Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**Звіт**

**Звіт**

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

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

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

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт № 6

**Виконав:**

Студент групи ШІ-12

Заставний Денис Миколайович

# **Тема роботи:**

Динамічні структури у програмуванні. Стек. Черга. Зв’язаний двонаправлений та однонаправлений списки. Бінарне дерево. Бінарне дерево пошуку.

# **Мета роботи:**

Зрозуміти принципи роботи із динамічними структурами даних. Детальніше дізнатися про поняття двонаправленого і однонапрaвленого списку, та операцій роботи з ними . Ознайомитися із теорією щодо бінарних дерев, та особливостей будови бінарного дерева пошуку. Ознайомитися з чергою та стеком. Застосувати на практиці отримані знання.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічний масив.
* Тема №2: Черга. Стек.
* Тема №3: Однонаправлений список. Двонаправлений список.
* Тема №4: Бінарні дерева. Бінарні дерева пошуку.

1. Індивідуальний план опрацювання теорії:

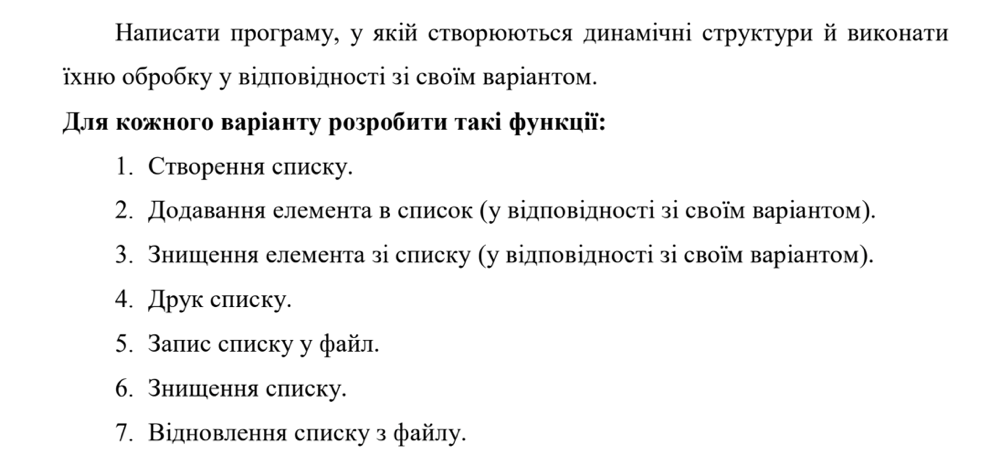
* Тема №1: Динамічний масив.
  + Джерела Інформації
    - <https://acode.com.ua/urok-90-dynamichni-masyvy/>
  + Що опрацьовано:
    - Поняття динамічного масиву, методи його задання.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 09.12.2023
  + Звершення опрацювання теми: 15.12.2023
* Тема №2: Однонаправлений список. Двонаправлений список.
  + Джерела Інформації:
    - <https://erudyt.net/navchalni-predmety/informatika/prohramuvannya/linijnyj-odnozvyaznyj-spysok.html>
    - <https://www.bestprog.net/uk/2022/02/16/c-linear-doubly-linked-bidirectional-list-general-concepts-ua/>
  + Що опрацьовано:
    - Поняття однозв’язного та двозвязного списків.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 21.11.2023
  + Звершення опрацювання теми: 15.12.2023
* Тема №3: Бінарні дерева. Бінарні дерева пошуку.
  + Джерела Інформації:
    - <https://www.programiz.com/dsa/binary-tree>
    - <https://uk.myservername.com/binary-search-tree-c>
    - <https://www.youtube.com/watch?v=Gt2yBZAhsGM>
  + Що опрацьовано:
    - Поняття дерев та їх структури.
  + Статус: Ознайомлений
  + Початок опрацювання теми:12.12.2023
  + Звершення опрацювання теми: 15.12.2023

# **Виконання роботи:**

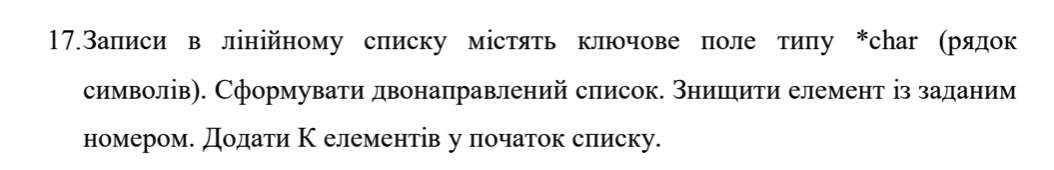
## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №1 VNS Lab 10

