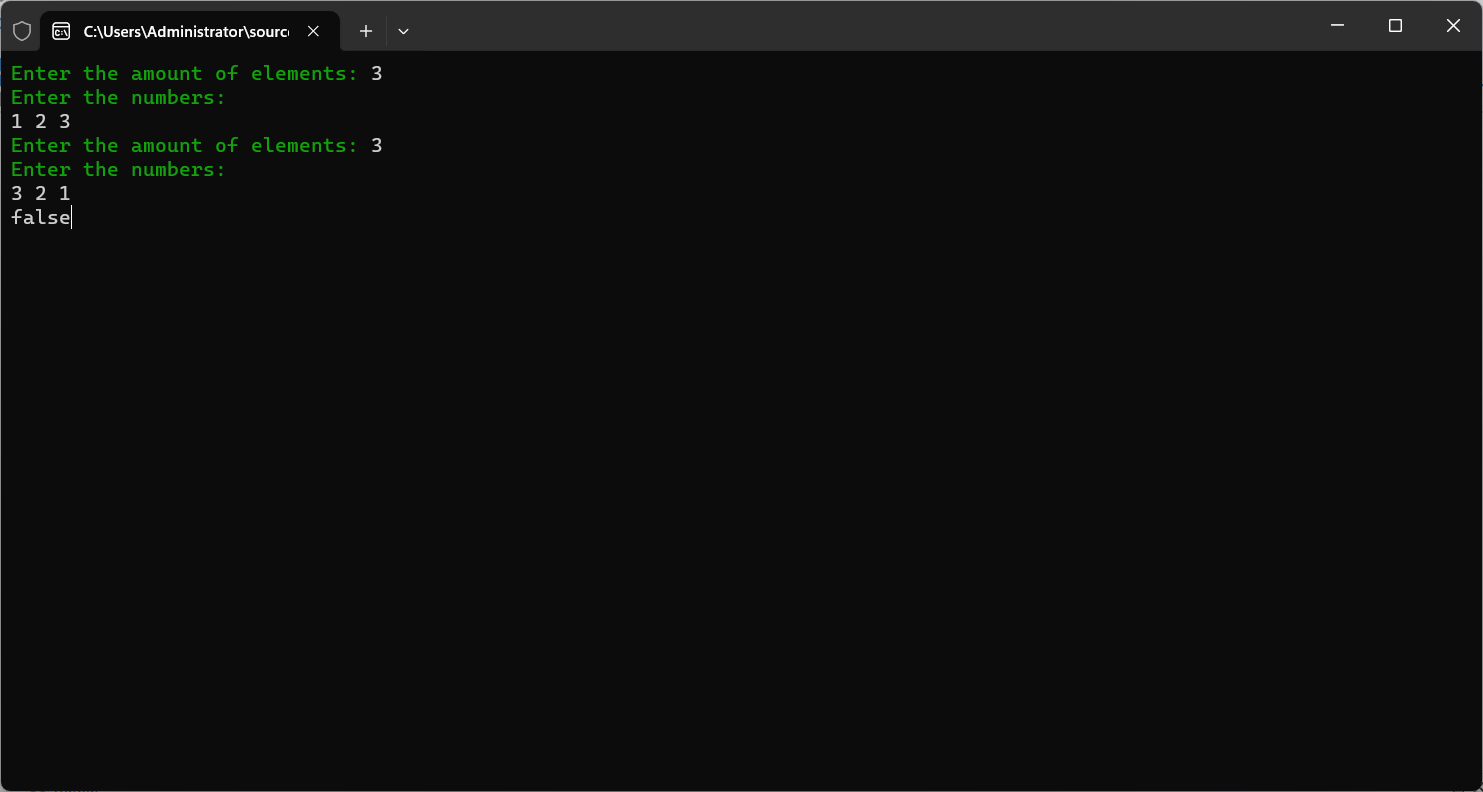
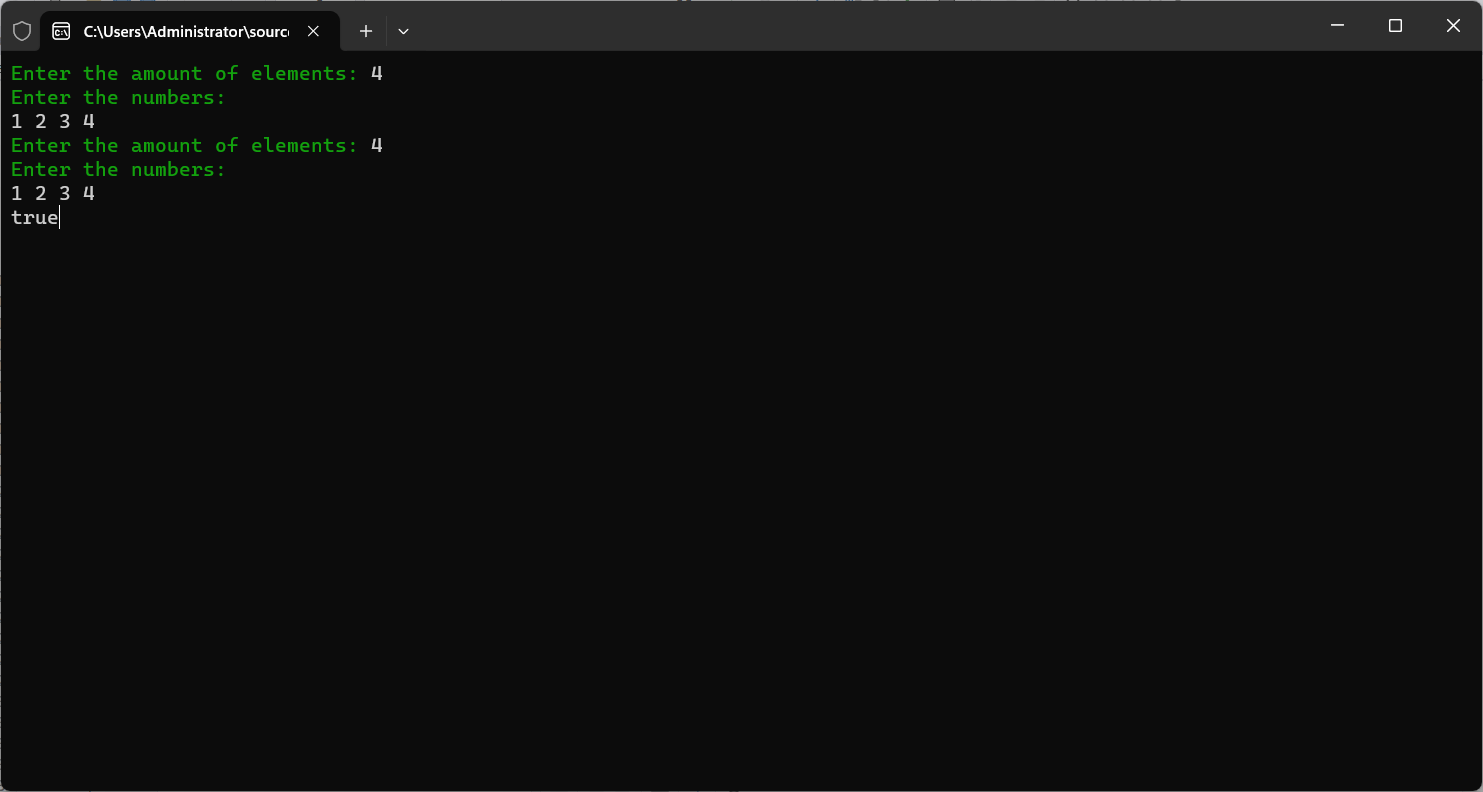
}

