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.»

***Виконала:***

студентка групи ШІ-13

Присліпська Софія Андріївна

# **Тема роботи:**

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

# **Мета роботи:**

Task 1 - Theory Education Activities

Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)

Task 3 - Lab# programming: VNS Lab 10

Task 4 - Lab# programming: Algotester Lab 5

Task 5 - Lab# programming: Algotester Lab 7-8

Task 6 - Practice# programming: Class Practice Task

Task 7 - Practice# programming: Self Practice Task

Task 8 - Result Documentation Report and Outcomes Placement Activities (Docs and Programs on GitHub)

Task 9 - Results Evaluation and Release

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Види динамічних структур
* Тема №2: Одно- та двозв’язні списки
* Тема №3: Бінарне дерево

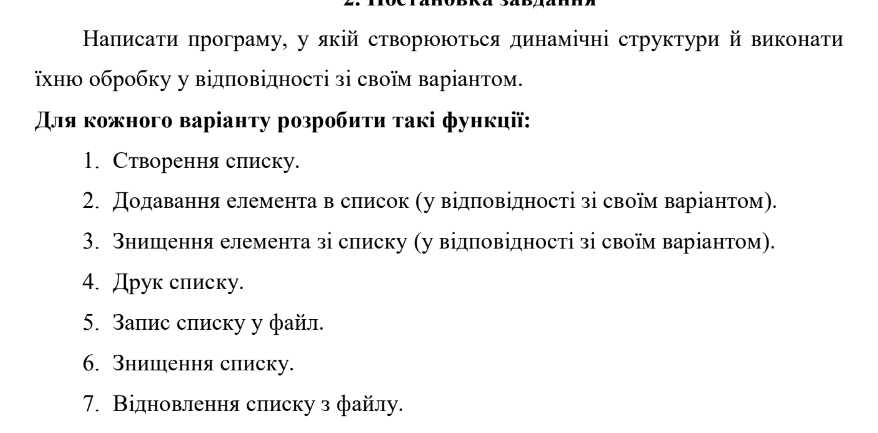
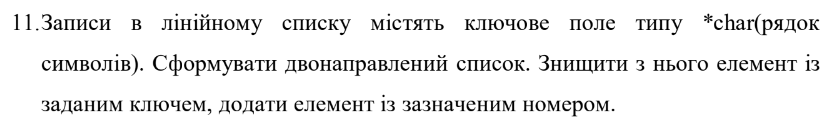
1. Індивідуальний план опрацювання теорії:

* Тема №1: Види динамічних структур
  + Джерела Інформації
    - <https://youtu.be/eSxLVD5vfqM?si=8OiPGfRKIazU-N8z>
    - <https://youtu.be/jH9ZE5uAal8?si=xdq9aK04b9Brv7Eq>
    - <https://youtu.be/5uAwaQL99cY?si=fn3dSlgevFYkswRp>
  + Що опрацьовано: Опрацьовано матеріали надані вище. Розібрано в чому різниця між різними динамічними структурами
  + Статус: Ознайомлена
  + Початок опрацювання теми: 6.12.2023
  + Звершення опрацювання теми: 14.12.2023
* Тема №2: Одно- та двозв’язні списки.
  + Джерела Інформації:
    - <https://youtu.be/N6dOwBde7-M?si=UrtPdTS4PvXcXLJc>
    - <https://youtu.be/vcQIFT79_50?si=9N_GPcCRgntvKedH>
  + Що опрацьовано: Опрацьовано матеріали надані вище. Попрактикувалась в імплементації linked list
  + Статус: Ознайомлена
  + Початок опрацювання теми: 6.12.2023
  + Звершення опрацювання теми: 14.12.2023
* Тема №3: Бінарне дерево
  + Джерела Інформації:
    - <https://youtu.be/Krr9J-_7IS0?si=A9QJWtCCxu9er9xI>
    - [https://youtu.be/qBFzNW0ALxQ?si=RgZR7\_W-PYHTQLz0](https://youtu.be/qBFzNW0ALxQ?si=RgZR7_W-PYHTQLz0%20)
    - <https://youtu.be/_IhTp8q0Mm0?si=xzi_Eb0rwYpFt3ib>
  + Що опрацьовано: Опрацьовано матеріали надані вище.
  + Статус: Ознайомлена
  + Початок опрацювання теми: 6.12.2023
  + Звершення опрацювання теми: 14.12.2023

