Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**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

**Виконала:**

Студентка групи ШІ-13

Фесенко Дар’я Максимівна

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

**Мета роботи:** Створити програму з використанням динамічних структур.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

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

1. Індивідуальний план опрацювання теорії:

* Тема №1: Динамічні структури (Черга, Стек, Списки, Дерево).
  + Джерела Інформації
    - <http://elcat.pnpu.edu.ua/docs/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8%20%D1%96%20%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%B8%D1%85/lab6_stack.html>
    - https://stud.com.ua/97380/informatika/dinamichni\_strukturi\_danih
  + Що опрацьовано:
    - Принцип роботи з динамічними структурами;
  + Статус: Ознайомлена
  + Початок опрацювання теми: 24.11.2023
  + Звершення опрацювання теми: 25.11.2023
* Тема №2: Алгоритми обробки динамічних структур.
  + Джерела Інформації:
  + <http://cpp.dp.ua/dynamichni-masyvy/>
  + Що опрацьовано:
    - Алгоритми обробки динамічних структур
  + Статус: Ознайомлена
  + Початок опрацювання теми: 25.11.2023
  + Звершення опрацювання теми: 27.11.2023

# **Виконання роботи:**

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

Завдання №1 "Інформаційні динамічні структури".

**Деталі завдання:**

1. 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.

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

символів). Сформувати двонаправлений список. Знищити елемент із заданим

номером. Додати по К елементів перед і після елемента із заданим ключем.

Завдання №2 Algotester lab 5.

**Деталі завдання:** 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. Input У першому рядку 2 числа N та M - розміри карти у другому рядку 2 числа x та y - координати піку гори Output N рядків по M елементів в рядку через пробіл - висоти карти.

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

**Constraints** 1≤N,M≤1031≤x≤N1≤y≤M

Завдання №3 Algotester lab 7-8.

**Деталі завдання:** 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 - i n s e r t Ви отримуєте ціле число i n d e x 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 i n d e x . Видалення: Ідентифікатор - e r a s e Ви отримуєте 2 цілих числа - i n d e x , індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити. Визначення розміру: Ідентифікатор - s i z e Ви не отримуєте аргументів. Ви виводите кількість елементів у динамічному масиві. Визначення кількості зарезервованої пам’яті: Ідентифікатор - c a p a c i t y 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 - g e t Ви отримуєте ціле число - i n d e x , індекс елемента. Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора [ ] Модифікація значення i -го елементу Ідентифікатор - s e t Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора [ ] Вивід динамічного масиву на екран Ідентифікатор - p r i n t Ви не отримуєте аргументів. Ви виводите усі елементи динамічного масиву через пробіл. Реалізувати використовуючи перегрузку оператора <<

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

Output Відповіді на запити у зазначеному в умові форматі.

Завдання №4.1 Practice Task

**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.

### 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 №4.2 Practice Task

**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.

### 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 №4.3 Practice Task

## **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.

***Розвинути навик вирішення проблем і увага до деталей:*** завдання покращує розуміння обмежень у представленні цілого числа сучасними комп’ютерами та пропонує спосіб його вирішення.

Завдання №4.4 Practice Task

**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.

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

Завдання №4.5 Practice Task

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.

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

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

Програма №4.2 Class Practice.