* Варіант 17

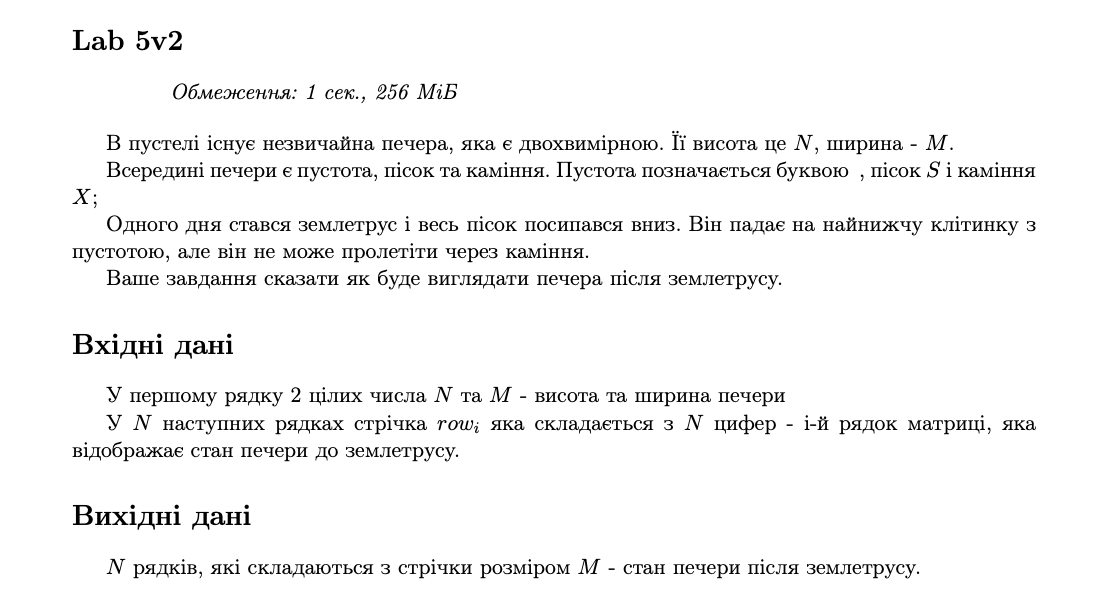


*Рисунок 1 - Умова Завдання №1 VNS Lab 10*



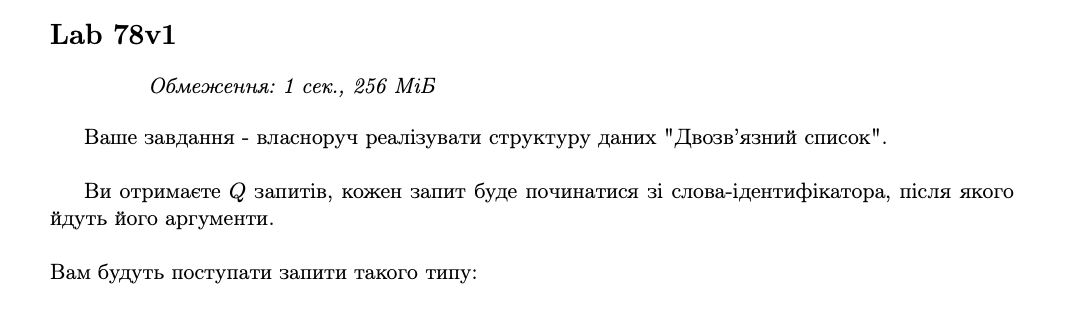
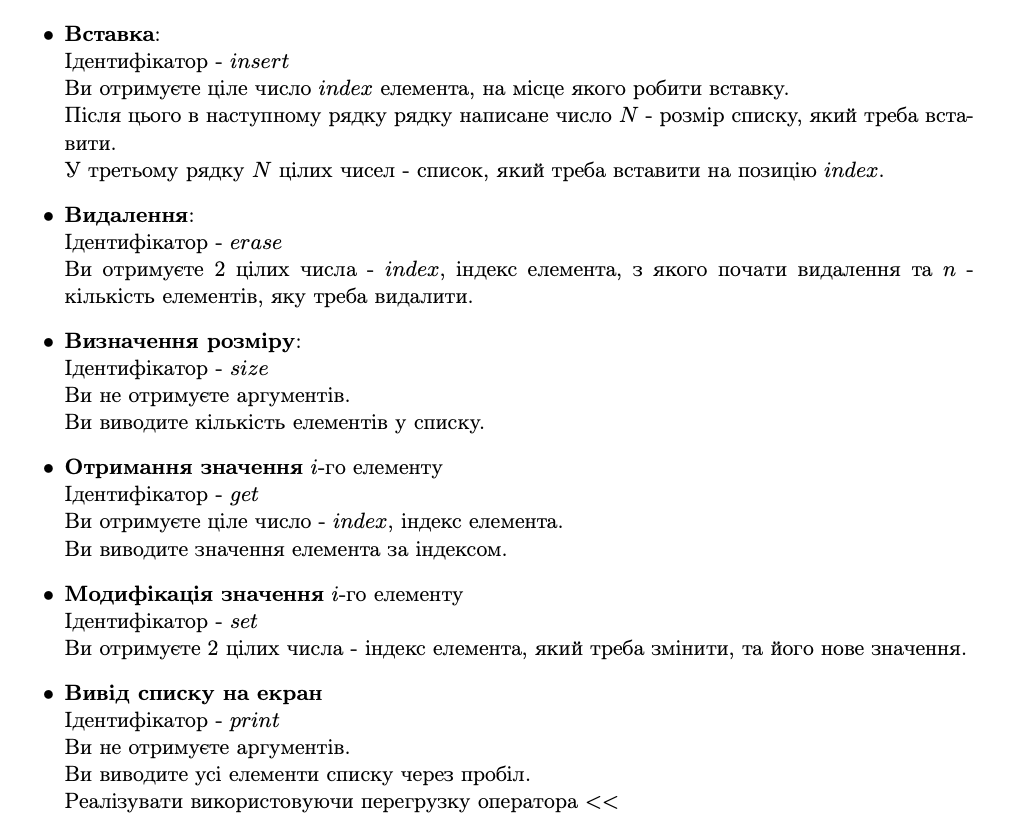
*Рисунок 2 - Умова Завдання №1 VNS Lab 10*

Завдання №2 Algotester Lab 5v2

* Варіант 2
* Деталі завдання
* 