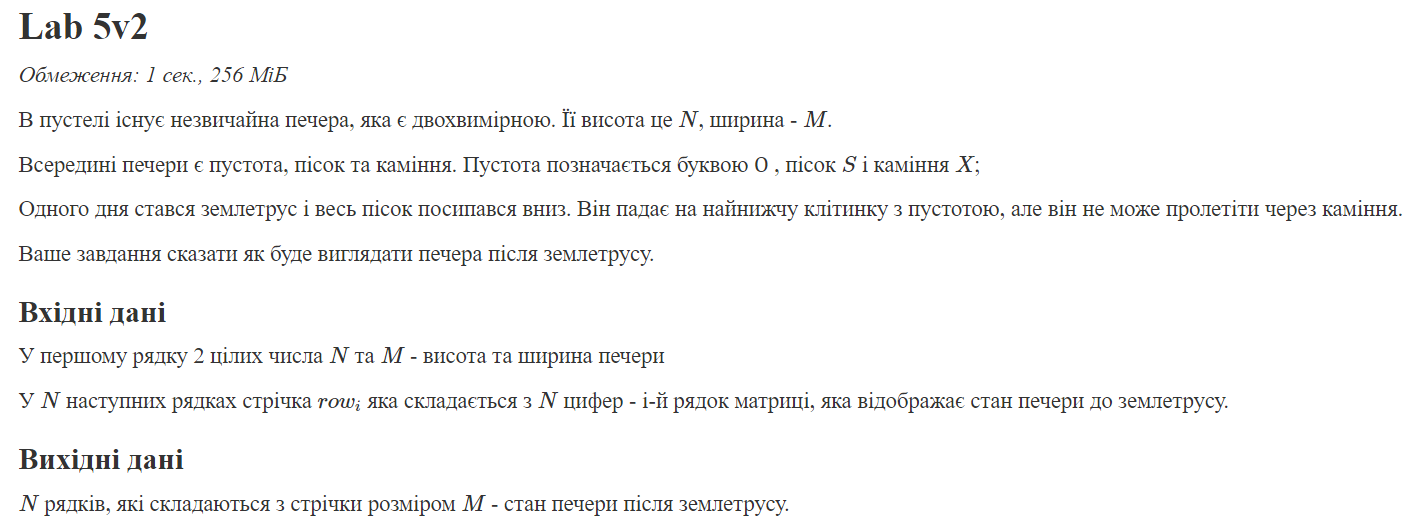
# **Виконання роботи:**

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

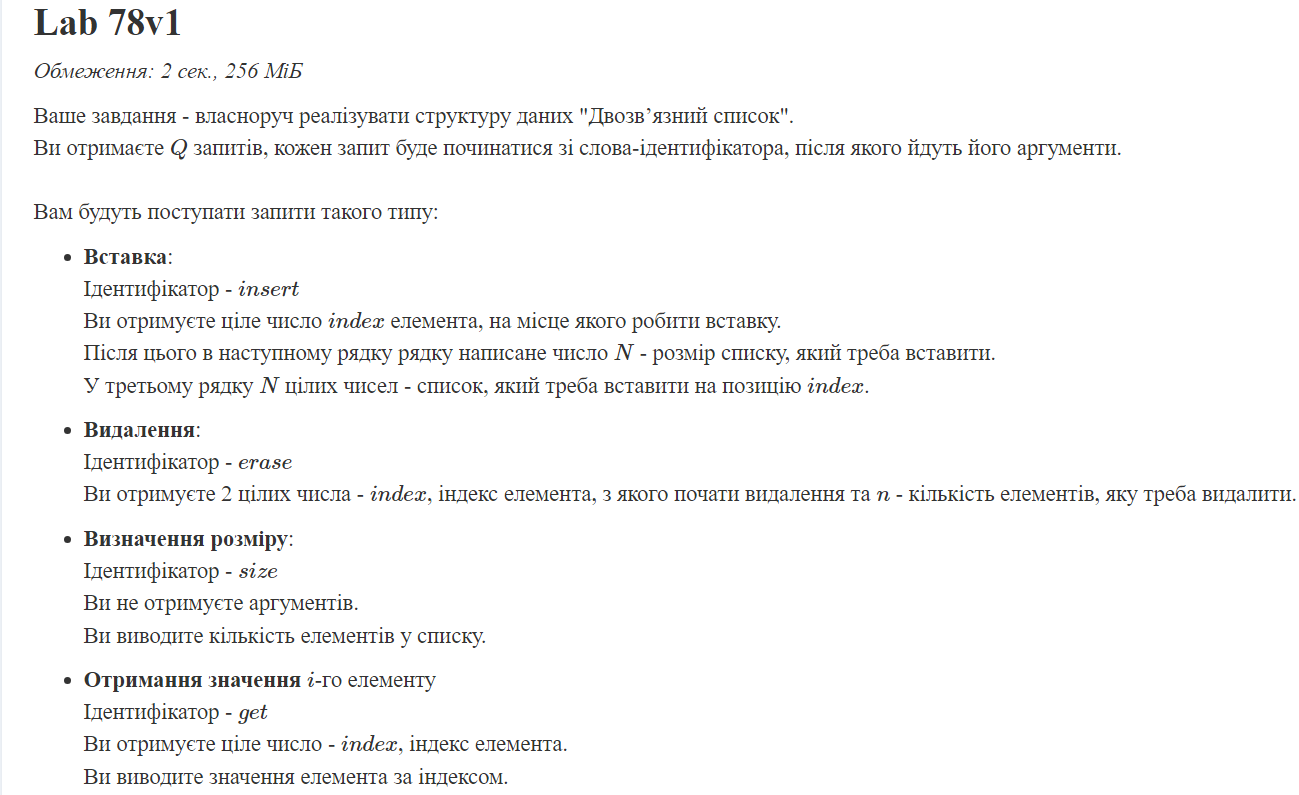
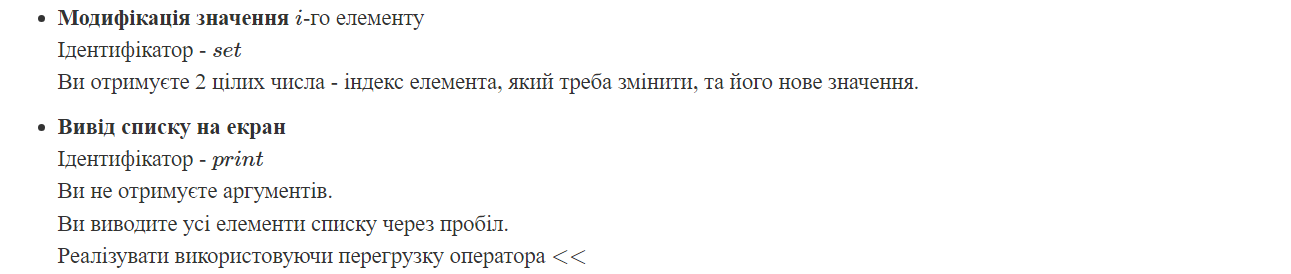
Завдання №1 **«Лабораторна робота №10 VNS»**

