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-13

Степанюк Артем Костянтинович

# **Тема роботи:** Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.

**Мета роботи:** Вивчити динамічні структури, навчитися користуватися чергою, списком, деревом. Вивчити використання алгоритмів обробки динамічних структур.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічні структури (Черга, Стек, Списки, Дерево).
* Тема №2: Поінтери
* Тема №3: Алгоритми обробки динамічних структур

1. Індивідуальний план опрацювання теорії:

* Тема №1: Динамічні структури (Черга, Стек, Списки, Дерево).
  + Джерела Інформації
    - [17.1. Колекції: стек, черга, список, двозв'язний список, словник. C# 11 .NET 7 - YouTube](https://www.youtube.com/watch?v=crwS3QvRYWk)
  + Що опрацьовано:
    - Вивчив користування динамічних структур по відео на ютубі
  + Статус: Ознайомлено
  + Початок опрацювання теми: 16.12.2023
  + Звершення опрацювання теми: 20.12.2023
* Тема №2: Поінтери
  + Джерела Інформації:
    - [you will never ask about pointers again after watching this video - YouTube](https://www.youtube.com/watch?v=2ybLD6_2gKM)
  + Що опрацьовано:
    - Вивчив використання поінтерів по відео на ютубі
  + Статус: Ознайомлено
  + Початок опрацювання теми: 16.12.2023
  + Звершення опрацювання теми: 20.12.2023
* Тема №3: Алгоритми обробки динамічних структур
  + Джерела Інформації:
    - [Basics of Dynamic Memory Allocation - YouTube](https://www.youtube.com/watch?v=udfbq4M2Kfc)
  + Що опрацьовано:
    - Ознайомився з алгоритмами обробки динамічних структур по роліку на ютубі
  + Статус: Ознайомлено
  + Початок опрацювання теми: 16.12.2023
  + Звершення опрацювання теми: 20.12.2023

# **Виконання роботи:**

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

Завдання №1 Інформаційні динамічні структури

**Деталі завдання:**

Написати функцію для створення списку. Функція може створювати

порожній список, а потім додавати в нього елементи.

2. Написати функцію для друку списку. Функція повинна передбачати вивід

повідомлення, якщо список порожній.

3. Написати функції для знищення й додавання елементів списку у

відповідності зі своїм варіантом.

4. Виконати зміни в списку й друк списку після кожної зміни.

5. Написати функцію для запису списку у файл.

6. Написати функцію для знищення списку.

7. Записати список у файл, знищити його й виконати друк (при друці повинне

бути видане повідомлення "Список порожній").

8. Написати функцію для відновлення списку з файлу.

9. Відновити список і роздрукувати його.

10.Знищити список.

**Важливі деталі для врахування в імплементації програми:**

Записи в лінійному списку містять ключове поле типу \*char (рядок

символів). Сформувати двонаправлений список. Знищити К елементів з

кінця списку. Додати елемент після елемента із заданим ключем.

**Завдання №2** Lab 5v2

**Деталі завдання:** В пустелi iснує незвичайна печера, яка є двохвимiрною. Її висота це N, ширина - M.

Всерединi печери є пустота, пiсок та камiння. Пустота позначається буквою , пiсок S i камiння

X;

Одного дня стався землетрус i весь пiсок посипався вниз. Вiн падає на найнижчу клiтинку з

пустотою, але вiн не може пролетiти через камiння.

Ваше завдання сказати як буде виглядати печера пiсля землетрусу.

Вхiднi данi

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

У N наступних рядках стрiчка rowi яка складається з N цифер - i-й рядок матрицi, яка

вiдображає стан печери до землетрусу.

Вихiднi данi

N рядкiв, якi складаються з стрiчки розмiром M - стан печери пiсля землетрусу.

**Важливі деталі для врахування в імплементації програми:**

Обмеження

1 ≤ N, M ≤ 1000

|rowi| = M

rowi ∈ {X, S, O}

**Завдання №3 Algotester Lab 7-8 V2**

**Деталі завдання:** Ваше завдання - власноруч реалiзувати структуру даних "Динамiчний масив".

Ви отримаєте Q запитiв, кожен запит буде починатися зi слова-iдентифiкатора, пiсля якого

йдуть його аргументи.

Вам будуть поступати запити такого типу:

• Вставка:

Iдентифiкатор - insert

Ви отримуєте цiле число index елемента, на мiсце якого робити вставку.

Пiсля цього в наступному рядку рядку написане число N - розмiр масиву, який треба вста-

вити.