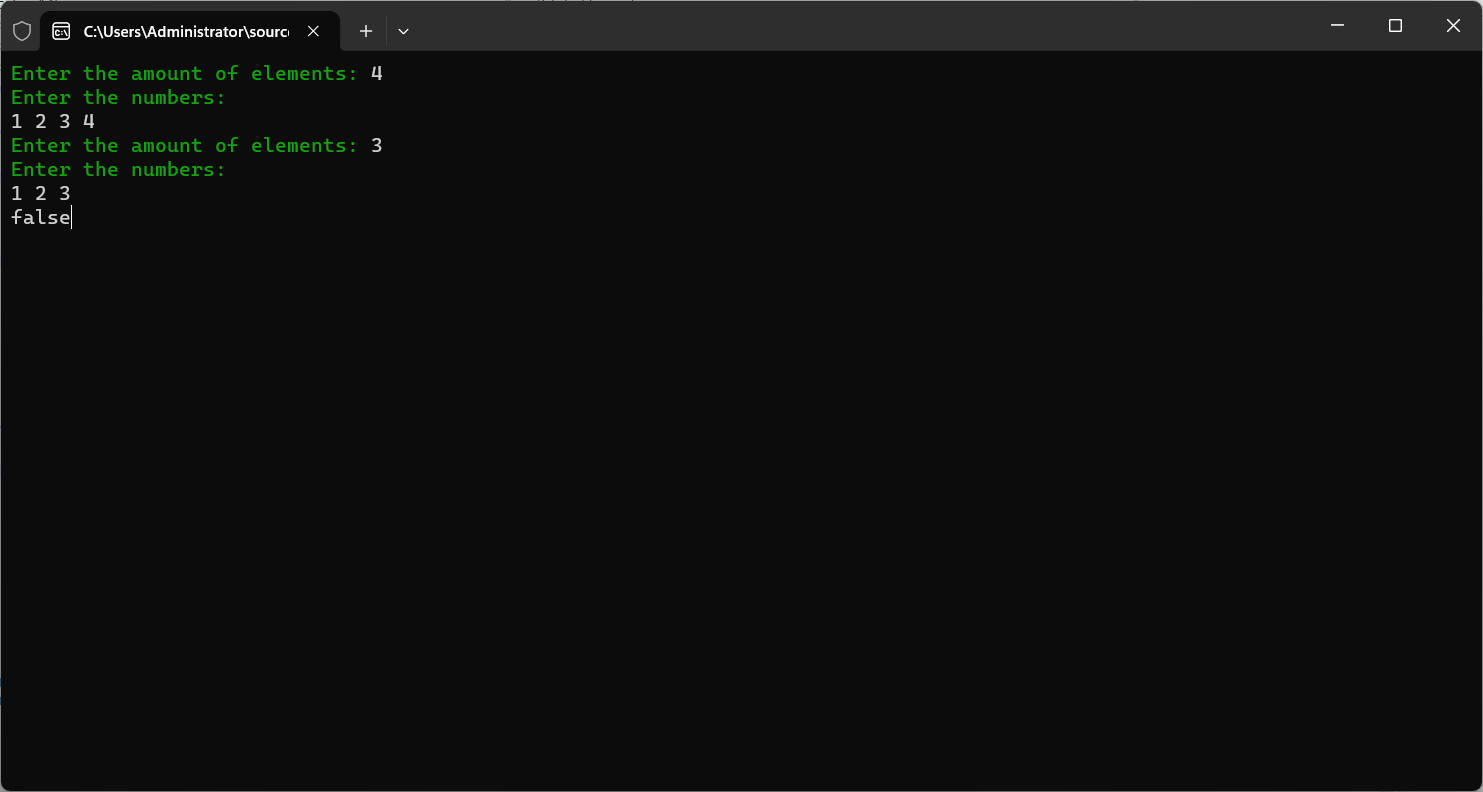
cout << boolalpha;