* Варіант завдання:11
* 
* *Figure 1.*
* 
* *Figure 2.*

Завдання №2 **«Algotester Lab 5v2»**

* Варіант завдання:2
* 
* *Figure 3.*

Завдання №3 **«Algotester 78v1»**

* Варіант завдання:1
* 
* 
* *Figure 4.*

Завдання №4 **«Practice Task 1»**

## **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 №5 **«Practice Task 2»**

## **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 №6 **«Practice Task 3»**

## **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 №7 **«Practice Task 4»**

## **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 №8 **«Practice Task 5»**

## **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

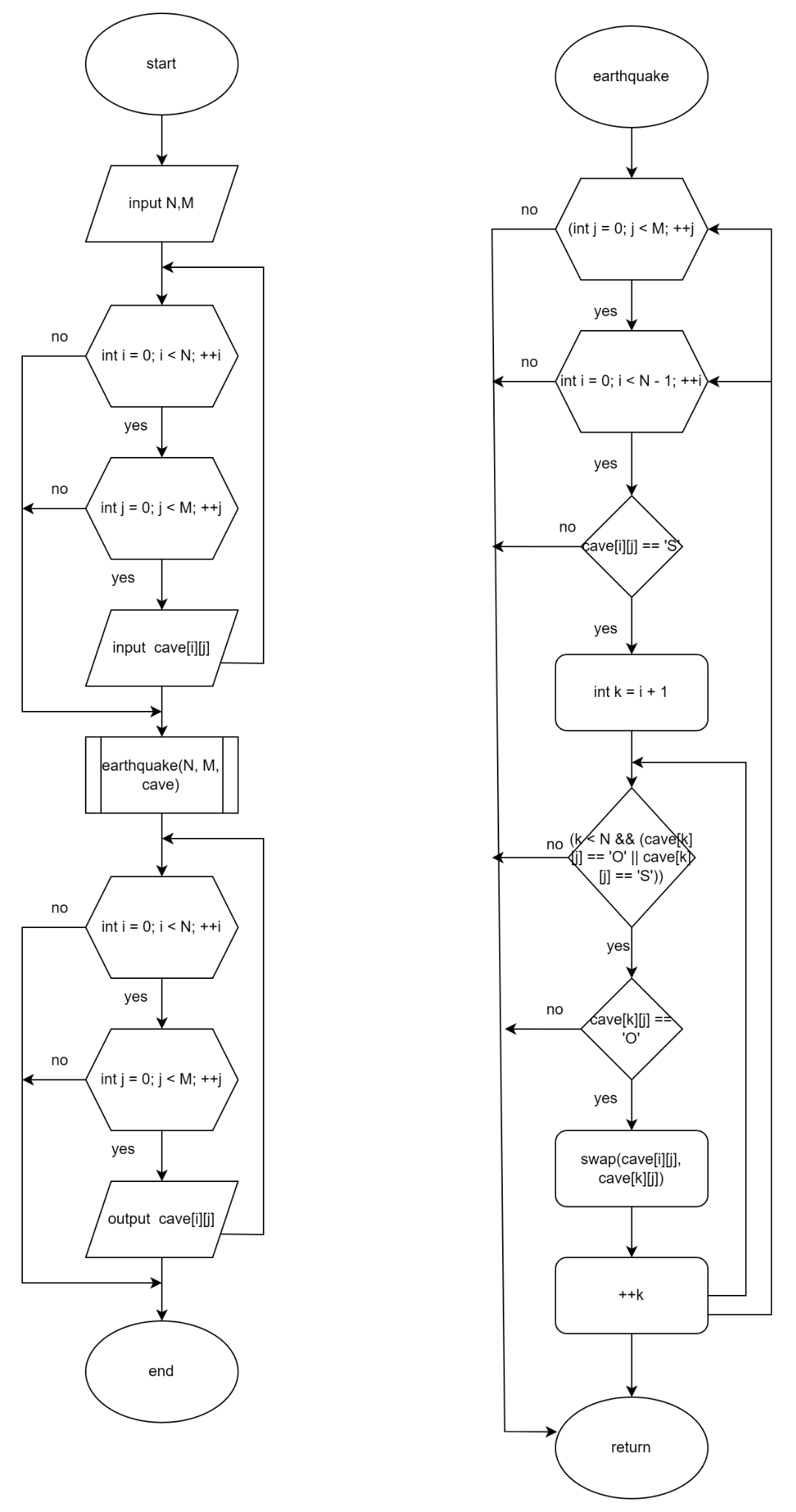
- значення змінюються від листків до кореня дерева

