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-11 Мацько Ілля Феліксович

Львів 2023

# **Тема роботи:**

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

# **Мета роботи:**

Опанувати роботу з динамічними структурами даних. Засвоїти черги, стеки, списки та дерева. Вивчити алгоритми обробки динамічних структур.

# **Теоретичні відомості:**

1) Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічні структури - черга/стек
* Тема №2: Динамічні структури - список
* Тема №3: Динамічні структури - дерево
* Тема №4: Алгоритми обробки динамічних структур.

2) Індивідуальний план опрацювання теорії

**Тема №1: Динамічні структури - черга/стек**

Стек:

Стек є абстрактною структурою даних, яка слідкує за принципом Last In, First Out (LIFO). Елементи можуть бути додані та вилучені тільки з вершини стеку. Це дозволяє ефективно використовувати його для вирішення задач, таких як обхід графів, виконання арифметичних операцій та інших.

Черга:

Черга є структурою даних за принципом First In, First Out (FIFO). Елементи можуть бути додані в кінець черги та вилучені з її початку. Вона часто використовується для організації обробки даних за принципом "першим прийшов, першим вийшов".

Посилання:

- [Стек (вікіпедія)](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA)

- [Черга (вікіпедія)](https://uk.wikipedia.org/wiki/%D0%A7%D0%B5%D1%80%D0%B3%D0%B0)

**Тема №2: Динамічні структури - список**

Список:

Список є структурою даних, що складається з вузлів, кожен з яких містить дані та посилання на наступний вузол в послідовності. Списки можуть бути однонаправленими (з кожного вузла є посилання лише на наступний вузол) або двонаправленими (з посиланням як на наступний, так і на попередній вузол).

Посилання:

- [Список (вікіпедія)](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA)

**Тема №3: Динамічні структури - дерево**

Дерево:

Дерево є ієрархічною структурою даних, яка складається з вузлів, з'єднаних гілками. Кожен вузол може мати кілька нащадків (дочірніх вузлів), але тільки один предок (батьківський вузол). Дерева використовуються для представлення ієрархій даних та використовуються в багатьох областях, включаючи інформаційні системи та операції пошуку.

Посилання:

- [Дерево (вікіпедія)](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)

**Тема №4: Алгоритми обробки динамічних структур**

Алгоритми:

Обробка динамічних структур даних включає в себе широкий спектр алгоритмів, таких як пошук, вставка, видалення та сортування. Наприклад, для дерев можна використовувати алгоритми обходу в глибину (DFS) або обходу в ширину (BFS). Для списків, стеків та черг можна використовувати різноманітні методи обробки, такі як пошук за значенням або індексом, вставка та видалення.

Посилання:

- [Алгоритми для динамічних структур даних (GeeksforGeeks)](https://www.geeksforgeeks.org/data-structures/)

# **Виконання роботи:**

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

Завдання №1 VNS Lab 10 - Task 1

* + Варіант завдання - 23
  + Деталі завдання:

Написати функцію для запису в лінійному списку що містить ключове поле типу \*char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим

ключем. Додати К елементів після елемента із заданим ключем.

Завдання № 2 Algotester Lab 5

* Варіант завдання - 2
* Деталі завдання:

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M. Всередині печери є пустота, пісок та каміння. Пустота позначається буквою О , пісок S і каміння X; Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння. Ваше завдання сказати як буде виглядати печера після землетрусу.

Завдання № 3 Algotester Lab 78

* Варіант завдання - 1
* Деталі завдання:

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

Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

Завдання № 4-1 Class practice work

* + Деталі завдання:

Задача №1 - Реверс списку (Reverse list)

Задача №2 - Порівняння списків

Завдання № 4-2 Class practice work

* + Деталі завдання:

Задача №3 – Додавання великих чисел

Задача №4 - Віддзеркалення дерева

Задача №5 - Записати кожному батьківському вузлу суму підвузлів

Завдання № 5 Self practice work

* + Деталі завдання

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. Клітинки 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.