У третьому рядку N цiлих чисел - масив, який треба вставити на позицiю index.

• Видалення:

Iдентифiкатор - erase

Ви отримуєте 2 цiлих числа - index, iндекс елемента, з якого почати видалення та n -

кiлькiсть елементiв, яку треба видалити.

• Визначення розмiру:

Iдентифiкатор - size

Ви не отримуєте аргументiв.

Ви виводите кiлькiсть елементiв у динамiчному масивi.

• Визначення кiлькостi зарезервованої пам’ятi:

Iдентифiкатор - capacity

Ви не отримуєте аргументiв.

Ви виводите кiлькiсть зарезервованої пам’ятi у динамiчному масивi.

Ваша реалiзацiя динамiчного масиву має мати фактор росту (Growth factor) рiвний 2.

• Отримання значення i-го елементу

Iдентифiкатор - get

Ви отримуєте цiле число - index, iндекс елемента.

Ви виводите значення елемента за iндексом. Реалiзувати використовуючи перегрузку опе-

ратора []

• Модифiкацiя значення i-го елементу

Iдентифiкатор - set

Ви отримуєте 2 цiлих числа - iндекс елемента, який треба змiнити, та його нове значення.

Реалiзувати використовуючи перегрузку оператора []

• Вивiд динамiчного масиву на екран

Iдентифiкатор - print

Ви не отримуєте аргументiв.

Ви виводите усi елементи динамiчного масиву через пробiл.

Реалiзувати використовуючи перегрузку оператора <<

Вхiднi данi

Цiле число Q - кiлькiсть запитiв.

У наступних рядках Q запитiв у зазначеному в умовi форматi.

Вихiднi данi

Вiдповiдi на запити у зазначеному в умовi форматi.

**Важливі деталі для врахування в імплементації програми:**

Обмеження

0 ≤ Q ≤ 105

0 ≤ li ≤ 105

klk ≤ 105

**Завдання №4 Практична(1)**

**Деталі завдання:** 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++.

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

**Важливі деталі для врахування в імплементації програми:**

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.

**Завдання №5 Практична(2)**

**Деталі завдання:** 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.

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, дозволяють 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.

Розвинути 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.

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

**Важливі деталі для врахування в імплементації програми:**

**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.

**Завдання №6 Практична(3)**

**Деталі завдання:** 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 операції 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.

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

**Завдання №7 Практична(4)**

**Деталі завдання:** 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.

**Важливі деталі для врахування в імплементації програми:**

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

**Завдання №8 Практична(5)**

**Деталі завдання:**

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 розумі

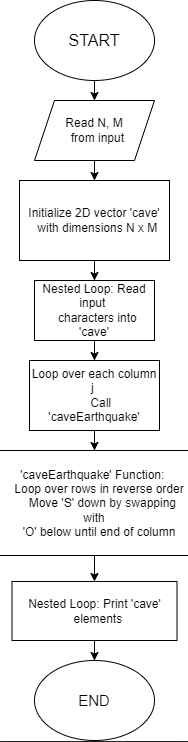
ння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

