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:*** «Основи програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт до блоку № 6

**Виконала:**

Студентка групи ШІ-13

Осінна Єлизавета Сергіївна

# **Тема роботи:**

“Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.”

# **Мета роботи:**

# Знайомство з динамічними інформаційними структурами на прикладі одно- і двонаправлених списків. Розуміння структур даних, розвиток алгоритмічного мислення, засвоїти механізми маніпуляції з покажчиками, розвинути навички розв’язувати задачі. Розуміння рівності в структурах даних, поглиблення розуміння зв’язаних списків, розуміння ефективності алгоритму, розвинути базові навики роботи з реальними програмами.

# **Теоретичні відомості:**

1. Основи Динамічних Структур Даних:
   * Вступ до динамічних структур даних: визначення та важливість
   * Виділення пам'яті для структур даних (stack і heap)
   * Приклади простих динамічних структур: динамічний масив
2. Стек:
   * Визначення та властивості стеку
   * Операції push, pop, top: реалізація та використання
   * Приклади використання стеку: обернений польський запис, перевірка балансу дужок
   * Переповнення стеку
3. Черга:
   * Визначення та властивості черги
   * Операції enqueue, dequeue, front: реалізація та застосування
   * Приклади використання черги: обробка подій, алгоритми планування
   * Розширення функціоналу черги: пріоритетні черги
4. Зв'язні Списки:
   * Визначення однозв'язного та двозв'язного списку
   * Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
   * Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
   * Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
5. Дерева:
   * Вступ до структури даних "дерево": визначення, типи
   * Бінарні дерева: вставка, пошук, видалення
   * Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
   * Застосування дерев: дерева рішень, хеш-таблиці
   * Складніші приклади дерев: AVL, Червоно-чорне дерево
6. Алгоритми Обробки Динамічних Структур:
   * Основи алгоритмічних патернів: ітеративні, рекурсивні
   * Алгоритми пошуку, сортування даних, додавання та видалення елементів

* Індивідуальний план опрацювання теорії:

### [C++ ⦁ Теорія ⦁ Урок 140 ⦁ ADT ⦁ Двозв'язний список](https://www.youtube.com/watch?v=QLzu2-_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=140&pp=iAQB)

### [C++ ⦁ Теорія ⦁ Урок 139 ⦁ ADT ⦁ Однозв'язний список](https://www.youtube.com/watch?v=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=139&pp=iAQB)

### [C++ ⦁ Теорія ⦁ Урок 141 ⦁ ADT ⦁ Стек](https://www.youtube.com/watch?v=ZYvYISxaNL0&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141&pp=iAQB)

### [C++ ⦁ Теорія ⦁ Урок 142 ⦁ ADT ⦁ Черга](https://www.youtube.com/watch?v=Yhw8NbjrSFA&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=142&pp=iAQB)

### [C++ ⦁ Теорія ⦁ Урок 143 ⦁ initializer\_list](https://www.youtube.com/watch?v=_LdFE_4L6gI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=143&pp=iAQB)

### [C++ ⦁ Теорія ⦁ Урок 144 ⦁ ADT ⦁ Бінарне дерево](https://www.youtube.com/watch?v=qBFzNW0ALxQ&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=144&pp=iAQB)

# **Виконання роботи:**

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

**Завдання № 1 VNS Lab 10**

* Варіант 5
* Деталі завдання:

Написати програму, у якій створюються динамічні структури й виконати

їхню обробку у відповідності зі своїм варіантом.

Для кожного варіанту розробити такі функції:

1. Створення списку.

2. Додавання елемента в список (у відповідності зі своїм варіантом).

3. Знищення елемента зі списку (у відповідності зі своїм варіантом).

4. Друк списку.

5. Запис списку у файл.

6. Знищення списку.

7. Відновлення списку з файлу.

Порядок виконання роботи

1. Написати функцію для створення списку. Функція може створювати

порожній список, а потім додавати в нього елементи.

2. Написати функцію для друку списку. Функція повинна передбачати вивід

повідомлення, якщо список порожній.

3. Написати функції для знищення й додавання елементів списку у

відповідності зі своїм варіантом.

4. Виконати зміни в списку й друк списку після кожної зміни.

5. Написати функцію для запису списку у файл.

6. Написати функцію для знищення списку.

7. Записати список у файл, знищити його й виконати друк (при друці повинне

бути видане повідомлення "Список порожній").

8. Написати функцію для відновлення списку з файлу.

9. Відновити список і роздрукувати його.

10.Знищити список.

Варіант 5. Записи в лінійному списку містять ключове поле типу int. Сформувати

однонаправлений список. Знищити з нього К елементів, починаючи із

заданого номера, додати К елементів, починаючи із заданого номера;

**Завдання** **№ 2 Algotester Lab 5**

* Варіант 2
* Деталі завдання:

В пустелi iснує незвичайна печера, яка є двохвимiрною. Її висота це N, ширина - M.

Всерединi печери є пустота, пiсок та камiння. Пустота позначається буквою , пiсок S i камiння

X;

Одного дня стався землетрус i весь пiсок посипався вниз. Вiн падає на найнижчу клiтинку з

пустотою, але вiн не може пролетiти через камiння.

Ваше завдання сказати як буде виглядати печера пiсля землетрусу.

Вхiднi данi

У першому рядку 2 цiлих числа N та M - висота та ширина печери

У N наступних рядках стрiчка rowi яка складається з N цифер - i-й рядок матрицi, яка

вiдображає стан печери до землетрусу.

Вихiднi данi

N рядкiв, якi складаються з стрiчки розмiром M - стан печери пiсля землетрусу.

Обмеження

1 ≤ N, M ≤ 1000

|rowi| = M

rowi ∈ {X, S, O}

**Завдання № 3 Algotester Lab 7-8**

* Варіант 1
* Деталі завдання:

Ваше завдання - власноруч реалiзувати структуру даних "Двозв’язний список".

Ви отримаєте Q запитiв, кожен запит буде починатися зi слова-iдентифiкатора, пiсля якого

йдуть його аргументи.

Вам будуть поступати запити такого типу:

• Вставка:

Iдентифiкатор - insert

Ви отримуєте цiле число index елемента, на мiсце якого робити вставку.

Пiсля цього в наступному рядку рядку написане число N - розмiр списку, який треба вста-

вити.

У третьому рядку N цiлих чисел - список, який треба вставити на позицiю index.

• Видалення:

Iдентифiкатор - erase

Ви отримуєте 2 цiлих числа - index, iндекс елемента, з якого почати видалення та n -

кiлькiсть елементiв, яку треба видалити.

• Визначення розмiру:

Iдентифiкатор - size

Ви не отримуєте аргументiв.

Ви виводите кiлькiсть елементiв у списку.

• Отримання значення i-го елементу

Iдентифiкатор - get

Ви отримуєте цiле число - index, iндекс елемента.

Ви виводите значення елемента за iндексом.

• Модифiкацiя значення i-го елементу

Iдентифiкатор - set

Ви отримуєте 2 цiлих числа - iндекс елемента, який треба змiнити, та його нове значення.

• Вивiд списку на екран

Iдентифiкатор - print

Ви не отримуєте аргументiв.

Ви виводите усi елементи списку через пробiл.

Реалiзувати використовуючи перегрузку оператора <<

Вхiднi данi

Цiле число Q - кiлькiсть запитiв.

У наступних рядках Q запитiв у зазначеному в умовi форматi.

Вихiднi данi

Вiдповiдi на запити у зазначеному в умовi форматi.

Обмеження

0 ≤ Q ≤ 103

0 ≤ li ≤ 103

|l| ≤ 103

**Завдання № 4 Algotester Lab 7-8**

* Варіант 3
* Деталі завдання:

Ваше завдання - власноруч реалiзувати структуру даних "Двiйкове дерево пошуку".