## **Дизайн та оцінка часу виконання завдань:**

Програма № 1 VNS Lab 10 - Task 1

* + Запланований час на реалізацію - 1 година

Програма № 2 Algotester Lab 5v2

* + Запланований час на реалізацію - 45 хвилин

Програма № 3 Algotester Lab 78v1

* + Запланований час на реалізацію - 2 години

Програма № 4-1 Class practice work

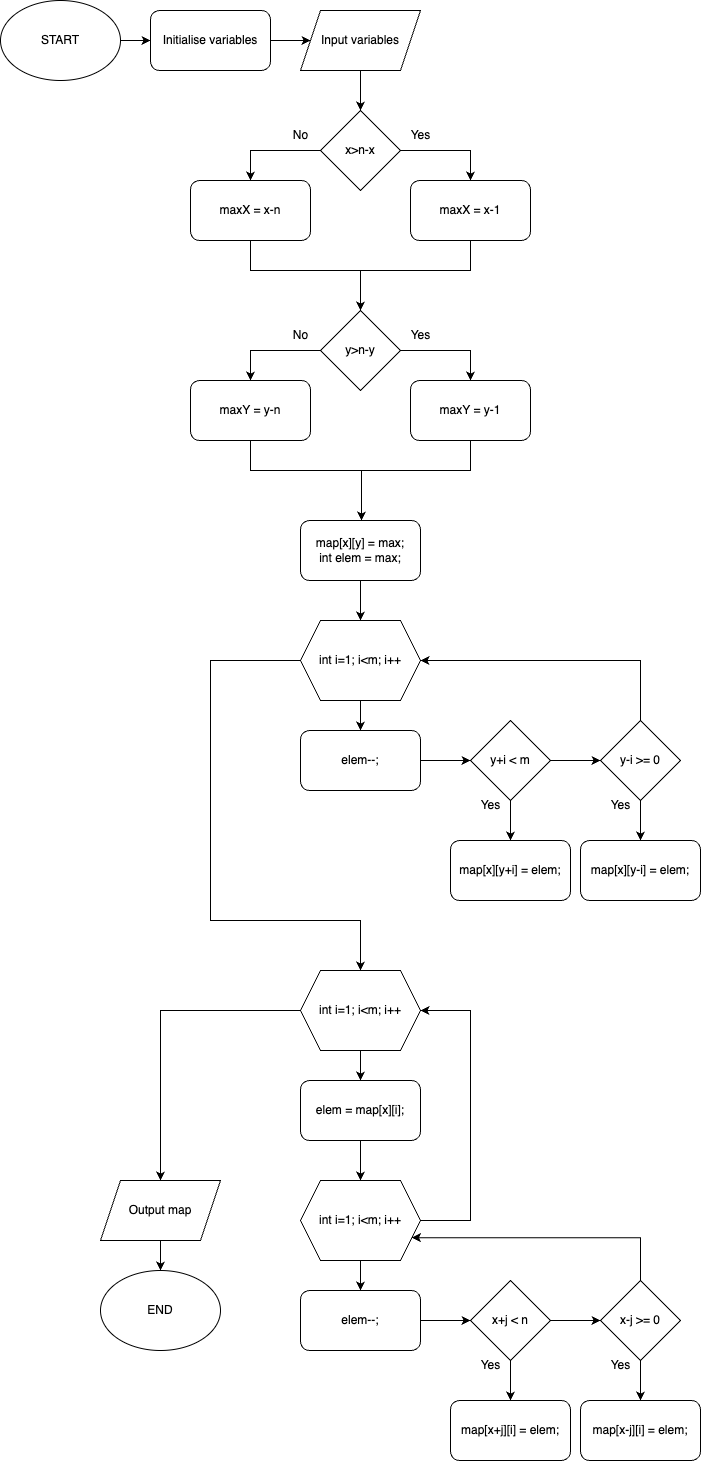
* + Запланований час на реалізацію - 2 години

Програма № 4-2 Class practice work

* + Запланований час на реалізацію - 2 години

Програма № 5 Self practice(Algotester Lab5v3)

* + Запланований час на реалізацію - 1 година



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

Завдання № 1 [VNS Lab 10 - Task 1](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/785/files#diff-2a533df19f11320da62e783cd8a3aa6f27ab1b199dcf01afb59cf2050cd56b80)

#include <iostream>

#include <fstream>

using namespace std;

template <class T>

class Node {

public:

T data;

Node\* prev;

Node\* next;

Node(T value) : data(value), prev(nullptr), next(nullptr) {}

};

template <class T>

class DoubleLinkedList {

private:

Node<T>\* head;

Node<T>\* tail;

public:

int size = 0;

void insert(int index, T value) {

Node<T>\* newNode = new Node<T>(value);

if (size == 0) {

head = tail = newNode;

tail->next = head;

head->prev = tail;

} else if (index == 0) {

newNode->next = head;

head->prev = newNode;

head = newNode;

tail->next = head;

head->prev = tail;

} else if(index == size) {

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

tail->next = head;

head->prev = tail;

} else {

Node<T>\* NodeToInsertBefore = head;

while (index > 0) {

NodeToInsertBefore = NodeToInsertBefore->next;

index--;

}

newNode->next = NodeToInsertBefore;

newNode->prev = NodeToInsertBefore->prev;

NodeToInsertBefore->prev->next = newNode;

NodeToInsertBefore->prev = newNode;

}

size++;

}

void erase(int index) {

Node<T>\* NodeToErase = head;

while (index > 0) {

NodeToErase = NodeToErase->next;

index--;

}

if(NodeToErase == head) {

head = NodeToErase->next;

tail->next = head;

head->prev = tail;

} else if(NodeToErase == tail) {

tail = NodeToErase->prev;

tail->next = head;

head->prev = tail;

} else {

NodeToErase->prev->next = NodeToErase->next;

NodeToErase->next->prev = NodeToErase->prev;

}

size--;

}

void print() {

if(size == 0)

{

cout << "List is empty" << endl;

return;

}

Node<T>\* current = head;

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

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

current = current->next;

}

cout << endl;