*Рисунок 3 - Умова Завдання №2 Algotester Lab 5v2*

Завдання №3 Algotester Lab 78

* Варіант 1
* Деталі завдання
* 
* 

Завдання №4 Class Practice Tasks

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:*

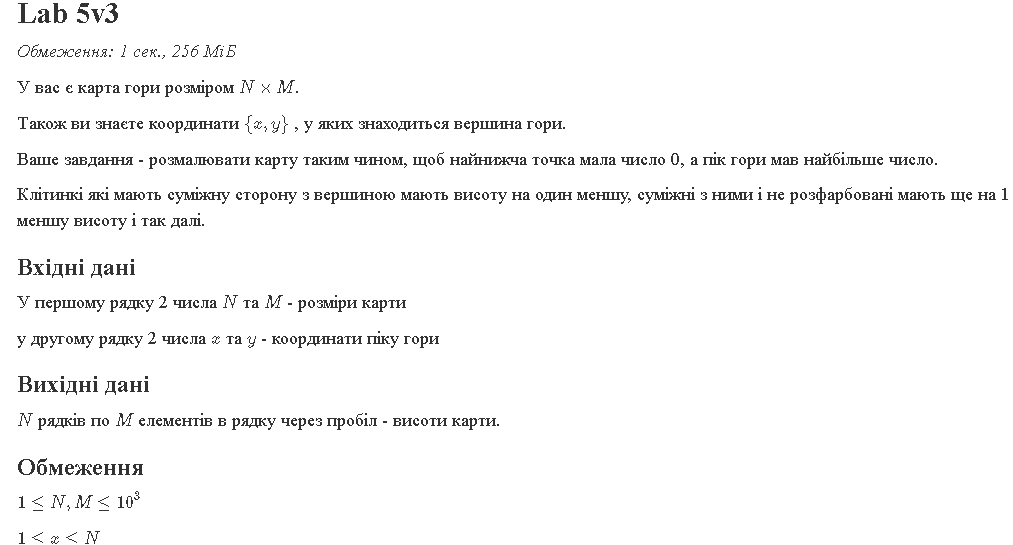
-       використовувати цілочисельні значення у вузлах дерева;