**Важливі деталі для врахування в імплементації програми:**

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

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

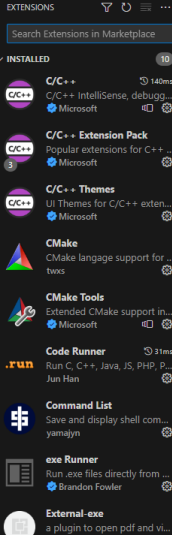
Завдання №2: Lab 5v2



*Блок-схема для* *алготестеру*

## **3. Конфігурація середовища до виконання завдань:**

Завдання №1. Конфігурація середовища

. ****

*Конфігурація мого середовища*

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

**Завдання №1: Інформаційні динамічні структури**

**#include <iostream>**

**#include <fstream>**

**#include <cstring>**

**struct Node {**

**char\* key;**

**Node\* next;**

**Node\* prev;**

**};**

**Node\* createNode(const char\* value) {**

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

**newNode->key = strdup(value);**

**newNode->next = nullptr;**

**newNode->prev = nullptr;**

**return newNode;**

**}**

**void appendNode(Node\*& head, const char\* value) {**

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

**if (!head) {**

**head = newNode;**

**} else {**

**Node\* current = head;**

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

**current = current->next;**

**}**

**current->next = newNode;**

**newNode->prev = current;**

**}**

**}**

**void printList(Node\* head) {**

**if (!head) {**

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

**return;**

**}**

**Node\* current = head;**

**while (current) {**

**std::cout << current->key << " ";**

**current = current->next;**

**}**

**std::cout << std::endl;**

**}**

**void saveListToFile(Node\* head, const char\* filename) {**

**std::ofstream file(filename);**

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

**std::cerr << "Не вдалося відкрити файл для запису." << std::endl;**

**return;**

**}**

**Node\* current = head;**

**while (current) {**

**file << current->key << std::endl;**

**current = current->next;**

**}**

**file.close();**

**}**

**void deleteKNodesFromEnd(Node\*& head, int k) {**

**if (!head) {**

**return;**

**}**

**Node\* current = head;**

**int length = 0;**

**while (current) {**

**length++;**

**current = current->next;**

**}**

**current = head;**

**for (int i = 0; i < length - k - 1; i++) {**

**current = current->next;**

**}**

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

**Node\* temp = current->next;**

**current->next = temp->next;**

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

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

**}**

**delete temp;**

**}**

**}**

**void insertAfterKey(Node\*& head, const char\* keyToFind, const char\* newValue) {**

**Node\* current = head;**

**while (current) {**

**if (strcmp(current->key, keyToFind) == 0) {**

**Node\* newNode = createNode(newValue);**

**newNode->next = current->next;**

**newNode->prev = current;**

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

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

**}**

**current->next = newNode;**

**return;**

**}**

**current = current->next;**

**}**

**}**

**void deleteList(Node\*& head) {**

**while (head) {**

**Node\* temp = head;**

**head = head->next;**

**delete[] temp->key;**

**delete temp;**

**}**

**}**

**int main() {**

**Node\* myList = nullptr;**

**appendNode(myList, "A");**

**appendNode(myList, "B");**

**appendNode(myList, "C");**

**appendNode(myList, "D");**

**std::cout << "Наш початковий список: ";**

**printList(myList);**

**int k = 2;**

**deleteKNodesFromEnd(myList, k);**

**std::cout << "Після видалення " << k << " елементів з кінця: ";**

**printList(myList);**

**const char\* keyToFind = "B";**

**const char\* newValue = "X";**

**insertAfterKey(myList, keyToFind, newValue);**

**std::cout << "Після вставки нового елемента після нашого ключа " << keyToFind << ": ";**

**printList(myList);**

**saveListToFile(myList, "moyfile.txt");**

**deleteList(myList);**

**return 0;**

**}**

**Завдання №2 Lab 5v2**

#include <iostream>

#include <vector>

using namespace std;

void caveEarthquake(vector<vector<char>>& cave, int col) {

for (int i = cave.size() - 1; i >= 0; --i) {

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

int k = i + 1;

while (k < cave.size() && cave[k][col] == 'O') {

swap(cave[k][col], cave[k - 1][col]);

++k;

}

}

}

}

int main() {

int N, M;

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];

}

}

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

caveEarthquake(cave, j);

}

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 7-8 V2**

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class DynamicArr {

T\* array;

int size;

int capacity;

public:

DynamicArr() {

capacity = 1;

array = new T[capacity];

size = 0;

}

~DynamicArr() {

delete[] array;

}

void get\_size() const {

cout << size << endl;

}

void get\_capacity() const {

cout << capacity << endl;

}

void get\_el(int index) const {

cout << array[index] << endl;

}

void set\_el(int index, T number) {

array[index] = number;

}

void insert\_el(int index, T\* numbers, int N) {

if (size + N >= capacity) {

while (size + N >= capacity) {

capacity \*= 2;

}

T\* newArray = new T[capacity];

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

newArray[i] = array[i];

}

delete[] array;

array = newArray;

}

T\* newArray = new T[capacity];

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

newArray[i] = array[i];

}

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

newArray[i] = numbers[j];

}

for (int i = index + N, j = index; j < size; i++, j++) {

newArray[i] = array[j];

}

size += N;

delete[] array;

array = newArray;

}

void erase\_el(int index1, int num) {

int index2 = index1 + num - 1;

T\* newArray = new T[capacity];

for (int i = 0; i < index1; i++) {

newArray[i] = array[i];

}

for (int i = index1, j = index2 + 1; j < size; i++, j++) {

newArray[i] = array[j];

}

size = size - num;

delete[] array;

array = newArray;

}

friend ostream& operator<<(ostream& out, const DynamicArr<T>& arr) {

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

out << arr.array[i] << " ";

}

return out;

}

};

int main() {

DynamicArr<int> arr;

int Q;

cin >> Q;

string requests;

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

cin >> requests;

if (requests == "insert") {

int index, N;

cin >> index >> N;

int\* numbers = new int[N];