Завдання №9 **«Self Practice»**

Робота зі стеками, додавання та видаляння елементів.

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

Програма №1 **«Algotester Lab 5v2»**

****

* *Figure 5.*

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

<https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1165>

Завдання №1 **Лабораторна робота №10 VNS»**

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**using namespace std;**

**// Структура елемента списку**

**struct Node {**

**string data;**

**Node\* prev;**

**Node\* next;**

**};**

**// Клас двонаправленого списку**

**class DoublyLinkedList {**

**private:**

**Node\* head;**

**Node\* tail;**

**public:**

**// Конструктор**

**DoublyLinkedList() : head(nullptr), tail(nullptr) {}**

**// Функція для додавання елемента в кінець списку**

**void addElement(const string& value) {**

**Node\* newNode = new Node();**

**newNode->data = value; // призначення значення value полю data**

**newNode->prev = tail;**

**newNode->next = nullptr;**

**if (!head) {**

**head = newNode;**

**} else {**

**tail->next = newNode;**

**}**

**tail = newNode;**

**}**

**// Функція для видалення елемента зі списку за заданим ключем (рядком символів)**

**void deleteElement(const string& key) {**

**Node\* current = head;**

**while (current) {**

**if (current->data == key) {**

**if (current->prev) {**

**current->prev->next = current->next;**

**} else {**

**head = current->next;**

**}**

**if (current->next) {**

**current->next->prev = current->prev;**

**} else {**

**tail = current->prev;**

**}**

**delete current;**

**cout << "Element with a key'" << key << "'deleted from the list." << endl;**

**return;**

**}**

**current = current->next;**

**}**

**cout << "Element with a key'" << key << "'not found in the list." << endl;**

**}**

**// Функція для виводу списку**

**void printList() const {**

**if (!head) {**

**cout << "The list is empty" << endl;**

**return;**

**}**

**Node\* current = head;**

**while (current) {**

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

**current = current->next;**

**}**

**cout << endl;**

**}**

**// Функція для запису списку у файл**

**void saveToFile(const string& filename) const {**

**ofstream outFile(filename);**

**if (outFile.is\_open()) {**

**Node\* current = head;**

**while (current) {**

**outFile << current->data << "\n";**

**current = current->next;**

**}**

**outFile.close();**

**cout << "List successfully saved to file '" << filename << "'." << endl;**

**} else {**

**cout << "Could not open file for writing." << endl;**

**}**

**}**

**// Функція для знищення списку**

**void clearList() {**

**Node\* current = head;**

**while (current) {**

**Node\* temp = current;**

**current = current->next;**

**delete temp;**

**}**

**head = nullptr;**

**tail = nullptr;**

**cout << "The list is empty." << endl;**

**}**

**// Функція для відновлення списку з файлу**

**void restoreFromFile(const string& filename) {**

**clearList();**

**ifstream inFile(filename);**

**if (inFile.is\_open()) {**

**string line;**

**while (getline(inFile, line)) {**

**addElement(line);**

**}**

**inFile.close();**

**cout << "List restored from file '" << filename << "'." << endl;**

**} else {**

**cout << "Failed to open recovery file." << endl;**

**}**

**}**

**};**

**int main() {**

**DoublyLinkedList myList;**

**// Додавання елементів у список**

**myList.addElement("element1");**

**myList.addElement("element2");**

**myList.addElement("element3");**

**// Друк списку**

**cout << "List after adding elements: ";**

**myList.printList();**

**// Видалення елемента зі списку**

**myList.deleteElement("element2");**

**// Друк списку після видалення елемента**

**cout << "List after deleting an element: ";**

**myList.printList();**

**// Запис списку у файл**

**myList.saveToFile("myList.txt");**

**// Знищення списку та вивід порожнього списку**

**myList.clearList();**

**myList.printList();**

**// Відновлення списку з файлу**

**myList.restoreFromFile("myList.txt");**

**// Друк відновленого списку**

**cout << "Restored list: ";**

**myList.printList();**

**return 0;**

**}**

Завдання №2 **«Algotester Lab 5v2»**

#include <iostream>

#include <vector>

using namespace std;

void earthquake(int N, int M, vector<vector<char>>& cave) {

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

for (int i = 0; i < N - 1; ++i) {

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

int k = i + 1;

while (k < N && (cave[k][j] == 'O' || cave[k][j] == 'S')) {

if (cave[k][j] == 'O') {

swap(cave[i][j], cave[k][j]);

break;

