Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт № 6**

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

***з розділу***: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур»

***Виконав:***

студент групи ШІ-13

Золотарчук Максим

# **Тема роботи:**

Динамічні структури. Їх типи, властивості, алгоритми обробки.

# **Мета роботи:**

Метою даної роботи є розуміння динамічних структур даних, проектування та реалізація структур даних, вивчення базових операцій, що виконуються над динамічними структурами (додавання, видалення, пошук, сортування), застосування динамічних структур у практичних задачах.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічний масив.
* Тема №2: Бінарне дерево.
* Тема №3: Однозв’язний та двозв’язний списки.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Динамічний масив.
  + Джерела Інформації
    - Відео - <https://www.youtube.com/watch?v=GHLv3fNLOh0>
    - Стаття 1 - [How to create a dynamic array of integers in c++](https://stackoverflow.com/questions/4029870/how-to-create-a-dynamic-array-of-integers-in-c)
    - Стаття 2 - [How do dynamic arrays work?](https://www.geeksforgeeks.org/how-do-dynamic-arrays-work/)

Що опрацьовано:

* + - Ознайомлення із Специфікацією Динамічного Масиву:
    - Розуміння основних властивостей динамічного масиву.
    - Знання процесу його створення та розширення в пам'яті за необхідності.
    - Операції Запису та Читання:
    - Вивчення процесу зчитування та запису даних у динамічний масив.
    - Розуміння індексації елементів в масиві та його взаємодії з пам'яттю.
    - Маніпулювання Розміром Масиву:
    - Освоєння способів зміни розміру динамічного масиву.
    - Розгляд ситуацій, коли потрібно збільшити чи зменшити розмір масиву.
    - Керування Пам'яттю:
    - Ознайомлення із процесом управління пам'яттю для динамічного масиву.
    - Розуміння важливості вивільнення пам'яті після завершення роботи.
    - Ефективне Використання Динамічного Масиву:
  + Статус: Ознайомлений
  + Початок опрацювання теми: 09.12.2023
  + Звершення опрацювання теми: 15.12.2023
* Тема №2: Бінарне дерево.
  + Джерела Інформації:
    - Відео - <https://www.youtube.com/watch?v=zuuAPYiMYDA>
    - Стаття 1 - [Introduction to binary tree data structure and algorithm](https://www.geeksforgeeks.org/introduction-to-binary-tree-data-structure-and-algorithm-tutorials/).
    - Стаття 2 - <https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/>
  + Що опрацьовано:
    - Розуміння того, що бінарне дерево - це ієрархічна структура даних, в якій кожен вузол має не більше двох нащадків (лівого та правого).
    - Операції на бінарному дерева
    - Розуміння властивостей BST, де для кожного вузла всі значення в лівому піддереві менше за його значення, а всі значення в правому піддереві більше.
    - Алгоритми для бінарного дерева:
    - Вивчення того, як бінарні дерева використовуються в різних областях програмування, таких як бази даних, обробка виразів, компілятори, графічні системи тощо.
    - Реалізація та оптимізація:
    - Реалізація власних класів або структур для представлення бінарного дерева в коді.
    - Оптимізація операцій та алгоритмів для досягнення більшої ефективності та швидкодії.
    - Розуміння можливих проблем та виправлення помилок.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 10.12.2023
  + Звершення опрацювання теми: 15.12.2023
* Тема №3: Однозв’язний та двозв’язний списки.
  + Джерела Інформації:
    - Відео - <https://www.youtube.com/watch?v=RCHGco2NvMk> .
    - Стаття 1 - [https://www.geeksforgeeks.org/difference-between-singly-linked-list-and-doubly-linked-list/.](https://www.geeksforgeeks.org/difference-between-singly-linked-list-and-doubly-linked-list/)
    - Стаття 2 - <https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/>
  + Що опрацьовано:
    - Однозв’язний список:
    - Структура вузла - Елементи однозв'язного списку складаються з вузлів, кожен із яких містить дані та вказівник на наступний вузол у списку.
    - Вставка - Додавання нового вузла на початок або в кінець списку.
    - - Вставка в середину списку на певну позицію.
    - Видалення - Видалення вузла з початку, кінця або з середини списку.
    - Пошук - Пошук вузла за значенням або за індексом.
    - Перебір - Проходження через всі вузли списку для операцій обробки.
    - Двозв'язний Список:
    - Структура вузла - Елементи однозв'язного списку складаються з вузлів, кожен із яких містить дані та вказівник на наступний вузол у списку.
    - Вставка - Додавання нового вузла на початок або в кінець списку.
    - - Вставка в середину списку на певну позицію.
    - Видалення - Видалення вузла з початку, кінця або з середини списку.
    - Пошук - Пошук вузла за значенням або за індексом.
    - Перебір - Проходження через всі вузли списку для операцій обробки.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 09.12.2023
  + Звершення опрацювання теми: 14.12.2023

# **Виконання роботи:**

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

Завдання №1 Опрацювання теорії

* Деталі завдання: опрацювання вище перечисленої теорії. Розбір статей, перегляд відео та імплементація теорії на практиці. Вивчення найосновніших тем.

Завдання №2 VNS Lab 10

* Варіант завдання – 3
* Записи в лінійному списку містять ключове поле типу int. Сформувати
* однонаправлений список. Знищити з нього К елементів, починаючи із
* заданого номера, додати елемент перед елементом із заданим ключем;
* 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.

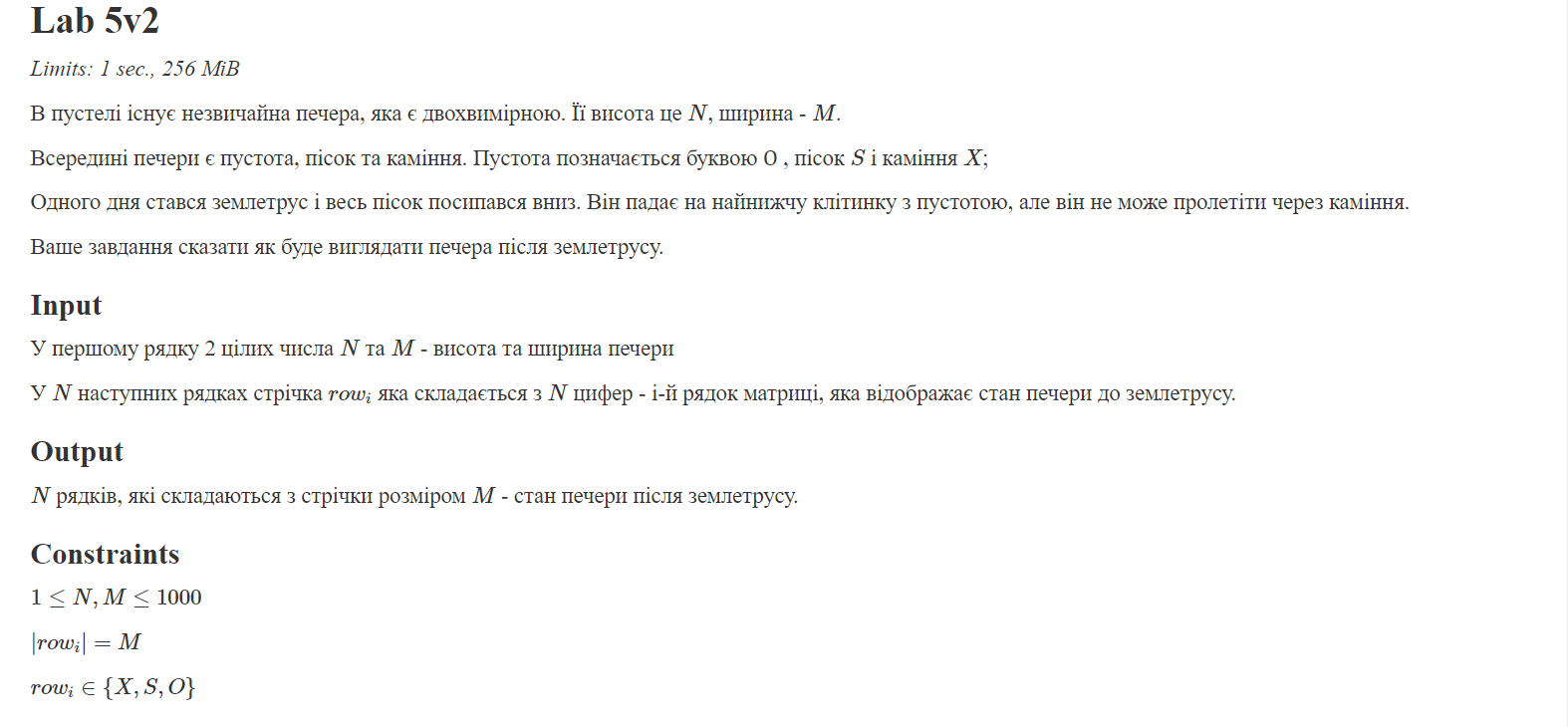
5. Запис списку у файл.

