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»

**Виконала:**

Студентка групи ШІ-12

Куць Софія Тарасівна

Львів – 2024 р.

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

Мета: Ознайомитись з основами та особливостями динамічних структур даних.

**Виконання завдань :**

Завдання №1 (VNS Lab 10)

Написати програму, у якій створюються динамічні структури й виконати

їхню обробку у відповідності зі своїм варіантом.

Для кожного варіанту розробити такі функції:

1. Створення списку.

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

3. Знищення елемента зі списку (у відповідності зі своїм варіантом).

4. Друк списку.

5. Запис списку у файл.

6. Знищення списку.

7. Відновлення списку з файлу.

Порядок виконання роботи

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

ключем. Додати по К елементів на початок й в кінець списку.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

char key[100];

struct Node\* next;

struct Node\* prev;

} Node;

Node\* createList() {

return NULL;

}

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

Node\* newNode = (Node\*)malloc(sizeof(Node));

strcpy(newNode->key, key);

newNode->next = NULL;

if (\*head == NULL) {

newNode->prev = NULL;

\*head = newNode;

} else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

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

Node\* newNode = (Node\*)malloc(sizeof(Node));

strcpy(newNode->key, key);

newNode->prev = NULL;

newNode->next = \*head;

if (\*head != NULL) {

(\*head)->prev = newNode;

}

\*head = newNode;

}

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

Node\* temp = \*head;

while (temp != NULL && strcmp(temp->key, key) != 0) {

temp = temp->next;

}

if (temp != NULL) {

if (temp->prev != NULL) {

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

} else {

\*head = temp->next;

}

if (temp->next != NULL) {

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

}

free(temp);

}

}

void printList(Node\* head) {

if (head == NULL) {

printf("Список порожній.\n");

return;

}

Node\* temp = head;

while (temp != NULL) {

printf("%s ", temp->key);

temp = temp->next;

}

printf("\n");

}

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

FILE\* file = fopen(filename, "w");

if (file == NULL) {

printf("Помилка відкриття файлу.\n");

return;

}

Node\* temp = head;

while (temp != NULL) {

fprintf(file, "%s\n", temp->key);

temp = temp->next;

}

fclose(file);

}

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

FILE\* file = fopen(filename, "r");

if (file == NULL) {

printf("Помилка відкриття файлу.\n");

return NULL;

}

Node\* head = NULL;

char key[100];

while (fgets(key, sizeof(key), file)) {

key[strcspn(key, "\n")] = '\0';

addToEnd(&head, key);

}

fclose(file);

return head;

}

void destroyList(Node\*\* head) {

Node\* temp = \*head;

while (temp != NULL) {

Node\* next = temp->next;

free(temp);

temp = next;

}

\*head = NULL;

}

int main() {

Node\* list = createList();

addToEnd(&list, "apple");

addToEnd(&list, "banana");

addToEnd(&list, "cherry");

printf("Список після додавання елементів:\n");

printList(list);

addToFront(&list, "kiwi");

addToFront(&list, "grape");

printf("Список після додавання елементів на початок:\n");

printList(list);

deleteNode(&list, "banana");

printf("Список після знищення елемента 'banana':\n");

printList(list);

saveToFile(list, "list.txt");

printf("Список записано у файл.\n");

destroyList(&list);

printf("Список знищено.\n");

printList(list);

list = loadFromFile("list.txt");

printf("Список відновлено з файлу:\n");

printList(list);

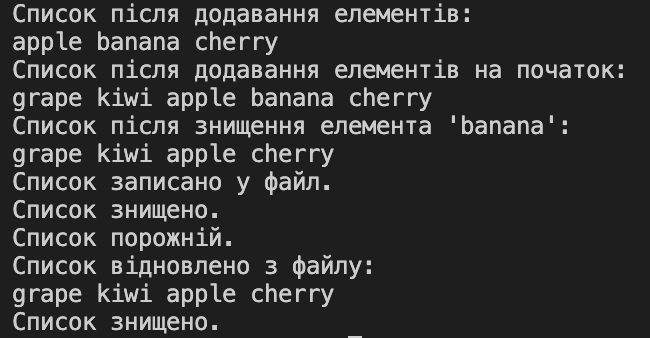
destroyList(&list);

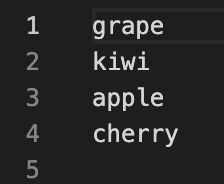
printf("Список знищено.\n");

return 0;

}

Результат:





Час виконання 2,5 год

Завдання №2 (Algotester Lab 5)

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.

