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(ла):**

Студент(ка) групи ШІ-11

Ковалець Владислав Миколайович

# **Тема роботи:**

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

# **Мета роботи:**

Ознайомлення та вивчення принципів роботи динамічних структур даних, таких як черга, стек, списки та дерево. Дослідження алгоритмів обробки цих структур з метою з'ясування їхніх особливостей, ефективності та використання в різноманітних областях програмування.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Стек
* Тема №2: Списки
* Тема №3: Дерева

1. Індивідуальний план опрацювання теорії:

* Тема №1: Стек

Джерела Інформації:

* + - Відео:

https://www.youtube.com/watch?v=ZYvYISxaNL0

* + Що опрацьовано:

Що таке стек і як він працює

* + Статус: Ознайомлений
  + Початок опрацювання теми: 05.12.2023
  + Звершення опрацювання теми: 05.11.2023
* Тема №2: Списки
  + Джерела Інформації:
    - Відео:

https://www.youtube.com/watch?v=-25REjF\_atI

* + Що опрацьовано:
    - Поняття однозв’язний список та двозв’язний список
  + Статус: Ознайомлений
  + Початок опрацювання теми: 05.12.2023
  + Звершення опрацювання теми:05.12.2023
* Тема №3: Дерева
  + Джерела Інформації:
    - Відео

https://www.youtube.com/watch?v=qBFzNW0ALxQ

* + Що опрацьовано:
    - Поняття бінарні дерева
  + Статус: Ознайомлений
  + Початок опрацювання теми: 05.12.2023
  + Звершення опрацювання теми: 05.12.2023

# **Виконання роботи:**

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

Завдання № VNS lab10

* 2 варіант

Написати програму, у якій створюються динамічні структури й виконати

їхню обробку. Додавання елемента в список (у відповідності зі своїм варіантом).

Завдання № algotester lab5

* 1 варіант

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. 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 -координати піку гори

Вихідні дані

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

Завдання № algotester lab7-8

* 1 варіант

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.

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 списку, який треба вставити. У третьому рядку N цілих чисел - список, 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 списку.

Отримання значення i-го елементу

Ідентифікатор - get

Ви отримуєте ціле число - index, індекс елемента. Ви виводите значення елемента за індексом.

Модифікація значення i-го елементу

Ідентифікатор – set

Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.

Вивід списку на екран

Ідентифікатор - print

Ви не отримуєте аргументів.

Ви виводите усі елементи списку 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 № practice 6

**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

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

Програма № algotester lab7-8

* Блок-схема

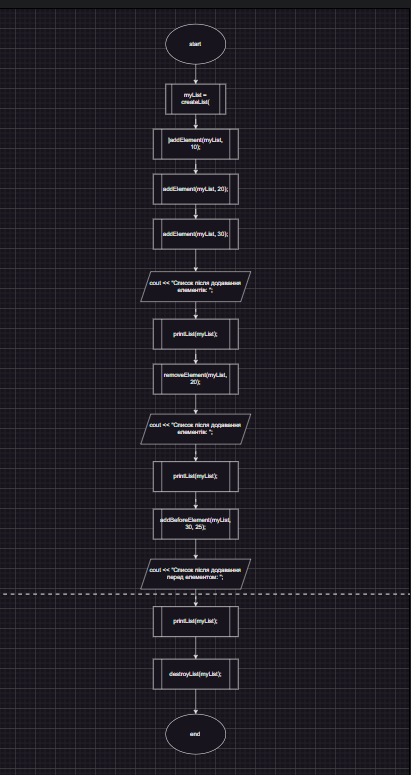


Рисунок Блок-схема до завдання algotester lab7-8

* Планований час на реалізацію

50 хвилин

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

**Завдання № vns lab10**

#include <iostream>

using namespace std;

// Структура для елемента списку

struct Node

{

    int key;

    Node\* next;

};

// Функція для створення порожнього списку

Node\* createList()

{

    return nullptr;

}

// Функція для додавання елемента в кінець списку

void addElement(Node\* &head, int key)

{

    Node\* newNode = new Node;

    newNode->key = key;

    newNode->next = nullptr;

    if (!head)

    {

        head = newNode;

    }

    else

    {

        Node\* current = head;

        while (current->next)

        {

            current = current->next;

        }

        current->next = newNode;