Ви отримаєте Q запитiв, кожен запит буде починатися зi слова-iдентифiкатора, пiсля якого

йдуть його параметри.

Вам будуть поступати запити такого типу:

• Вставка:

Iдентифiкатор - insert

Ви отримуєте цiле число value - число, яке треба вставити в дерево.

• Пошук:

Iдентифiкатор - contains

Ви отримуєте цiле число value - число, наявнiсть якого у деревi необхiдно перевiрити.

Якщо value наявне в деревi - ви виводите Y es, у iншому випадку No.

• Визначення розмiру:

Iдентифiкатор - size

Ви не отримуєте аргументiв.

Ви виводите кiлькiсть елементiв у деревi.

• Вивiд дерева на екран

Iдентифiкатор - print

Ви не отримуєте аргументiв.

Ви виводите усi елементи дерева через пробiл.

Реалiзувати використовуючи перегрузку оператора <<

Вхiднi данi

Цiле число Q - кiлькiсть запитiв.

У наступних рядках Q запитiв у зазначеному в умовi форматi.

Вихiднi данi

Вiдповiдi на запити у зазначеному в умовi форматi.

Обмеження

0 ≤ Q ≤ 103

0 ≤ ti ≤ 103

**Завдання № 5 Class Practice Work**

* Деталі завдання:

## 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;

## 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***.

## 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.

## 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

## 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

**Завдання № 6 Self Practice Work**

* Деталі завдання:

# Lab 5v3

У вас є карта гори розміром N×M.

Також ви знаєте координати {x,y}, у яких знаходиться вершина гори.

Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

# Input

У першому рядку 2 числа N та M - розміри карти

у другому рядку 2 числа x та y - координати піку гори

# Output

N рядків по M елементів в рядку через пробіл - висоти карти.

# Constraints

1≤N,M≤103

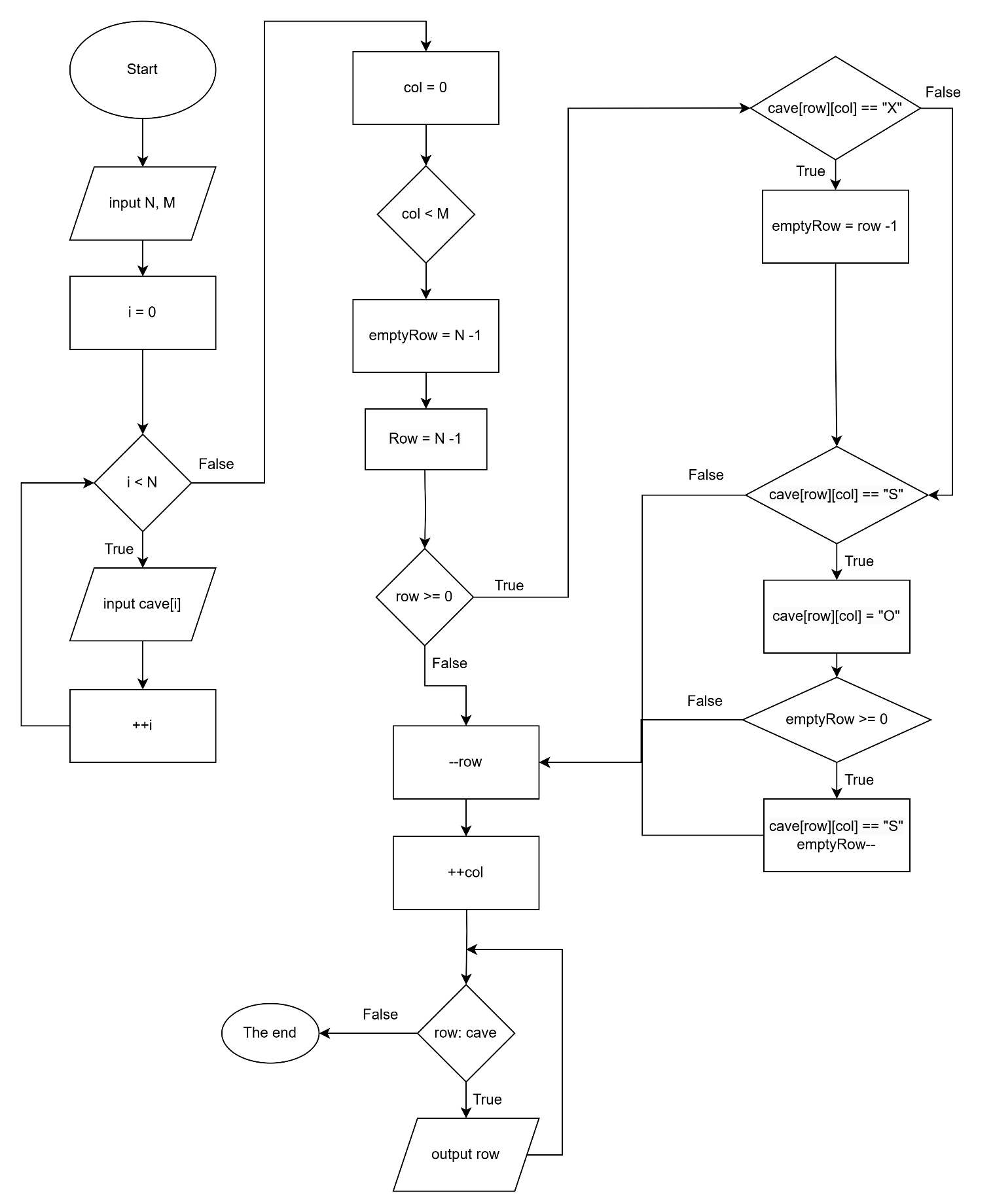
1≤x≤N

1≤y≤M

## **2. Дизайн та планована оцінка часу виконання завдань:**

Програма № 2 Algotester Lab 5 Варіант 2

* Блок-схема



* Планований час на реалізацію кожної програми – 40-50 хв

## **4. Код програм:**

Завдання № 1 VNS Lab 10

#include <iostream>

#include <fstream>

#include <memory>

using namespace std;

struct Node {

    int key;

    shared\_ptr<Node> next;

};

class LinkedList {

private:

    shared\_ptr<Node> head;

public:

    LinkedList() : head(nullptr) {}

    // 1. Додавання елемента в список

    void addElement(int key, int position = -1) {

        auto newNode = make\_shared<Node>();

        newNode->key = key;

        newNode->next = nullptr;

        if (!head || position == 0) {

            newNode->next = head;

            head = newNode;

            return;

        }

        auto current = head;

        int index = 0;

        while (current->next && (position == -1 || index < position - 1)) {

            current = current->next;

            index++;

        }

        newNode->next = current->next;

        current->next = newNode;

    }

    // 2. Видалення K елементів із заданого номера

    void deleteElements(int position, int k) {

        if (!head) return;

        auto current = head;

        shared\_ptr<Node> prev = nullptr;

        for (int i = 0; i < position && current; ++i) {

            prev = current;

            current = current->next;

        }

        for (int i = 0; i < k && current; ++i) {

            if (prev) prev->next = current->next;

            else head = current->next;

            current = current->next;

        }

    }

    // 3. Друк списку

    void printList() const {

        if (!head) {

            cout << "Список порожній" << endl;

            return;

        }

        auto current = head;

        while (current) {

            cout << current->key << " ";

            current = current->next;

        }

        cout << endl;

    }

    // 4. Запис списку у файл

    void saveToFile(const string& filename) const {

        ofstream file(filename);

        if (!file) {

            cerr << "Помилка відкриття файлу для запису!" << endl;

            return;

        }

        auto current = head;

        while (current) {

            file << current->key << " ";

            current = current->next;