cout << compare(test, test2);

}

## **3 . Результати виконання завдань, тестування та фактично затрачений час:**



## **TASK #3**

## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Node

{

public:

int data;

Node\* prev;

Node\* next;

public:

Node(int input)

{

this->data = input;

this->prev = nullptr;

this->next = nullptr;

}

};

class DoubleLinkedList

{

public:

Node\* head;

Node\* tail;

DoubleLinkedList()

{

head = nullptr;

tail = nullptr;

}

~DoubleLinkedList()

{

while (head != nullptr)

{

pop\_front();

}

}

void push\_front(int input)

{

Node\* node = new Node(input);

node->next = head;

if (head != nullptr)

{

head->prev = node;

}

if (tail == nullptr)

{

tail = node;

}

head = node;

}

void push\_back(int input)

{

Node\* node = new Node(input);

node->prev = tail;

if (tail != nullptr)

{

tail->next = node;

}

if (head == nullptr)

{

head = node;

}

tail = node;

}

void pop\_front()

{

if (head == nullptr)

{

return;

}

Node\* node = head->next;

if (node != nullptr)

{

node->prev = nullptr;

}

else

{

tail = node;

}

delete head;

head = node;

}

void pop\_back()

{

if (tail == nullptr)

{

return;

}

Node\* node = tail->prev;

if (node != nullptr)

{

node->next = nullptr;

}

else

{

head = node;

}

delete tail;

tail = node;

}

Node\* get(int index)

{

if (index < 0)

{

return nullptr;

}

Node\* node = head;

int N = 0;

while (node != nullptr && N != index && node->next != nullptr)

{

node = node->next;

N++;

}

if (N == index)

{

return node;

}

else

{

return nullptr;

}

}

int& getdata(int index)

{

return get(index)->data;

}

int& operator [](int index)

{

if (get(index) == nullptr)

{

return head->data;

}

return get(index)->data;

}

void insert(int index, int input)

{

Node\* left = get(index);

if (left == nullptr)

{

push\_front(input);

}

Node\* right = left->next;

Node\* node = new Node(input);

left->next = node;

node->next = right;

node->prev = left;

node->next = right;

if (right == nullptr)

{

tail = node;

}

}

void erase(int index)

{

Node\* node = get(index);

if (node == nullptr)

{

return;

}

if (node->prev == nullptr)

{

pop\_front();

return;

}

if (node->next == nullptr)

{

pop\_back();

return;

}

Node\* left = node->prev;

Node\* right = node->next;

left->next = right;

right->prev = left;

delete node;

node = nullptr;

}

void print()

{

for (Node\* node = head; node != nullptr; node = node->next)

{

cout << node->data << " ";

}

if (head == nullptr)

{

cout << "\033[31m" << "The list is empty" << endl << "\033[0m";

}

}

int get\_front()

{

if (head == nullptr)

{

return NAN;

}

int x = head->data;

return x;

}

int get\_back()

{

if (tail == nullptr)

{

return NAN;

}

int x = tail->data;

return x;

}

};

#include <iostream>

using namespace std;

int main()

{

DoubleLinkedList test;

int Q;

cout << "\033[32m" << "Enter the amount of elements: " << "\033[0m";

cin >> Q;

if (Q > 0)

{

cout << "\033[32m" << "Enter the numbers:" << endl << "\033[0m";

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

{

int x;

cin >> x;

test.push\_back(x);

}

}

DoubleLinkedList test2;

int Q2;

cout << "\033[32m" << "Enter the amount of elements: " << "\033[0m";

cin >> Q2;

if (Q2 > 0)

{

cout << "\033[32m" << "Enter the numbers:" << endl << "\033[0m";

for (int i = 0; i < Q2; i++)