    }

}

// Функція для видалення елемента зі списку за ключем

void removeElement(Node\* &head, int key)

{

    if (!head)

    {

        cout << "Список порожній" << endl;

        return;

    }

    Node\* current = head;

    Node\* prev = nullptr;

    while (current && current->key != key)

    {

        prev = current;

        current = current->next;

    }

    if (!current)

    {

        cout << "Елемент не знайдений у списку" << endl;

        return;

    }

    if (!prev)

    {

        head = current->next;

    }

    else

    {

        prev->next = current->next;

    }

    delete current;

}

// Функція для додавання елемента перед елементом за ключем

void addBeforeElement(Node\*& head, int key, int newKey)

{

    Node\* newNode = new Node;

    newNode->key = newKey;

    newNode->next = nullptr;

    if (!head)

    {

        cout << "Список порожній" << endl;

        delete newNode;

        return;

    }

    Node\* current = head;

    Node\* prev = nullptr;

    while (current && current->key != key)

    {

        prev = current;

        current = current->next;

    }

    if (!current)

    {

        cout << "Елемент не знайдений у списку" << endl;

        delete newNode;

        return;

    }

    if (!prev)

    {

        newNode->next = head;

        head = newNode;

    }

    else

    {

        prev->next = newNode;

        newNode->next = current;

    }

}

// Функція для друку списку

void printList(const Node\* head)

{

    if (!head)

    {

        cout << "Список порожній" << endl;

        return;

    }

    const Node\* current = head;

    while (current)

    {

        cout << current->key << " ";

        current = current->next;

    }

    cout << endl;

}

// Функція для знищення списку

void destroyList(Node\* &head)

{

    Node\* current = head;

    while (current)

    {

        Node\* nextNode = current->next;

        delete current;

        current = nextNode;

    }

    head = nullptr;

}

int main()

{

    Node\* myList = createList();

    // Додавання елементів до списку

    addElement(myList, 10);

    addElement(myList, 20);

    addElement(myList, 30);

    // Друк списку

    cout << "Список після додавання елементів: ";

    printList(myList);

    // Видалення елемента зі списку

    removeElement(myList, 20);

    // Друк списку після видалення елемента

    cout << "Список після видалення елемента: ";

    printList(myList);

    // Додавання елемента перед елементом з заданим ключем

    addBeforeElement(myList, 30, 25);

    // Друк списку після додавання перед елементом

    cout << "Список після додавання перед елементом: ";

    printList(myList);

    // Знищення списку після використання

    destroyList(myList);

    return 0;

}

**Завдання № algotester lab5**

#include <iostream>

#include <cmath>

int main()

{

    int n, m, x, y, maxelement=0;

    std::cin >> n >> m >> x >> y;

    int arr[n][m];

    for(int i=0; i<n; i++)

    {

        for(int j=0; j<m; j++)

        {

            if(maxelement < abs(x-1-i) + abs(y-1-j) )

            maxelement = abs(x-1-i) + abs(y-1-j);

        }

    }

    for(int i=0; i<n; i++)

    {

        for(int j=0; j<m; j++)

        {

            arr[i][j] = maxelement - (abs(x-1-i) + abs(y-1-j));

        }

    }

    arr[x-1][y-1]=maxelement;

    for(int i=0; i<n; i++)

    {

        for(int j=0; j<m; j++)

        {

            std::cout << arr[i][j] << " ";

        }

        std::cout<< std::endl;

    }

}

**Завдання № algotester lab7-8**

#include <iostream>

// Структура для представлення вузла двозв'язного списку

struct Node

{

    int data;      // Дані, які зберігаються в вузлі

    Node\* next;    // Вказівник на наступний вузол

    Node\* prev;    // Вказівник на попередній вузол

    // Конструктор ініціалізації вузла з вказаним значенням даних

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

};

// Клас для реалізації двозв'язного списку

class DoublyLinkedList

{

private:

    Node\* head;    // Вказівник на початковий вузол

    Node\* tail;    // Вказівник на кінцевий вузол

    int size;      // Кількість елементів у списку

public:

    // Конструктор за замовчуванням ініціалізує вказівники як nullptr і розмір як 0

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

    // Функція вставки елементів у вказану позицію

    void insert(int index, int N, int\* values)