#include <iostream>

#include <cmath>

#include <algorithm>

int main()

{

int N, M;

std::cin >> N >> M;

int x, y;

std::cin >> x >> y;

int\*\* height = new int\*[N];

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

height[i] = new int[M];

}

int max\_value = std::max(N - x, x - 1) + std::max(M - y, y - 1);

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

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

height[i][j] = max\_value - (std::abs(i - (x - 1)) + std::abs(j - (y - 1)));

}

}

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

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

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

}

std::cout << std::endl;

}

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

delete[] height[i];

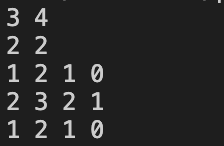
}

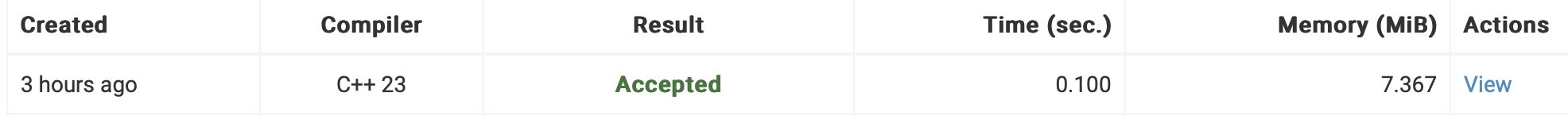
delete[] height;

return 0;

}

Результат:





Час виконання 1,5 год

Завдання №3 (Algotester Lab 7\_8)

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 <<

# 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.

#include <iostream>

#include <stdexcept>

template <typename T>

class DoublyLinkedList {

private:

struct Node {

T data;

Node\* prev;

Node\* next;

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

};

Node\* head;

Node\* tail;

int size;

Node\* getNode(int index) const {

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

throw std::out\_of\_range("Index out of bounds");

}

Node\* current = head;

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

current = current->next;

}

return current;

}

public:

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

~DoublyLinkedList() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

int getSize() const {

return size;

}

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

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

throw std::out\_of\_range("Index out of bounds");

}

Node\* prevNode = (index == 0) ? nullptr : getNode(index - 1);

Node\* nextNode = (index == size) ? nullptr : getNode(index);

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

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

if (prevNode) {

prevNode->next = newNode;

newNode->prev = prevNode;

} else {

head = newNode;

}

prevNode = newNode;

}

if (prevNode) {

prevNode->next = nextNode;

}

if (nextNode) {

nextNode->prev = prevNode;

} else {

tail = prevNode;

}

size += n;

}

void erase(int index, int n) {

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

throw std::out\_of\_range("Index out of bounds");

}

Node\* startNode = getNode(index);

Node\* endNode = (index + n < size) ? getNode(index + n) : nullptr;

Node\* prevNode = startNode->prev;

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

Node\* temp = startNode;

startNode = startNode->next;

delete temp;

}

if (prevNode) {

prevNode->next = endNode;

} else {

head = endNode;

}

if (endNode) {

endNode->prev = prevNode;

} else {

tail = prevNode;

}

size -= n;

}

T get(int index) const {

return getNode(index)->data;

}

void set(int index, T value) {

getNode(index)->data = value;

}

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

Node\* current = list.head;

while (current) {

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

current = current->next;

}

return os;

}

};

int main() {

int Q;

std::cin >> Q;

DoublyLinkedList<int> list;

while (Q--) {

std::string command;

std::cin >> command;

if (command == "insert") {

int index, n;

std::cin >> index >> n;

int\* values = new int[n];

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

std::cin >> values[i];

}

list.insert(index, n, values);

delete[] values;

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

int index, n;

std::cin >> index >> n;

list.erase(index, n);

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

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

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

int index;

std::cin >> index;

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

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

int index, value;

std::cin >> index >> value;

list.set(index, value);

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

std::cout << list << std::endl;