        }

    }

    // 5. Відновлення списку з файлу

    void loadFromFile(const string& filename) {

        ifstream file(filename);

        if (!file) {

            cerr << "Помилка відкриття файлу для читання!" << endl;

            return;

        }

        head = nullptr;

        int key;

        while (file >> key) {

            addElement(key);

        }

    }

    // 6. Знищення списку

    void clearList() {

        head = nullptr;

    }

};

int main() {

    LinkedList list;

    // 1. Створення списку

    cout << "Створення списку:" << endl;

    list.addElement(10);

    list.addElement(20);

    list.addElement(30);

    list.addElement(40);

    list.printList();

    // 2. Додавання елемента у список

    cout << "Додавання елемента в список:" << endl;

    list.addElement(25, 2); // Додаємо 25 на позицію 2

    list.printList();

    // 3. Видалення елементів зі списку

    cout << "Видалення 2 елементів, починаючи з позиції 1:" << endl;

    list.deleteElements(1, 2);

    list.printList();

    // 4. Запис у файл

    cout << "Запис списку у файл 'list.txt':" << endl;

    list.saveToFile("list.txt");

    // 5. Знищення списку

    cout << "Знищення списку:" << endl;

    list.clearList();

    list.printList();

    // 6. Відновлення списку з файлу

    cout << "Відновлення списку з файлу 'list.txt':" << endl;

    list.loadFromFile("list.txt");

    list.printList();

    // 7. Знищення списку остаточно

    cout << "Остаточне знищення списку:" << endl;

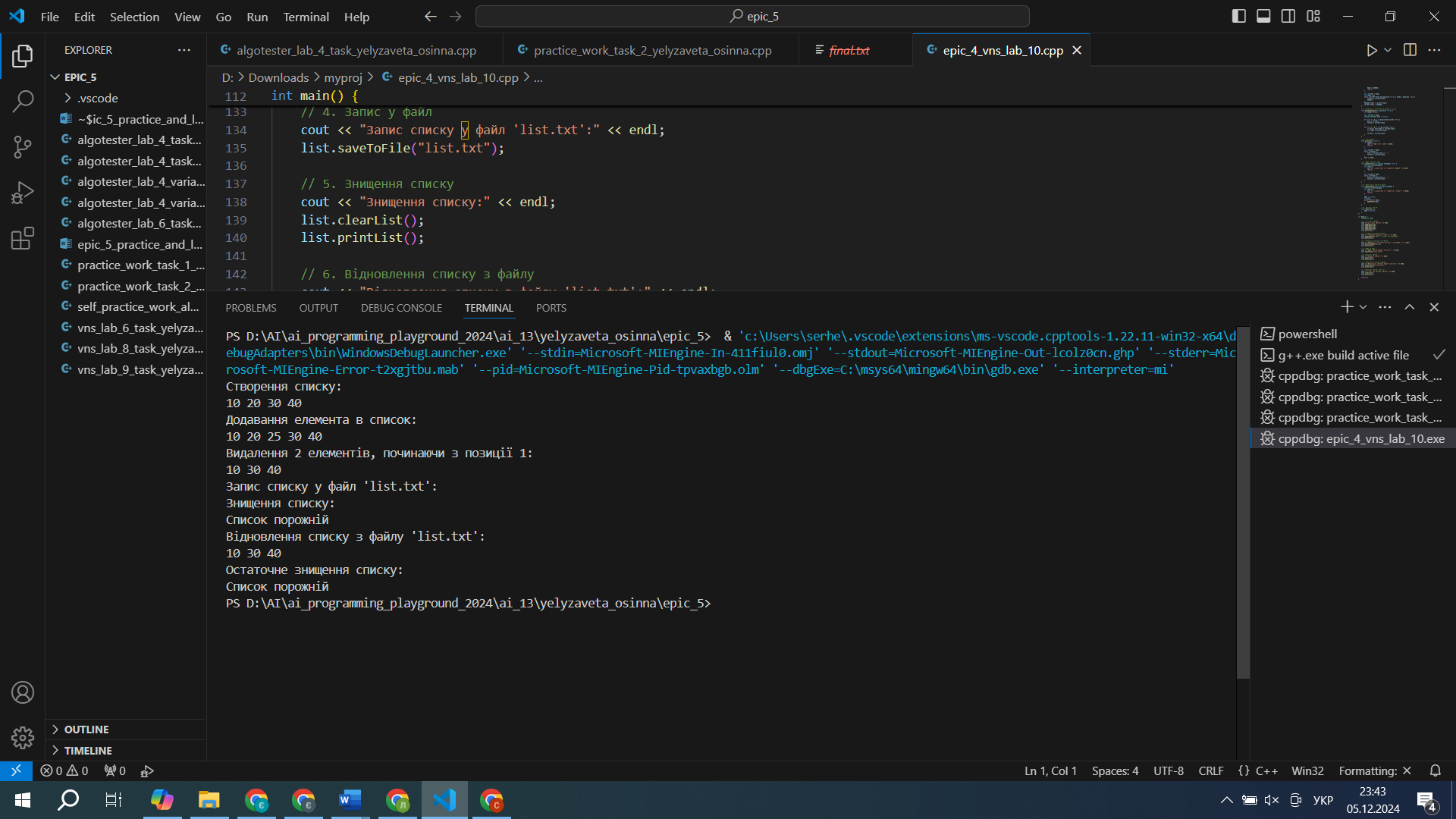
    list.clearList();

    list.printList();

    return 0;

}

## **Result:**



Завдання № 2 Algotester Lab 5

#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main() {

    int N, M;

    cin >> N >> M;

    vector<string> cave(N); // Матриця печери

    for (int i = 0; i < N; ++i) {

        cin >> cave[i];

    }

    // Симуляція падіння піску

    for (int col = 0; col < M; ++col) {

        int emptyRow = N - 1; // Починаємо з найнижчого рядка

        for (int row = N - 1; row >= 0; --row) {

            if (cave[row][col] == 'X') {

                // Камінь блокує падіння піску

                emptyRow = row - 1;

            } else if (cave[row][col] == 'S') {

                // Пісок падає на рівень пустоти

                cave[row][col] = 'O'; // Очищаємо поточну позицію

                if (emptyRow >= 0) {

                    cave[emptyRow][col] = 'S';

                    emptyRow--;

                }

            }

        }

    }

    cout << endl;