6. Знищення списку.

7. Відновлення списку з файлу.

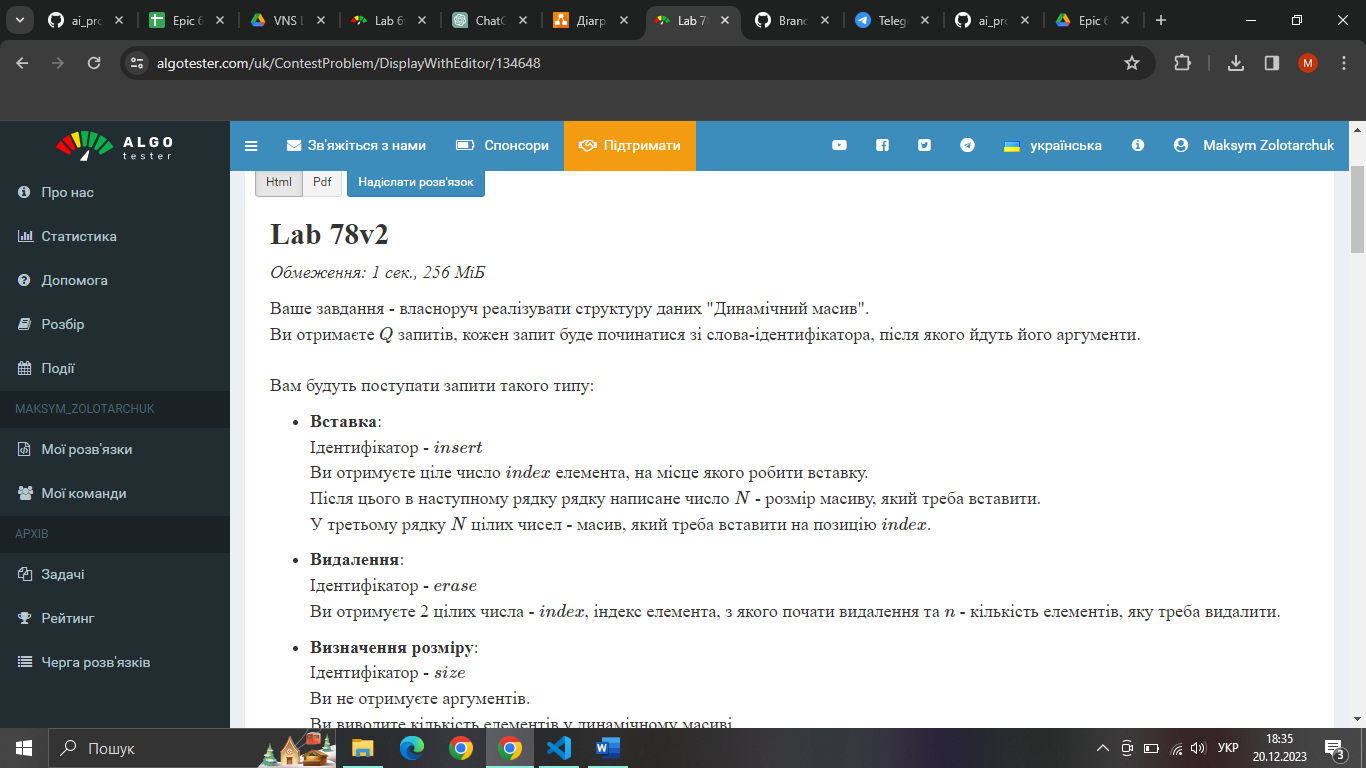
Завдання №3 Algotester Lab 5

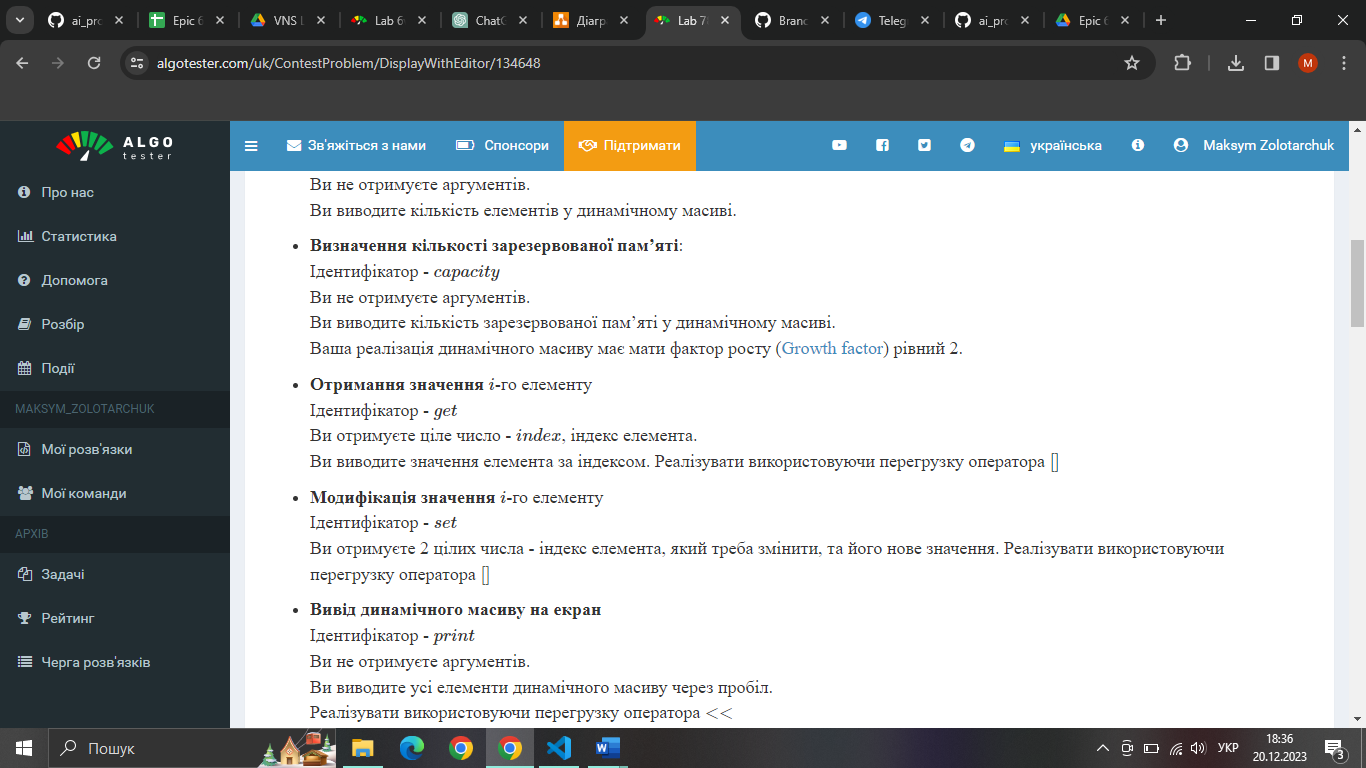
* Варіант завдання - 2
* Деталі завдання



Завдання №4 Algotester Lab 78

* Варіант завдання - 2
* Деталі завдання





Завдання №5 Class Practice 1

* 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;

Завдання №6 Class Practice 2

* Деталі завдання

## 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***.

Завдання №7 Class Practice 3

* Деталі завдання

## 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.

Завдання №8 Class Practice 4

* Деталі завдання

## 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

Завдання №9 Class Practice 5

* Деталі завдання

## 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

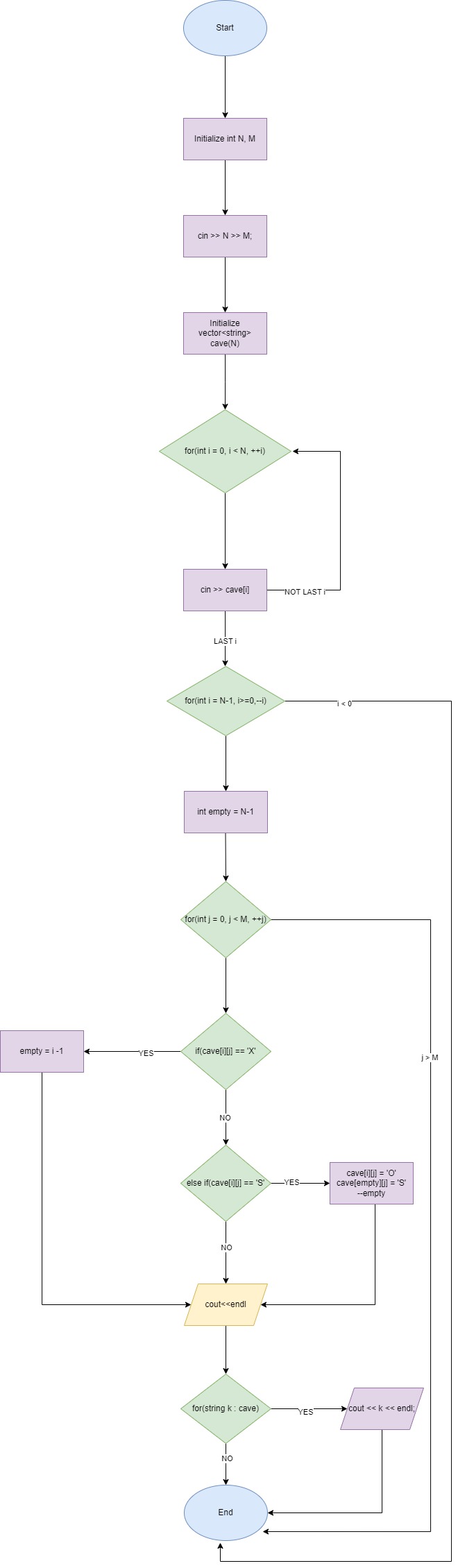
## **2. Дизайн та планована оцінка часу виконання завдань:**