{

int x;

cin >> x;

test2.push\_back(x);

}

}

int max = 0;

if (Q > Q2)

{

max = Q;

int count = Q - Q2;

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

{

test2.push\_front(0);

}

}

else

{

max = Q2;

int count = Q2 - Q;

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

{

test.push\_front(0);

}

}

test.print();

cout << endl;

test2.print();

cout << endl;

DoubleLinkedList test3;

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

{

test3.push\_back(0);

}

for (int i = max - 1; i >= 0; i--)

{

test3[i] = test[i] + test2[i];

if (test3[i] >= 10)

{

if (i >= 1)

{

test3.getdata(i) = test3.getdata(i) - 10;

test.getdata(i - 1) = test.getdata(i - 1) + 1;

}

if (i < 1)

{

test3.getdata(i) = test3.getdata(i) - 10;

test3.push\_front(1);

}

}

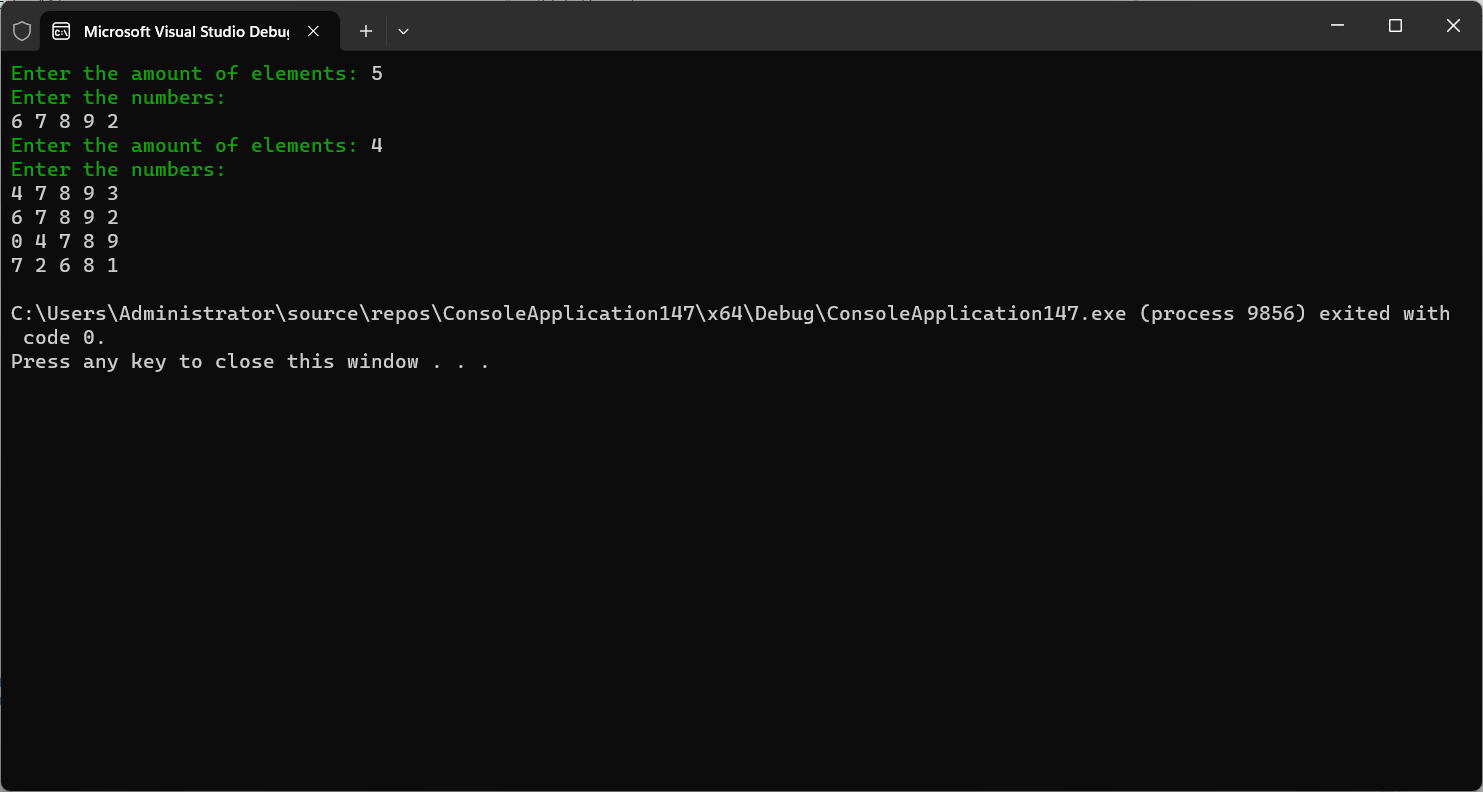
}

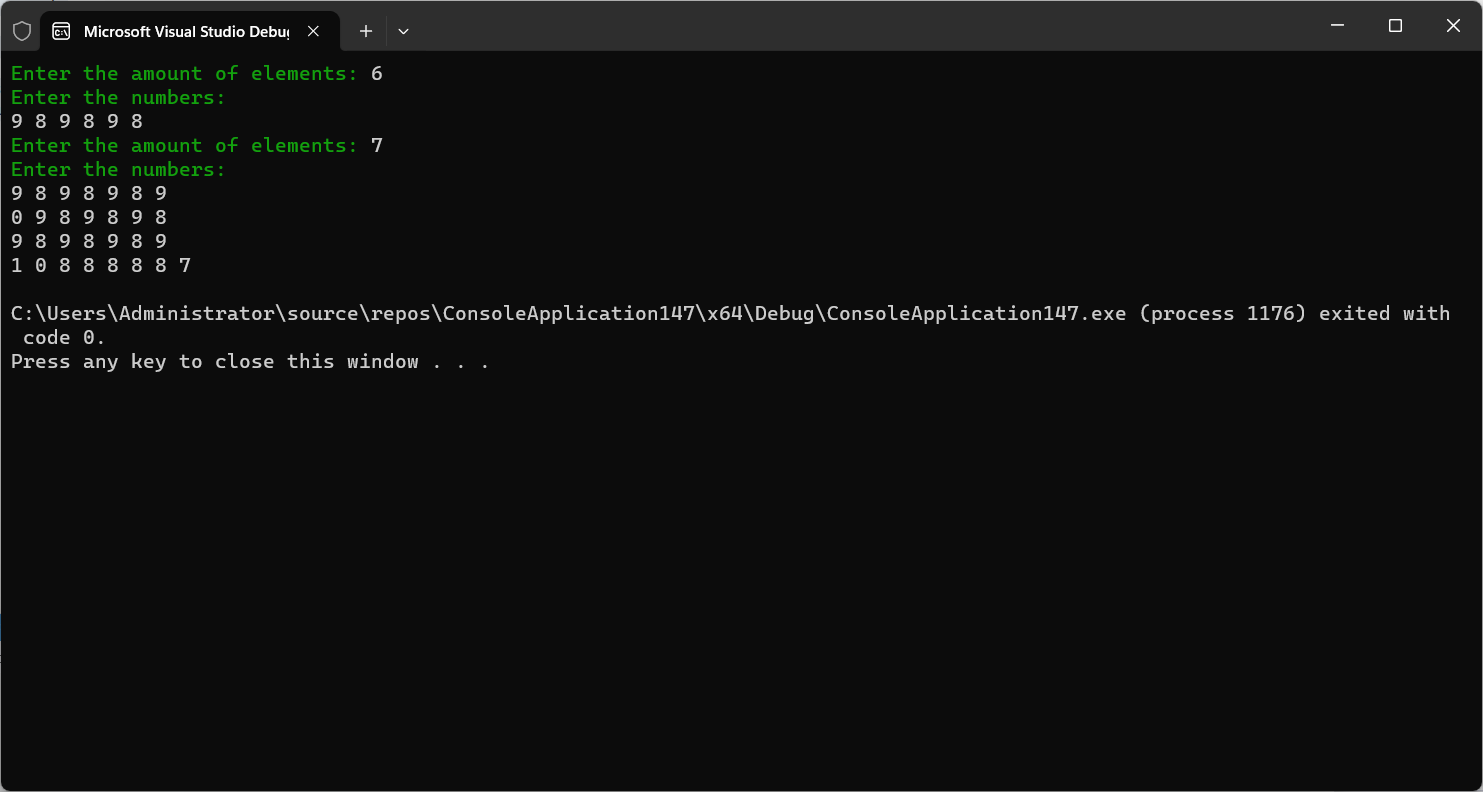
test3.print();

cout << endl;

}

## **3 . Результати виконання завдань, тестування та фактично затрачений час:**





TASK #4

## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

using namespace std;

class Node

{

public:

int key;

Node\* left;

Node\* right;

Node(int item)

{

this->key = item;

this->left = nullptr;

this->right = nullptr;

}

};

class Tree

{

public:

Node\* root;

Tree()

{

this->root = nullptr;

}

private:

Node\* insert(Node\* node, int num)

{

if (node == nullptr)

{

return new Node(num);

}

if (num < node->key)

{

node->left = insert(node->left, num);

}

else if (num > node->key)

{

node->right = insert(node->right, num);

}

return node;

}

void inorderbypass(Node\* node)

{

if (node != nullptr)

{

inorderbypass(node->left);

cout << node->key << " ";

inorderbypass(node->right);

}

}

void preorderbypass(Node\* node)

{

if (node != nullptr)

{

cout << node->key << " ";

preorderbypass(node->left);

preorderbypass(node->right);

}

}

public:

void insert(int num)

{

root = insert(root, num);

}

void inorderbypass()

{

inorderbypass(root);

}

void preorderbypass()

{

preorderbypass(root);

}

void reflect(Node\* root)

{

if (root == nullptr)

{

return;

}

Node\* temporary;

temporary = root->left;

root->left = root->right;

root->right = temporary;

reflect(root->left);

reflect(root->right);

}

};

Tree reflecttree(Tree input)

{

input.reflect(input.root);

return input;

}

int main()