    {

        Node\* newNode;

        Node\* current;

        for (int i = 0; i < N; ++i)

        {

            newNode = new Node(values[i]);

            if (size == 0)

            {

                head = tail = newNode;

            }

            else if (index == 0)

            {

                newNode->next = head;

                head->prev = newNode;

                head = newNode;

            }

            else if (index == size)

            {

                tail->next = newNode;

                newNode->prev = tail;

                tail = newNode;

            }

            else

            {

                current = head;

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

                {

                    current = current->next;

                }

                newNode->prev = current->prev;

                newNode->next = current;

                current->prev->next = newNode;

                current->prev = newNode;

            }

            ++size;

            ++index;

        }

    }

    // Функція видалення елементів з вказаної позиції

    void erase(int index, int n)

    {

        while (n > 0 && index < size)

        {

            Node\* current = head;

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

            {

                current = current->next;

            }

            Node\* nextNode = current->next;

            if (current->prev)

            {

                current->prev->next = nextNode;

            }

            else

            {

                head = nextNode;

            }

            if (nextNode)

            {

                nextNode->prev = current->prev;

            }

            else

            {

                tail = current->prev;

            }

            delete current;

            --size;

            --n;

        }

    }

    // Функція отримання розміру списку

    void getSize() const

    {

        std::cout << size << std::endl;

    }

    // Функція отримання значення елементу за індексом

    void get(int index) const

    {

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

        {

            Node\* current = head;

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

            {

                current = current->next;

            }

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

        }

    }

    // Функція зміни значення елементу за індексом

    void set(int index, int value)

    {

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

        {

            Node\* current = head;

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

            {

                current = current->next;

            }

            current->data = value;

        }

    }

    // Перегрузка оператора виводу для виведення списку

    friend std::ostream& operator<<(std::ostream& os, const DoublyLinkedList& list)

    {

        Node\* current = list.head;

        while (current) {

            os << current->data << " ";

            current = current->next;

        }

        os << std::endl;

        return os;

    }

    // Деструктор для вивільнення пам'яті відведеної під вузли

    ~DoublyLinkedList()

    {

        Node\* current = head;

        while (current)

        {

            Node\* nextNode = current->next;

            delete current;

            current = nextNode;

        }

        size = 0;

    }

};

int main()

{

    int Q;

    std::cin >> Q;

    DoublyLinkedList list;

    for (int i = 0; i < Q; i++)

    {

        std::string userprint;

        std::cin >> userprint;

        if (userprint == "insert")

        {

            int index, N;

            std::cin >> index >> N;

            int\* values = new int[N];

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

                std::cin >> values[j];

            }

            list.insert(index, N, values);

            delete[] values;

        }

        else if (userprint == "erase")

        {

            int index, n;

            std::cin >> index >> n;

            list.erase(index, n);

        }

        else if (userprint == "print")

        {

            std::cout << list;

        }

        else if (userprint == "size")

        {

            list.getSize();

        }

        else if (userprint == "get")

        {

            int index;

            std::cin >> index;

            list.get(index);

        }

        else if (userprint == "set")

        {

            int index, value;

            std::cin >> index >> value;

            list.set(index, value);

        }

    }

    return 0;

}

**№ algotester lab4**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

    int N, M;

    // Зчитуємо розмір та елементи масиву 1

    cin >> N;

    vector<int> array1(N);

    for (int i = 0; i < N; i++)

    {

        cin >> array1[i];

    }

    // Зчитуємо розмір та елементи масиву 2

    cin >> M;

    vector<int> array2(M);

    for (int i = 0; i < M; i++)

    {

        cin >> array2[i];

    }

    // Сортуємо масиви

    sort(array1.begin(), array1.end());

    sort(array2.begin(), array2.end());

    // Різниця N-M

    vector<int> diffNM;

    set\_difference(array1.begin(), array1.end(), array2.begin(), array2.end(), back\_inserter(diffNM));

    // Різниця M-N

    vector<int> diffMN;

    set\_difference(array2.begin(), array2.end(), array1.begin(), array1.end(), back\_inserter(diffMN));

    // Перетин

    vector<int> intersection;

    set\_intersection(array1.begin(), array1.end(), array2.begin(), array2.end(), back\_inserter(intersection));