Програма №1 VNS Lab 10

* Планований час на реалізацію - 3 година
* Час затрачений на виконання завдання – 4 години

Програма №2 Algotester Lab 5

* Планований час на реалізацію – 2 година
* Час затрачений на виконання завдання – 2 години
* Блок-схема



Програма №3 Algotester Lab 78

* Планований час на реалізацію - 3 година
* Час затрачений на виконання завдання – 6 години

Програма №4 Class Practice 1

* Планований час на реалізацію - 1 година
* Час затрачений на виконання завдання – 30 хвилин

Програма №5 Class Practice 2

* Планований час на реалізацію – 1 година
* Час затрачений на виконання завдання – 30 хвилин

Програма №6 Class Practice 3

* Планований час на реалізацію – 1 година
* Час затрачений на виконання завдання – 30 хвилин

Програма №7 Class Practice 4

* Планований час на реалізацію – 1 година
* Час затрачений на виконання завдання – 30 хвилин

Програма №8 Class Practice 5

* Планований час на реалізацію – 1 година
* Час затрачений на виконання завдання – 30 хвилин

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

Завдання №1 VNS Lab 10

#include <iostream>

#include <list>

using namespace std;

        // Функція для видалення К елементів, починаючи з заданого номера

        void deleteElements(list<int>&*lst*, int *start*, int *K*)

        {

        if (*start* < 0, *start* >= *lst*.size(), *K* <= 0) return;

        auto it = *lst*.begin();

        advance(it, *start*);

        while (*K*-- > 0 && it != *lst*.end())

        {

            it = *lst*.erase(it);

        }

    }

    // Функція для додавання елемента перед елементом із заданим ключем

    void addElement(std::list<int>&*lst*, int *key*, int *newElement*)

    {

        for (auto it = *lst*.begin(); it != *lst*.end(); ++it)

        {

            if (\*it == *key*)

            {

*lst*.insert(it, *newElement*);

                break;

            }

        }

    }

    int main()

    {

       list<int> lst = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        // Видалити 3 елементи, починаючи з 2-го номера

        deleteElements(lst, 2, 3);

        // Додати елемент 100 перед елементом із ключем 5

        addElement(lst, 5, 100);

        for (int x : lst)

        {

            cout << x << ' ';

        }

        cout << endl;

        return 0;

    }

*Код програми №1 VNS Lab 10*

Завдання №2 Algotester Lab 5

#include <iostream>

#include <string>

#include <vector>

using namespace std;

int main()

{

    int N, M;

    cin >> N;

    cin>> M;

    vector<*string*> cave(N);

    for (int i = 0; i < N; ++i)

    {

      cin >> cave[i];

    }

    for (int i = N - 1; i >= 0; --i)

    {

        int empty = N - 1;

        for (int j = 0; j < M; ++j)

        {

            if (cave[i][j] == 'X')

            {

                empty = i - 1; // встановлюю позицію найнижчої пустоти

            }

            else if (cave[i][j] == 'S')

            {

                cave[i][j] = 'O';

                cave[empty][j] = 'S';

                --empty;

            }

        }

    }

    cout << endl;

    for (*string* k : cave)

    {

        cout << k << endl;

    }

    return 0;

}

*Код програми №2 Algotester Lab 5*

Завдання №3 Algotester Lab 78

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

struct *DynamicArray* {

    long int\* massive;

    long int size;

    long int capacity;

    DynamicArray() : size(0), capacity(1) {

        massive = **new** long int[capacity];

    }

    ~DynamicArray() {

**delete[]** massive;

    }

    void insert(long int *a*, long int *N*, long int\* *value*) {

        if (*a* < 0 || *a* > size)

            return;

        if (size + *N* >= capacity) {

            long int changedCapacity = capacity;

            while (size + *N* >= changedCapacity) {

                changedCapacity \*= 2;

            }

            long int\* changedArr = **new** long int[changedCapacity];

            size += *N*;

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

                changedArr[i] = massive[i];

            }

**delete[]** massive;

            massive = changedArr;

            copy\_backward(massive + *a*, massive + size - *N*, massive + size);

            for (*size\_t* i = 0; i < *N*; ++i) {

                massive[*a* + i] = *value*[i];

            }

            capacity = changedCapacity;

        } else {

            for (int i = size - 1; i >= *a*; --i) {

                massive[i + *N*] = massive[i];

            }

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

                massive[*a* + i] = *value*[i];

            }

            size += *N*;

        }

    }

    void erase(long int *a*, long int *n*) {

        long int deleteCount = min(*n*, size - *a*);

        for (int i = *a*; i < size - deleteCount; ++i) {

            massive[i] = massive[i + deleteCount];

        }

        size -= deleteCount;

    }

    long int Size() const {

        return size;

    }

    long int Capacity() const {

        return capacity;

    }

    void print() const {

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

            cout << massive[i] << " ";

        }

    }

    long int& operator[](long int *a*) {

        return massive[*a*];

    }

};

int main() {

    long int N;

    cin >> N;

*DynamicArray* dynArray;

    while (N--) {

*string* answer;

        cin >> answer;

        char sym = answer[0];

        long int a, n;

        long int\* value;

        switch (sym) {

            case 'i':

                cin >> a >> n;

                value = **new** long int[n];

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

                    cin >> value[i];

                }

                dynArray.insert(a, n, value);

**delete[]** value;

                break;

            case 'e':

                cin >> a >> n;

                dynArray.erase(a, n);

                break;

            case 's':

                if (answer == "size") {

                    cout << dynArray.Size() << endl;

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

                    cin >> a >> n;

                    if (a >= 0 && a < dynArray.Size()) {

                        dynArray[a] = n;

                    }

                }

                break;

            case 'c':

                cout << dynArray.Capacity() << endl;

                break;

            case 'g':

                cin >> a;

                if (a >= 0 && a < dynArray.Size()) {

                    cout << dynArray[a] << endl;

                }

                break;

            case 'p':

                dynArray.print();

                cout << endl;

                break;

            default:

                cout << "Error" << endl;

        }

    }

    return 0;

}

*Код програми №4 Algotester Lab 78*

Завдання №4 Class Practice 1

#include <iostream>

using namespace std;

struct *Node*

{

    int data;

*Node* \*next;

};

using *NodePtr* = *Node* \*;

*NodePtr* createList()

{

    return nullptr;

}

void insert(*NodePtr* &*head*, int *data*)

{

*NodePtr* tempPtr = **new** *Node*;

    tempPtr->data = *data*;

    tempPtr->next = *head*;

*head* = tempPtr;

}

*NodePtr* reverse(*NodePtr* *head*)

{

*NodePtr* current = *head*;

*NodePtr* prev = nullptr;

*NodePtr* next = nullptr;

    while (current != nullptr)

    {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    return prev;

}

void printList(const *NodePtr* *head*)

{

    if (*head* == nullptr)

    {

        cout << "List is empty";

    }

    else

    {

        for (*NodePtr* current = *head*; current != nullptr; current = current->next)

        {

            cout << current->data << " ";

        }

    }

    cout << endl;

}

void deleteList(*NodePtr* &*head*)

{

    while (*head* != nullptr)

    {

*NodePtr* temp = *head*;

*head* = *head*->next;

**delete** temp;

    }

}

int main()

{

*NodePtr* myList = createList();

    insert(myList, 8);

    insert(myList, 3);

    insert(myList, 5);

    cout << "Original List: ";

    printList(myList);

*NodePtr* reversedList = reverse(myList);

    cout << "Reversed List: ";

    printList(reversedList);