}

void writeToFile(string filename) {

ofstream file(filename, ios::binary | ios::trunc);

if (!file.is\_open()) {

cerr << "Error opening" << endl;

return;

}

Node<T>\* current = head;

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

file.write(reinterpret\_cast<char\*>(&current->data), sizeof(T));

current = current->next;

}

file.close();

}

void readFromFile(string filename) {

ifstream file(filename, ios::binary);

if (!file.is\_open()) {

cerr << "Error opening" << endl;

return;

}

while (!file.eof()) {

T value;

file.read(reinterpret\_cast<char\*>(&value), sizeof(T));

if (!file.fail()) {

insert(size, value);

}

}

file.close();

}

};

int main()

{

DoubleLinkedList<int> list;

int k, index;

cin >> k;

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

{

string input;

cin >> input;

if(input == "insert") {

int n;

cin >> index >> n;

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

{

int value;

cin >> value;

list.insert(index, value);

index++;

}

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

int n;

cin >> index >> n;

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

{

list.erase(index);

}

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

list.print();

}

}

/\*====================================================================================================\*/

/\*====================================================================================================\*/

list.writeToFile("list.bin");

int size = list.size;

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

{

list.erase(0);

}

cout << "List after erasing:" << endl;

list.print();

list.readFromFile("list.bin");

cout << "List after reading from file:" << endl;

list.print();

return 0;

}

Завдання № 2 [Algotester Lab 5v2](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/785/files#diff-fcd3d6cc33b29a4bea67fd0cf856922007135b8e12604c5f5d67db4e3436480b)

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n, m;

cin >> n >> m;

char cave[n][m];

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

{

cin.ignore();

string input;

cin >> input;

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

{

cave[i][j] = input[j];

}

}

for(int i=n-2; i>=0; i--)

{

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

{

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

{

int counter = 1, newRow = i;

while(i+counter<n && cave[i+counter][j] == 'O')

{

newRow++;

counter++;

}

cave[i][j] = 'O';

cave[newRow][j] = 'S';

}

}

}

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

{

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

{

cout << cave[i][j];

}

cout << endl;

}

return 0;

}

Завдання № 3 [Algotester Lab 78v1](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/785/files#diff-0125609c23706ed3aced8760b202fe8f67f384293a4dacad2557bbe0a7de56b5)

#include <iostream>

using namespace std;

template <class T>

class Node {

public:

T data;

Node\* prev;

Node\* next;

Node(T value) : data(value), prev(nullptr), next(nullptr) {}

};

template <class T>

class DoubleLinkedList {

private:

Node<T>\* head;

Node<T>\* tail;

public:

int size = 0;

void insert(int index, T value) {

Node<T>\* newNode = new Node<T>(value);

if (size == 0) {

head = tail = newNode;

tail->next = head;

head->prev = tail;

} else if (index == 0) {

newNode->next = head;

head->prev = newNode;

head = newNode;

tail->next = head;

head->prev = tail;

} else if(index == size) {

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

tail->next = head;

head->prev = tail;

} else {

Node<T>\* NodeToInsertBefore = head;

while (index > 0) {

NodeToInsertBefore = NodeToInsertBefore->next;

index--;

}

newNode->next = NodeToInsertBefore;

newNode->prev = NodeToInsertBefore->prev;

NodeToInsertBefore->prev->next = newNode;

NodeToInsertBefore->prev = newNode;

}

size++;

}

void erase(int index, int n) {

Node<T>\* NodeToErase = head;

while (index > 0) {

NodeToErase = NodeToErase->next;

index--;

}

if(NodeToErase == head) {

head = NodeToErase->next;

tail->next = head;

head->prev = tail;

} else if(NodeToErase == tail) {

tail = NodeToErase->prev;

tail->next = head;

head->prev = tail;

} else {

NodeToErase->prev->next = NodeToErase->next;

NodeToErase->next->prev = NodeToErase->prev;

}

size--;

}

void get(int index) {

Node<T>\* NodeToGet = head;

while (index > 0) {

NodeToGet = NodeToGet->next;

index--;

}

cout << NodeToGet->data << endl;

}

void set(int index, T value) {

Node<T>\* NodeToSet = head;

while (index > 0) {

NodeToSet = NodeToSet->next;

index--;

}

NodeToSet->data = value;

}

void print() {

Node<T>\* current = head;

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

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

current = current->next;

}

cout << std::endl;

}

};

int main()

{

DoubleLinkedList<int> list;

int k, index;

cin >> k;

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

{

string input;

cin >> input;

if(input == "insert") {

int n;

cin >> index >> n;

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

{

int value;

cin >> value;

list.insert(index, value);

index++;

}

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

int n;

cin >> index >> n;

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

{

list.erase(index, n);

}

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

cout << list.size << endl;

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

cin >> index;

list.get(index);

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

int value;

cin >> index >> value;

list.set(index, value);

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

list.print();

}

}

return 0;

}

Завдання № [4-1 Class practice work](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/785/files#diff-832c76f3f08b69bc2a5ba4011b36ce9767a90c63de6e4f61c28d1749a00019f5)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;

};

int size(Node\* head)