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

cin >> numbers[i];

}

arr.insert\_el(index, numbers, N);

delete[] numbers;

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

arr.get\_size();

} else if (requests == "capacity") {

arr.get\_capacity();

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

int index;

cin >> index;

arr.get\_el(index);

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

int index, number;

cin >> index >> number;

arr.set\_el(index, number);

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

cout << arr;

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

int index1, num;

cin >> index1 >> num;

arr.erase\_el(index1, num);

}

}

return 0;

}

**Завдання №4 Практична(1)**

#include <iostream>

#include <memory>

template <typename T>

struct Node {

T data;

std::shared\_ptr<Node<T>> next;

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

};

template <typename T>

std::shared\_ptr<Node<T>> reverse(std::shared\_ptr<Node<T>> head) {

std::shared\_ptr<Node<T>> prev = nullptr;

auto current = head;

std::shared\_ptr<Node<T>> next = nullptr;

while (current != nullptr) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev;

}

template <typename T>

void printList(const std::shared\_ptr<Node<T>>& head) {

auto current = head;

while (current != nullptr) {

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

current = current->next;

}

std::cout << std::endl;

}

int main() {

auto head = std::make\_shared<Node<int>>(1);

head->next = std::make\_shared<Node<int>>(2);

head->next->next = std::make\_shared<Node<int>>(3);

head->next->next->next = std::make\_shared<Node<int>>(4);

head->next->next->next->next = std::make\_shared<Node<int>>(5);

std::cout << "Вихідний список: ";

printList(head);

head = reverse(head);

std::cout << "Обернений список: ";

printList(head);

return 0;

}

**Завдання №5 Практична (2)**

#include <iostream>

#include <fstream>

struct ListNode {

int value;

ListNode\* next;

ListNode(int val) : value(val), next(nullptr) {}

};

bool areListsEqual(ListNode\* head1, ListNode\* head2) {

while (head1 != nullptr && head2 != nullptr) {

if (head1->value != head2->value) {

return false;

}

head1 = head1->next;

head2 = head2->next;

}

return (head1 == nullptr && head2 == nullptr);

}

void writeListToFile(const ListNode\* head, const std::string& filename) {

std::ofstream file(filename);

while (head != nullptr) {

file << head->value << " ";

head = head->next;

}

file.close();

}

ListNode\* readListFromFile(const std::string& filename) {

std::ifstream file(filename);

ListNode\* head = nullptr;

ListNode\* current = nullptr;

int value;

while (file >> value) {

if (head == nullptr) {

head = new ListNode(value);

current = head;

} else {

current->next = new ListNode(value);

current = current->next;

}

}

file.close();

return head;

}

int main() {

ListNode\* list1 = new ListNode(10);

list1->next = new ListNode(20);

list1->next->next = new ListNode(30);

writeListToFile(list1, "list1.txt");

ListNode\* list2 = new ListNode(10);

list2->next = new ListNode(20);

list2->next->next = new ListNode(30);

writeListToFile(list2, "list2.txt");

ListNode\* readList1 = readListFromFile("list1.txt");

ListNode\* readList2 = readListFromFile("list2.txt");

if (areListsEqual(readList1, readList2)) {

std::cout << "Списки ідентичні.\n";

} else {

std::cout << "Списки різні.\n";

}

delete list1;

delete list2;

delete readList1;

delete readList2;

return 0;

}

**Завдання №6 Практична (3)**

**#include <iostream>**

**struct Node {**

**int data;**

**Node \*next;**

**Node(int val) : data(val), 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 val1 = (n1 != nullptr) ? n1->data : 0;**