    // Очистка пам'яті після використання списку

    deleteList(myList);

    deleteList(reversedList);

    return 0;

}

*Код програми №4 Class Practice 1*

Завдання №5 Class Practice 2

#include <iostream>

#include <vector>

using namespace std;

class *List*

{

private:

    struct *Element*

    {

        int value;

*Element* \*nextElement;

        // Конструктор для створення елемента зі значенням та наступним елементом

        Element(int *val*, *Element* \**next*) : value(*val*), nextElement(*next*) {}

    };

*Element* \*firstElement;

public:

    // Конструктор за замовчуванням

    List() : firstElement(nullptr) {}

    // Додавання елемента на початок списку

    void addElement(int *value*)

    {

        firstElement = **new** *Element*(*value*, firstElement);

    }

    // Порівняння двох списків на рівність

    bool isEqual(const *List* &*other*) const

    {

*Element* \*currentElement1 = firstElement;

*Element* \*currentElement2 = *other*.firstElement;

        while (currentElement1 != nullptr && currentElement2 != nullptr)

        {

            if (currentElement1->value != currentElement2->value)

            {

                return false;

            }

            currentElement1 = currentElement1->nextElement;

            currentElement2 = currentElement2->nextElement;

        }

        // Перевірка, чи обидва елемента дійшли кінця свого списку

        return currentElement1 == nullptr && currentElement2 == nullptr;

    }

};

int main()

{

*List* listA;

    listA.addElement(47);

    listA.addElement(19);

    listA.addElement(54);

    listA.addElement(95);

*List* listB;

    listB.addElement(47);

    listB.addElement(19);

    listB.addElement(54);

    listB.addElement(94);

    if (listA.isEqual(listB))

    {

        cout << "Lists are equal\n";

    }

    else

    {

        cout << "Lists are not equal\n";

    }

    return 0;

}

*Код програми №5 Class Practice 2*

Завдання №6 Class Practice 3

#include <iostream>

using namespace std;

struct *Node*

{

  int data;

*Node* \*next;

  explicit Node(int *val*) : data(*val*), next(nullptr) {}

};

*Node* \*createNode(int *data*)

{

  return **new** *Node*(*data*);

}

void push(*Node* \*&*head*, int *data*)

{

*Node* \*newNode = createNode(*data*);

  newNode->next = *head*;

*head* = newNode;

}

*Node* \*addTwoLists(const *Node* \**first*, const *Node* \**second*)

{

*Node* \*result = nullptr;

*Node* \*temp = nullptr;

*Node* \*previous = nullptr;

  int carry = 0, sum;

  while (*first* != nullptr || *second* != nullptr)

  {

    sum = carry + (*first* ? *first*->data : 0) + (*second* ? *second*->data : 0);

    carry = (sum >= 10) ? 1 : 0;

    sum = sum % 10;

    temp = createNode(sum);

    if (result == nullptr)

    {

      result = temp;

    }

    else

    {

      previous->next = temp;

    }

    previous = temp;

    if (*first*)

    {

*first* = *first*->next;

    }

    if (*second*)

    {

*second* = *second*->next;

    }

  }

  if (carry > 0)

  {

    temp->next = createNode(carry);

  }

  return result;

}

void printList(const *Node* \**node*)

{

  while (*node* != nullptr)

  {

    cout << *node*->data << " ";

*node* = *node*->next;

  }

  cout << std::endl;

}

void deleteList(*Node* \*&*head*)

{

  while (*head* != nullptr)

  {

*Node* \*temp = *head*;

*head* = *head*->next;

**delete** temp;

  }

}

int main()

{

*Node* \*firstList = nullptr;

*Node* \*secondList = nullptr;

  push(firstList, 9);

  push(firstList, 7);

  push(firstList, 3);

  push(secondList, 9);

  push(secondList, 7);

  push(secondList, 3);

*Node* \*resultList = addTwoLists(firstList, secondList);

  printList(resultList);

  // Очистка пам'яті після використання списків

  deleteList(firstList);

  deleteList(secondList);

  deleteList(resultList);

  return 0;

}

*Код програми №6 Class Practice 3*

Завдання №7 Class Practice 4

#include <iostream>

using namespace std;

struct *TreeNode*

{

    int value;

*TreeNode* \*leftChild;

*TreeNode* \*rightChild;

    TreeNode(int *val*) : value(*val*), leftChild(nullptr), rightChild(nullptr) {}

};

void insertNode(*TreeNode* \*&*rootNode*, int *val*)

{

    if (*rootNode* == nullptr)

    {

*rootNode* = **new** *TreeNode*(*val*);

    }

    else if (*val* < *rootNode*->value)

    {

        insertNode(*rootNode*->leftChild, *val*);

    }

    else

    {

        insertNode(*rootNode*->rightChild, *val*);

    }

}

*TreeNode* \*createMirrorFlip(const *TreeNode* \**rootNode*)

{

    if (*rootNode* == nullptr)

    {

        return nullptr;

    }

*TreeNode* \*mirrorRootNode = **new** *TreeNode*(*rootNode*->value);

    mirrorRootNode->leftChild = createMirrorFlip(*rootNode*->rightChild);

    mirrorRootNode->rightChild = createMirrorFlip(*rootNode*->leftChild);

    return mirrorRootNode;

}

void inOrderTraversal(const *TreeNode* \**rootNode*)

{

    if (*rootNode*)

    {

        inOrderTraversal(*rootNode*->leftChild);

        cout << *rootNode*->value << " ";

        inOrderTraversal(*rootNode*->rightChild);

    }

}

void deleteTree(*TreeNode* \*&*rootNode*)

{

    if (*rootNode*)

    {

        deleteTree(*rootNode*->leftChild);

        deleteTree(*rootNode*->rightChild);

**delete** *rootNode*;

*rootNode* = nullptr;

    }

}

int main()

{

*TreeNode* \*rootNode = nullptr;

    int nodeValues[] = {5, 3, 7, 2, 4, 6, 8};

    for (int val : nodeValues)

    {

        insertNode(rootNode, val);

    }

    cout << "Original Tree: ";

    inOrderTraversal(rootNode);

    cout << "\n";

*TreeNode* \*mirrorRootNode = createMirrorFlip(rootNode);

    cout << "Mirrored Tree: ";

    inOrderTraversal(mirrorRootNode);

    cout << "\n";

    // Очистка пам'яті після використання дерева

    deleteTree(rootNode);

    deleteTree(mirrorRootNode);

    return 0;

}

*Код програми №7 Class Practice 4*

Завдання №8 Class Practice 5

#include <iostream>

#include <vector>

using namespace std;

struct *TreeNode*

{

  int data;

*TreeNode* \*left;

*TreeNode* \*right;

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

};

*TreeNode* \*insertNode(*TreeNode* \**root*, int *value*)

{

  if (*root* == nullptr)

  {

    return **new** *TreeNode*(*value*);

  }

  if (*value* < *root*->data)

  {

*root*->left = insertNode(*root*->left, *value*);

  }

  else

  {

*root*->right = insertNode(*root*->right, *value*);

  }

  return *root*;

}

void inOrderTraversal(*TreeNode* \**root*)

{

  if (*root* != nullptr)

  {

    inOrderTraversal(*root*->left);

    cout << *root*->data << " ";

    inOrderTraversal(*root*->right);

  }

}

void calculateTreeSum(*TreeNode* \**root*)

{

  if (*root* == nullptr || (*root*->left == nullptr && *root*->right == nullptr))

  {

    return;

  }

  calculateTreeSum(*root*->left);

  calculateTreeSum(*root*->right);

*root*->data = (*root*->left ? *root*->left->data : 0) + (*root*->right ? *root*->right->data : 0);

}

int main()

{

  vector<int> values = {5, 3, 7, 2, 4, 6, 8};

*TreeNode* \*root = nullptr;

  for (int value : values)

  {

    root = insertNode(root, value);

  }

  cout << "Original Tree: ";

  inOrderTraversal(root);

  cout << endl;

  calculateTreeSum(root);

  cout << "Tree After Sum Calculation: ";

  inOrderTraversal(root);

  cout << endl;