{

Node\* current = head;

int size = 0;

while (current != head->prev)

{

size++;

current = current->next;

}

size++;

return size;

}

Node\* reverse(Node\* head)

{

Node\* newHead = new Node;

Node\* newTail = newHead;

Node\* currentData = head->prev;

newHead->data = currentData->data;

newHead->next = nullptr;

newHead->prev = newTail;

newTail->next = newHead;

newTail = newHead;

currentData = currentData->prev;

while(currentData != head->prev)

{

Node\* currentInRevList = new Node;

currentInRevList->data = currentData->data;

currentInRevList->next = nullptr;

currentInRevList->prev = newTail;

newTail->next = currentInRevList;

newTail = currentInRevList;

currentData = currentData->prev;

}

newHead->prev = newTail;

newTail->next = newHead;

return newHead;

}

bool comapre(Node\* head1, Node\* head2)

{

int size1 = size(head1);

int size2 = size(head2);

if (size1 != size2)

{

return false;

}

Node\* current1 = head1;

Node\* current2 = head2;

while(current1 != head1->prev)

{

if (current1->data != current2->data)

{

return false;

}

current1 = current1->next;

current2 = current2->next;

}

return true;

}

Node\* add(Node\* head1, Node\* head2) {

Node\* sumHead = new Node;

Node\* sumTail = sumHead;

Node\* tail1 = head1->prev;

Node\* tail2 = head2->prev;

Node\* current1 = tail1;

Node\* current2 = tail2;

vector<int> sum;

int counter = 0;

int curr2data = current2->data;

int curr1data = current1->data;

if(size(head1) >= size(head2))

{

while(current1 != head1)

{

if(counter >= size(head2))

{

curr2data = 0;

} else {

curr2data = current2->data;

}

sum.push\_back(current1->data + curr2data);

current1 = current1->prev;

current2 = current2->prev;

counter++;

}

if(counter >= size(head2))

{

curr2data = 0;

} else {

curr2data = current2->data;

}

sum.push\_back(current1->data + curr2data);

} else {

while(current2 != head2)

{

if(counter >= size(head1))

{

curr1data = 0;

} else {

curr1data = current1->data;

}

sum.push\_back(current2->data + curr1data);

current1 = current1->prev;

current2 = current2->prev;

counter++;

}

if(counter >= size(head1))

{

curr1data = 0;

} else {

curr1data = current1->data;

}

sum.push\_back(current2->data + curr1data);

}

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

{

if(sum[i] > 9)

{

sum[i] -= 10;

if(i+1 == sum.size())

{

sum.push\_back(1);

} else {

sum[i+1] += 1;

}

}

}

sumHead->next = nullptr;

sumHead->prev = nullptr;

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

{

Node\* current = new Node;

current->data = sum[i];

current->next = nullptr;

current->prev = sumTail;

sumTail->next = current;

sumTail = current;

}

sumHead = sumHead->next;

sumHead->prev = sumTail;

sumTail->next = sumHead;

return sumHead;

}

void print(Node\* head)

{

Node\* current = head;

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

current = current->next;

while (current != head)

{

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

current = current->next;

}

cout << endl;

}

int main()

{

int n;

cout << "Enter n: ";

cin >> n;

cout << "Enter list #1: ";

Node\* head1 = new Node;

Node\* tail1 = head1;

cin >> head1->data;

head1->next = nullptr;

head1->prev = nullptr;

for (int i = 1; i < n; i++) {

Node\* current1 = new Node;

cin >> current1->data;

current1->next = nullptr;

current1->prev = tail1;

tail1->next = current1;

tail1 = current1;

}

head1->prev = tail1;

tail1->next = head1;

//LIST #2

cout << "Enter k: ";

cin >> n;

cout << "Enter list #2: ";

Node\* head2 = new Node;

Node\* tail2 = head2;

cin >> head2->data;

head2->next = nullptr;

head2->prev = nullptr;

for (int i = 1; i < n; i++) {

Node\* current2 = new Node;

cin >> current2->data;

current2->next = nullptr;

current2->prev = tail2;

tail2->next = current2;

tail2 = current2;

}

head2->prev = tail2;

tail2->next = head2;

cout << endl;

cout << "Reversed list #1: ";

print(reverse(head1));

cout << endl;

if(comapre(head1, head2))

{

cout << "Lists are equal" << endl;

}

else

{

cout << "Lists are not equal" << endl;

}

cout << endl;

cout << "Sum of lists: ";

print(add(head1, head2));

}

Завдання № [4-2 Class practice work](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/785/files#diff-771ee414bd91a7707d0713f41009b35f718282c3ef328de6b99fbdb6c6c8ceb1)

#include <iostream>

#include <stack>

using namespace std;

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

TreeNode\* parent;

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

};

TreeNode\* mirror(TreeNode\* root) {

if (root == nullptr) {

return nullptr;

}

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

newRoot->right = mirror(root->left);

newRoot->left = mirror(root->right);

return newRoot;

}

void tree\_sum(TreeNode\* Node) {

if (Node == nullptr) {

return;