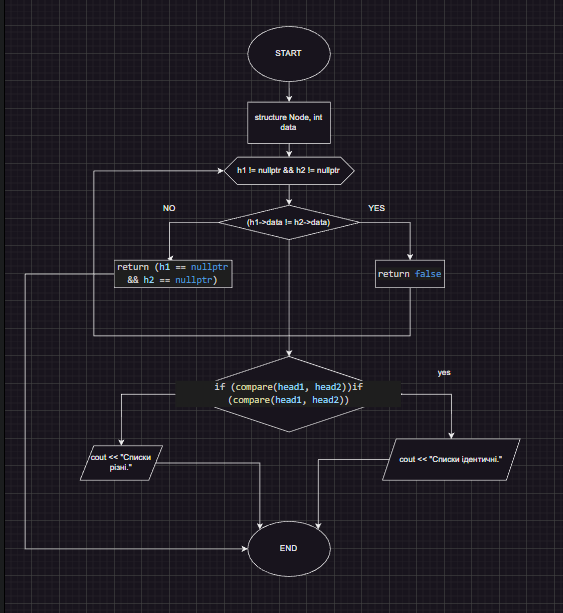


Рисунок Блок - схема до програми 4.2

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

Завдання №1. Конфігурація середовища

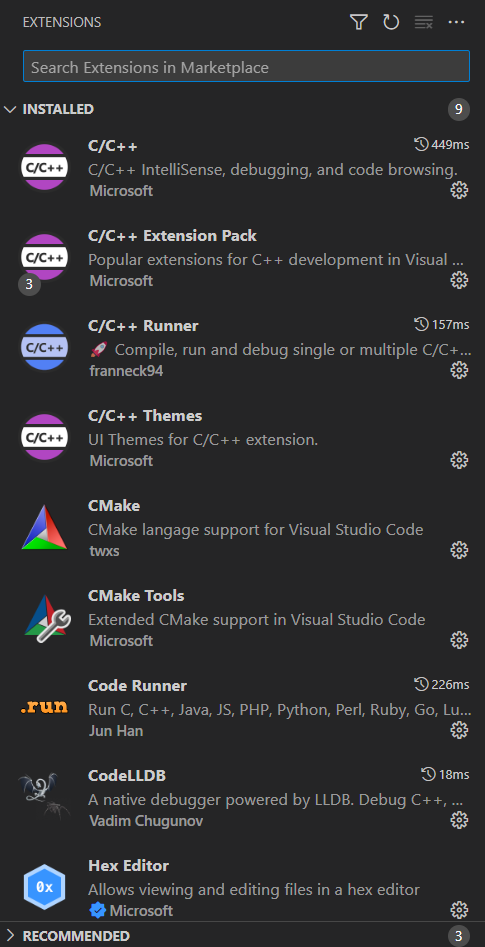
. 

Рисунок Встановлені розширення

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

Завдання №1: " Інформаційні динамічні структури "

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

using namespace std;

struct Node {

    char key[100];

    Node\* prev;

    Node\* next;

};

Node\* createList() {

    return nullptr;

}

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

    Node\* newNode = new Node;

    strcpy(newNode->key, key);

    newNode->next = nullptr;

    if (head == nullptr) {

        newNode->prev = nullptr;

        head = newNode;

    } else {

        Node\* temp = head;

        while (temp->next != nullptr) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

    }

}

void deleteNodeByIndex(Node\*& head, int index) {

    if (head == nullptr) {

        cout << "Список порожній\n";

        return;

    }

    Node\* temp = head;

    for (int i = 0; i < index && temp != nullptr; i++) {

        temp = temp->next;

    }

    if (temp == nullptr) {

        cout << "Елемент з номером " << index << " не існує\n";

        return;

    }

    if (temp->prev != nullptr) {

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

    } else {

        head = temp->next;

    }

    if (temp->next != nullptr) {

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

    }

    delete temp;

}

void printList(Node\* head) {

    if (head == nullptr) {

        cout << "Список порожній\n";

        return;

    }

    Node\* temp = head;

    while (temp != nullptr) {

        cout << temp->key << " ";

        temp = temp->next;

    }

    cout << endl;

}

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

    ofstream outFile(filename);

    if (!outFile.is\_open()) {

        cout << "Помилка відкриття файлу\n";

        return;

    }

    Node\* temp = head;

    while (temp != nullptr) {

        outFile << temp->key << endl;

        temp = temp->next;

    }

    outFile.close();

}

void deleteList(Node\*& head) {

    while (head != nullptr) {

        Node\* temp = head;

        head = head->next;

        delete temp;

    }

}

Node\* restoreListFromFile(const char\* filename) {

    Node\* head = nullptr;

    ifstream inFile(filename);

    if (!inFile.is\_open()) {

        cout << "Помилка відкриття файлу\n";

        return nullptr;

    }

    char key[100];

    while (inFile.getline(key, sizeof(key))) {

        appendNode(head, key);

    }

    inFile.close();

    return head;

}

int main() {

    Node\* myList = createList();

    appendNode(myList, "Елемент1");

    appendNode(myList, "Елемент2");

    appendNode(myList, "Елемент3");

    cout << "Список до змін:\n";

    printList(myList);

    deleteNodeByIndex(myList, 1);

    cout << "Список після видалення:\n";

    printList(myList);

    appendNode(myList, "Елемент4");

    cout << "Список після додавання елемента:\n";

    printList(myList);

    writeListToFile(myList, "myList.txt");

    deleteList(myList);

    cout << "Список після видалення (повинно бути порожній):\n";

    printList(myList);

    myList = restoreListFromFile("myList.txt");

    cout << "Відновлений список:\n";

    printList(myList);

    deleteList(myList);

    return 0;

}

Завдання №2 Algotester lab 5.