    // Вивід результату

    for (const string& row : cave) {

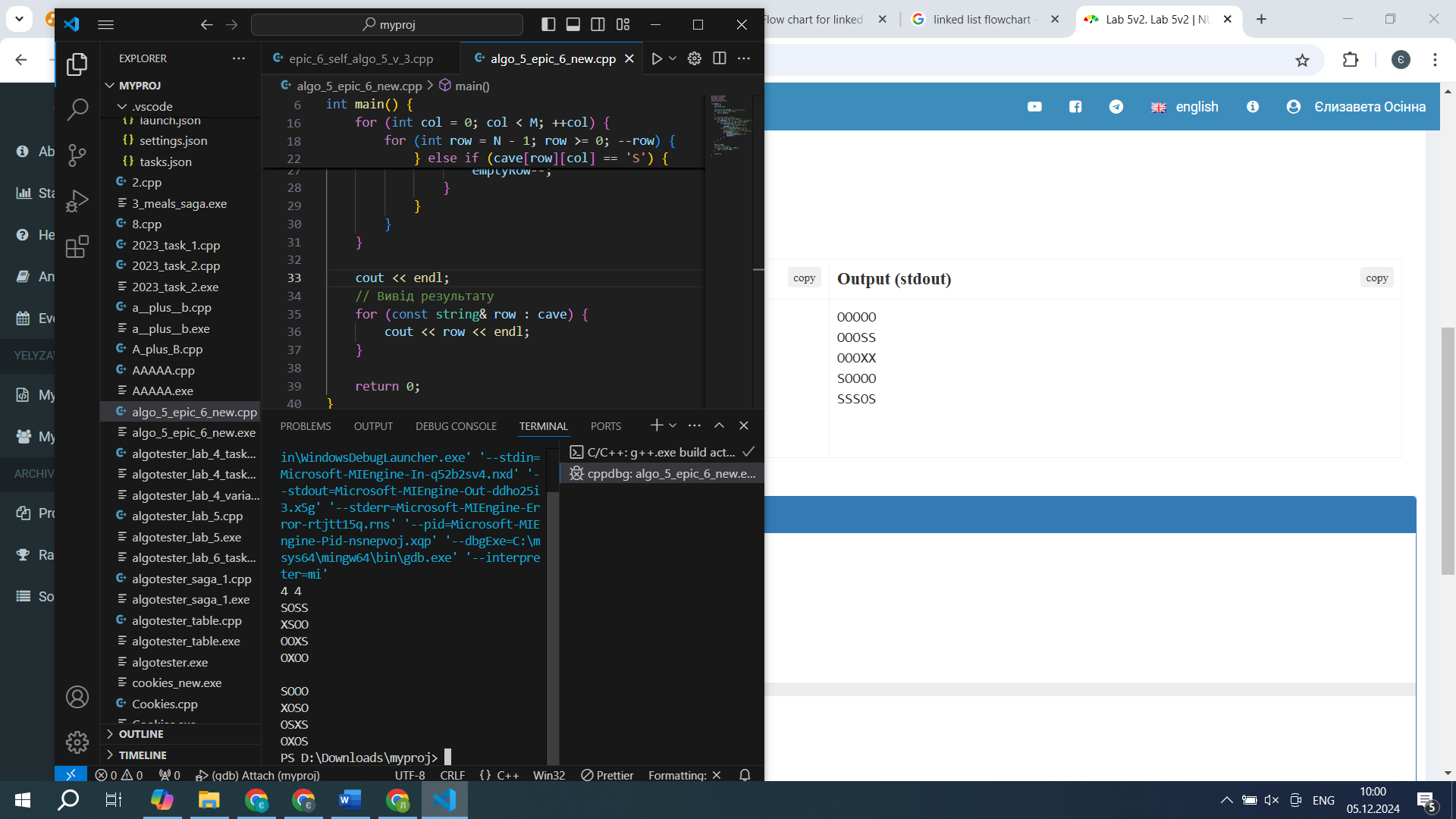
        cout << row << endl;

    }

    return 0;

}

## **Result:**



Завдання № 3 Algotester Lab 7-8 V1

#include <iostream>

#include <string>

#include <stdexcept>

#include <vector>

#include <sstream>

using namespace std;

// Структура вузла двозв'язного списку

struct Node {

    int value;

    Node\* prev;

    Node\* next;

    Node(int val) : value(val), prev(nullptr), next(nullptr) {}

};

// Клас двозв'язного списку

class DoublyLinkedList {

private:

    Node\* head;

    Node\* tail;

    size\_t listSize;

    // Отримання вузла за індексом

    Node\* getNodeAt(int index) const {

        if (index < 0 || index >= listSize) {

            throw out\_of\_range("Index out of range");

        }

        Node\* current;

        if (index < listSize / 2) {

            current = head;

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

                current = current->next;

            }

        } else {

            current = tail;

            for (int i = listSize - 1; i > index; --i) {

                current = current->prev;

            }

        }

        return current;

    }

public:

    DoublyLinkedList() : head(nullptr), tail(nullptr), listSize(0) {}

    // Вставка на позицію index

    void insert(int index, int n, const int\* values) {

        if (index < 0 || index > listSize) {

            throw out\_of\_range("Index out of range");

        }

        for (int i = 0; i < n; ++i) {

            Node\* newNode = new Node(values[i]);

            if (index == 0) {

                // Вставка на початок

                newNode->next = head;

                if (head) head->prev = newNode;

                head = newNode;

                if (!tail) tail = newNode;

            } else if (index == listSize) {

                // Вставка в кінець

                newNode->prev = tail;

                if (tail) tail->next = newNode;

                tail = newNode;

            } else {

                // Вставка в середину

                Node\* nextNode = getNodeAt(index);

                Node\* prevNode = nextNode->prev;

                newNode->next = nextNode;

                newNode->prev = prevNode;

                prevNode->next = newNode;

                nextNode->prev = newNode;

            }

            index++;

            listSize++;

        }

    }

    // Видалення n елементів з позиції index

    void erase(int index, int n) {

        if (index < 0 || index >= listSize || n <= 0) {

            throw out\_of\_range("Invalid arguments for erase");

        }

        for (int i = 0; i < n; ++i) {

            if (index >= listSize) break;

            Node\* target = getNodeAt(index);

            if (target->prev) target->prev->next = target->next;

            if (target->next) target->next->prev = target->prev;

            if (target == head) head = target->next;

            if (target == tail) tail = target->prev;

            delete target;

            listSize--;

        }

    }

    // Розмір списку

    size\_t size() const {

        return listSize;

    }

    // Отримання значення за індексом

    int get(int index) const {

        return getNodeAt(index)->value;

    }

    // Модифікація значення за індексом

    void set(int index, int value) {

        getNodeAt(index)->value = value;

    }

    // Перевантаження оператора << для виводу списку

    friend ostream& operator<<(ostream& os, const DoublyLinkedList& list) {

        Node\* current = list.head;

        while (current) {

            os << current->value << " ";

            current = current->next;

        }

        return os;

    }

};

int main() {

    DoublyLinkedList list;

    int Q;

    cin >> Q;

    // Буфер для збереження результатів

    ostringstream outputBuffer;

    while (Q--) {

        string command;

        cin >> command;

        if (command == "insert") {

            int index, N;

            cin >> index >> N;

            int\* values = new int[N];

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

                cin >> values[i];

            }

            try {

                list.insert(index, N, values);

            } catch (out\_of\_range& e) {

                outputBuffer << "Error: " << e.what() << endl;

            }

            delete[] values;

        } else if (command == "erase") {

            int index, n;

            cin >> index >> n;

            try {

                list.erase(index, n);

            } catch (out\_of\_range& e) {

                outputBuffer << "Error: " << e.what() << endl;

            }

        } else if (command == "size") {

            outputBuffer << list.size() << endl;

        } else if (command == "get") {

            int index;

            cin >> index;

            try {

                outputBuffer << list.get(index) << endl;

            } catch (out\_of\_range& e) {

                outputBuffer << "Error: " << e.what() << endl;

            }

        } else if (command == "set") {

            int index, value;

            cin >> index >> value;

            try {

                list.set(index, value);

            } catch (out\_of\_range& e) {

                outputBuffer << "Error: " << e.what() << endl;

            }

        } else if (command == "print") {

            outputBuffer << list << endl;