    // Об'єднання

    vector<int> unionSet;

    set\_union(array1.begin(), array1.end(), array2.begin(), array2.end(), back\_inserter(unionSet));

    // Симетрична різниця

    vector<int> symmetricDiff;

    set\_symmetric\_difference(array1.begin(), array1.end(), array2.begin(), array2.end(), back\_inserter(symmetricDiff));

    // Виведення результатів

    cout << diffNM.size() << endl;

    for (int value : diffNM)

    {

        cout << value << " ";

    }

    cout << endl;

    cout << diffMN.size() << endl;

    for (int value : diffMN)

    {

        cout << value << " ";

    }

    cout << endl;

    cout << intersection.size() << endl;

    for (int value : intersection)

    {

        cout << value << " ";

    }

    cout << endl;

    cout << unionSet.size() << endl;

    for (int value : unionSet)

    {

        cout << value << " ";

    }

    cout << endl;

    cout << symmetricDiff.size() << endl;

    for (int value : symmetricDiff)

    {

        cout << value << " ";

    }

    cout << endl;

    return 0;

}

**Завдання № practice 6 task1**

#include <iostream>

// Вузол списку

struct Node

{

    int data;      // Дані які вузол містить

    Node\* next;    // Вказівник на наступний вузол у списку

    // Конструктор для створення вузла з заданим значенням даних

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

};

// Метод для реверсу списку

Node\* reverse(Node\* head)

{

    Node\* prev = nullptr;

    Node\* current = head;

    Node\* next = nullptr;

    while (current != nullptr)

    {

        next = current->next; //  Зберігаємо посилання на наступний вузол у змінній next щоб не втратити його при зміні посилань в наступних кроках

        current->next = prev; // змінюємо вказівник next поточного вузла,вказуючи його на попередній вузол

        prev = current; // Переміщуємо prev на поточний вузол тепер це стає попереднім вузлом для наступного кроку

        current = next; // Переміщуємо current на наступний вузол у списку

    }

    return prev;  // новий початок списку

}

// метод для виведення списку

void printList(Node\* head)

{

    while (head != nullptr)

    {

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

        head = head->next;

    }

    std::cout << std::endl;

}

int main()

{

    // Створюємо вхідний список 1 - 2 - 3 - 4 - 5

    Node\* head = new Node(1);

    head->next = new Node(2);

    head->next->next = new Node(3);

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

    head->next->next->next->next = new Node(5);

    // Виводимо вхідний список

    std::cout << "список";

    printList(head);

    // Реверсуємо список

    head = reverse(head);

    // Виводимо обернений список

    std::cout << "Реверс списку(обернений)";

    printList(head);

    // Звільняємо пам'ять від списку

    while (head != nullptr)

    {

        Node\* temp = head;

        head = head->next;

        delete temp;

    }

    return 0;

}

**Завдання № practice6 task2**

#include <iostream>

// Вузол списку

struct Node

{

    int data;      // Дані які вузол містить

    Node\* next;    // Вказівник на наступний вузол у списку

    // Конструктор для створення вузла з заданим значенням даних

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

};

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

{

    if (h1->data != h2->data)

    {

        // Якщо дані в вузлах не збігаються

        return false;

    }

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

    {

        // Переходимо до наступних вузлів у обох списках

        h1 = h1->next;

        h2 = h2->next;

    }

    // Одна з умов: або обидва списки закінчилися одночасно, або один з них не закінчився, а інший закінчився раніше

    // У обох випадках список однаковий до цього моменту

    return (h1 == nullptr && h2 == nullptr);

}

// метод для виведення списку

void printList(Node\* head)

{

    while (head != nullptr)

    {

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

        head = head->next;

    }

    std::cout << std::endl;

}

int main()

{

    // Створюємо перший список 1 - 2 - 3 - 4 - 5

    Node\* head1 = new Node(1);

    head1->next = new Node(2);

    head1->next->next = new Node(3);

    head1->next->next->next = new Node(4);

    head1->next->next->next->next = new Node(5);

    // Створюємо другий список 6 - 7 - 8 - 9 -10

    Node\* head2 = new Node(6);

    head2->next = new Node(7);

    head2->next->next = new Node(8);

    head2->next->next->next = new Node(9);

    head2->next->next->next->next = new Node(10);