{

Tree test;

test.insert(5);

test.insert(3);

test.insert(6);

test.insert(7);

test.insert(9);

test.insert(8);

test.insert(11);

test.insert(2);

cout << "Output in depth" << endl;

test.inorderbypass();

cout << endl;

cout << "Output in width" << endl;

test.preorderbypass();

cout << endl;

Tree test2 = reflecttree(test);

cout << "Output in depth after reflecting" << endl;

test2.inorderbypass();

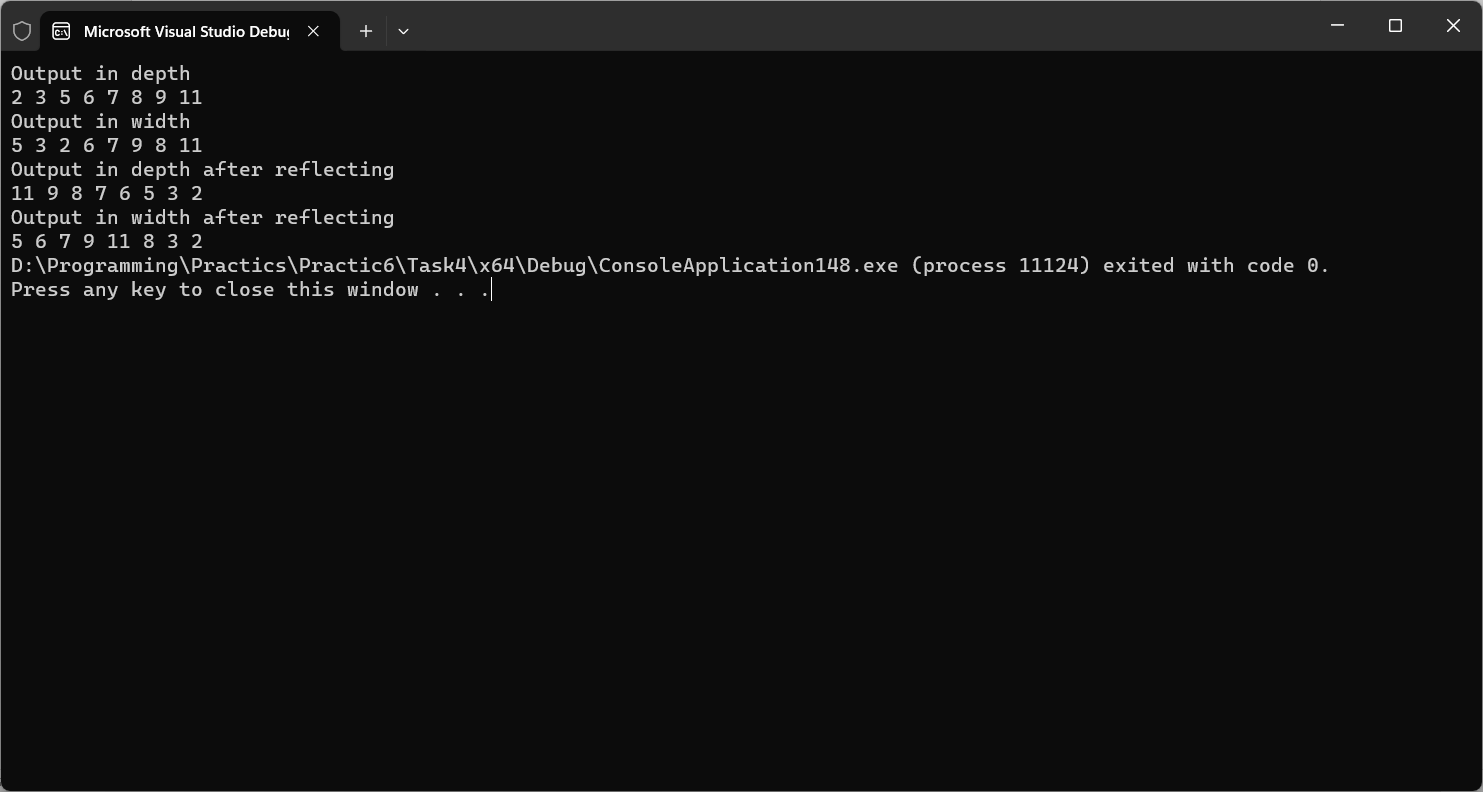
cout << endl;

cout << "Output in width after reflecting" << endl;

test2.preorderbypass();

}

## **3 . Результати виконання завдань, тестування та фактично затрачений час:**



## **Task #5**

## **2. Код програм з посиланням на зовнішні ресурси:**

#include <iostream>

using namespace std;

#include <iostream>

using namespace std;

class Node

{

public:

int key;

Node\* left;

Node\* right;

Node(int item)

{

this->key = item;

this->left = nullptr;

this->right = nullptr;

}

};

class Tree

{

public:

Node\* root;

Tree()

{

this->root = nullptr;

}

private:

Node\* insert(Node\* node, int num)

{

if (node == nullptr)

{

return new Node(num);

}

if (num < node->key)

{

node->left = insert(node->left, num);

}

else if (num > node->key)

{

node->right = insert(node->right, num);

}

return node;

}

void inorderbypass(Node\* node)

{

if (node != nullptr)

