Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра систем штучного інтелекту

Звіт
про виконання лабораторних та практичних робіт блоку № 6
На тему: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур»
з дисципліни: «Основи програмування»
до:
ВНС Лабораторної Роботи № 10
Алготестер Лабораторної Роботи №5
Алготестер Лабораторної Роботи № 7-8
Практичних робіт до блоку №6

Виконав(ла):
Студентка групи ШІ-11
Ільящук Марта Тарасівна

Львів 2024

Тема роботи. Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур
Мета роботи: Ознайомитися з динамічними структурами: чергою, стеком, списками, деревами; навчитися працювати з ними та застосовувати різноманітні алгоритми обробки динамічних структур

Теоретичні відомості:

1. Теоретичні відомості з переліком важливих тем:
    - Тема №1. Основи динамічних структур даних
    - Тема №2. Стек
    - Тема №3. Черга
    - Тема №4. Зв'язні списки
    - Тема №5. Дерева
    - Тема №6. Алгоритми обробки динамічних структур

2. Індивідуальний план опрацювання теорії:
    - Тема №1. Основи динамічних структур даних
      Джерела інформації: Stack vs Heap Memory Allocation - GeeksforGeeks
      Що опрацьовано: Поняття stack, heap, прості динамічні структури
    - Тема №2. Стек
      Джерела інформації: Stack Data Structure - GeeksforGeeks
      Що опрацьовано: Стек, операції над ним, приклади використання
    - Тема №3. Черга
      Джерела інформації: C++ Queues
      Що опрацьовано: Черги, їх функціонал, використання, пріоритетні черги
    - Тема №4. Зв'язні списки
      Джерела інформації: Linked List in C++ - GeeksforGeeks
      Що опрацьовано: Зв'язні списки та їх види, операції над ними, використання
    - Тема №5. Дерева
      Джерела інформації: Binary Tree in C++ - GeeksforGeeks
      Що опрацьовано: Бінарні дерева, обхід дерева, основні операції, застосування
    - Тема №6. Алгоритми обробки динамічних структур
      Джерела інформації: Recursive Algorithms - GeeksforGeeks
      Що опрацьовано: Ітеративні, рекурсивні алгоритми

Виконання роботи:

1. Опрацювання завдань та вимог до середовища:

Завдання №1. VNS lab 10 variant 8

. Записи в лінійному списку містять ключове поле типу int. Сформувати двонаправлений список. Знищити з нього елемент після елемента із заданим номером, додати К елементів у початок списку.

Завдання №2. Algotester lab 5 variant 3

У вас є карта гори розміром $N \times M$.

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

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

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

Вхідні дані

У першому рядку 2 числа N та M - розміри карти

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

Вихідні дані

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

Завдання №3. Algotester lab 7-8 variant 2

Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - ***insert***
  Ви отримуєте ціле число ***index*** елемента, на місце якого робити вставку.
  Після цього в наступному рядку рядку написане число N - розмір масиву, який треба вставити.
  У третьому рядку N цілих чисел - масив, який треба вставити на позицію ***index***.
- **Видалення**:
  Ідентифікатор - ***erase***
  Ви отримуєте 2 цілих числа - ***index***, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити.
- **Визначення розміру**:
  Ідентифікатор - ***size***
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у динамічному масиві.
- **Визначення кількості зарезервованої пам'яті**:
  Ідентифікатор - ***capacity***
  Ви не отримуєте аргументів.
  Ви виводите кількість зарезервованої пам'яті у динамічному масиві.
  Ваша реалізація динамічного масиву має мати фактор росту (Growth factor) рівний 2.
- **Отримання значення i-го елементу**
  Ідентифікатор - ***get***
  Ви отримуєте ціле число - ***index***, індекс елемента.
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора []
- **Модифікація значення i-го елементу**
  Ідентифікатор - ***set***
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора []
- **Вивід динамічного масиву на екран**
  Ідентифікатор - ***print***
  Ви не отримуєте аргументів.
  Ви виводите усі елементи динамічного масиву через пробіл.
  Реалізувати використовуючи перегрузку оператора <<