}

tree\_sum(Node->left);

tree\_sum(Node->right);

int newData = 0;

if (Node->left != nullptr) {

newData += Node->left->data;

}

if (Node->right != nullptr) {

newData += Node->right->data;

}

if(Node->left != nullptr || Node->right != nullptr)

{

Node->data = newData;

}

}

void insert(TreeNode\* root, int value) {

TreeNode\* current = root;

TreeNode\* parent = nullptr;

while (current != nullptr) {

parent = current;

if (value < current->data) {

current = current->left;

}

else {

current = current->right;

}

}

TreeNode\* newNode = new TreeNode(value);

newNode->parent = parent;

if (value < parent->data) {

parent->left = newNode;

}

else {

parent->right = newNode;

}

}

// "In order" output

void print(TreeNode\* Node) {

if (Node != nullptr) {

print(Node->left);

cout << Node->data << " ";

print(Node->right);

}

}

int main() {

cout << "Enter the number of elements: ";

int n;

cin >> n;

cout << "Enter the elements: ";

int value;

cin >> value;

TreeNode\* root = new TreeNode(value);

for (int i = 1; i < n; i++) {

cin >> value;

insert(root, value);

}

cout << "Original tree: ";

print(root);

cout << endl;

cout << "Mirrored tree: ";

print(mirror(root));

cout << endl;

cout << "Updated tree: ";

tree\_sum(root);

print(root);

return 0;

}

Завдання № 6 [Self-practice(Algotester Lab5v3)](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/785/files#diff-bec6d4040a7cadcc3c30b9cf030515ad883236bf3a30555bbdd45f060b616907)

#include <iostream>

using namespace std;

int main()

{

int n, m, x, y;

cin >> n >> m >> x >> y;

int map[n][m];

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

{

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

{

map[i][j] = 0;

}

}

int maxX;

if(x > n-x)

{

maxX = x-1;

} else {

maxX = n-x;

}

int maxY;

if(y > m-y)

{

maxY = y-1;

} else {

maxY = m-y;

}

int max = maxX + maxY;

x--;

y--;

map[x][y] = max;

int elem = max;

for(int i=1; i<m; i++)

{

elem--;

if(y+i < m)

{

map[x][y+i] = elem;

}

if(y-i >= 0)

{

map[x][y-i] = elem;

}

}

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

{

elem = map[x][i];

for(int j=1; j<n; j++)

{

elem--;

if(x+j < n)

{

map[x+j][i] = elem;

}

if(x-j >= 0)

{

map[x-j][i] = elem;

}

}

}

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

{

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

{

cout << map[i][j] << ' ';

}

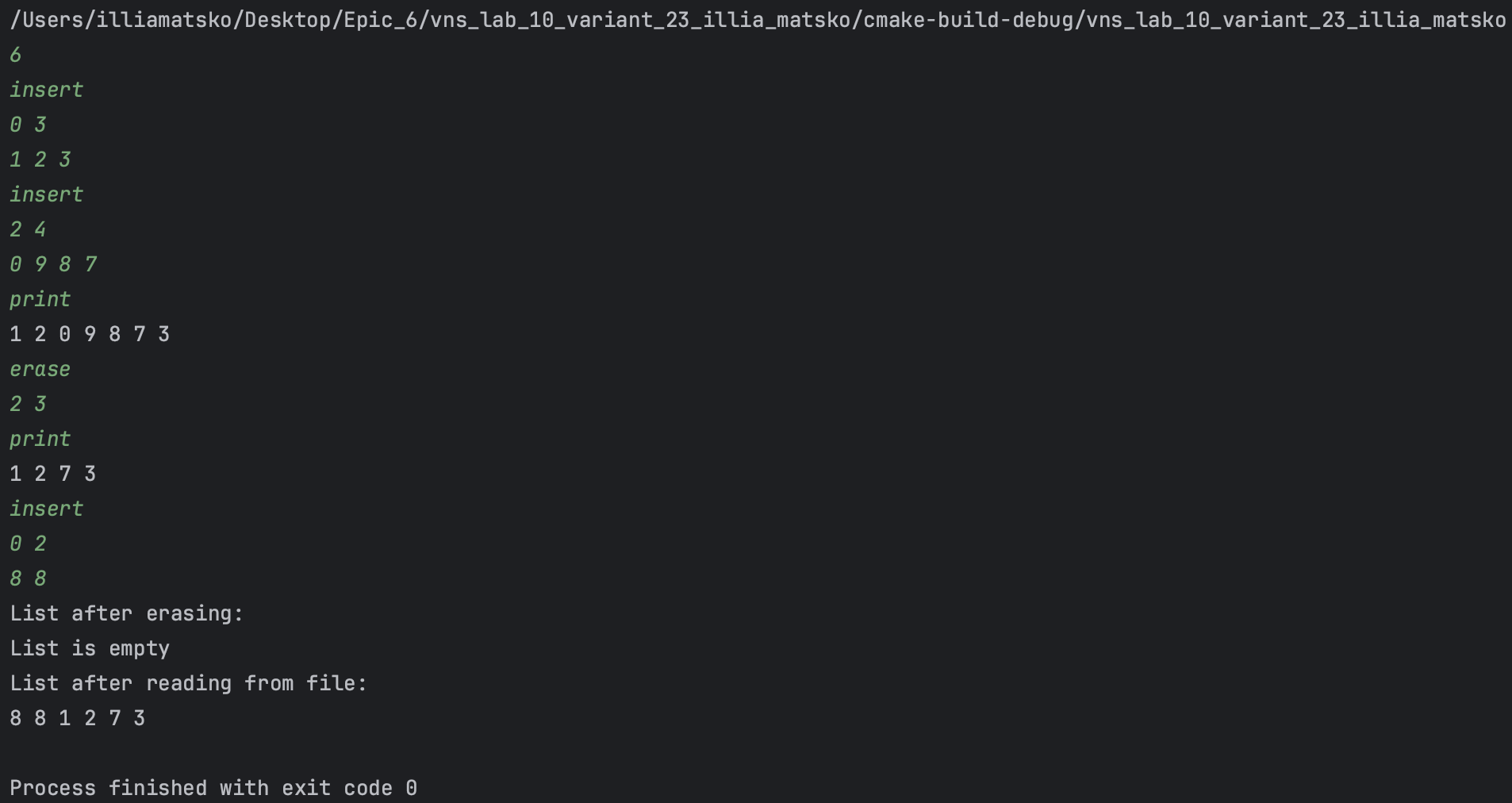
cout << endl;