}

++k;

}

}

}

}

}

int main() {

int N, M; //висота і ширина

cin >> N >> M;

vector<vector<char>> cave(N, vector<char>(M));

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

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

cin >> cave[i][j];

}

}

earthquake(N, M, cave);

// Виведення результату

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 78v1»**

#include <iostream>

using namespace std;

class DoublyLinkedList

{

private:

struct Node

{

public:

int data;

Node \*prev;

Node \*next;

};

Node \*head;

Node \*tail;

int size;

public:

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

void insert(int index, int N)

{

Node \*newNode = new Node{N};

if (index == 0)

{

newNode->next = head;

if (head != nullptr)

head->prev = newNode;

head = newNode;

if (tail == nullptr)

tail = newNode;

}

else if (index != size)

{

Node \*temp = head;

for (int i = 0; i < index - 1; ++i)

{

temp = temp->next;

}

newNode->next = temp->next;

newNode->prev = temp;

temp->next->prev = newNode;

temp->next = newNode;

}

else

{

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

size++;

}

void erase(int index, int n)

{

Node \*temp = head;

while (index--)

{

temp = temp->next;

}

while (n--)

{

if (temp->prev == nullptr)

{

head = temp->next;

}

else

{

temp->prev->next = temp->next;

}

if (temp->next == nullptr)

{

tail = temp->prev;

}

else

{

temp->next->prev = temp->prev;

}

Node \*tmp = temp;

temp = temp->next;

delete tmp;

size--;

}

}

int getsize() const

{

return size;

}

int get(int index) const

{

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

return -1;

Node \*temp = head;

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

{

temp = temp->next;

}

return temp->data;

}

void set(int index, int value)

{

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

return;

Node \*temp = head;

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

{

temp = temp->next;

}

temp->data = value;

}

void print() const

{

Node \*temp = head;

while (temp != nullptr)

{

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

};

int main()

{

int Q;

cin >> Q;

DoublyLinkedList list;

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

{

string request;

cin >> request;

if (request == "insert")

{

int index, N;

cin >> index >> N;

int arr[N];

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

{

cin >> arr[i];

}

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

{

list.insert(index, arr[i]);

}

}

else if (request == "erase")

{

int index, n;

cin >> index >> n;

list.erase(index, n);

}

else if (request == "size")

{

cout << list.getsize() << endl;

}

else if (request == "get")

{

int index;

cin >> index;

int value = list.get(index);

if (value != -1)

cout << value << endl;

}

else if (request == "set")

{

int index, value;

cin >> index >> value;

list.set(index, value);

}

else if (request == "print")

{

list.print();

}

}

return 0;

}

Завдання №4 **«Practice Task 1»**

//Реверс списку

#include <iostream>

using namespace std;

// Вузол однозв'язного списку

struct Node {

int data;

Node\* next;

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

};

// Клас для роботи зі списком

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

// Метод додавання елемента в кінець списку

void addNode(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Метод виведення списку

void printList(Node\* start) {

while (start != nullptr) {

cout << start->data << " ";

start = start->next;

}

cout << endl;

}

// Метод реверсу списку

Node\* reverse(Node\* head) {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev; // Нова голова списку (після реверсу)

}

// Метод реверсування списку та виведення його на екран

void reverseAndPrint() {

cout << "Incoming list: ";

printList(head);

head = reverse(head);

cout << "reversed list: ";

printList(head);

}

};

int main() {

LinkedList list;

list.addNode(1);

list.addNode(2);

list.addNode(3);

list.addNode(4);

list.addNode(5);

list.reverseAndPrint();

return 0;

}

Завдання №5 **«Practice Task 2»**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

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

};

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

while (h1 != nullptr && h2 != nullptr) {

if (h1->data != h2->data) {

return false; // Якщо дані в вузлах не співпадають

}

h1 = h1->next;

h2 = h2->next;

}

// Якщо довжина списків різна, повертаємо false

if (h1 != nullptr || h2 != nullptr) {

return false;

}

return true; // Якщо всі елементи співпадають і списки однакової довжини, повертаємо true

}

void printList(Node\* head) {

while (head != nullptr) {

cout << head->data << " ";

head = head->next;

}

cout << endl;

}

int main() {

//створити два списки по три вузла

Node\* head1 = new Node(1);

head1->next = new Node(2);

head1->next->next = new Node(3);

Node\* head2 = new Node(1);

head2->next = new Node(2);

head2->next->next = new Node(3);

// Додати додатковий вузол до другого списку, щоб довжини були різні

head2->next->next->next = new Node(4);

cout << "list 1: ";

printList(head1);

cout << "list 2: ";

printList(head2);

bool result = compare(head1, head2);

if (result) {

cout << "The lists are the same." << endl;

} else {

cout << "The lists are different." << endl;