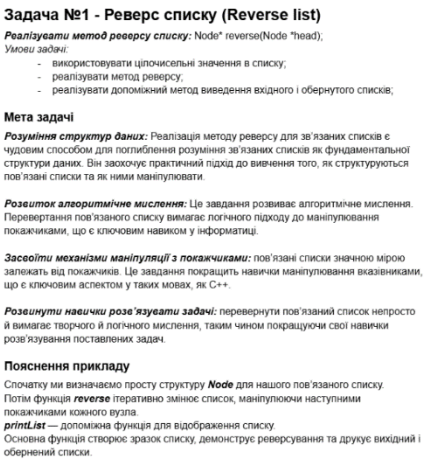
Вхідні дані

Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Вихідні дані

Відповіді на запити у зазначеному в умові форматі.

Завдання №4. Class Practice Task



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;

Мета задачі

Розуміння структур даних: Реалізація методу реверсу для зв'язаних списків є чудовим способом для поглиблення розуміння зв'язаних списків як фундаментальної структури даних. Він заохочує практичний підхід до вивчення того, як структуруються пов'язані списки та як ними маніпулювати.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Перевертання пов'язаного списку вимагає логічного підходу до маніпулювання покажчиками, що є ключовим навиком у інформатиці.

Засвоїти механізми маніпуляції з покажчиками: пов'язані списки значною мірою залежать від покажчиків. Це завдання покращить навички маніпулювання вказівниками, що є ключовим аспектом у таких мовах, як C++.

Розширити навички розв'язувати задачі: перевернути пов'язаний список непросто й вимагає творчого й логічного мислення, таким чином покращуючи свої навички розв'язування поставлених задач.

Пояснення прикладу

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов'язаного списку.
Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.
printList — допоміжна функція для відображення списку.
Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

Задача №2 - Порівняння списків

bool compare(Node *h1, Node *h2);
Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати функцію, яка ітеративно проходиться по обох спискам і порівнює дані в кожному вузлі;

- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

Мета задачі

Розуміння рівності в структурах даних: це завдання допомагає зрозуміти, як визначається рівність у складних структурах даних, таких як зв'язані списки. На відміну від примітивних типів даних, рівність пов'язаного списку передбачає порівняння кожного елемента та їх порядку.

Поглиблення розуміння зв'язаних списків: Порівнюючи зв'язані списки, дозволять покращити своє розуміння обходу, фундаментальної операції в обробці зв'язаних списків.

Розуміння ефективність алгоритму: це завдання також вводить поняття ефективності алгоритму. Студенти вчаться ефективно порівнювати елементи, що є навичкою, важливою для оптимізації та зменшення складності обчислень.

Розширити базові навики роботи з реальними програми: функції порівняння мають вирішальне значення в багатьох реальних програмах, таких як виявлення змін у даних, синхронізація структур даних або навіть у таких алгоритмах, як сортування та пошук.

Розширити навик вирішення проблем і уважа до деталей: це завдання закінчує скрупульозний підхід до програмування, оскільки навіть найменша неуважність може призвести до неправильних результатів порівняння. Це покращує навички вирішення проблем і увагу до деталей.

Пояснення прикладу

- Для пов'язаного списку визначено структуру ***Node***.
- Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.
- Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.
- Основна функція ***main*** створює два списки та демонструє порівняння.

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.

Мета задачі

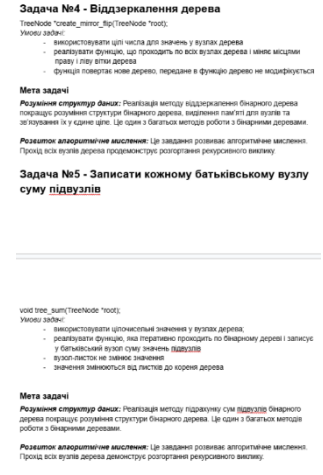
Розуміння операцій зі структурами даних: це завдання унаочнює практичне використання списка для обчислювальних потреб. Арифметичні операції з великими числами це окремий клас задач, для якого використання списків допомагає обійти обмеження у представленні цілого числа сучасними комп'ютерами.