-       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 №5 Algotester Self practice Algotester lab5v3

* 

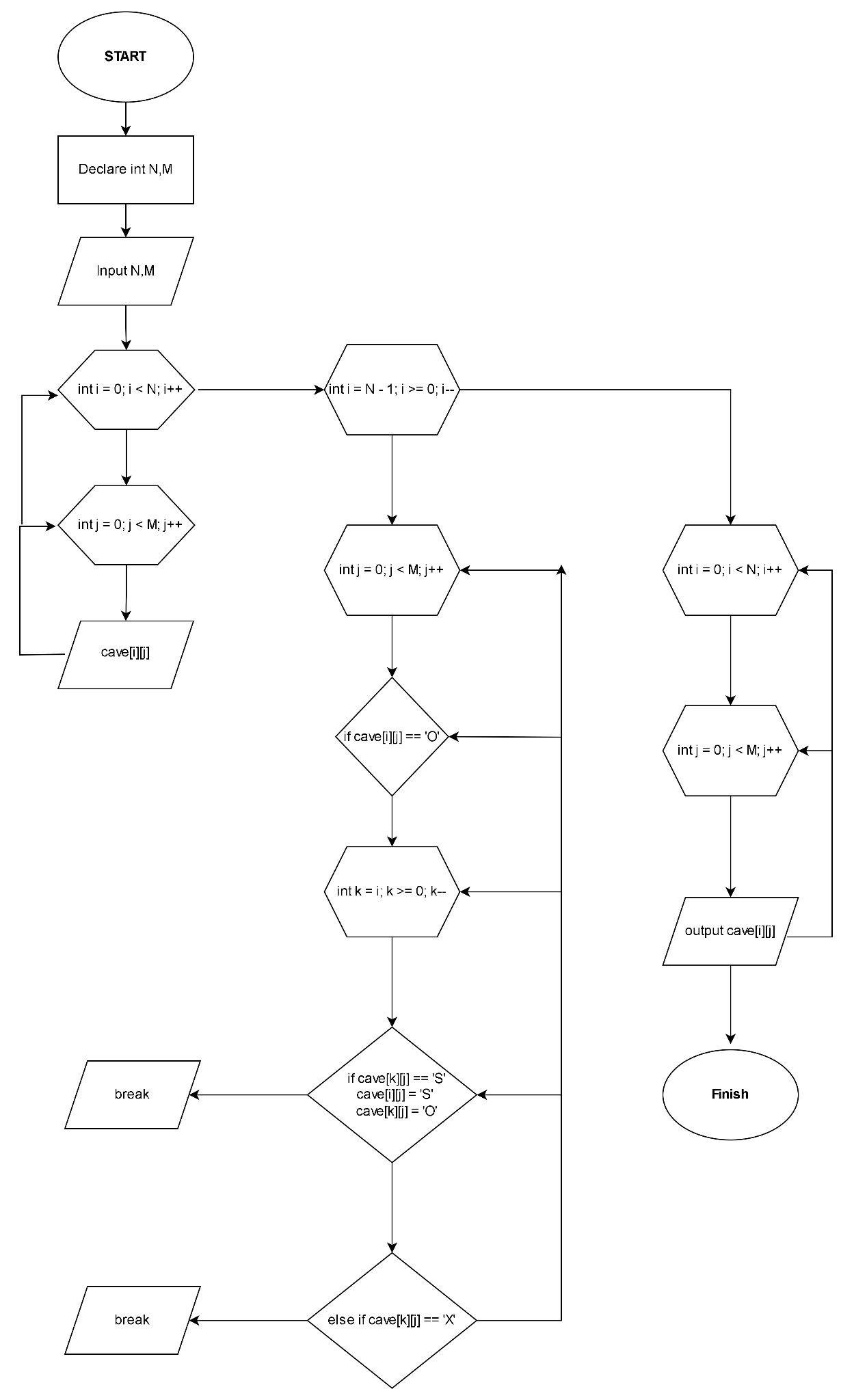
## **2. Дизайн та планована оцінка часу виконання завдань:**