        }

    }

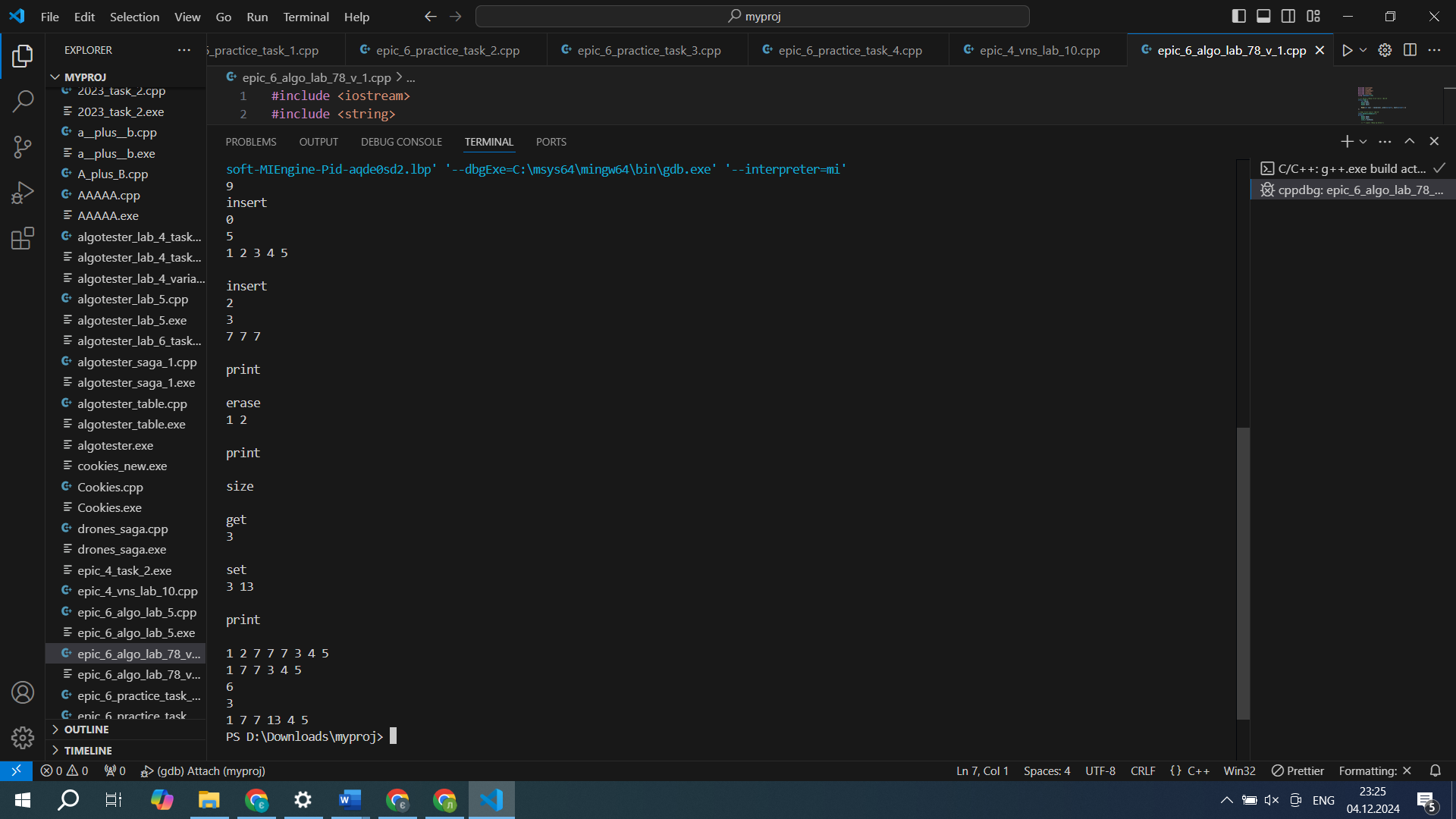
    // Вивід усіх результатів одним блоком

    cout << "\n" << outputBuffer.str();

    return 0;

}

Result:



Час затрачений на виконання завдання

Завдання № 4 Algotester Lab 7-8 V3

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

// Вузол дерева

struct Node {

    int value;

    Node\* left;

    Node\* right;

    Node(int v) : value(v), left(nullptr), right(nullptr) {}

};

// Клас двійкового дерева пошуку

class BinarySearchTree {

private:

    Node\* root;

    int tree\_size;

    // Вставка елемента

    Node\* insert(Node\* node, int value) {

        if (!node) {

            tree\_size++;

            return new Node(value);

        }

        if (value < node->value)

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

        else if (value > node->value)

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

        return node;

    }

    // Перевірка наявності елемента

    bool contains(Node\* node, int value) const {

        if (!node) return false;

        if (node->value == value) return true;

        if (value < node->value)

            return contains(node->left, value);

        return contains(node->right, value);

    }

    // Рекурсивний обхід дерева для виводу

    void inOrderTraversal(Node\* node, ostream& out) const {

        if (!node) return;

        inOrderTraversal(node->left, out);

        out << node->value << " ";

        inOrderTraversal(node->right, out);

    }

public:

    BinarySearchTree() : root(nullptr), tree\_size(0) {}

    void insert(int value) {

        root = insert(root, value);

    }

    bool contains(int value) const {

        return contains(root, value);

    }

    int size() const {

        return tree\_size;

    }

    friend ostream& operator<<(ostream& out, const BinarySearchTree& tree) {

        tree.inOrderTraversal(tree.root, out);

        return out;

    }

};

int main() {

    int Q;

    cin >> Q;

    BinarySearchTree bst;

    ostringstream output; // Буфер для збереження результатів

    while (Q--) {

        string command;

        cin >> command;

        if (command == "insert") {

            int value;

            cin >> value;

            bst.insert(value);

        } else if (command == "contains") {

            int value;

            cin >> value;

            output << (bst.contains(value) ? "Yes" : "No") << endl;

        } else if (command == "size") {

            output << bst.size() << endl;

        } else if (command == "print") {

            output << bst << endl;

        }

    }

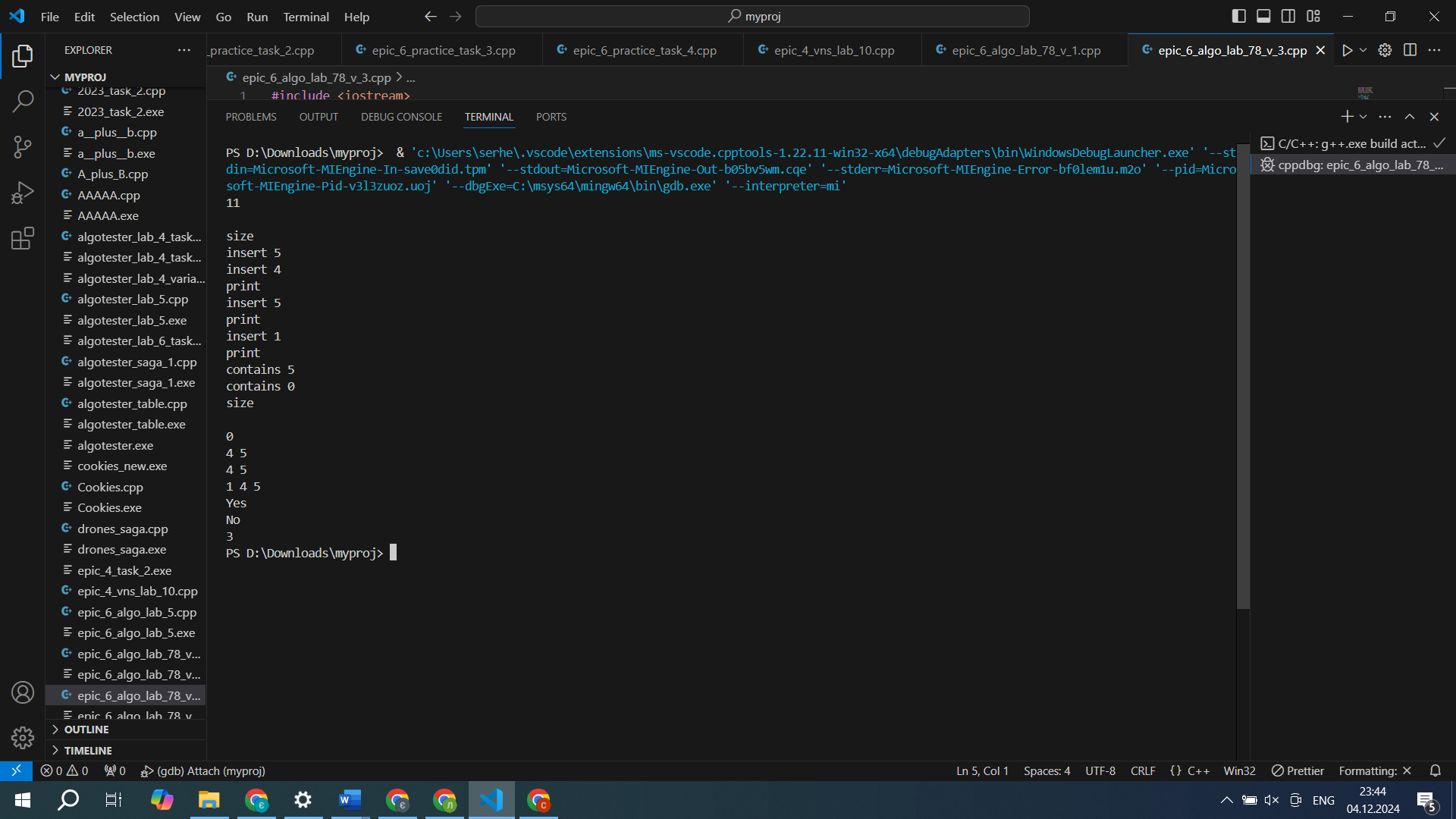
    // Вивід зібраних результатів одним блоком