Поглиблення розуміння зв'язаних списків: Застосовування зв'язаних списків для арифметичних операцій з великими числами дозволяє покращити розуміння операцій з обробки зв'язаних списків.

Розуміння ефективність алгоритму: це завдання дозволяє порівняти швидкість алгоритму додавання з використанням списків зі швидкістю вбудованих арифметичних операцій. Студенти вчаться розрізняти позитивні та негативні ефекти при виборі структур даних для реалізації практичних програм.

Розширити базові навики роботи з реальними програми: арифметичні операції з великими числами використовуються у криптографії, теорії чисел, астрономії, та ін.

Розширити навик вирішення проблем і уважа до деталей: завдання покращує розуміння обмежень у представленні цілого числа сучасними комп'ютерами та пропонує спосіб його вирішення.



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

Мета задачі

Розуміння структур даних: Реалізація методу віддзеркалення бінарного дерева покращує розуміння структури бінарного дерева, виділення пам'яті для вузлів та зв'язування їх у єдине ціле. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева продемонструє розогрітання рекурсивного виклику.

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

Мета задачі

Розуміння структур даних: Реалізація методу підрахунку сум підвузлів бінарного дерева покращує розуміння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розогрітання рекурсивного виклику.

Завдання №5. Self-Practice task Algotester lab 5 variant 2

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M.

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою O , 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.

Вхідні дані

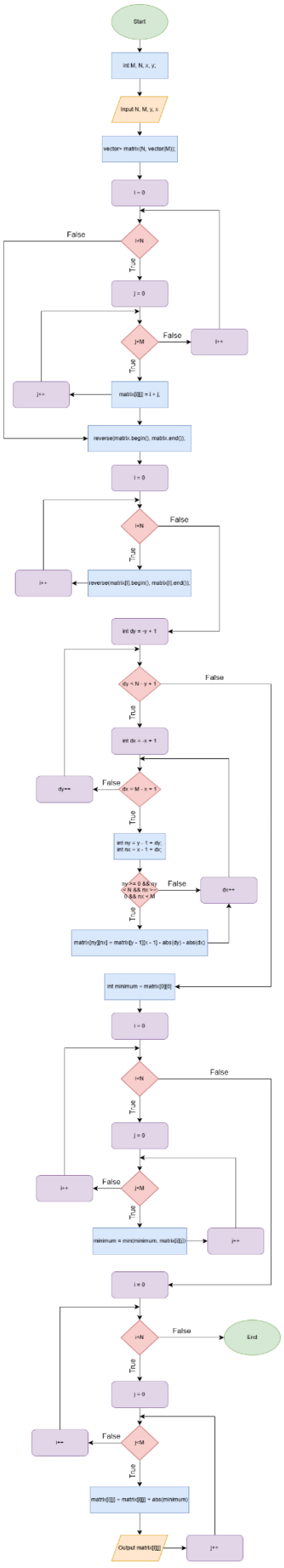
У першому рядку 2 цілих числа N та M - висота та ширина печери

У N наступних рядках стрічка row_i яка складається з N цифер - і-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

N рядків, які складаються з стрічки розміром M - стан печери після землетрусу.