Програма №1 VNS Lab 10

* Планований час на реалізацію: 3 години

Програма №2 Algotester Lab 5v2

* Блок-схема:



* Планований час на реалізацію: 2 години

Програма №3 Algotester Lab 78

* Планований час на реалізацію: 4 години

Програма №4 Class Practice Tasks 1-5

* Планований час на реалізацію: 2 години

Програма №6 Algotester Self practice Algotester lab5v3

* Планований час на реалізацію: 2 години

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

Завдання №1 VNS Lab 10 v17

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

struct Node {

char\* data;

Node\* next;

Node\* prev;

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

void addElement(const char\* data) {

Node\* newNode = new Node{strdup(data), nullptr, nullptr};

if (head == nullptr) {

head = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

void printList() const {

if (head == nullptr) {

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

} else {

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

}

void destroyList() {

Node\* current = head;

while (current != nullptr) {

Node\* next = current->next;

delete[] current->data;

delete current;

current = next;

}

head = nullptr;

}

void writeToFile(const char\* filename) const {

ofstream file(filename);

Node\* current = head;

while (current != nullptr) {

file << current->data << endl;

current = current->next;

}

file.close();

}

void restoreFromFile(const char\* filename) {

destroyList();

ifstream file(filename);

string line;

while (getline(file, line)) {

addElement(line.c\_str());

}

file.close();

}

void deleteElementByNumber(int position) {

if (head == nullptr) {

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

return;

}

Node\* current = head;

for (int i = 1; i < position && current != nullptr; ++i) {

current = current->next;

}

if (current == nullptr) {

cout << "Element with position " << position << " does not exist" << endl;

return;

}

if (current->prev != nullptr) {

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

} else {

head = current->next;

}

if (current->next != nullptr) {

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

}

delete[] current->data;

delete current;

}

void addKElementsToBeginning(int k) {

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

cout << "Enter a string for a new element: ";

string input;

getline(cin, input);

addElement(input.c\_str());

}

}

~LinkedList() {

destroyList();

}

};

int main() {

LinkedList list;

list.addElement("One");

list.addElement("Two");

list.addElement("Three");

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

list.printList();

list.writeToFile("list.txt");

list.destroyList();

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

list.printList();

list.restoreFromFile("list.txt");

cout << "Restored list:" << endl;

list.printList();

list.destroyList();

LinkedList doublyList;

doublyList.addElement("A");

doublyList.addElement("B");

doublyList.addElement("C");

doublyList.addElement("D");

cout << "Doubly linked list before changes:" << endl;

doublyList.printList();

doublyList.deleteElementByNumber(2);

cout << "Doubly linked list after deleting element with number 2:" << endl;

doublyList.printList();

int k = 2;

doublyList.addKElementsToBeginning(k);

cout << "Doubly linked list after adding " << k << " elements to the beginning:" << endl;

doublyList.printList();

return 0;

}

Завдання №2 Algotester Lab 5v2

#include <iostream>

using namespace std;

int main() {

int N, M;

cin >> N >> M;

char cave[N][M];

// Зчитування стану печери до землетрусу

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

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

cin >> cave[i][j];

}

}

// Проходження печери від низу до верху та видалення піску

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

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

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

for (int k = i; k >= 0; k--) {

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

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

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

break;

} else if (cave[k][j] == 'X') {

break;

}

}

}

}

}

// Виведення стану печери після землетрусу

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 7-8v1

#include <iostream>

template <typename T>

class Node {

public:

T data;

Node\* prev;

Node\* next;

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

};

template <typename T>

class DoublyLinkedList {

private:

Node<T>\* head;

Node<T>\* tail;

int size;

public:

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

~DoublyLinkedList() {

clear();

}

void insert(int index, int count, T\* values) {

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

return;

}

Node<T>\* current = getNode(index);

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

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

if (current == nullptr) {

if (tail == nullptr) {

head = tail = newNode;

} else {

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

}

} else {

if (current->prev == nullptr) {

newNode->next = head;

head->prev = newNode;

head = newNode;