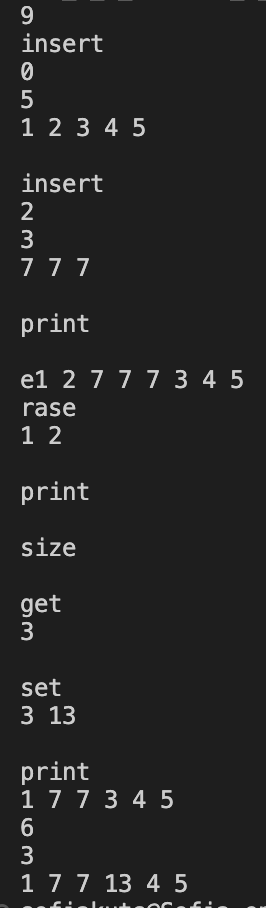
}

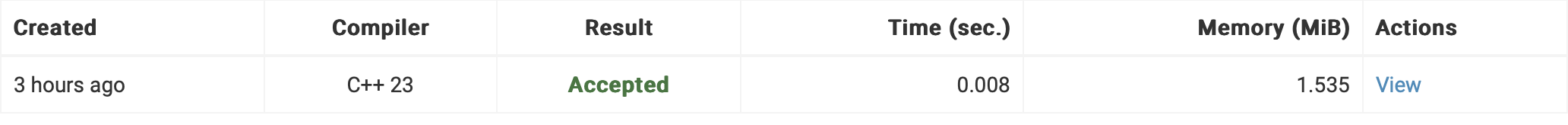
}

return 0;