2. Дизайн виконання завдань



3. Код програм з посиланням на зовнішні ресурси
   [Epic 6 - Marta Iliashchuk by martailiashchuk · Pull Request #640 · artificial-intelligence-department/ai_programming_playground_2024 · GitHub]

Завдання №1. VNS lab 10 variant 8

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
    Node* prev;


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

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:

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

    void addNodeAtStart(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void addNode(int value) {
        Node* newNode = new Node(value);
        if (tail == nullptr) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void removeElementAfter(int value) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == value && current->next != nullptr) {
```

```
                Node* x = current->next;
                current->next = x->next;
                if (x->next != nullptr) {
                    x->next->prev = current;
                } else {
                    tail = current;
                }
                delete x;
                return;
            }
            current = current->next;
        }
    }

    void printList() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

// Free memory deconstructor
    ~DoublyLinkedList() {
        while (head != nullptr) {
            Node* x = head;
            head = head->next;
            delete x;
        }
    }
};

int main() {
    DoublyLinkedList list;

    list.addNode(1);
    list.addNode(2);
    list.addNode(3);
    list.addNode(4);
    list.addNode(5);

    cout << "Original list: ";
    list.printList();


    list.removeElementAfter(2);
    cout << "List after removing element after 2: ";
    list.printList();


    int K = 3;
    for (int i = 0; i < K; ++i) {
```

```
        list.addNodeAtStart(10 + i);
    }
    cout << "List after adding 3 elements at the start: ";
    list.printList();

    return 0;
}
```

Завдання №2. Algotester lab 5 variant 3

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    int M, N, x, y;
    cin >> N >> M >> y >> x;

    vector<vector<int>> matrix(N, vector<int>(M));

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            matrix[i][j] = i + j;
        }
    }

    reverse(matrix.begin(), matrix.end());
    for (int i = 0; i < N; i++) {
        reverse(matrix[i].begin(), matrix[i].end());
    }

    for (int dy = -y + 1; dy < N - y + 1; dy++) {
        for (int dx = -x + 1; dx < M - x + 1; dx++) {
            int ny = y - 1 + dy;
            int nx = x - 1 + dx;

            if (ny >= 0 && ny < N && nx >= 0 && nx < M) {
                matrix[ny][nx] = matrix[y - 1][x - 1] - abs(dy) - abs(dx);
            }
        }
    }

    int minimum = matrix[0][0];
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            minimum = min(minimum, matrix[i][j]);
        }
    }

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

```
        for (int j = 0; j < M; j++) {
            matrix[i][j] = matrix[i][j] + abs(minimum);
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Завдання №3. Algotester lab 7-8 variant 2

```
#include <iostream>
using namespace std;

template <class T>
class DynamicArray {
private:
    T *data;
//Growth factor 2
    void ResizeIfNeeded(int N) {
        while (size + N >= capacity) {
            capacity *= 2;
        }
    }

public:
    int size;
    int capacity;

    DynamicArray() {
        size = 0;
        capacity = 1;
        data = new T[1];
    }

    void insert(int index, int N, T *elements) {
        ResizeIfNeeded(N);
// Create an array with all previous elements[i<index], add new elements starting
with index, add the rest data, index of the last added element - start point
        T *temporary = new T[capacity];

        for (int i = 0; i < index; i++) {
            temporary[i] = data[i];
        }

        for (int i = 0; i < N; i++) {
            temporary[index + i] = elements[i];
        }

        for (int i = index; i < size; i++) {
            temporary[i + N] = data[i];
```

```
        }

        size += N;
        delete[] data;
        data = temporary;
    }

    void erase(int index, int N) {
        T *temporary = new T[capacity];
        int newSize = 0;
//Delete imitation(rebuild data without element[index])
        for (int i = 0; i < size; i++) {
            if (i < index || i >= index + N) {
                temporary[newSize] = data[i];
                newSize++;
            }
        }

        size -= N;
        delete[] data;
        data = temporary;
    }

    int Size() const {
        return size;
    }

    int Capacity() const {
        return capacity;
    }

    T get(int index) const {
        return data[index];
    }

//Change element[index]
    void set(int index, T value) {
        data[index] = value;
    }


    void print(const string separator = " ") const {
        for (int i = 0; i < size; i++) {
            cout << data[i];
            if (i < size - 1) {
                cout << separator;
            }
        }
        cout << endl;
    }
};

int main() {
```

```
//data (typename)
    DynamicArray<int> array;

    int Q;
    cin >> Q;

    while (Q--) {
        string command;
        cin >> command;

        if (command == "insert") {
            int index, N;
            cin >> index >> N;
            int *elements = new int[N];

            for (int i = 0; i < N; i++) {
                cin >> elements[i];
            }

            array.insert(index, N, elements);
            delete[] elements;
        }
        else if (command == "erase") {
            int index, N;
            cin >> index >> N;
            array.erase(index, N);
        }
        else if (command == "size") {
            cout << array.Size() << endl;
        }
        else if (command == "capacity") {
            cout << array.Capacity() << endl;

        }
        else if (command == "get") {
            int index;
            cin >> index;
            cout << array.get(index) << endl;
        }
        else if (command == "set") {
            int index, N;
            cin >> index >> N;
            array.set(index, N);
        }
        else if (command == "print") {
            array.print();
        }
    }

    return 0;
}
```