} else {

newNode->prev = current->prev;

newNode->next = current;

current->prev->next = newNode;

current->prev = newNode;

}

}

++size;

}

}

void erase(int index, int count) {

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

return;

}

Node<T>\* current = getNode(index);

Node<T>\* end = getNode(index + count - 1);

if (current != nullptr && end != nullptr) {

if (current->prev != nullptr) {

current->prev->next = end->next;

} else {

head = end->next;

}

if (end->next != nullptr) {

end->next->prev = current->prev;

} else {

tail = current->prev;

}

while (current != nullptr && current != end->next) {

Node<T>\* temp = current;

current = current->next;

delete temp;

--size;

}

}

}

int getSize() const {

return size;

}

T get(int index) const {

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

return T();

}

Node<T>\* current = getNode(index);

if (current != nullptr) {

return current->data;

}

return T();

}

void set(int index, T value) {

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

return;

}

Node<T>\* current = getNode(index);

if (current != nullptr) {

current->data = value;

}

}

void print() const {

Node<T>\* current = head;

while (current != nullptr) {

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

current = current->next;

}

std::cout << std::endl;

}

void clear() {

Node<T>\* current = head;

while (current != nullptr) {

Node<T>\* temp = current;

current = current->next;

delete temp;

}

head = tail = nullptr;

size = 0;

}

private:

Node<T>\* getNode(int index) const {

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

return nullptr;

}

Node<T>\* current = head;

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

if (current != nullptr) {

current = current->next;

}

}

return current;

}

};

int main() {

int Q;

std::cin >> Q;

DoublyLinkedList<int> dll;

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

std::string identifier;

std::cin >> identifier;

if (identifier == "insert") {

int index, count;

std::cin >> index >> count;

int\* values = new int[count];

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

std::cin >> values[j];

}

dll.insert(index, count, values);

delete[] values;

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

int index, count;

std::cin >> index >> count;

dll.erase(index, count);

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

std::cout << dll.getSize() << std::endl;

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

int index;

std::cin >> index;

std::cout << dll.get(index) << std::endl;

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

int index, value;

std::cin >> index >> value;

dll.set(index, value);

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

dll.print();

}

}

return 0;

}

Завдання №4 Class Practice Tasks 1-5

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

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

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

void addNode(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

} else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

}

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;

}

void printList(Node\* list) {

Node\* current = list;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

void printOriginalAndReversed() {

cout << "Original List: ";

printList(head);

Node\* reversedHead = reverse(head);

cout << "Reversed List: ";

printList(reversedHead);

}

};

int main() {

LinkedList list;

list.addNode(1);

list.addNode(2);

list.addNode(3);

list.addNode(4);

list.addNode(5);

list.printOriginalAndReversed();

return 0;