{

inorderbypass(node->left);

cout << node->key << " ";

inorderbypass(node->right);

}

}

void preorderbypass(Node\* node)

{

if (node != nullptr)

{

cout << node->key << " ";

preorderbypass(node->left);

preorderbypass(node->right);

}

}

public:

void insert(int num)

{

root = insert(root, num);

}

void inorderbypass()

{

inorderbypass(root);

}

void preorderbypass()

{

preorderbypass(root);

}

void reflect(Node\* root)

{

if (root == nullptr)

{

return;

}

Node\* temporary;

temporary = root->left;

root->left = root->right;

root->right = temporary;

reflect(root->left);

reflect(root->right);

}

void replaceparent(Node\* node)

{

if (node != nullptr)

{

replaceparent(node->left);

replaceparent(node->right);

int leftsum = 3;

int rightsum = 3;

if (node->left != nullptr)

{

leftsum = node->left->key;

}

else

{

leftsum = 0;

}

if (node->right != nullptr)

{

rightsum = node->right->key;

}

else

{

rightsum = 0;

}

node->key = leftsum + rightsum + node->key;

}

}

void sumreplace()

{

replaceparent(root);

}

};

int main()

{

Tree test;

test.insert(5);

test.insert(3);

test.insert(6);

test.insert(7);

test.insert(9);

test.insert(8);

test.insert(11);

test.insert(2);

cout << "Output in depth" << endl;

test.inorderbypass();

cout << endl;

cout << "Output in width" << endl;

test.preorderbypass();

cout << endl;

test.sumreplace();

cout << "Output in depth after modificating" << endl;

test.inorderbypass();

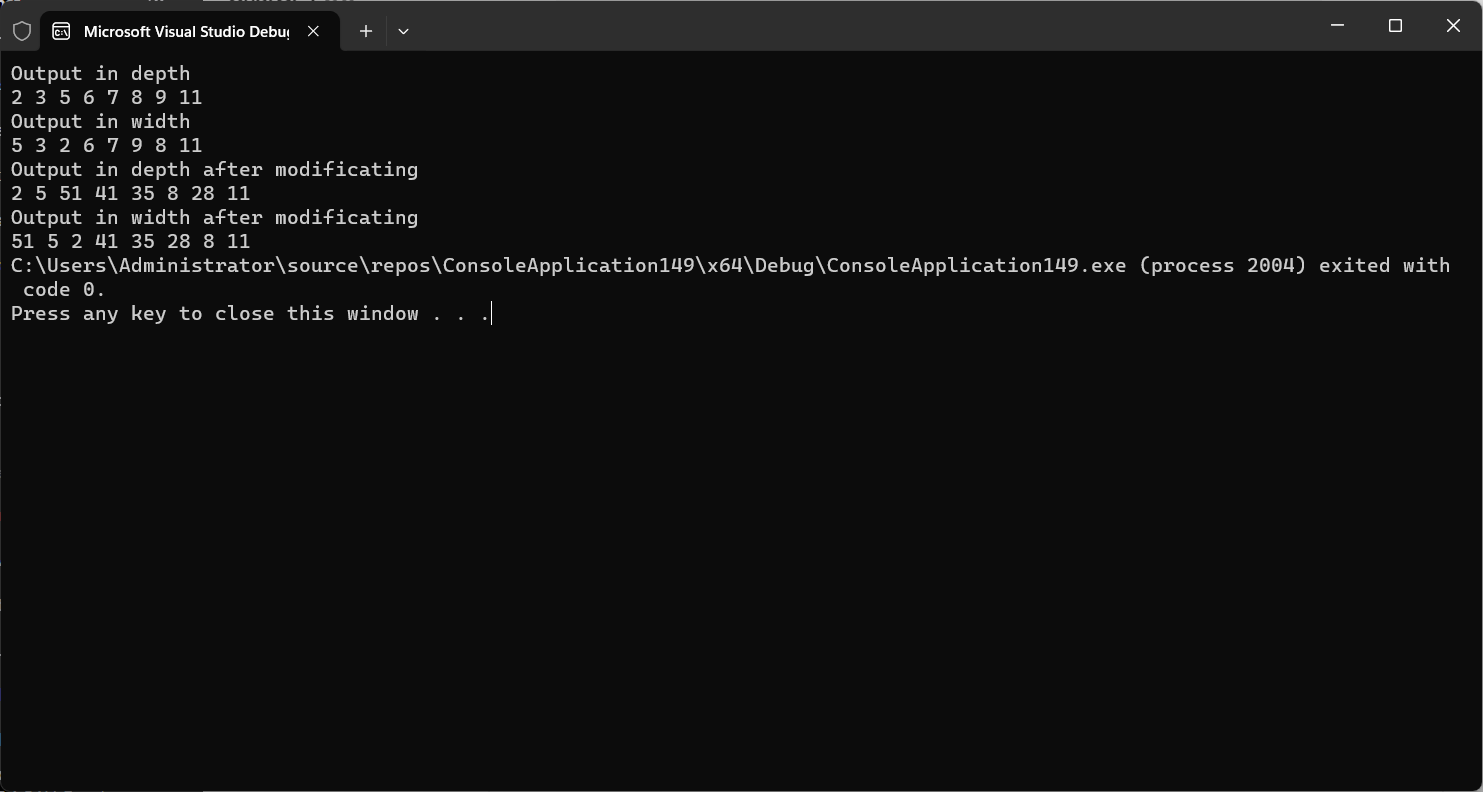
cout << endl;

cout << "Output in width after modificating" << endl;

test.preorderbypass();