Завдання №4. Class Practice Task

1.

```
#include <iostream>
using namespace std;

struct Node {
    int nodeValue;
    Node* nextNodeValue;


    Node(int value) : nodeValue(value), nextNodeValue(nullptr) {}
};

// Task 1: Reversing linked list
Node* reverse(Node*& head) {
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;

    while (current != nullptr) {
        next = current->nextNodeValue;
        current->nextNodeValue = prev; // 1 -> nullptr, next - current...4 - >... 2
->...
        prev = current;
        current = next;
    }

    return prev;
}

// Task 2: Comparing two linked lists
// Going through all elements while element = element
bool compare(Node* h1, Node* h2) {
    while (h1 != nullptr && h2 != nullptr) {
        if (h1->nodeValue != h2->nodeValue) {
            return false;
        }
        h1 = h1->nextNodeValue;
        h2 = h2->nextNodeValue;
    }
    return h1 == nullptr && h2 == nullptr;
}

void addToList(Node*& head, int value) {
    Node* newNode = new Node(value);

    if (head == nullptr) {
        head = newNode;
    } else {
        Node* x = head;
        while (x->nextNodeValue != nullptr) {
```

```
            x = x->nextNodeValue;
        }
        x->nextNodeValue = newNode;
    }
}

// Task 3: Adding two linked lists
Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* x = nullptr;
    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry != 0) {
        int sum = carry;
        if (n1 != nullptr) {
            sum += n1->nodeValue;
            n1 = n1->nextNodeValue;
        }

        if (n2 != nullptr) {
            sum += n2->nodeValue;
            n2 = n2->nextNodeValue;
        }

        carry = sum / 10;
        int digit = sum % 10;

        addToList(result, digit);
    }

    return result;
}

void printList(Node* head) {
    while (head != nullptr) {
        cout << head->nodeValue << " ";
        head = head->nextNodeValue;
    }
    cout << endl;
}

void printElements(Node* head) {
    head = reverse(head);
    while (head != nullptr) {
        cout << head->nodeValue;
        head = head->nextNodeValue;
    }
    cout << endl;
}

// Free memory
void clearList(Node*& head) {
    while (head != nullptr) {
```

```
        Node* temp = head;
        head = head->nextNodeValue;
        delete temp;
    }
}

int main() {
    Node* head = nullptr;
    addToList(head, 1);
    addToList(head, 2);
    addToList(head, 3);
    addToList(head, 4);

    cout << "Task 1" << endl;
    printList(head);

    head = reverse(head);

    cout << "Reversed list: " << endl;
    printList(head);

    Node* h1 = nullptr;
    addToList(h1, 1);
    addToList(h1, 3);
    addToList(h1, 7);
    addToList(h1, 5);

    cout << endl << "Task 2: " << endl;
    cout << "First list: " << endl;
    printList(h1);

    Node* h2 = nullptr;
    addToList(h2, 1);
    addToList(h2, 4);
    addToList(h2, 7);

    cout << "Second list: " << endl;
    printList(h2);

    if (compare(h1, h2)) {
        cout << "=" << endl;
    } else {
        cout << "!=" << endl;
    }

    cout << endl << "Task 3: " << endl;
    Node* result = add(h1, h2);

    cout << "Number: ";
    printElements(h1);
    cout << "Next number: ";
    printElements(h2);
    cout << "Sum: ";
```

```
    printElements(result);

// Free memory
    clearList(head);
    clearList(h1);
    clearList(h2);
    clearList(result);

    return 0;
}
```