    cout  << "\n" << output.str();

    return 0;

}

Result:



Завдання № 5 Class Practice Work

#include <iostream>

using namespace std;

// ------------------------------------

// Задача 1 - Реверс списку

// ------------------------------------

struct Node {

    int value;

    Node\* next;

    Node(int v) : value(v), next(nullptr) {}

};

Node\* reverse(Node\* head) {

    Node\* prev = nullptr;

    Node\* current = head;

    while (current) {

        Node\* next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    return prev;

}

void printList(Node\* head) {

    Node\* current = head;

    while (current) {

        cout << current->value << " ";

        current = current->next;

    }

    cout << endl;

}

// ------------------------------------

// Задача 2 - Порівняння списків

// ------------------------------------

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

    while (h1 && h2) {

        if (h1->value != h2->value) return false;

        h1 = h1->next;

        h2 = h2->next;

    }

    return h1 == nullptr && h2 == nullptr;

}

// ------------------------------------

// Задача 3 - Додавання великих чисел

// ------------------------------------

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

    Node\* result = nullptr;

    Node\* tail = nullptr;

    int carry = 0;

    while (n1 || n2 || carry) {

        int sum = carry + (n1 ? n1->value : 0) + (n2 ? n2->value : 0);

        carry = sum / 10;

        Node\* newNode = new Node(sum % 10);

        if (!result) {

            result = tail = newNode;

        } else {

            tail->next = newNode;

            tail = newNode;

        }

        if (n1) n1 = n1->next;

        if (n2) n2 = n2->next;

    }

    return result;

}

// ------------------------------------

// Задача 4 - Віддзеркалення дерева

// ------------------------------------

struct TreeNode {

    int value;

    TreeNode\* left;

    TreeNode\* right;

    TreeNode(int v) : value(v), left(nullptr), right(nullptr) {}

};

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

    if (!root) return nullptr;

    TreeNode\* newRoot = new TreeNode(root->value);

    newRoot->left = create\_mirror\_flip(root->right);

    newRoot->right = create\_mirror\_flip(root->left);

    return newRoot;

}

// ------------------------------------

// Задача 5 - Сума підвузлів

// ------------------------------------

int calculateSum(TreeNode\* root) {

    if (!root) return 0;

    if (!root->left && !root->right) return root->value;

    int leftSum = calculateSum(root->left);

    int rightSum = calculateSum(root->right);

    root->value = leftSum + rightSum;

    return root->value;

}

void tree\_sum(TreeNode\* root) {

    calculateSum(root);

}

// ------------------------------------

// Основна програма

// ------------------------------------

int main() {

    // Задача 1: Реверс списку

    Node\* head = new Node(1);

    head->next = new Node(2);

    head->next->next = new Node(3);

    cout << "Original List: ";

    printList(head);

    Node\* reversed = reverse(head);

    cout << "Reversed List: ";

    printList(reversed);

    // Задача 2: Порівняння списків

    Node\* list1 = new Node(1);

    list1->next = new Node(2);

    Node\* list2 = new Node(1);

    list2->next = new Node(2);

    cout << "Lists are " << (compare(list1, list2) ? "equal" : "not equal") << endl;

    // Задача 3: Додавання великих чисел

    Node\* num1 = new Node(3);

    num1->next = new Node(7);

    num1->next->next = new Node(9);

    Node\* num2 = new Node(9);

    num2->next = new Node(2);

    Node\* sumList = add(num1, num2);

    cout << "Sum List: ";

    printList(sumList);

    // Задача 4: Віддзеркалення дерева

    TreeNode\* root = new TreeNode(1);

    root->left = new TreeNode(2);

    root->right = new TreeNode(3);

    root->left->left = new TreeNode(4);

    root->left->right = new TreeNode(5);

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

    cout << "Original Tree Root: " << root->value << ", Mirrored Tree Root: " << mirrored->value << endl;

    // Задача 5: Сума підвузлів

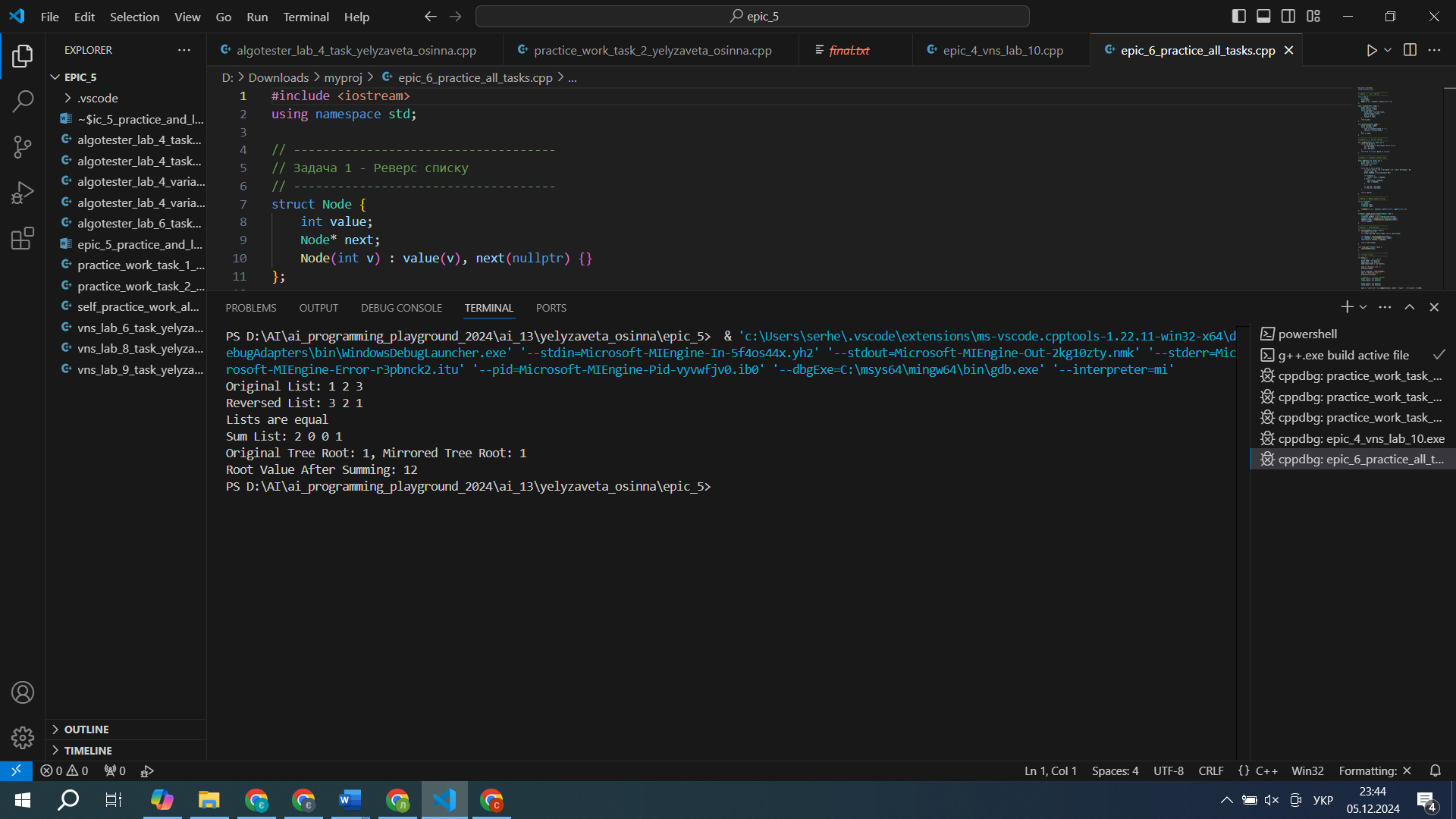
    tree\_sum(root);