}

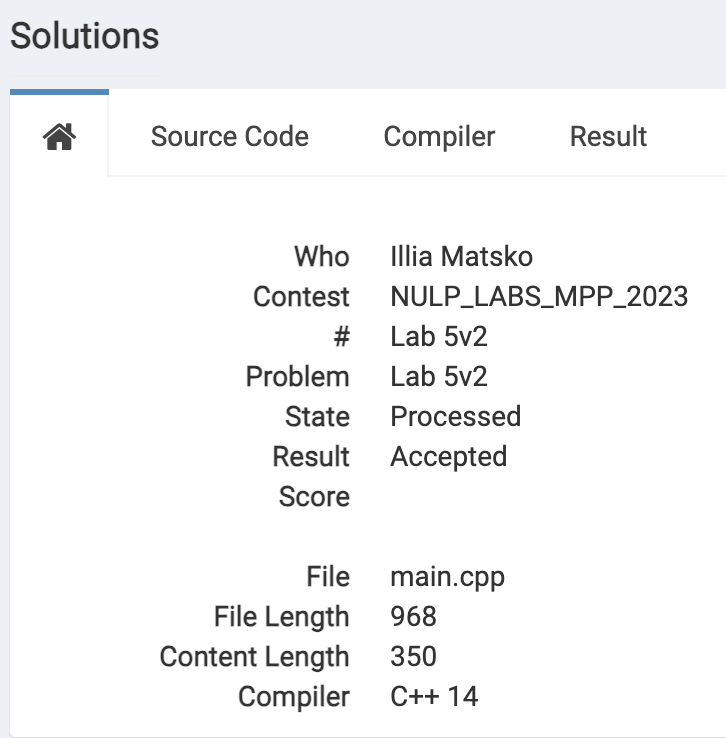
}

1. **Результати виконання завдань, тестування та фактично витрачений час:**

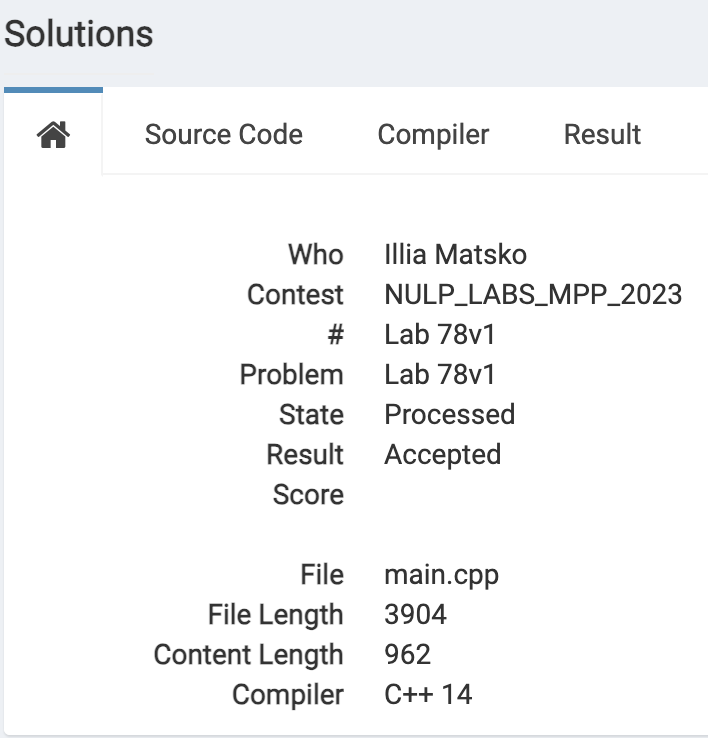
**Завдання № 1**

****

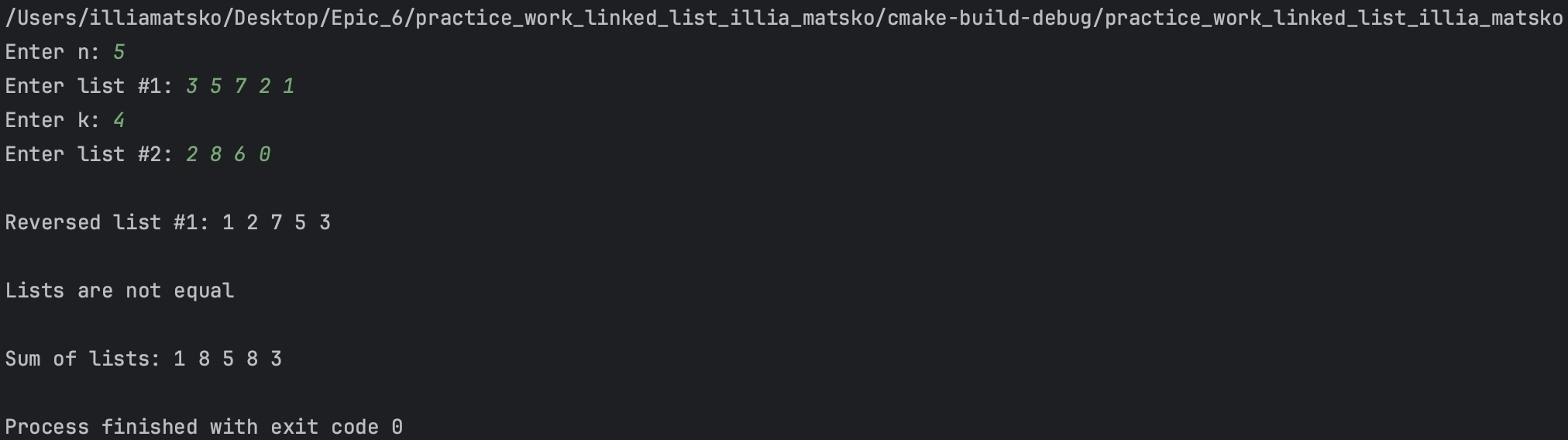
**Завдання № 2**

****

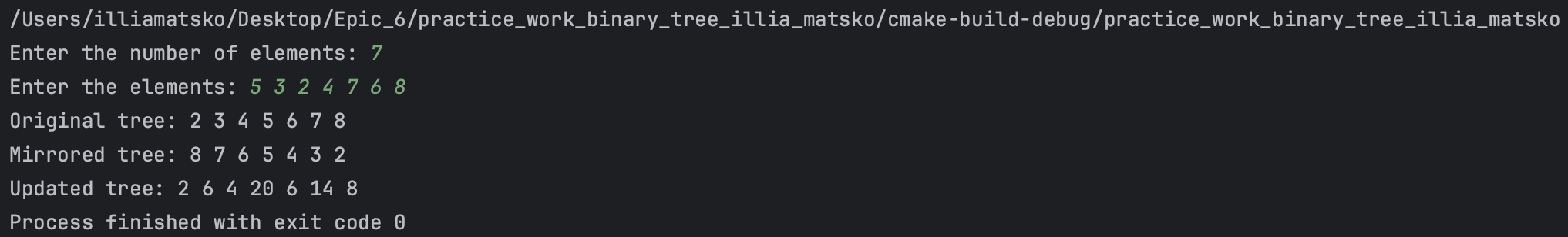
**Завдання № 3**

****

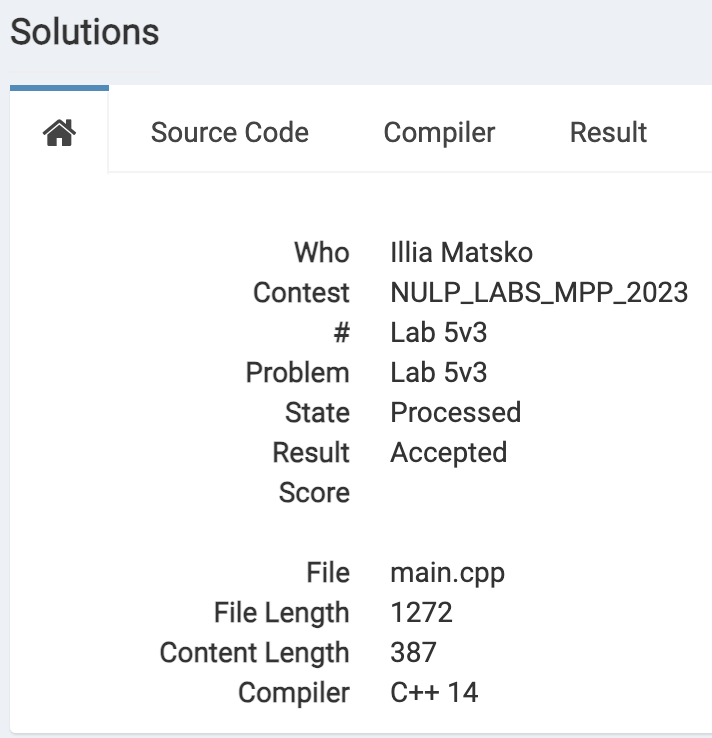
**Завдання № 4-1**

****

**Завдання № 4-2**

****

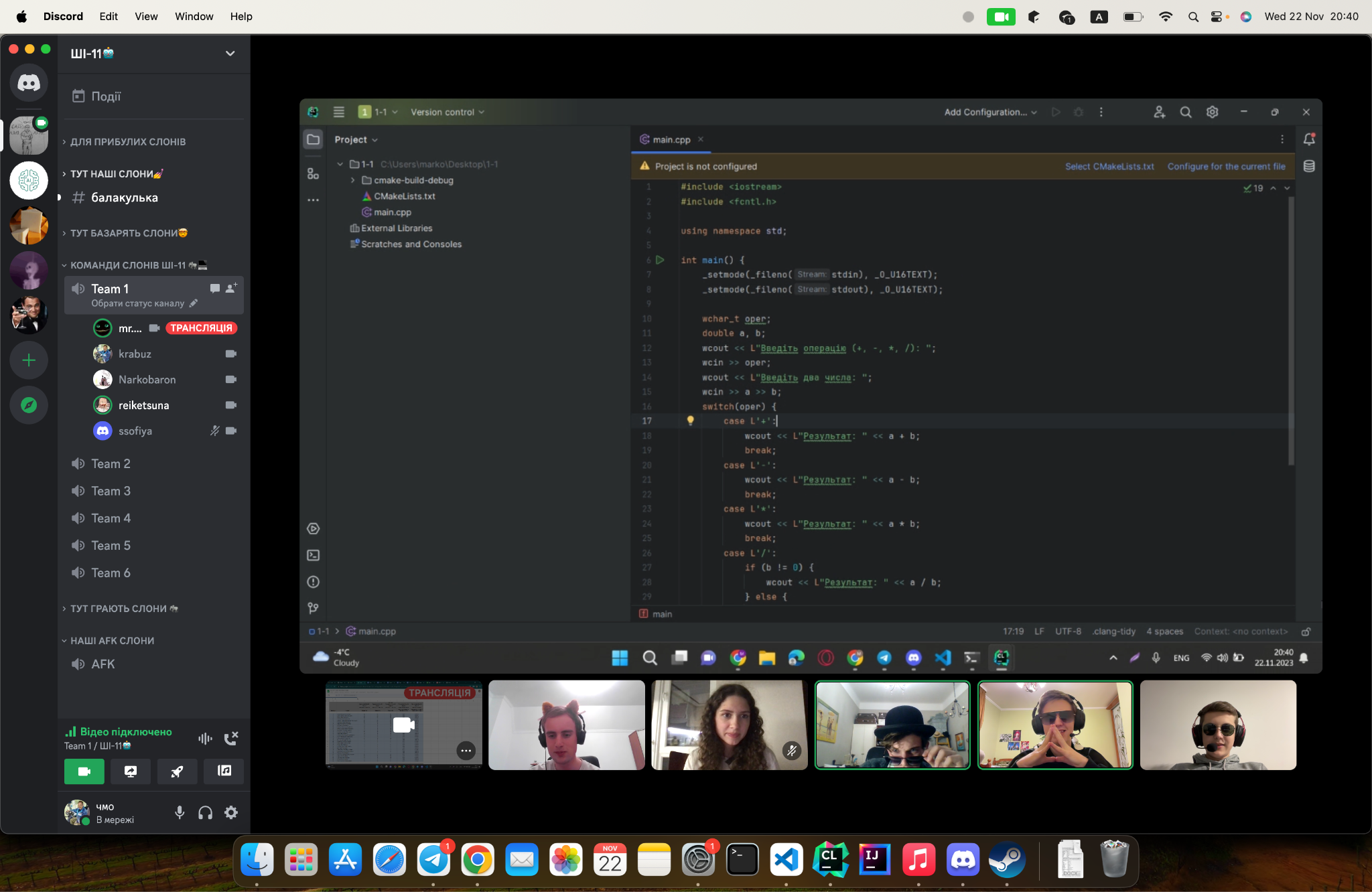
**Завдання № 5**

****

[**Посилання на Pull-request**](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/785/files)

Робота з командою:





# **Висновки:**

У ході даної роботи я розглянув основні динамічні структури даних у програмуванні, такі як стек, черга, список та дерево. Вивчив їхні принципи роботи та застосування. Далі, розглянув основні алгоритми обробки цих структур, включаючи пошук, вставку, видалення та сортування. Ця інформація стане основою для розробки програм на мові програмування C++, де використовуються ці динамічні структури для оптимального управління та обробки даних. За допомогою наданого матеріалу можна створювати ефективні програми, враховуючи особливості кожної структури та вибираючи відповідні алгоритми для конкретних завдань.