    // Виводимо обидва списки

    std::cout << "Список №1";

    printList(head1);

    std::cout << "Список №2";

    printList(head2);

    // Порівнюємо списки і виводимо результат

    if (compare(head1, head2))

    {

        std::cout << "Списки однакові" << std::endl;

    }

    else

    {

        std::cout << "Списки різні" << std::endl;

    }

    // Звільняємо пам'ять від обох списків

    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 № practice6 task3**

#include <iostream>

// Вузол списку

struct Node

{

    int data;      // Дані, які вузол містить

    Node\* next;    // Вказівник на наступний вузол у списку

    // Конструктор для створення вузла з заданим значенням даних

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

};

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

{

    Node\* head = nullptr;

    Node\* tail = nullptr;

    int suma = 0;

    while (n1 != nullptr || n2 != nullptr || suma != 0)

    {

        int value0 = (n1 != nullptr) ? n1->data : 0;

        int value1 = (n2 != nullptr) ? n2->data : 0;

        int sum = value0 + value1 + suma;

        suma = sum / 10;  // Визначаємо перенос

        int digit = sum % 10;  // Отримуємо значення розряду

        // Створюємо новий вузол з отриманим значенням розряду

        Node\* newNode = new Node(digit);

        // Додаємо новий вузол до кінця результативного списку

        if (head == nullptr)

        {

            head = newNode;

            tail = newNode;

        }

        else

        {

            tail->next = newNode;

            tail = newNode;

        }

        // Переходимо до наступного розряду у обох вхідних списках (якщо вони не закінчилися)

        if (n1 != nullptr) n1 = n1->next;

        if (n2 != nullptr) n2 = n2->next;

    }

    return head;

}

void printList(Node\* head)

{

    while (head != nullptr)

    {

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

        head = head->next;

    }

    std::cout << std::endl;

}

int main()

{

    Node\* num1 = new Node(3);

    num1->next = new Node(7);

    num1->next->next = new Node(9);

    Node\* num2 = new Node(9);

    num2->next = new Node(2);

    num2->next->next = new Node(5);

    Node\* result = add(num1, num2);

    // Вивід результату

    std::cout << "Список " << std::endl;

    printList(num1);

    std::cout << "Список " << std::endl;

    printList(num2);

    std::cout << "Сума " << std::endl;

    printList(result);

    // Звільняємо пам'ять від списків

    while (num1 != nullptr)

    {

        Node\* temp = num1;

        num1 = num1->next;

        delete temp;

    }

    while (num2 != nullptr)

    {

        Node\* temp = num2;

        num2 = num2->next;

        delete temp;

    }

    while (result != nullptr)

    {

        Node\* temp = result;

        result = result->next;

        delete temp;

    }

    return 0;

}

**Завдання № practice task 4**

#include <iostream>

// Структура вузла бінарного дерева

struct TreeNode

{

    int data;         // Значення вузла

    TreeNode\* left;    // Вказівник на ліву вітку

    TreeNode\* right;   // Вказівник на праву вітку

    // Конструктор для ініціалізації вузла з заданим значенням та вказівниками на ліву та праву вітки

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

};

// Функція для створення дерева-дзеркала

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

{

    if (root == nullptr)

    {

        return nullptr;

    }

    // Створюємо новий вузол з тим же значенням

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

    // Рекурсивно обертаємо праву і ліву вітки

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

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

    return newRoot;

}

// Допоміжна функція для виведення дерева в префіксному порядку

void print\_tree(TreeNode\* root)

{

    if (root == nullptr) {

        return;

    }

    std::cout << root->data << " ";

    print\_tree(root->left);

    print\_tree(root->right);

}

int main() {

    // Створення дерева для тестування

    TreeNode\* root = new TreeNode(1);

    root->left = new TreeNode(2);

    root->right = new TreeNode(3);

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

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

    root->left->left = new TreeNode(4);

    root->left->right = new TreeNode(5);

    std::cout << "Оригінальне дерево: ";

    print\_tree(root);

    std::cout << std::endl;

    // Створення дерева-дзеркала

    TreeNode\* mirror= create\_mirror(root);

    std::cout << "Дерево після обертання: ";

    print\_tree(mirror);

    std::cout << std::endl;

    return 0;