2.

```
#include <iostream>
using namespace std;

struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

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

void printTree(TreeNode* root) {
    if (root != nullptr) {
        cout << root->data << " ";
        printTree(root->left);
        printTree(root->right);
    }
}

TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* left = create_mirror_flip(root->left);
    TreeNode* right = create_mirror_flip(root->right);

    root->left = right;
    root->right = left;

    return root;
}

int tree_sum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }

    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);
```

```
    if (root->left != nullptr || root->right != nullptr) {
        root->data += leftSum + rightSum;
    }

    return root->data;
}

//Free memory
void deleteTree(TreeNode* root) {
    if (root == nullptr) {
        return;
    }

// Delete left and right subtrees
    deleteTree(root->left);
    deleteTree(root->right);

// Delete current node
    delete root;
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(4);
    root->left->left = new TreeNode(8);
    root->right->left = new TreeNode(9);
    root->right->right = new TreeNode(7);

    cout << "Task 4: " << endl;
    printTree(root);

    root = create_mirror_flip(root);
    cout << endl << "Mirrored tree: " << endl;
    printTree(root);

    cout << endl << "Task 5: " << endl;
    tree_sum(root);

    cout << "Tree sum: ";
    printTree(root);
    cout << endl;

// Free memory
    deleteTree(root);

    return 0;
}
```

Завдання №5. Self-Practice task Algotester lab 5 variant 2

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int N, M;
    cin >> N >> M;

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

    for (int i = 0; i < N; i++) {
        string row;
        cin >> row;
        for (int j = 0; j < M; j++) {
            rows[i][j] = row[j];
        }
    }

    for (int j = 0; j < M; j++) {
        for (int i = N - 2; i >= 0; i--) {
            if (rows[i][j] == 'S') {
                int k = i;
                while (k + 1 < N && rows[k + 1][j] == 'O') {
                    k++;
                }
                if (k != i && rows[k][j] != 'X') {
                    rows[k][j] = 'S';
                    rows[i][j] = 'O';
                }
            }
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << rows[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

4. Результати виконання завдань

Завдання №1. VNS lab 10 variant 8

```
Original list: 1 2 3 4 5
List after removing element after 2: 1 2 4 5
List after adding 3 elements at the start: 12 11 10 1 2 4
```

Завдання №2. Algotester lab 5 variant 3

```
3
9
1
2
8 9 8 7 6 5 4 3 2
7 8 7 6 5 4 3 2 1
6 7 6 5 4 3 2 1 0
```