    cout << "Root Value After Summing: " << root->value << endl;

    return 0;

}

## **Result:**



Завдання № 6 Self Practice Work

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int main() {

    int N, M, x, y;

    cin >> N >> M;

    cin >> x >> y;

    // Обчислення максимальної висоти (відстані до найвіддаленішого кута)

    int max\_height = 0;

    if (x > N / 2) {

        max\_height += x - 1;  // Відстань до верхнього або нижнього краю

    } else {

        max\_height += N - x;

    }

    if (y > M / 2) {

        max\_height += y - 1;  // Відстань до лівого або правого краю

    } else {

        max\_height += M - y;

    }

    // Матриця для висот

    vector<vector<int>> map(N, vector<int>(M, -1));

    // BFS

    queue<pair<int, int>> q;

    map[x - 1][y - 1] = max\_height; // Встановлюємо висоту вершини

    q.push({x - 1, y - 1});

    // Напрямки руху: вверх, вниз, вліво, вправо

    int dx[] = {-1, 1, 0, 0};

    int dy[] = {0, 0, -1, 1};

    while (!q.empty()) {

        auto [cx, cy] = q.front();

        q.pop();

        for (int i = 0; i < 4; i++) {

            int nx = cx + dx[i];

            int ny = cy + dy[i];

            // Перевірка меж карти

            if (nx >= 0 && nx < N && ny >= 0 && ny < M && map[nx][ny] == -1) {

                map[nx][ny] = map[cx][cy] - 1; // Висота на 1 менша

                q.push({nx, ny});

            }

        }

    }

    // Вивід результату

    cout << endl;

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < M; j++) {

            cout << map[i][j] << " ";

        }

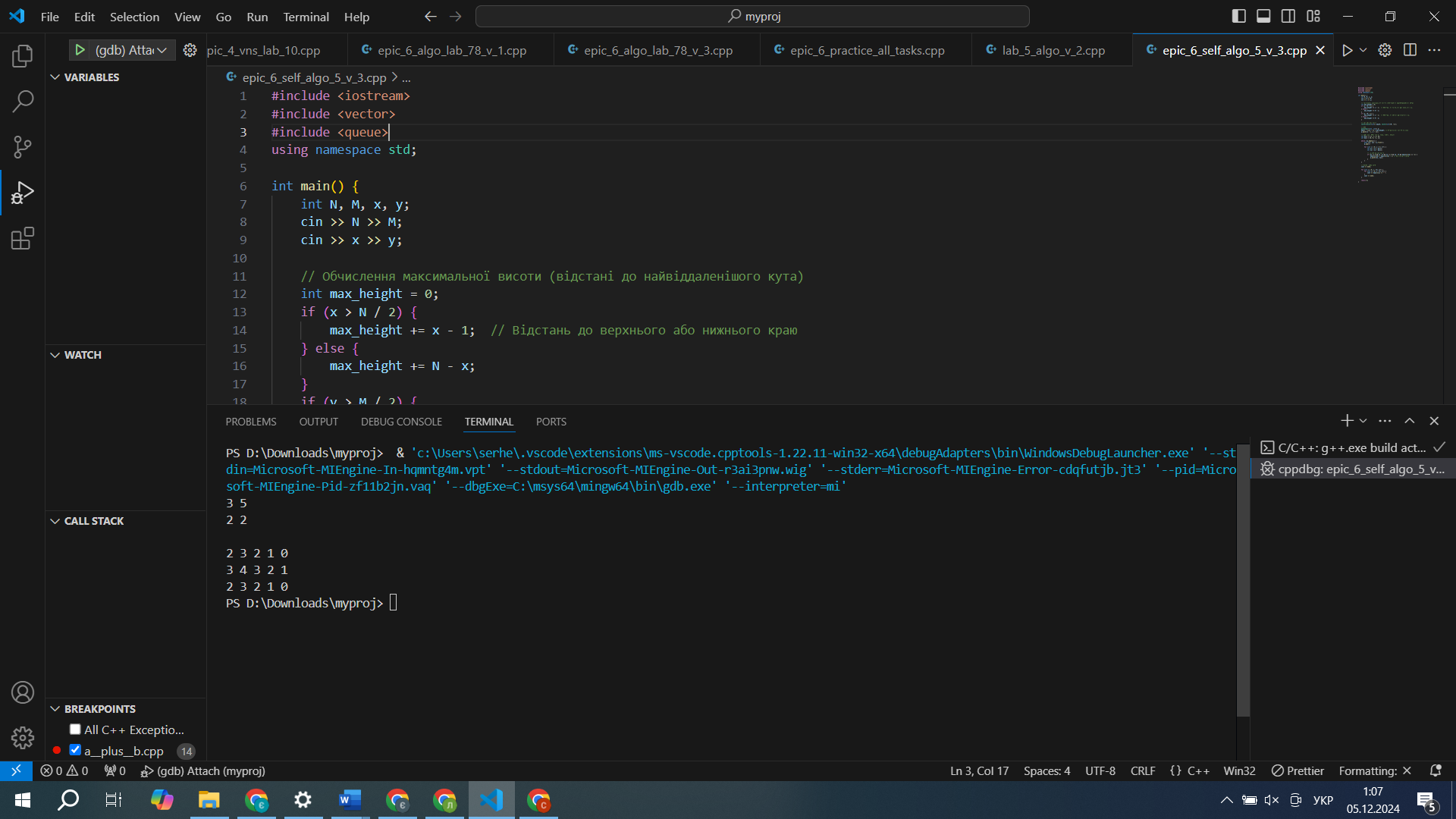
        cout << endl;