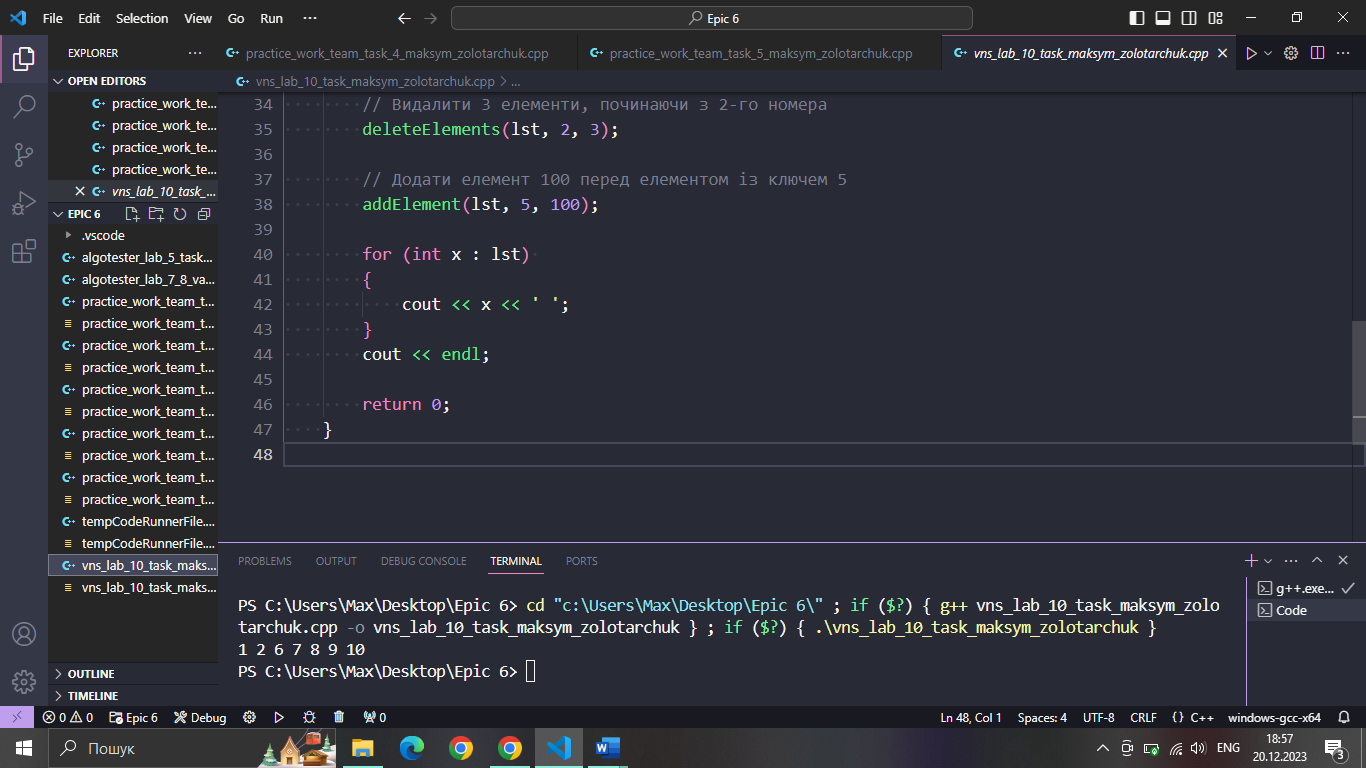
  return 0;

}

*Код програми №8 Class Practice 5*

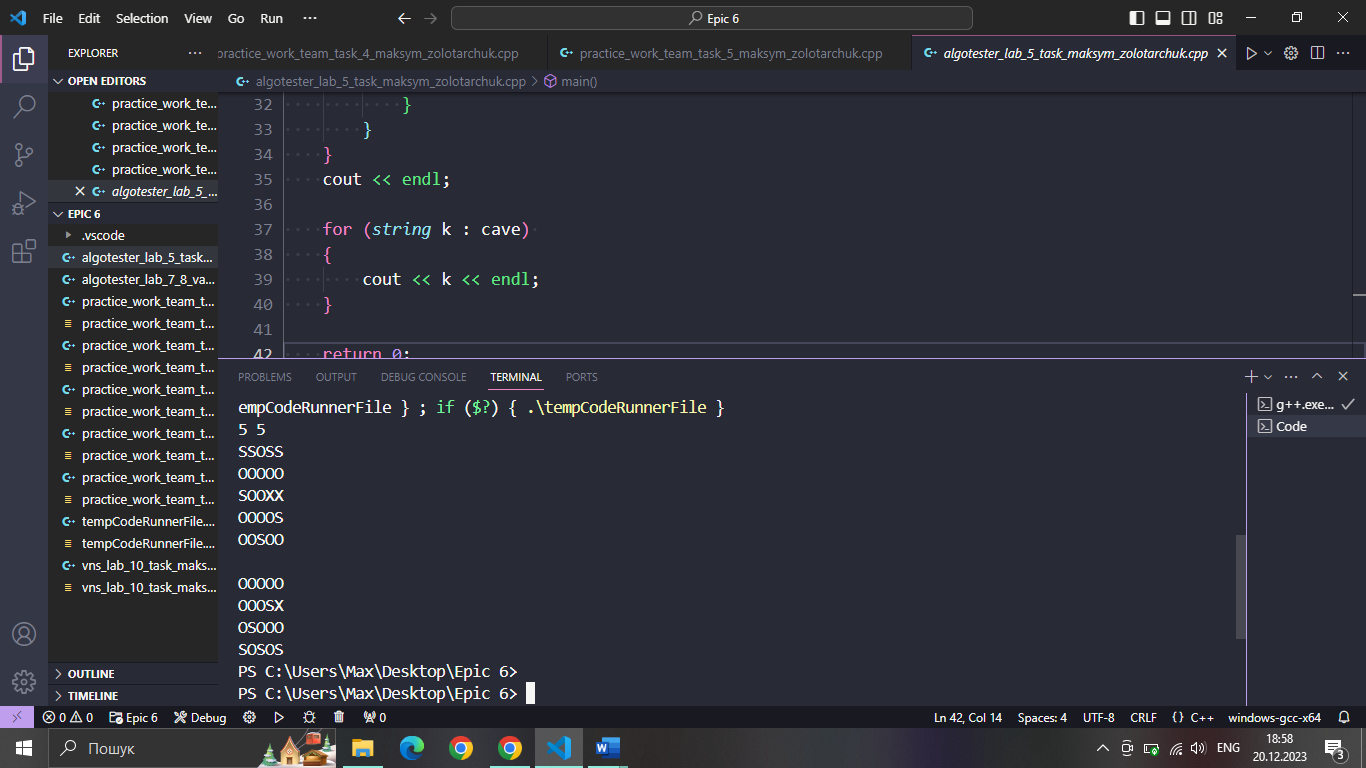
## **4. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1 VNS Lab 10