}

// Очищення пам'яті

while (head1 != nullptr) {

Node\* temp = head1;

head1 = head1->next;

delete temp;

}

while (head2 != nullptr) {

Node\* temp = head2;

head2 = head2->next;

delete temp;

}

return 0;

}

Завдання №6 **«Practice Task 3»**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

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

};

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

Node\* result = nullptr;

Node\* current = nullptr;

int carry = 0;

while (n1 != nullptr || n2 != nullptr || carry != 0) {

int sum = carry;

if (n1 != nullptr) {

sum += n1->data;

n1 = n1->next;

}

if (n2 != nullptr) {

sum += n2->data;

n2 = n2->next;

}

carry = sum / 10; // Отримуємо розряд для переносу

sum = sum % 10; // Отримуємо поточний розряд для нового вузла

if (result == nullptr) {

result = new Node(sum);

current = result;

} else {

current->next = new Node(sum);

current = current->next;

}

}

return result;

}

void printList(Node\* head) {

while (head != nullptr) {

cout << head->data << " ";

head = head->next;

}

cout << endl;

}

int main() {

Node\* num1 = new Node(9);

num1->next = new Node(7);

num1->next->next = new Node(3);

Node\* num2 = new Node(6);

num2->next = new Node(8);

num2->next->next = new Node(2);

cout << "number 1: ";

printList(num1);

cout << "number 2: ";

printList(num2);

Node\* sum = add(num1, num2);

cout << "the sum of numbers: ";

printList(sum);

// Очищення пам'яті

while (num1 != nullptr) {

Node\* temp = num1;

num1 = num1->next;

delete temp;

}

while (num2 != nullptr) {

Node\* temp = num2;

num2 = num2->next;

delete temp;

}

while (sum != nullptr) {

Node\* temp = sum;

sum = sum->next;

delete temp;

}

return 0;

Завдання №7 **«Practice Task 4»**

#include <iostream>

using namespace std;

// Структура вузла бінарного дерева

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

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

};

// Функція для створення дерева з віддзеркаленням

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

if (root == nullptr) {

return nullptr;

}

// Створення нового вузла з обміненими лівим і правим піддеревами

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

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

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

return newNode;

}

// Функція для виводу бінарного дерева (приклад)

void printTree(TreeNode\* root) {

if (root == nullptr) {

return;

}

cout << root->data << " ";

printTree(root->left);

printTree(root->right);

}

int main() {

// Створення бінарного дерева для прикладу

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);

cout << "A tree before reflection: ";

printTree(root);

cout << endl;

// Створення віддзеркаленого дерева

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

cout << "A tree after reflection: ";

printTree(mirroredRoot);

cout << endl;

return 0;

}

Завдання №8 **«Practice Task 5»**

#include <iostream>

#include <queue>

using namespace std;

struct TreeNode { //представляє вузол бін дерева

int val; //значення вузла

TreeNode\* left; //покажчики на дочіпні вузли

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

void tree\_sum(TreeNode\* root) {

if (root == nullptr) {

return;

}

queue<TreeNode\*> q;

q.push(root); //додає поч. вузол до черги

while (!q.empty()) {

TreeNode\* current = q.front();//повертає посилання на перший елемент у черзі

q.pop();// видаляє перший елемент з черги

if (current->left != nullptr) {

q.push(current->left);

}

if (current->right != nullptr) {

q.push(current->right);

}

if (current->left != nullptr || current->right != nullptr) {

int sum = 0;

if (current->left != nullptr) {

sum += current->left->val;

}

if (current->right != nullptr) {

sum += current->right->val;

}

current->val = sum;

}

}

}

void printTree(TreeNode\* root) {

if (root == nullptr) {

return;

}

printTree(root->left);

cout << root->val << " ";

printTree(root->right);

}

int main() {

TreeNode\* root = new TreeNode(10);

root->left = new TreeNode(5);

root->right = new TreeNode(15);

root->left->left = new TreeNode(3);

root->left->right = new TreeNode(8);

root->right->left = new TreeNode(12);

root->right->right = new TreeNode(20);

cout << "Before applying tree\_sum: ";

printTree(root);

cout << endl;

tree\_sum(root);

cout << "After applying tree\_sum: ";

printTree(root);

cout << endl;