    }

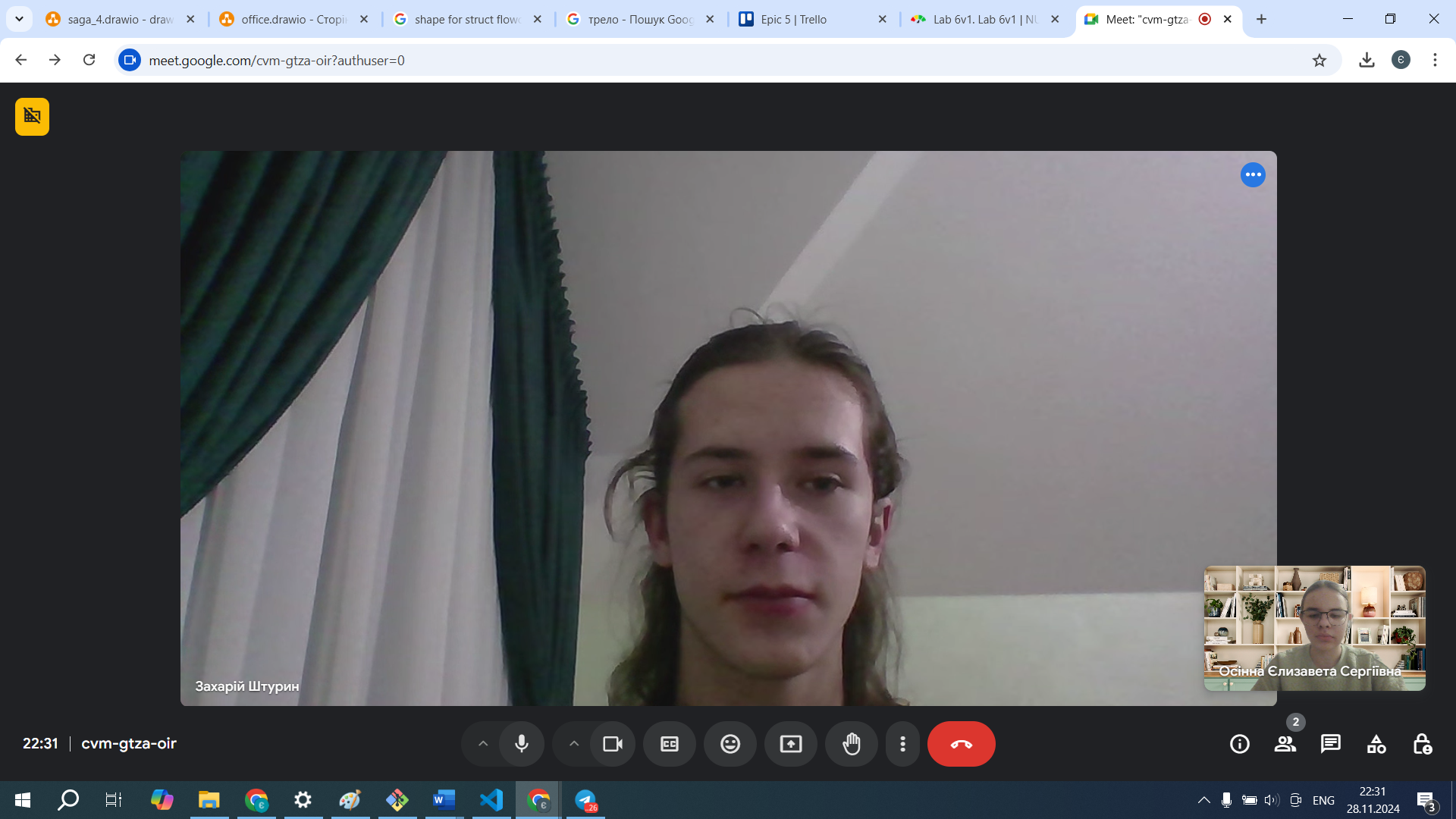
    return 0;

}

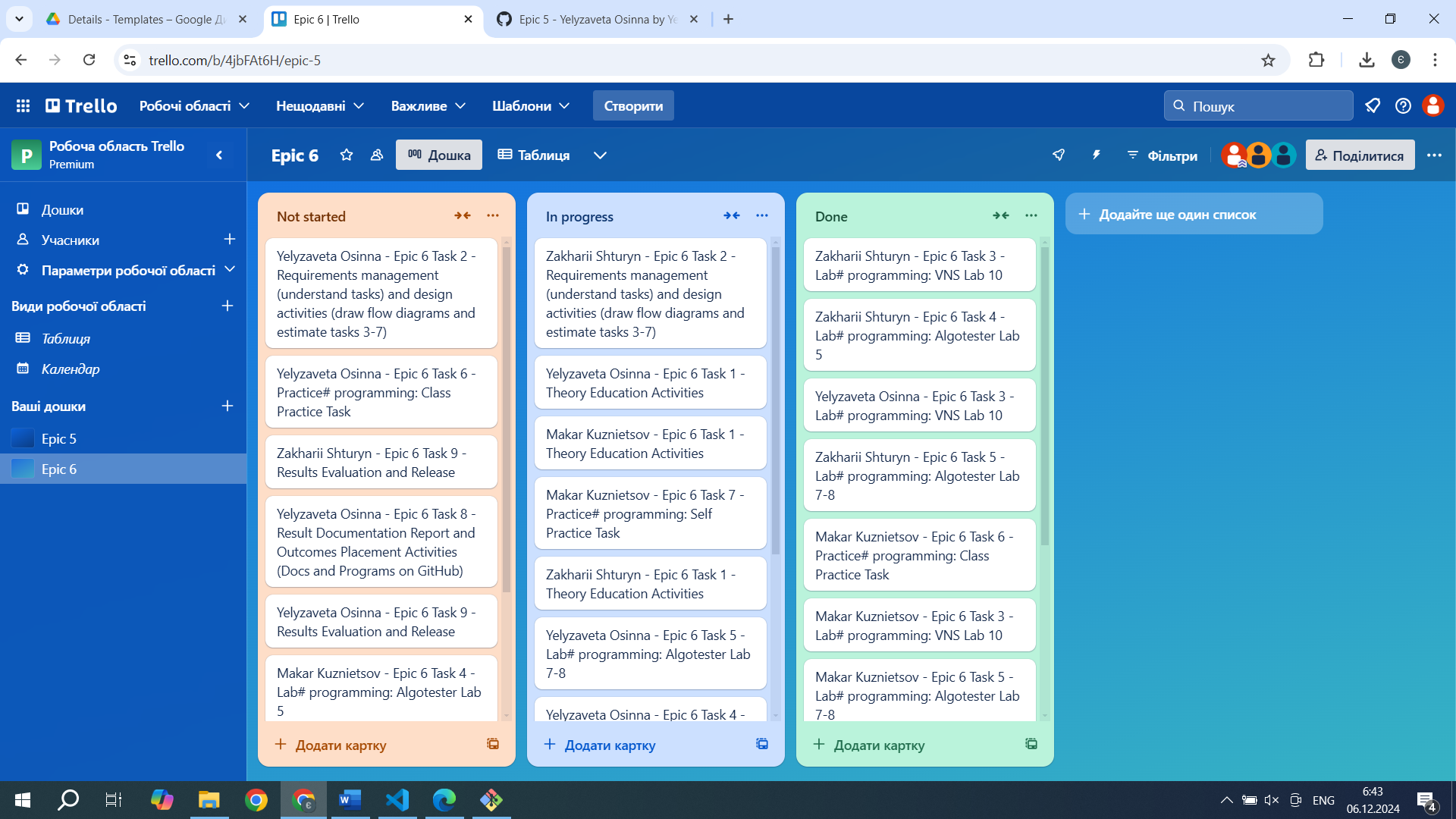
Result:



## **6. Кооперація з командою:**



Trello:



# **Висновки:**

Виконавши цю роботу я ознайомилася з динамічними інформаційними структурами на прикладі одно- і двонапрямлених списків, попрацювала над розумінням структур даних, розвитком алгоритмічного мислення, засвоїла механізми маніпуляції з покажчиками, розвинула розуміння рівності в структурах даних, поглибила розуміння зв’язаних списків, розуміння ефективності алгоритму