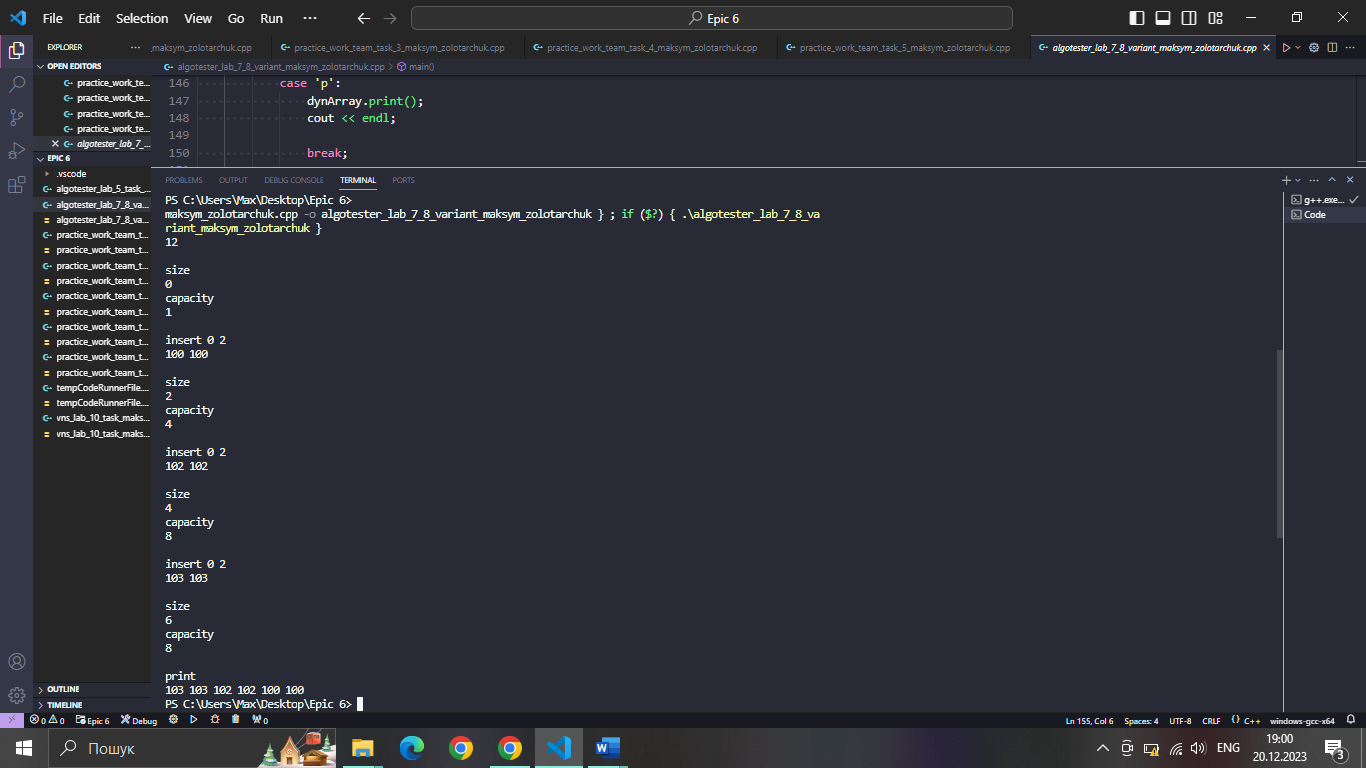
VNS Lab 10

Завдання №2 Algotester Lab 5



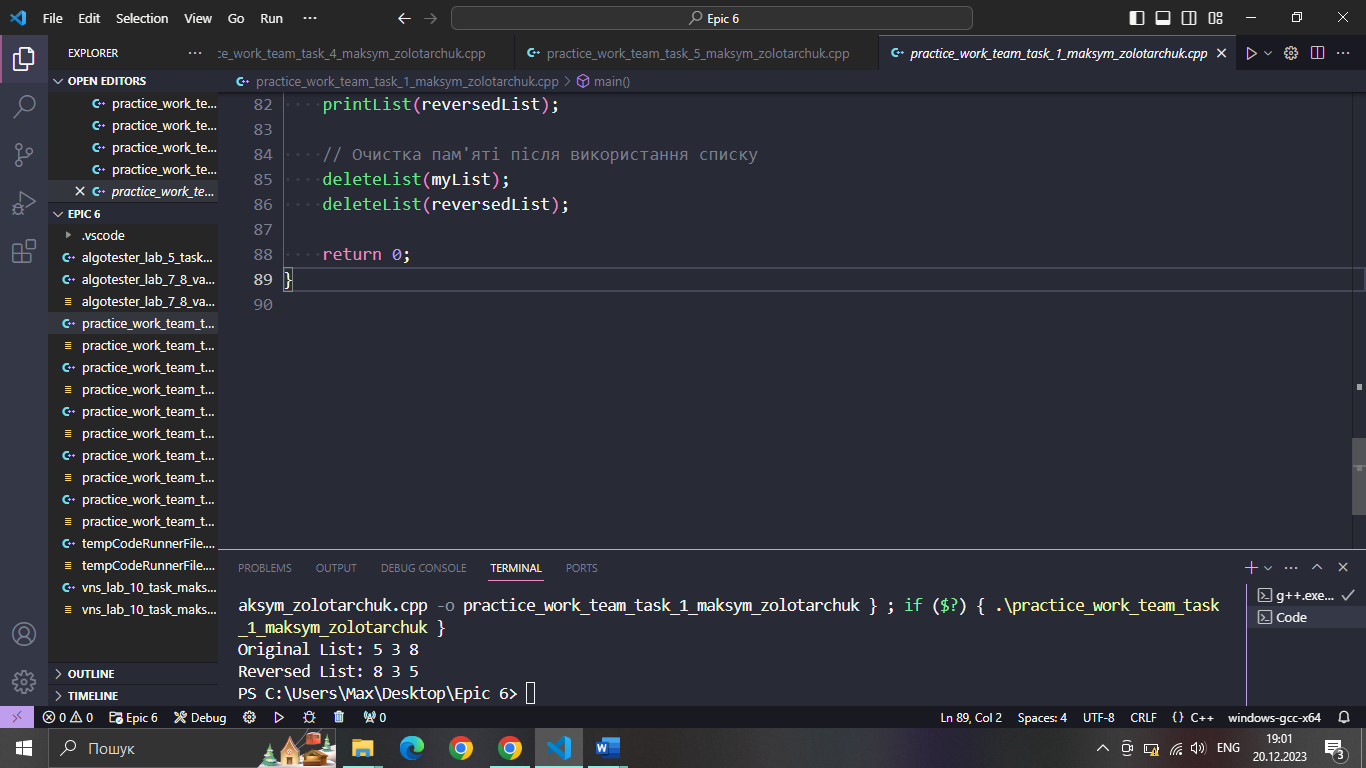
Algotester Lab 5

Завдання №3 Algotester Lab 78



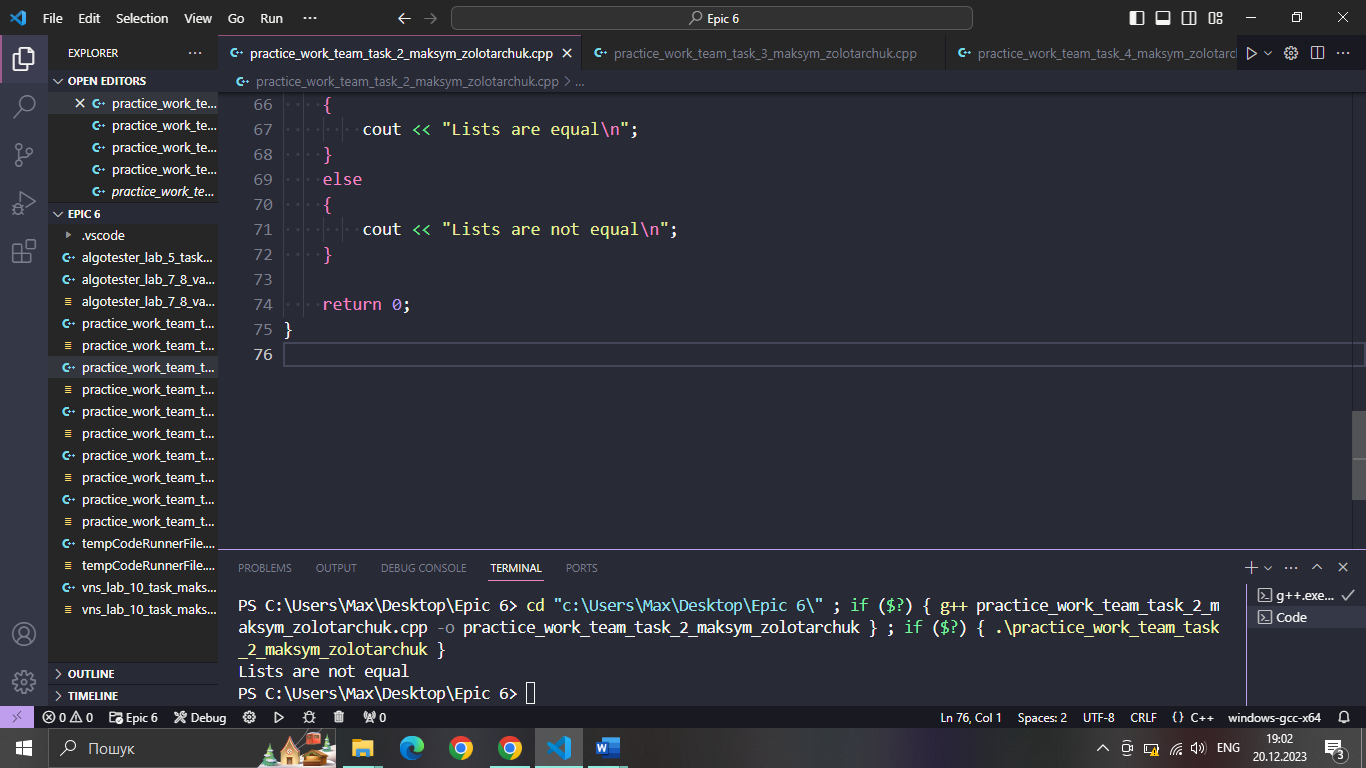
Algotester Lab 78

Завдання №4 Class Practice 1



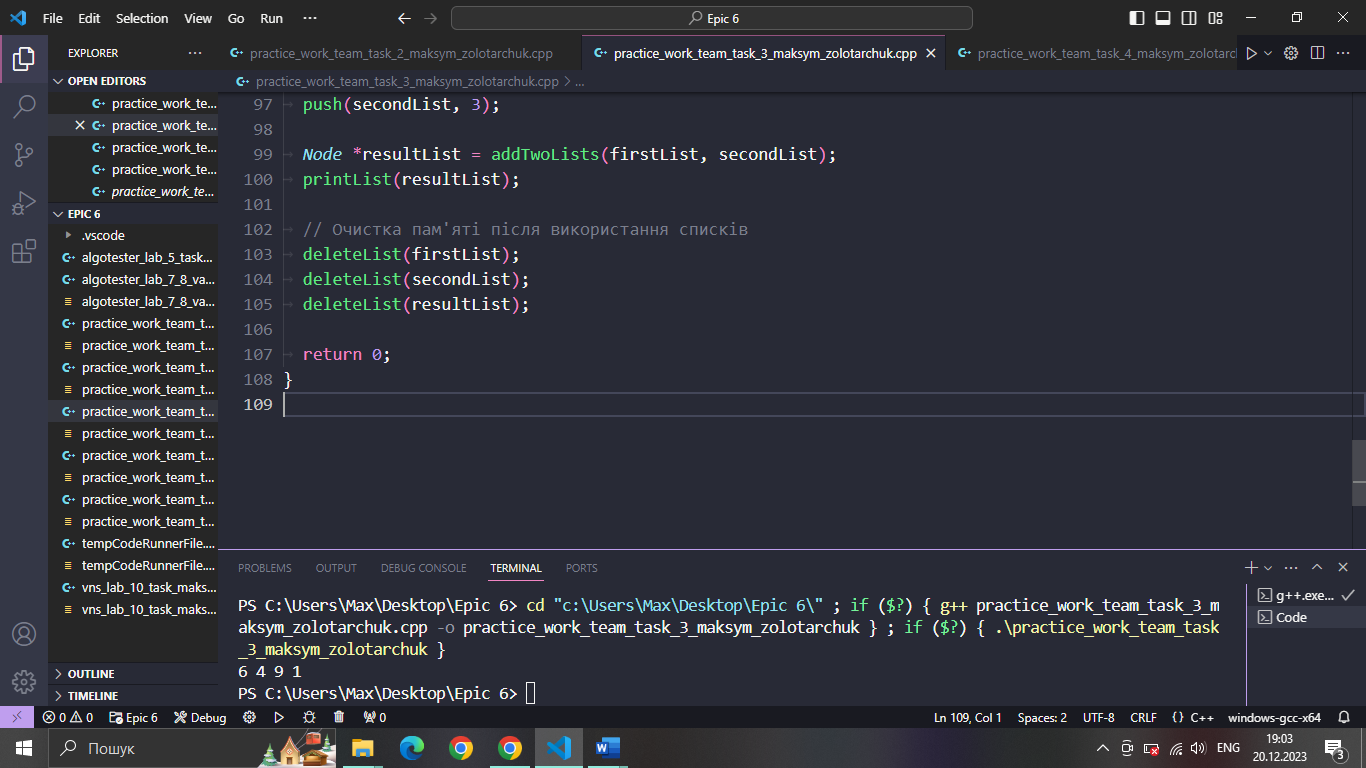
Class Practice 1

Завдання №5 Class Practice 2



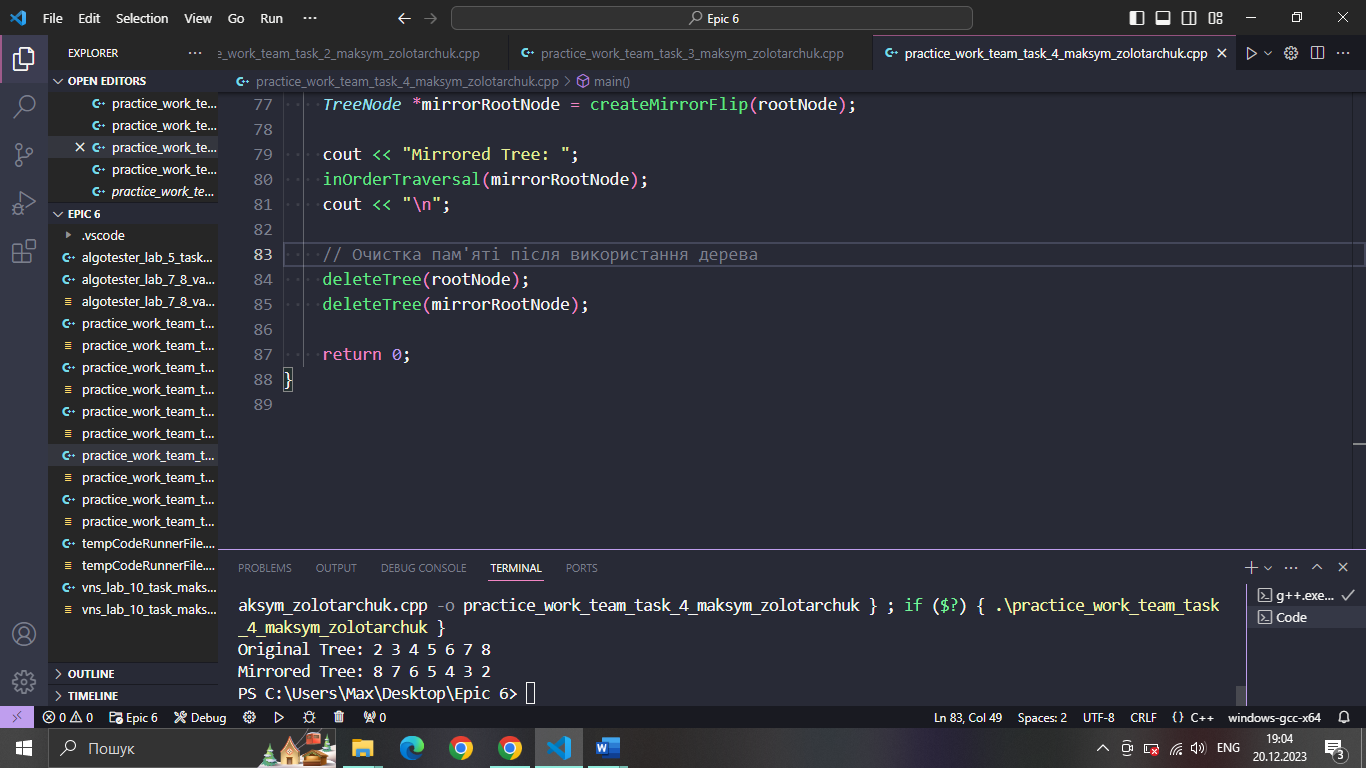
Class Practice 2

Завдання №6 Class Practice 3



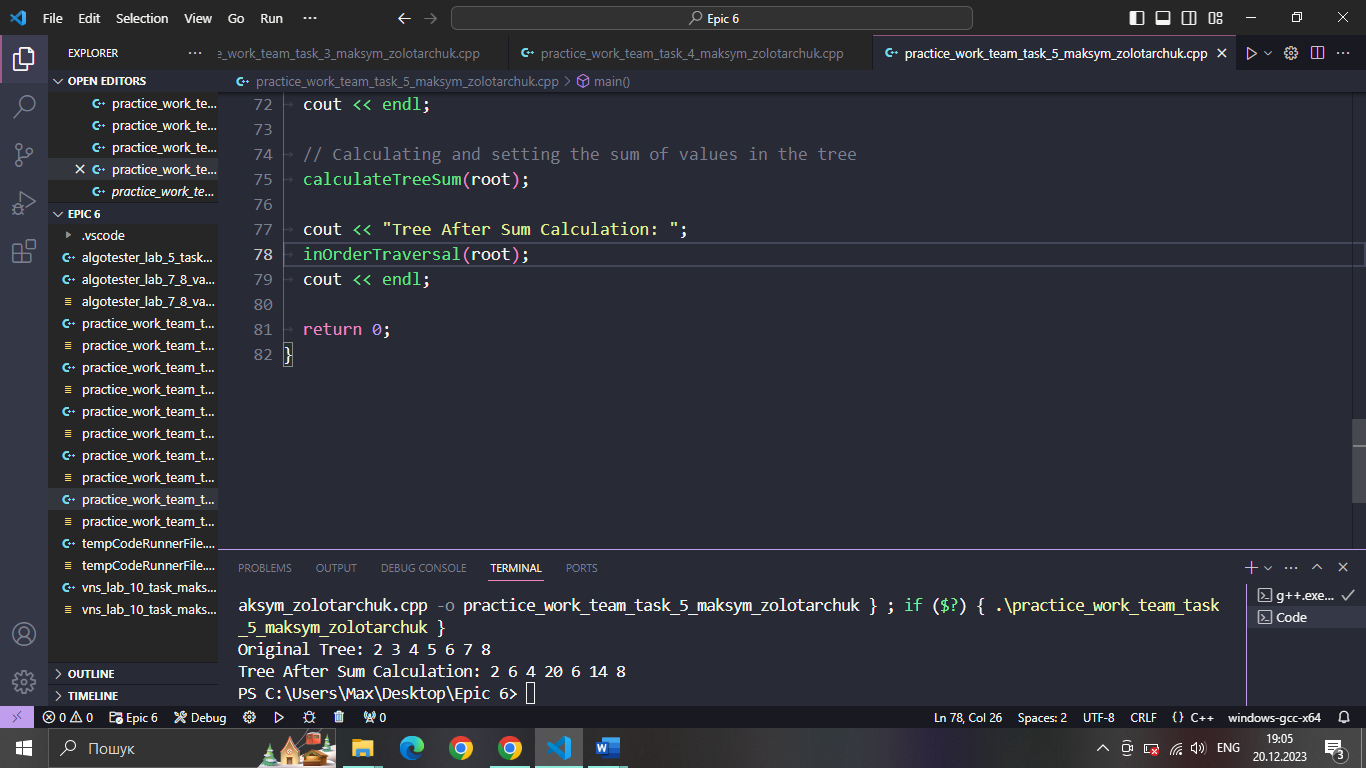
Class Practice 3

Завдання №7 Class Practice 4



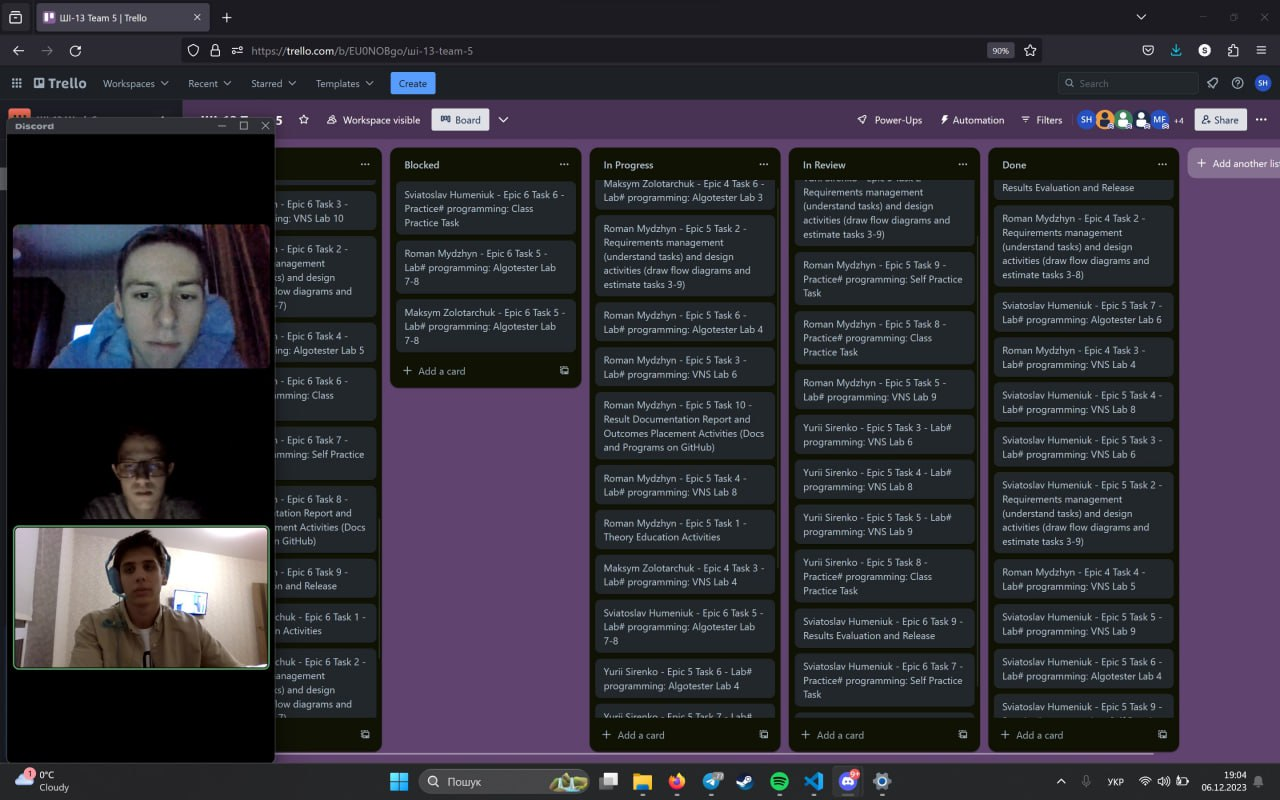
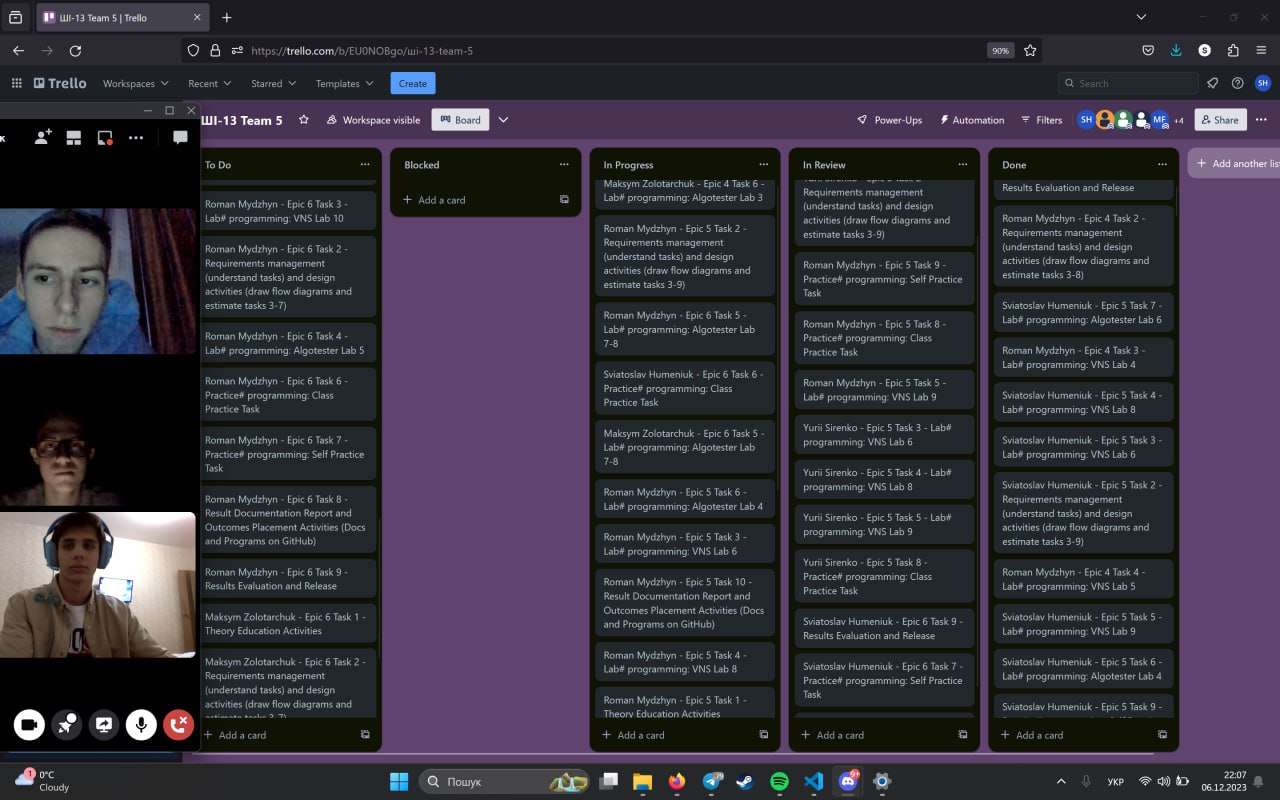
Class Practice 4

Завдання №8 Class Practice 5



Class Practice 5