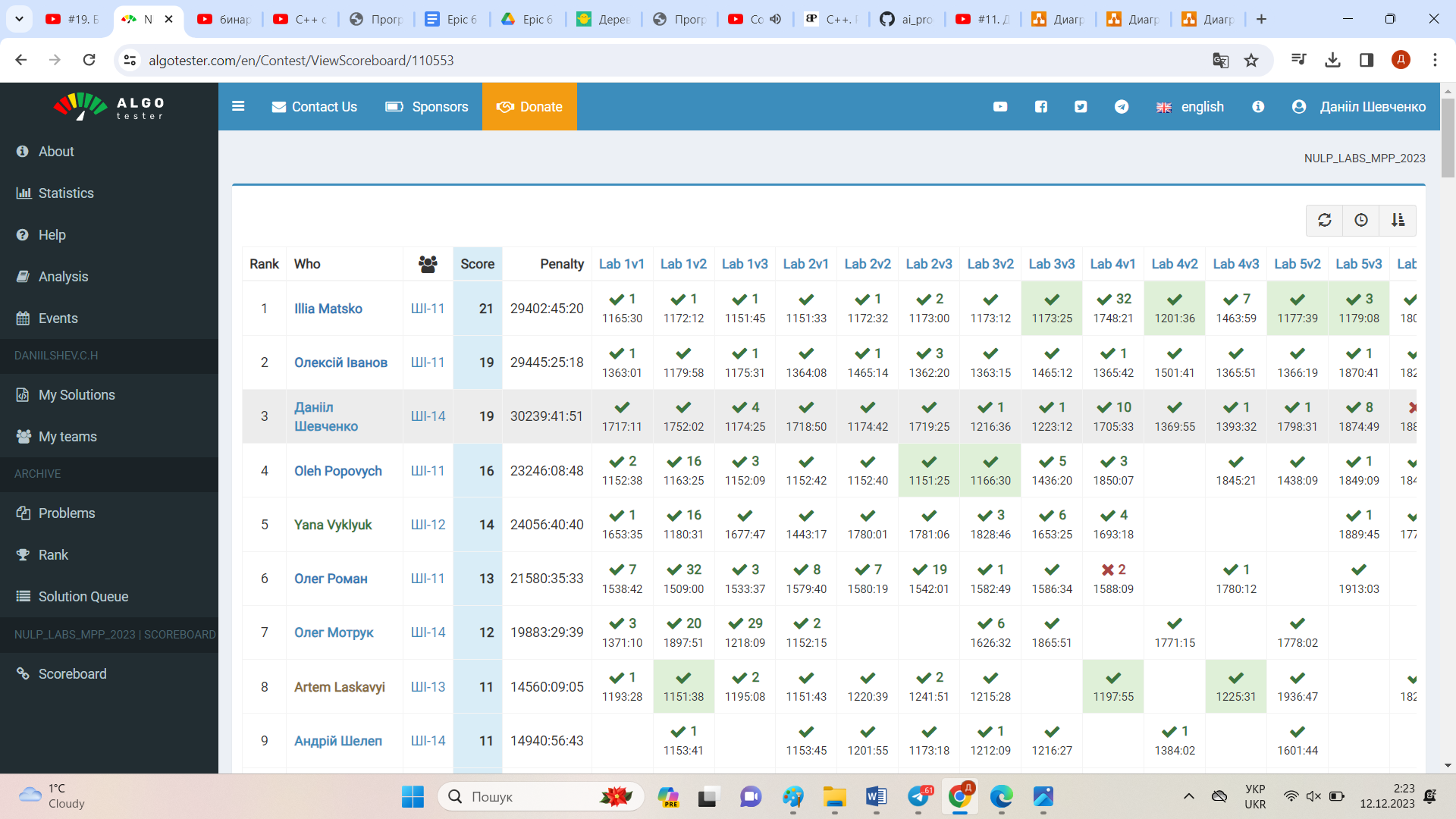
}

## **3 . Результати виконання завдань, тестування та фактично затрачений час:**



# **Висновки:**

Отже, в ході роботи над епіком 6 я отримав початкові уявлення про динамічні структури даних, зокрема стек, чергу, хіп(купу) та 4 види зв’язних списків. Відбулось перше ознайомлення з об’єктно-орієнтованим програмуванням та його елементами: класами, об’єктами, методами. Я отримав уявлення про конструктори та деструктори, геттери та сеттери, специфікатори доступу публічності та приватизації. За допомогою методів ООП спробував реалізувати 2 види лінійних списків. Також було ознайомлено з поняттям перевантаження операторів.



Pull-Request: https://github.com/artificial-intelligence-department/ai\_programming\_playground/pull/824