}

*#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;*

*}*

*h1 = h1->next;*

*h2 = h2->next;*

*}*

*// Якщо довжина списків різна*

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

*return false;*

*}*

*return true;*

*}*

*int main() {*

*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);*

*bool result = compare(head1, head2);*

*if (result) {*

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

*} else {*

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

*}*

*// Очистка 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;*

*}*

*#include <iostream>*

*using namespace std;*

*struct Node {*

*int data;*

*Node\* next;*

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

*};*

*// Допоміжна функція для додавання нового вузла до кінця списку*

*void append(Node\*& head, int value) {*

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

*if (head == nullptr) {*

*head = newNode;*

*} else {*

*Node\* current = head;*

*while (current->next != nullptr) {*

*current = current->next;*

*}*

*current->next = newNode;*

*}*

*}*

*// Функція для додавання двох чисел, представлених у вигляді списків*

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

*Node\* result = nullptr;*

*Node\* current = nullptr;*

*int carry = 0;*

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

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

*carry = sum / 10;*

*int digit = sum % 10;*

*append(result, digit);*

*if (n1) n1 = n1->next;*

*if (n2) n2 = n2->next;*

*if (current == nullptr) {*

*current = result;*

*} else {*

*current = current->next;*

*}*

*}*

*return result;*

*}*

*// Функція для виведення списку*

*void printList(Node\* list) {*

*Node\* current = list;*

*while (current != nullptr) {*

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

*current = current->next;*

*}*

*cout << endl;*

*}*

*int main() {*

*Node\* num1 = new Node(2);*

*num1->next = new Node(4);*

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

*Node\* num2 = new Node(5);*

*num2->next = new Node(6);*

*num2->next->next = new Node(4);*

*cout << "Number 1: ";*

*printList(num1);*

*cout << "Number 2: ";*

*printList(num2);*

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

*cout << "Sum: ";*

*printList(sum);*

*// Очистка 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;*

*}*

*#include <iostream>*

*using namespace std;*

*struct TreeNode {*

*int data;*

*TreeNode\* left;*

*TreeNode\* right;*

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

*};*

*TreeNode\* createTree() {*

*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);*

*root->right->left = new TreeNode(6);*

*root->right->right = new TreeNode(7);*

*return root;*

*}*

*void printTree(TreeNode\* root) {*

*if (root != nullptr) {*

*printTree(root->left);*

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

*printTree(root->right);*

*}*

*}*

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

*if (root == nullptr) {*

*return nullptr;*

*}*

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

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

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

*return newRoot;*

*}*

*int main() {*

*TreeNode\* originalTree = createTree();*

*cout << "Original Tree (Inorder): ";*

*printTree(originalTree);*

*cout << endl;*

*TreeNode\* mirroredTree = create\_mirror\_flip(originalTree);*

*cout << "Mirrored Tree (Inorder): ";*

*printTree(mirroredTree);*

*cout << endl;*

*return 0;*

*}*

*#include <iostream>*

*#include <stack>*

*using namespace std;*

*struct TreeNode {*

*int data;*

*TreeNode\* left;*

*TreeNode\* right;*

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

*};*

*TreeNode\* createTree() {*

*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);*

*root->right->left = new TreeNode(6);*

*root->right->right = new TreeNode(7);*

*return root;*

*}*

*void printTree(TreeNode\* root) {*

*if (root != nullptr) {*

*printTree(root->left);*

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

*printTree(root->right);*

*}*

*}*

*void tree\_sum(TreeNode\* root) {*

*stack<TreeNode\*> nodes;*

*TreeNode\* current = root;*

*TreeNode\* prev = nullptr;*

*while (current != nullptr || !nodes.empty()) {*

*while (current != nullptr) {*

*nodes.push(current);*

*current = current->left;*

*}*

*current = nodes.top();*

*nodes.pop();*

*if (current->right == nullptr || current->right == prev) {*

*int sum = 0;*

*if (current->left != nullptr) {*

*sum += current->left->data;*

*}*

*if (current->right != nullptr) {*

*sum += current->right->data;*

*}*

*current->data += sum;*

*prev = current;*

*current = nullptr;*

*} else {*

*nodes.push(current);*

*current = current->right;*

*}*

*}*

*}*

*int main() {*

*TreeNode\* root = createTree();*

*cout << "Original Tree (Inorder): ";*

*printTree(root);*

*cout << endl;*

*tree\_sum(root);*

*cout << "Modified Tree (Inorder): ";*

*printTree(root);*

*cout << endl;*

*return 0;*

*}*

Завдання №6 Algotester Self practice Algotester lab 5v3

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

void generate\_landscape(int rows, int cols, int start\_row, int start\_col) {

vector<vector<int>> heights(rows, vector<int>(cols, -1));

queue<pair<int, int>> mountain\_queue;

mountain\_queue.push({start\_row - 1, start\_col - 1});

heights[start\_row - 1][start\_col - 1] = 0;

const int dx[] = {1, -1, 0, 0};

const int dy[] = {0, 0, 1, -1};

while (!mountain\_queue.empty()) {

pair<int, int> current = mountain\_queue.front();

mountain\_queue.pop();

for (int i = 0; i < 4; ++i) {

int nx = current.first + dx[i];

int ny = current.second + dy[i];

if (nx >= 0 && nx < rows && ny >= 0 && ny < cols && heights[nx][ny] == -1) {

heights[nx][ny] = heights[current.first][current.second] + 1;

mountain\_queue.push({nx, ny});

}

}

}

int max\_height = 0;

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

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

max\_height = max(max\_height, heights[i][j]);

}

}

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

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

cout << max\_height - heights[i][j] << " ";

}

cout << endl;