}

**Завдання № practice task 5**

#include <iostream>

#include <queue>

struct TreeNode

{

    int data;         // Значення вузла

    TreeNode\* left;    // Вказівник на ліву вітку

    TreeNode\* right;   // Вказівник на праву вітку

    // Конструктор для ініціалізації вузла з заданим значенням та вказівниками на ліву та праву вітки

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

};

// Функція для ітеративного обчислення сум значень підвузлів та запису їх у батьківський вузол

void tree\_sum(TreeNode\* root)

{

    if (root == nullptr)

    {

        return;

    }

    std::queue<TreeNode\*> levelOrderQueue;

    levelOrderQueue.push(root);

    while (!levelOrderQueue.empty())

    {

        TreeNode\* current = levelOrderQueue.front();

        levelOrderQueue.pop();

        // Обчислення суми значень підвузлів

        int sum = 0;

        if (current->left != nullptr)

        {

            sum += current->left->data;

            levelOrderQueue.push(current->left);

        }

        if (current->right != nullptr)

        {

            sum += current->right->data;

            levelOrderQueue.push(current->right);

        }

        // Запис суми у дерево

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

        {

            current->data = sum;

        }

    }

}

// Допоміжна функція для виведення дерева в префіксному порядку

void print\_tree(TreeNode\* root)

{

    if (root == nullptr) {

        return;

    }

    std::cout << root->data << " ";

    print\_tree(root->left);

    print\_tree(root->right);

}

int main()

{

    // Створення дерева для тестування

    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);

    std::cout << "Оригінальне дерево: ";

    print\_tree(root);

    std::cout << std::endl;

    // Виклик функції для обчислення сум та зміни значень вузлів

    tree\_sum(root);

    std::cout << "Дерево після змін: ";

    print\_tree(root);

    std::cout << std::endl;

    return 0;

}

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

Завдання № vns lab10

Програма використовує структуру Node, яка містить ключ та вказівник на наступний елемент у списку. Функції, які працюють зі списком, додають, видаляють та друкують елементи, а також забезпечують коректну роботу з пам'яттю.

Кожна операція в коді ілюструється в функції main, де створюється порожній список, до якого додаються елементи, один елемент видаляється, додається новий перед іншим елементом, і, нарешті, список знищується після використання.

Час затрачений на виконання завдання

30 хвилин

Завдання № algotester lab5

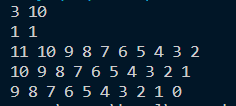
 Програма працює коректно. Відповідно до умов.

Рисунок Результати виконання програми

Час затрачений на виконання завдання

30 хвилин

Завдання № algotester lab7-8

Реалізовано клас DoublyLinkedList для роботи з двозв'язним списком. Клас включає функції для вставки, видалення, отримання розміру, отримання значення за індексом, зміни значення за індексом та виведення списку.

Програма використовує об'єкт цього класу для обробки різних запитів, таких як вставка, видалення, отримання розміру, отримання значення, зміна значення і виведення списку. Введені команди обробляються у циклі, де викликаються відповідні функції класу.

Час затрачений на виконання завдання

4-5 годин

Завдання № practice task1

Відбувається створення список з елементами 1, 2, 3, 4, 5, а потім виклакається функція реверсу. Після чого виводиться оригінальний та реверсований список.

Час затрачений на виконання завдання

45 хвилин

Завдання № practice task2

Відбувається створення двох списків (список №1 та список №2), після чого вони заповнюються деякими значеннями. Потім виводяться обидва списки і викликається функція порівняння їх за допомогою compare. Потім виводить результат порівняння.

Час затрачений на виконання завдання

45 хвилин

Завдання № practice task3

Реалізовано функцію add, яка додає два числа, представлені у вигляді однозв'язних списків. Кожен вузол списку представляє один розряд числа, а значення вузла - це цифра числа. Програма створює два вхідні числа (3->7->9 та 9->2->5), додає їх та виводить результат у вигляді однозв'язного списку (2->0->5->1, що відповідає сумі 973 та 529).

Час затрачений на виконання завдання

45 хвилин

Завдання № practice task4

Використовується структура TreeNode для представлення вузлів бінарного дерева. Програма створює оригінальне бінарне дерево, виводить його у префіксному порядку, створює дерево-дзеркало (дерево, в якому ліва та права вітки обернуті), та виводить дерево-дзеркало також у префіксному порядку.