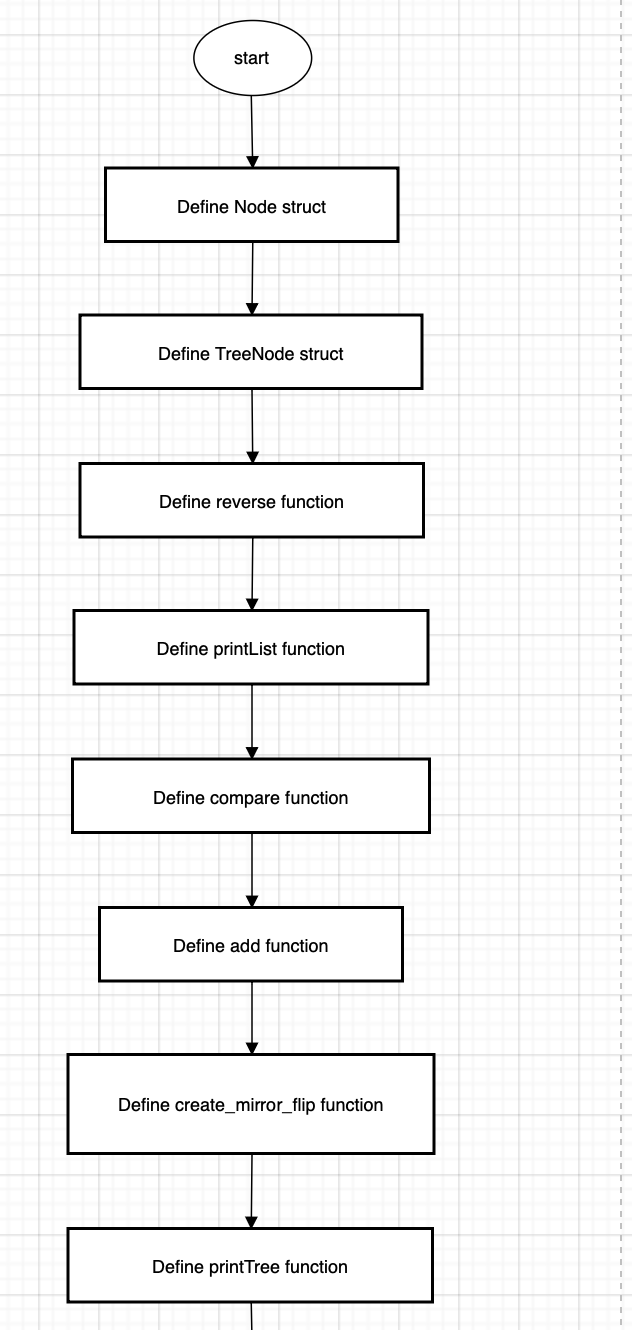
}

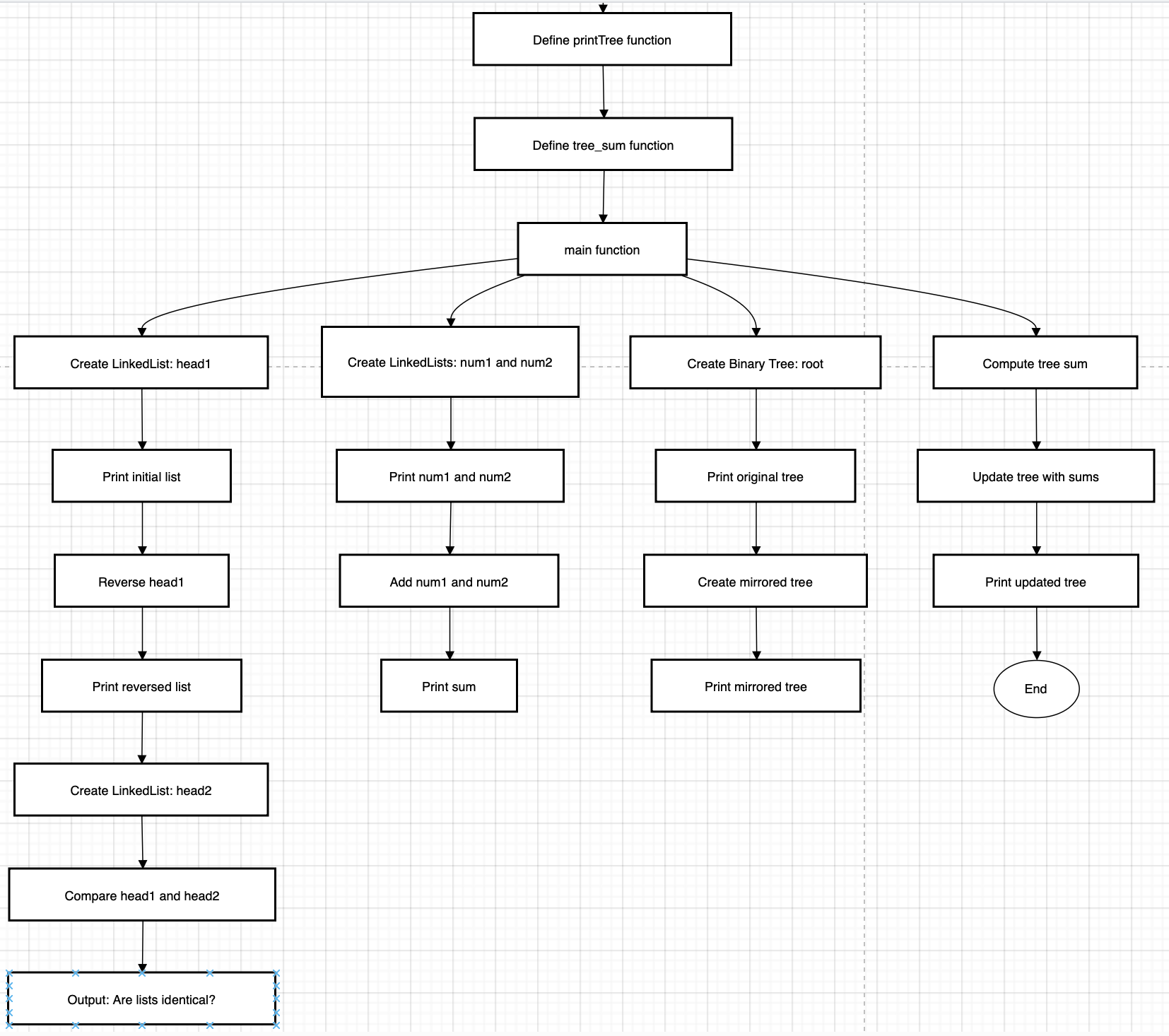
Результат :





Завдання №4 Class Practice Work )





#include <iostream>

using namespace std;

// Структура для зв'язаного списку

struct Node {

int data;

Node\* next;

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

};

// Структура для бінарного дерева

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

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

};

// завдання №1

Node\* reverse(Node\* head) {

Node\* prev = NULL;

Node\* current = head;

Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev;

}

void printList(Node\* head) {

while (head != NULL) {

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

head = head->next;

}

cout << endl;

}

// завдання №2

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

while (h1 != NULL && h2 != NULL) {

if (h1->data != h2->data) return false;

h1 = h1->next;

h2 = h2->next;

}

return h1 == NULL && h2 == NULL;

}

// завдання №3

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

Node\* result = NULL;

Node\* tail = NULL;

int carry = 0;

while (n1 != NULL || n2 != NULL || carry) {

int sum = carry;

if (n1 != NULL) {

sum += n1->data;

n1 = n1->next;

}

if (n2 != NULL) {

sum += n2->data;

n2 = n2->next;

}

Node\* newNode = new Node(sum % 10);

carry = sum / 10;

if (result == NULL) {

result = tail = newNode;

} else {

tail->next = newNode;

tail = newNode;

}

}

return result;

}

// завдання №4

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

if (!root) return NULL;

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) return;

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

printTree(root->left);

printTree(root->right);