#include <iostream>

#include <cmath>

#include <vector>

int main() {

    size\_t n, m;

    std::cin >> n >> m;

    size\_t x, y, distance = 0;

    std::cin >> x >> y;

    std::vector<std::vector<size\_t>> mountain(n, std::vector<size\_t>(m, 0));

    for (size\_t i = 0; i < n; i++){

        for (size\_t j = 0; j < m; j++){

            size\_t tempdist = abs(x - 1 - i) + abs(y - 1 - j);

            mountain[i][j] = tempdist;

            if (tempdist > distance){

                distance = tempdist;

            }

        }

    }

    for (size\_t i = 0; i < n; i++){

        for (size\_t j = 0; j < m; j++){

            std::cout << distance - mountain[i][j] << ' ';

        }

        std::cout << std::endl;

    }

    return 0;

}

Завдання №3 Algotester lab 7-8.

#include <iostream>

class DynamicArray {

private:

    int\* array;

    int currentSize;

    int currentCapacity;

    void resize(int newCapacity) {

        int\* tempArray = new int[newCapacity];

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

            tempArray[i] = array[i];

        }

        delete[] array;

        array = tempArray;

        currentCapacity = newCapacity;

    }

public:

    DynamicArray() : array(nullptr), currentSize(0), currentCapacity(1) {

        array = new int[currentCapacity];

    }

    ~DynamicArray() {

        delete[] array;

    }

    void insert(int index, int N, int\* elements) {

        if (currentSize + N >= currentCapacity) {

            while (currentSize + N >= currentCapacity) {

                currentCapacity \*= 2;

            }

            resize(currentCapacity);

        }

        for (int i = currentSize - 1; i >= index; --i) {

            array[i + N] = array[i];

        }

        for (int i = 0; i < N; ++i) {

            array[index + i] = elements[i];

        }

        currentSize += N;

    }

    void erase(int index, int n) {

        for (int i = index; i < currentSize - n; ++i) {

            array[i] = array[i + n];

        }

        currentSize -= n;

    }

    int get(int index) const {

        return array[index];

    }

    void set(int index, int value) {

        array[index] = value;

    }

    int size() const {

        return currentSize;

    }

    int capacity() const {

        return currentCapacity;

    }

    void print() const {

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

            std::cout << array[i] << " ";

        }

        std::cout << std::endl;

    }

};

int main() {

    int Q;

    std::cin >> Q;

    DynamicArray dynamicArray;

    for (int i = 0; i < Q; ++i) {

        std::string query;

        std::cin >> query;

        if (query == "insert") {

            int index, N;

            std::cin >> index >> N;

            int\* elements = new int[N];

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

                std::cin >> elements[j];

            }

            dynamicArray.insert(index, N, elements);

            delete[] elements;

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

            int index, n;

            std::cin >> index >> n;

            dynamicArray.erase(index, n);

        } else if (query == "size") {

            std::cout << dynamicArray.size() << std::endl;

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

            std::cout << dynamicArray.capacity() << std::endl;

        } else if (query == "get") {

            int index;

            std::cin >> index;

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

        } else if (query == "set") {

            int index, value;

            std::cin >> index >> value;

            dynamicArray.set(index, value);

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

            dynamicArray.print();

        }

    }

    return 0;

}

Завдання №4.1 Class Practice.

#include <iostream>

using namespace std;

struct Node {

    int data;

    Node\* next;

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

};

Node\* reverse(Node\* head) {

    Node \*prev = nullptr, \*current = head, \*nextNode = nullptr;

    while (current != nullptr) {

        nextNode = current->next;

        current->next = prev;

        prev = current;

        current = nextNode;

    }

    return prev;

}

void printList(Node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    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);

    cout << "Оригінальний список: ";

    printList(head);

    head = reverse(head);

    cout << "Реверсований список: ";

    printList(head);

    while (head != nullptr) {

        Node\* temp = head;

        head = head->next;

        delete temp;

    }

    return 0;

}

Завдання №4.2 Class Practice.

#include <iostream>

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;

    }

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

}

void printList(Node\* head) {

    while (head != nullptr) {

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

        head = head->next;

    }

    std::cout << std::endl;

}

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);

    if (compare(head1, head2)) {

        std::cout << "Списки ідентичні." << std::endl;

    } else {

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

    }

    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 №4.3 Class Practice.

#include <iostream>

struct Node {

    int data;

    Node\* next;

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

};

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

    Node\* result = nullptr;

    Node\* current = nullptr;

    int carry = 0;

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

        int sum = carry;

        if (n1 != nullptr) {

            sum += n1->data;

            n1 = n1->next;

        }

        if (n2 != nullptr) {

            sum += n2->data;

            n2 = n2->next;

        }

        carry = sum / 10;

        sum %= 10;

        if (result == nullptr) {

            result = new Node(sum);

            current = result;

        } else {

            current->next = new Node(sum);

            current = current->next;

        }

    }

    return result;

}

void printList(Node\* head) {

    while (head != nullptr) {

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

        head = head->next;

    }

    std::cout << std::endl;

}

int main() {

    Node\* num1 = new Node(3);

    num1->next = new Node(2);

    num1->next->next = new Node(1);

    Node\* num2 = new Node(6);

    num2->next = new Node(5);

    num2->next->next = new Node(4);

    Node\* result = add(num1, num2);

    std::cout << "Сума чисел: ";

    printList(result);

    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 №4.4 Class Practice.