Час затрачений на виконання завдання

45 хвилин

Завдання № practice task5

Використовує структура TreeNode для представлення вузлів бінарного дерева. Програма створює оригінальне бінарне дерево, виводить його у префіксному порядку, викликає функцію tree\_sum для обчислення та зміни значень вузлів (сума значень підвузлів), і виводить змінене дерево.

Час затрачений на виконання завдання

45 хвилин

## **5. Кооперація з командою:**

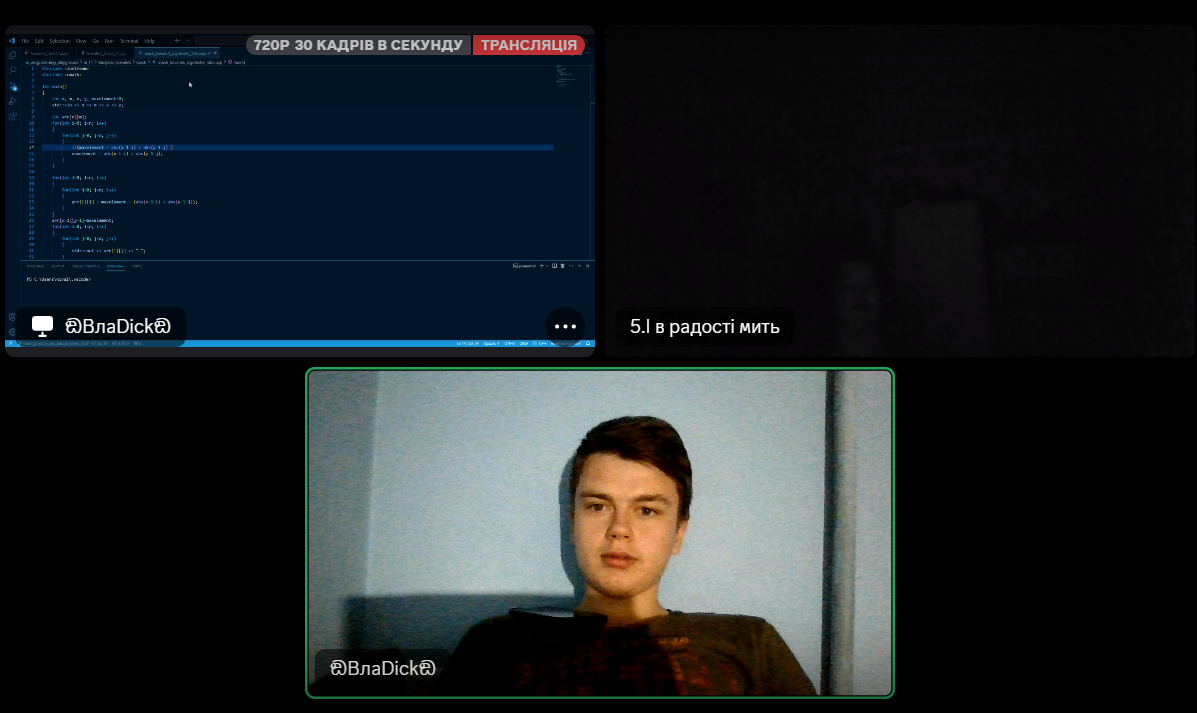
****

Рисунок 3 Скріншот з конференції 11.12.2023

## **Висновки:**

1. \*\*Робота зі списками:\*\*

- Реалізовано структуру списку, використовуючи концепції вказівників та динамічної пам'яті.

- Написано функцію для реверсу списку (Reverse list), яка змінює порядок елементів у зворотньому напрямку.

- Розроблено функцію порівняння двох списків (Compare lists), яка ітеративно порівнює значення елементів у вузлах обох списків.

2. \*\*Робота з числами, представленими у вигляді списків:\*\*

- Створено функцію для додавання двох чисел, представлених у вигляді списків. При цьому кожен вузол списку містить один розряд числа, а список представляє число, записане в зворотньому порядку.

3. \*\*Робота з бінарним деревом:\*\*

- Реалізовано структуру бінарного дерева та метод для створення його дзеркального варіанту (віддзеркалення).

- Написано функцію для обчислення суми значень підвузлів та запису цих сум у батьківські вузли дерева.