Завдання №3. Algotester lab 7-8 variant 2

```
2
insert
0
2
3
9
print
3 9
```

Завдання №4. Class Practice Task

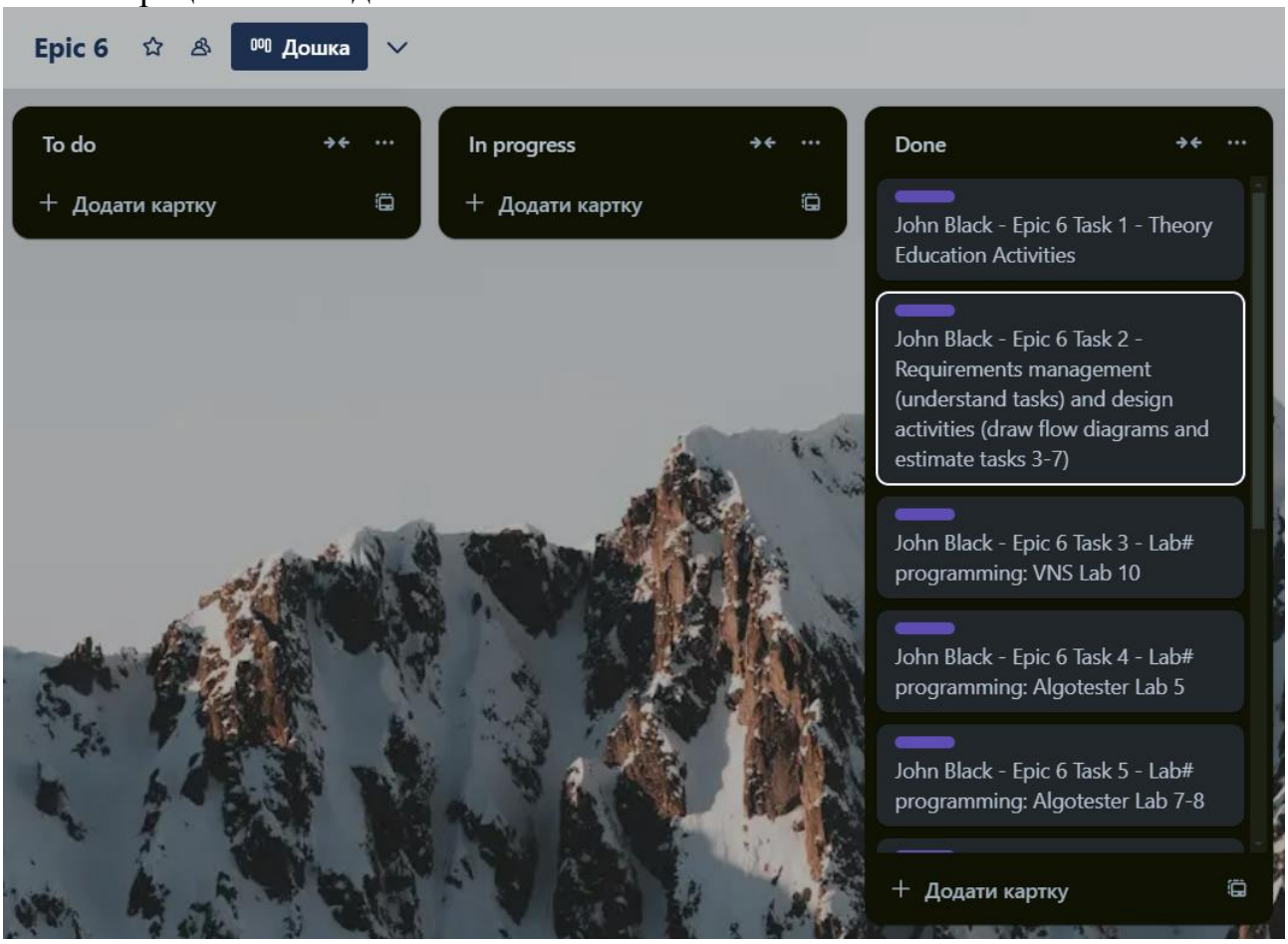
```
Task 1
1 2 3 4
Reversed list:
4 3 2 1

Task 2:
First list:
1 3 7 5
Second list:
1 4 7
!=

Task 3:
Number: 5731
Next number: 741
Sum: 6472
```

```
Task 4:
1 2 8 4 9 7
Mirrored tree:
1 4 7 9 2 8
Task 5:
Tree sum: 31 20 7 9 10 8
```

Завдання №5. Self-Practice task Algotester lab 5 variant 2

```
5
5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO
OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```

5. Кооперація з командою



Висновок: Під час виконання роботи я ознайомилася з динамічними структурами та навчилася використовувати їх на практиці.