return 0;}

Завдання №9 **«Self Practice»**

**#include <iostream>**

**using namespace std;**

**struct Node {**

**int data;**

**Node\* next;**

**};**

**class Stack {**

**private:**

**Node\* top; // верхівка стеку**

**public:**

**Stack() : top(nullptr) {}**

**// Перевірка на порожність стеку**

**bool isEmpty() {**

**if (top == nullptr)**

**{**

**return true;**

**}**

**return false;**

**}**

**// Додавання елемента до стеку**

**void push(int value) {**

**Node\* newNode = new Node;**

**newNode->data = value;**

**newNode->next = top;**

**top = newNode;**

**cout << value << " added to the stack." << endl;**

**}**

**// Видалення елемента зі стеку**

**void pop() {**

**if (isEmpty()) {**

**cout << "stack is empty.It is impossible to add the element." << endl;**

**return;**

**}**

**Node\* temp = top;**

**top = top->next;**

**cout << temp->data << " deleted from the stack." << endl;**

**delete temp;**

**}**

**// Повернення верхнього елемента стеку (без видалення)**

**void peek() {**

**if (isEmpty()) {**

**cout << "stack is empty." << endl;**

**return;**

**}**

**cout << "the top of the stack: " << top->data << endl;**

**}**

**// Деструктор для звільнення пам'яті, використаної для стеку**

**~Stack() {**

**while (!isEmpty()) {**

**pop();**

**}**

**}**

**};**

**int main() {**

**Stack stack;**

**stack.push(5);**

**stack.push(10);**

**stack.push(15);**

**stack.peek();**

**stack.pop();**

**stack.peek();**

**stack.pop();**

**stack.pop();**

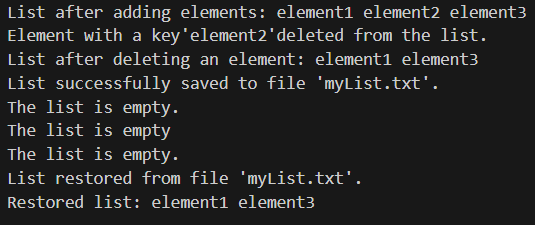
**stack.pop();**

**return 0;**

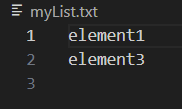
**}**

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

Завдання №1 **Лабораторна робота №10 VNS»**

****

*Figure 6.*

****

*Figure 7.*

Завдання №2 **«Algotester Lab 5v2»**

<https://algotester.com/uk/ProblemSolution/Display/1527117>

Завдання №3 **«Algotester 78v1»**

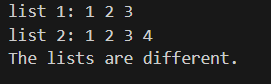
<https://algotester.com/uk/ContestProblem/DisplayWithEditor/134647>