}

// завдання №5

int tree\_sum(TreeNode\* root) {

if (!root) return 0;

if (!root->left && !root->right) return root->data;

int leftSum = tree\_sum(root->left);

int rightSum = tree\_sum(root->right);

root->data += leftSum + rightSum;

return root->data;

}

int main() {

Node\* head1 = new Node(1);

head1->next = new Node(2);

head1->next->next = new Node(3);

cout << "Вихідний список: ";

printList(head1);

head1 = reverse(head1);

cout << "Реверсований список: ";

printList(head1);

Node\* head2 = new Node(1);

head2->next = new Node(2);

head2->next->next = new Node(3);

cout << "Списки однакові? " << (compare(head1, head2) ? "Так" : "Ні") << endl;

Node\* num1 = new Node(9);

num1->next = new Node(7);

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

Node\* num2 = new Node(6);

num2->next = new Node(4);

num2->next->next = new Node(8);

cout << "Число 1: ";

printList(num1);

cout << "Число 2: ";

printList(num2);

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

cout << "Сума: ";

printList(sum);

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);

cout << "Оригінальне дерево: ";

printTree(root);

cout << endl;

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

cout << "Віддзеркалене дерево: ";

printTree(mirrored);

cout << endl;

tree\_sum(root);

cout << "Дерево з сумами: ";

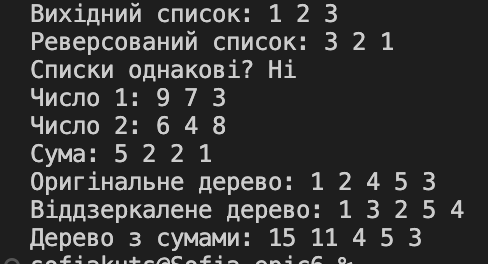
printTree(root);

cout << endl;

return 0;

}

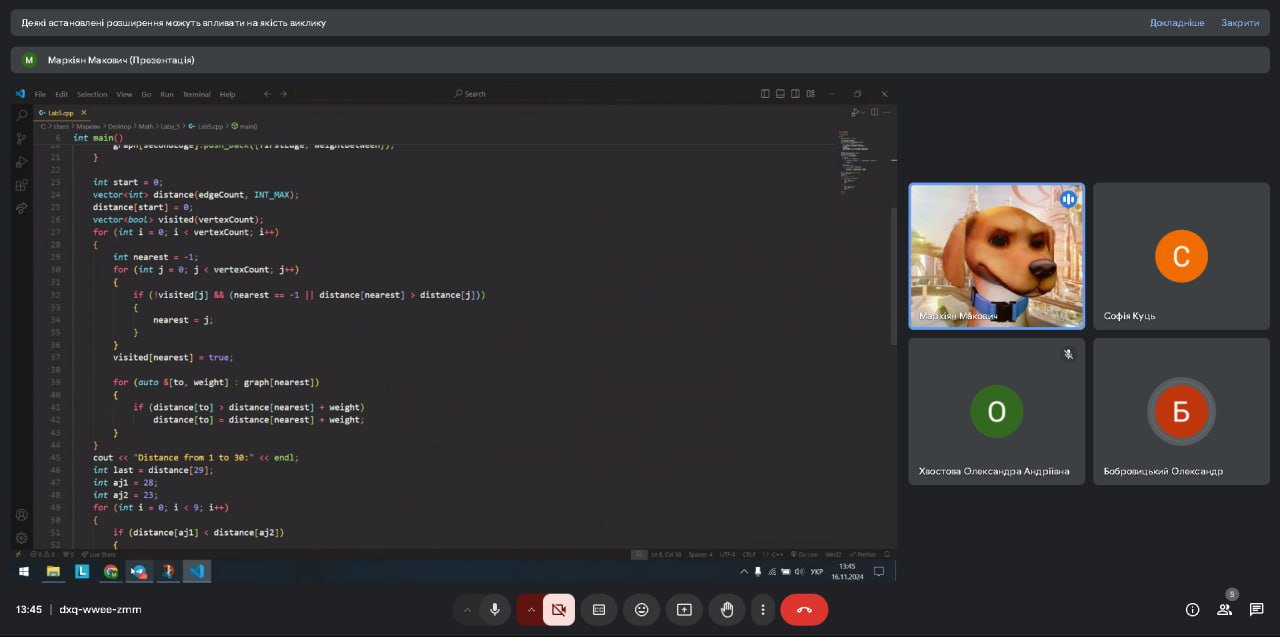
Результат:



Час виконання 3 год

Кооперація з командою

Скрін з онлайн зустрічі



Скрін дошки в Trello