}

}

int main() {

int min\_val = 1;

int max\_val = 1000;

int rows, cols, start\_row, start\_col;

cin >> rows >> cols;

cin >> start\_row >> start\_col;

if (rows >= min\_val && rows <= max\_val && cols >= min\_val && cols <= max\_val &&

start\_row >= min\_val && start\_row <= rows && start\_col >= min\_val && start\_col <= cols) {

generate\_landscape(rows, cols, start\_row, start\_col);

} else {

cout << "Invalid input, please try again.";

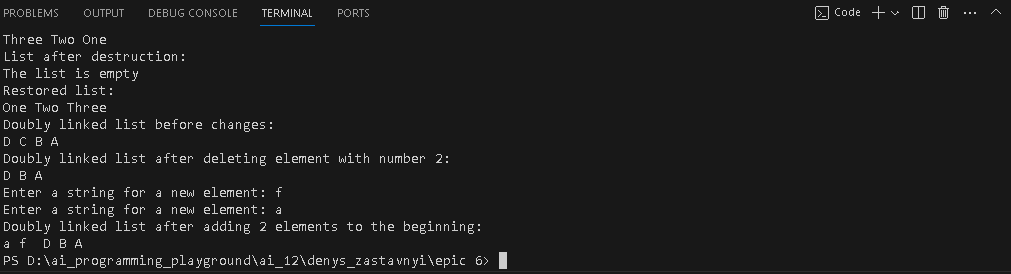
}

return 0;

}

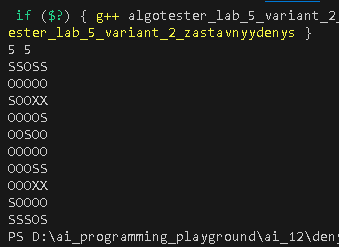
## **4. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1 VNS Lab 10

**

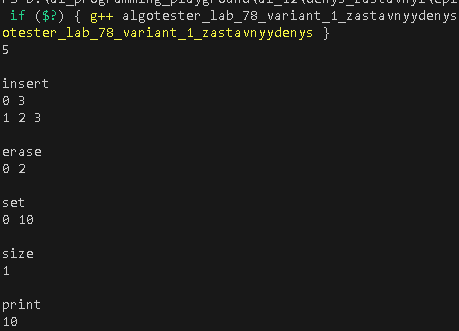
Час затрачений на виконання завдання: 1 день

Завдання №2 Algotester Lab 5



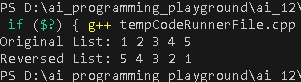
Час затрачений на виконання завдання: 1 день

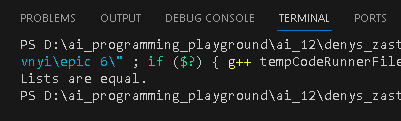
Завдання №3 Algotester Lab 7-8

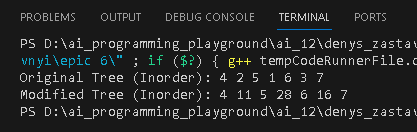
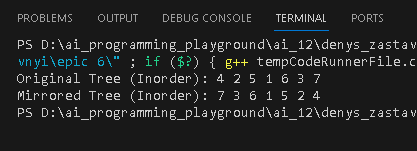
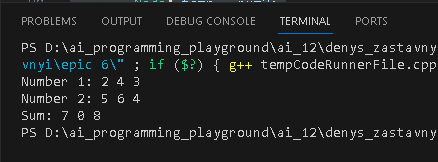
**

Час затрачений на виконання завдання: 1 день

Завдання №4 Class Practice Tasks 1-5

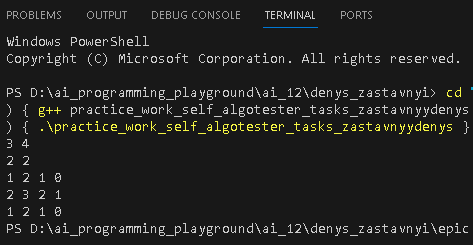






Час затрачений на виконання завдання: 1 день

Завдання №5 Algotester Self practice Algotester lab 5v3



# **Висновки:**

Під час виконання даної лабораторної роботи я детальніше дізнався про методи роботи із однонаправленими та двонаправленими списками. Ознайомився із поняттям бінарного дерева і застосувавце на практиці.