Завдання №4 **«Practice Task 1»**

****

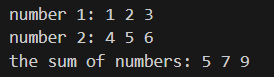
*Figure 8.*

Завдання №5 **«Practice Task 2»**

****

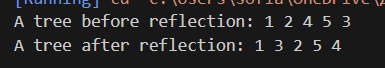
*Figure 9.*

Завдання №6 **«Practice Task 3»**

****

*Figure 10.*

Завдання №7 **«Practice Task 4»**

****

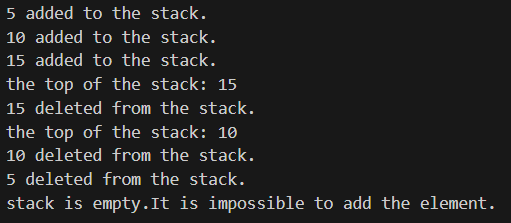
*Figure 11.*

Завдання №8 **«Practice Task 5»**

****

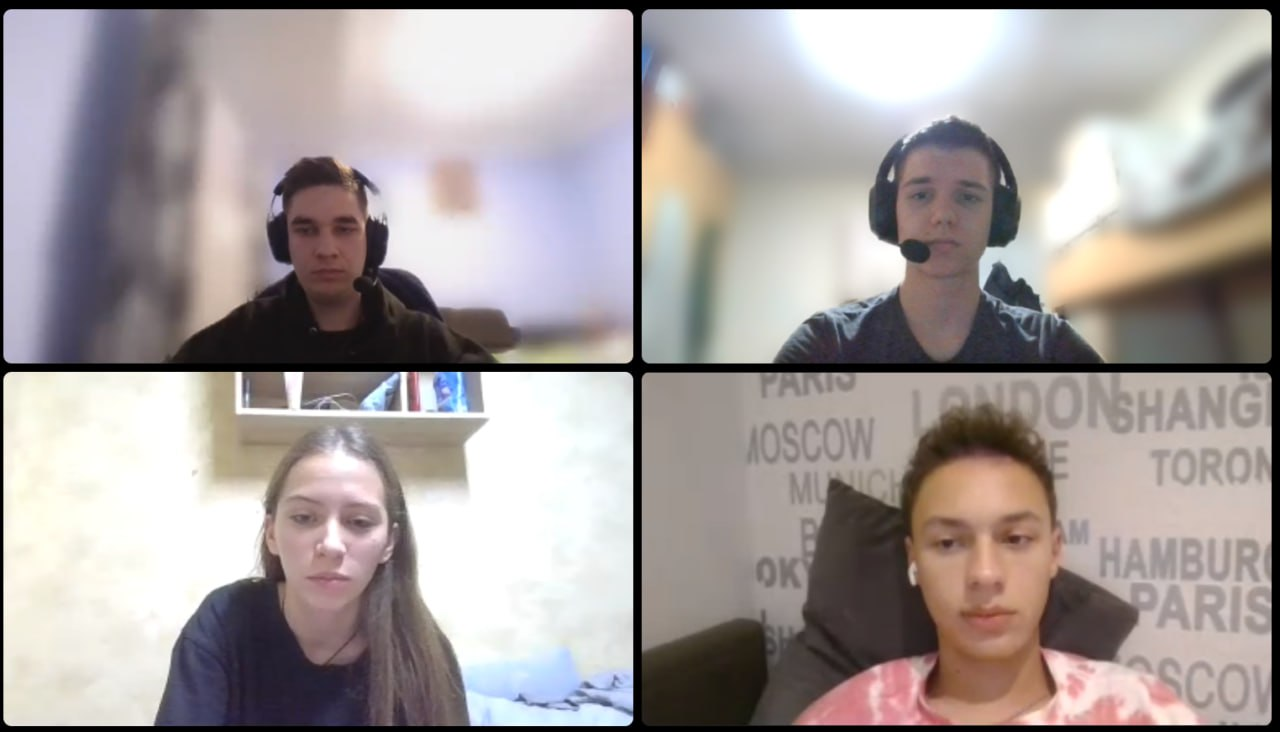
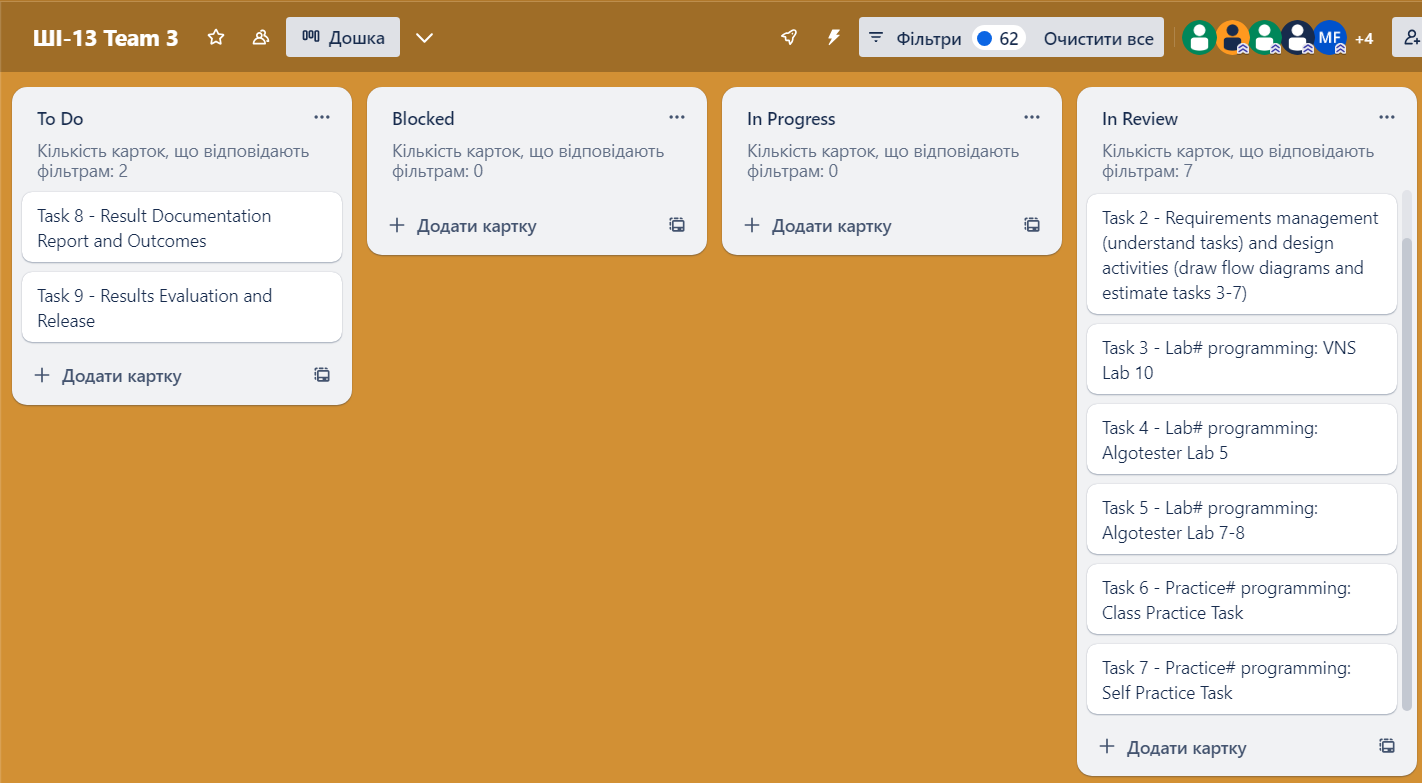
*Figure 12.*

Завдання №9 **«Self Practice»**

****

*Figure 12.*

## **5. Кооперація з командою:**

* 
* *Figure 13.*
* 
* *Figure 14.*

# *Figure 15.*

# **Висновки:**

Під час виконання даних практичних та лабораторних робіт з епіка 6 я змогла зрозуміти основні відмінності в різних типах динамічних структур. Також вдалось попрактикуватись в реалізації зв’язних списків та бінарних дерев. Було пропрацьовано різні алгоритми обробки динамічних структур, а саме: створення списку, друк списку, вставка елемента в список, пошук по бінарному дереву, обхід дерева.