## **5. Кооперація з командою:**

* Скрін з 1-ї зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло
* 
* Скрін з 2-ї зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло
* 

GitHub: <https://github.com/artificial-intelligence-department/ai_programming_playground/tree/epic_6_practice_and_labs_maksym_zolotarchuk/ai_13/maksym_zolotarchuk/Epic_6_Maksym_Zolotarchuk/Code>

# **Висновок:**

В ході виконання даної роботи було отримано важливі знання та навички у сфері динамічних структур даних. Основною метою було розуміння принципів роботи динамічних структур, проектування та реалізація їхніх операцій, таких як додавання, видалення, пошук та сортування.

Вивчення базових операцій, які можна виконувати над динамічними структурами даних, надало мені розуміння того, як ефективно працювати з такими структурами у реальних програмах. Реалізація алгоритмів для додавання, видалення та сортування даних дозволила мені відчути практичний аспект роботи з цими структурами.

Застосування динамічних структур даних у практичних задачах стало ключовим етапом у розвитку моєї програмістської експертизи. Це включає в себе використання таких структур для ефективного управління та організації даних у різноманітних програмах.

В цілому, вивчення динамічних структур даних було важливим етапом у моєму навчанні, і я тепер володію необхідними знаннями та вміннями для успішної роботи з цим аспектом програмування в майбутньому.