**int val2 = (n2 != nullptr) ? n2->data : 0;**

**int sum = val1 + val2 + carry;**

**carry = sum / 10;**

**sum = sum % 10;**

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

**if (result == nullptr) {**

**result = newNode;**

**current = result;**

**} else {**

**current->next = newNode;**

**current = newNode;**

**}**

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

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

**}**

**return result;**

**}**

**void printList(Node \*head) {**

**while (head != nullptr) {**

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

**head = head->next;**

**}**

**std::cout << std::endl;**

**}**

**void deleteList(Node \*head) {**

**while (head != nullptr) {**

**Node \*temp = head;**

**head = head->next;**

**delete temp;**

**}**

**}**

**int main() {**

**Node \*num1 = new Node(3);**

**num1->next = new Node(7);**

**num1->next->next = new Node(9);**

**Node \*num2 = new Node(4);**

**num2->next = new Node(5);**

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

**std::cout << "Перше число: ";**

**printList(num1);**

**std::cout << "Друге число: ";**

**printList(num2);**

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

**std::cout << "Сума: ";**

**printList(sum);**

**deleteList(num1);**

**deleteList(num2);**

**deleteList(sum);**

**return 0;**

**}**

**Завдання №7 Практична (4)**

**#include <iostream>**

**struct TreeNode {**

**int data;**

**TreeNode\* left;**

**TreeNode\* right;**

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

**};**

**TreeNode\* createMirror(TreeNode\* root) {**

**if (root == nullptr) {**

**return nullptr;**

**}**

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

**mirroredNode->left = createMirror(root->right);**

**mirroredNode->right = createMirror(root->left);**

**return mirroredNode;**

**}**

**void printInOrder(TreeNode\* root) {**

**if (root != nullptr) {**

**printInOrder(root->left);**

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

**printInOrder(root->right);**

**}**

**}**

**int main() {**

**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 (зворотній обхід): ";**

**printInOrder(root);**

**std::cout << std::endl;**

**TreeNode\* mirroredRoot = createMirror(root);**

**std::cout << "Віддзеркалене дерево (зворотній обхід): ";**

**printInOrder(mirroredRoot);**

**std::cout << std::endl;**

**return 0;**

**}**

**Завдання №8 Практична (5)**

#include <iostream>

#include <stack>

struct BinaryTreeNode {

int value;

BinaryTreeNode\* left;

BinaryTreeNode\* right;

BinaryTreeNode(int val) : value(val), left(nullptr), right(nullptr) {}

};

void calculateSubtreeSum(BinaryTreeNode\* root) {

if (!root) {

return;

}

std::stack<BinaryTreeNode\*> nodes;

BinaryTreeNode\* current = root;

BinaryTreeNode\* prev = nullptr;

while (current || !nodes.empty()) {

while (current) {

nodes.push(current);

current = current->left;

}

current = nodes.top();

nodes.pop();

if (current->right && current->right != prev) {

nodes.push(current);

current = current->right;

} else {

current->value += (current->left ? current->left->value : 0) +

(current->right ? current->right->value : 0);

prev = current;

current = nullptr;

}

}

}

BinaryTreeNode\* createSampleBinaryTree() {

BinaryTreeNode\* root = new BinaryTreeNode(1);

root->left = new BinaryTreeNode(2);

root->right = new BinaryTreeNode(3);

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

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

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

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

return root;

}

void printBinaryTree(BinaryTreeNode\* root) {

if (!root) {

return;

}

std::stack<BinaryTreeNode\*> nodes;

BinaryTreeNode\* current = root;

while (current || !nodes.empty()) {

while (current) {

nodes.push(current);

current = current->left;

}

current = nodes.top();

nodes.pop();

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

current = current->right;

}

}

int main() {

BinaryTreeNode\* root = createSampleBinaryTree();

std::cout << "Дерево до зміни значень: ";

printBinaryTree(root);

std::cout << std::endl;

calculateSubtreeSum(root);

std::cout << "Дерево після зміни значень: ";

printBinaryTree(root);

std::cout << std::endl;

return 0;

}

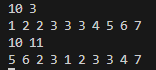
## **5. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1 Інформаційні динамічні структури

**

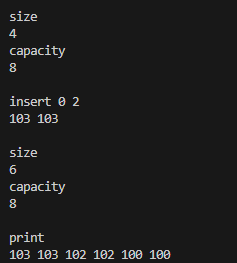
*Те, що мені видає програма 1*

Завдання №2 **Lab 5v2**

**

*Те, що мені видає програма 2*

Завдання №3: **Algotester Lab 7-8 V2**



*Те, що мені видає програма 3*

Завдання №4 Практична(1)

**

*Те, що мені видає програма 4*

Завдання №5 Практична(2)



*Те, що мені видає програма 5*

Завдання №6 Практична(3)



*Те, що мені видає програма 6*

Завдання №7 Практична(4)



*Те, що мені видає програма 7*

Завдання №8 Практична(5)

****

*Те, що мені видає програма 8*

**Висновки:**

У ході виконання лабораторної роботи з програмування на мові C++ з теми "Динамічні структури" були вивчені та реалізовані основні алгоритми обробки динамічних структур, таких як черга, стек, списки та дерево.Черга, Стек, Списки, Дерево. Завдяки виконанню цієї лабораторної роботи, отримані навички роботи з динамічними структурами даних та їх алгоритмами обробки. Розуміння принципів роботи черги, стеку, списків і дерева є важливим етапом у розвитку навичок програмування.