#include <iostream>

struct TreeNode {

    int data;

    TreeNode\* left;

    TreeNode\* right;

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

};

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;

}

void printTree(TreeNode\* root) {

    if (root == nullptr) {

        return;

    }

    std::cout << root->data << " ";

    printTree(root->left);

    printTree(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);

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

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

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

    printTree(root);

    std::cout << std::endl;

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

    std::cout << "Віддзеркалене бінарне дерево: ";

    printTree(mirroredRoot);

    std::cout << std::endl;

    return 0;

}

Завдання №4.5 Class Practice.

#include <iostream>

#include <queue>

struct TreeNode {

    int data;

    TreeNode\* left;

    TreeNode\* right;

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

};

void tree\_sum(TreeNode\* root) {

    if (root == nullptr) {

        return;

    }

    std::queue<TreeNode\*> q;

    q.push(root);

    while (!q.empty()) {

        TreeNode\* current = q.front();

        q.pop();

        int sum = 0;

        if (current->left != nullptr) {

            sum += current->left->data;

            q.push(current->left);

        }

        if (current->right != nullptr) {

            sum += current->right->data;

            q.push(current->right);

        }

        current->data = sum;

    }

}

void printTree(TreeNode\* root) {

    if (root == nullptr) {

        return;

    }

    std::queue<TreeNode\*> q;

    q.push(root);

    while (!q.empty()) {

        TreeNode\* current = q.front();

        q.pop();

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

        if (current->left != nullptr) {

            q.push(current->left);

        }

        if (current->right != nullptr) {

            q.push(current->right);

        }

    }

    std::cout << std::endl;

}

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);

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

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

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

    printTree(root);

    std::cout << std::endl;

    tree\_sum(root);

    std::cout << "Бінарне дерево після запису сум підвузлів: ";

    printTree(root);

    std::cout << std::endl;

    return 0;

}

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

Завдання №1: " Інформаційні динамічні структури "

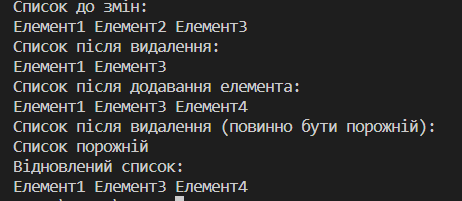


Рисунок Результат виконання програми 1

Час затрачений на виконання завдання: 25 хв.

Завдання №2 Algotester lab 5.

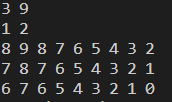


Рисунок Результат виконання програми 2

Час затрачений на виконання завдання: 2 години.

Завдання №3 Algotester lab 7-8.

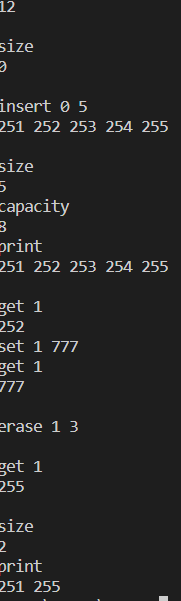


Рисунок Результат виконання програми 3

Час затрачений на виконання завдання: 2 години.

Завдання №4.1 Class Practice.



Рисунок Результат виконання програми 4.1

Час затрачений на виконання завдання: 2 години.

Завдання №4.2 Class Practice.



Рисунок Результат виконання програми 4.2

Час затрачений на виконання завдання: 1 година.

Завдання №4.3 Class Practice.



Рисунок Результат виконання програми 4.3

Час затрачений на виконання завдання: 1 година.

Завдання №4.4 Class Practice.



Рисунок Результат виконання програми 4.4

Час затрачений на виконання завдання: 1 година.

Завдання №4.5 Class Practice.

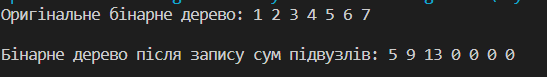


Рисунок Результат виконання програми 4.5

Час затрачений на виконання завдання: 1 година.

# **Висновки:** Створити програму з використанням динамічних структур.

**Pull request:** https://github.com/artificial-intelligence-department/ai\_programming\_playground/pull/468