# Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №5 з дисципліни

«Основи програмування-2.

Модульне програмування»

«Дерева»

Варіант 29

Виконав студент ІП-11 Тарасьонок Дмитро Євгенович

(шифр, прізвище, ім'я, по батькові)

Перевірив Вітковська Ірина Іванівна

( прізвище, ім'я, по батькові)

Київ 2022

Мета: вивчити механізми організації і обробки дерев

Умова задачі: Побудувати дерево, елементами якого є символи. Визначити і вивести на друк усі термінальні вершини (листя) цього дерева

Постановка задачі: для виконання поставленої задачі нам треба створити клас Node, що буде реалізацією вершини дерева. Ця вершина має містити символ, а також вказівник на лівий та правий нащадки. Для заповнення дерева пропонується використовувати введення з клавіатури. Для визначення термінальних вершин дерева необхідно перевірити, чи має вершина нащадки, і, якщо так, виконати цю ж умову для нащадків, а інакше позначити цю вершину як термінальну й повернути символ, що вона містить.

Програма мовою C++:

main.cpp

#include "Node.h"

#include <locale>

#include <Windows.h>

int main()

{

locale::global(locale("rus"));

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Вводьте символи, що треба додати в дерево (Ctrl + D для зупинки): ";

Node root;

for (char c; cin >> c && c != 4;)

{

root.addNode(c);

}

cout << "Сформоване дерево:" << endl;

printTree(&root, nullptr, false);

cout << "Термінальні вершини (листя) дерева: ";

vector<char> leaves = root.getLeaves();

for (char c : leaves)

{

cout << c << ' ';

}

cout << endl;

return 0;

}

Node.h

#pragma once

#include <vector>

#include "Trunk.h"

#include <iostream>

using namespace std;

class Node

{

public:

Node(char data);

Node();

~Node();

char getData() const { return \*m\_data; }

void addNode(char n);

const Node\* left() const { return m\_left; }

const Node\* right() const { return m\_right; }

vector<char> getLeaves() const;

private:

char\* m\_data;

Node\* m\_left, \* m\_right;

};

void printTree(const Node\* root, Trunk\* prev, bool isRight);

Node.cpp

#include "Node.h"

void printTree(const Node\* root, Trunk\* prev, bool isRight)

{

if (root != nullptr) {

string prev\_str = " ";

Trunk\* trunk = new Trunk(prev, prev\_str);

printTree(root->right(), trunk, true);

if (!prev)

{

trunk->m\_str = "---";

}

else if (isRight)

{

trunk->m\_str = ".---";

prev\_str = " |";

}

else

{

trunk->m\_str = "`---";

prev->m\_str = prev\_str;

}

cout << \*trunk;

cout << " " << root->getData() << endl;

if (prev)

{

prev->m\_str = prev\_str;

}

trunk->m\_str = " |";

printTree(root->left(), trunk, false);

}

}

Node::Node(char data)

:m\_data{ new char },

m\_left{ nullptr },

m\_right{ nullptr }

{

\*m\_data = data;

}

Node::Node()

:m\_data{ nullptr },

m\_left{ nullptr },

m\_right{ nullptr }

{}

Node::~Node()

{

if (m\_data)

{

delete m\_data;

}

if (m\_left)

{

delete m\_left;

}

if (m\_right)

{

delete m\_right;

}

}

void Node::addNode(char n)

{

if (!m\_data)

{

m\_data = new char{ n };

}

else if (n < \*m\_data)

{

if (m\_left)

{

m\_left->addNode(n);

}

else

{

m\_left = new Node(n);

}

}

else

{

if (m\_right)

{

m\_right->addNode(n);

}

else

{

m\_right = new Node(n);

}

}

}

vector<char> Node::getLeaves() const

{

vector<char> leaves;

if (m\_left)

{

vector<char> leftLeaves = m\_left->getLeaves();

for (char c : leftLeaves)

{

leaves.push\_back(c);

}

}

if (m\_right)

{

vector<char> rightLeaves = m\_right->getLeaves();

for (char c : rightLeaves)

{

leaves.push\_back(c);

}

}

if (!m\_left && !m\_right)

{

leaves.push\_back(\*m\_data);

}

return leaves;

}

Trunk.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

struct Trunk

{

Trunk(Trunk\* prev, string str);

Trunk\* m\_prev;

string m\_str;

};

ostream& operator << (ostream& os, const Trunk& trunk);

Trunk.cpp

#include "Trunk.h"

Trunk::Trunk(Trunk\* prev, string str)

:m\_prev{ prev },

m\_str{ str }

{

}

ostream& operator << (ostream& os, const Trunk& trunk)

{

if (trunk.m\_prev)

{

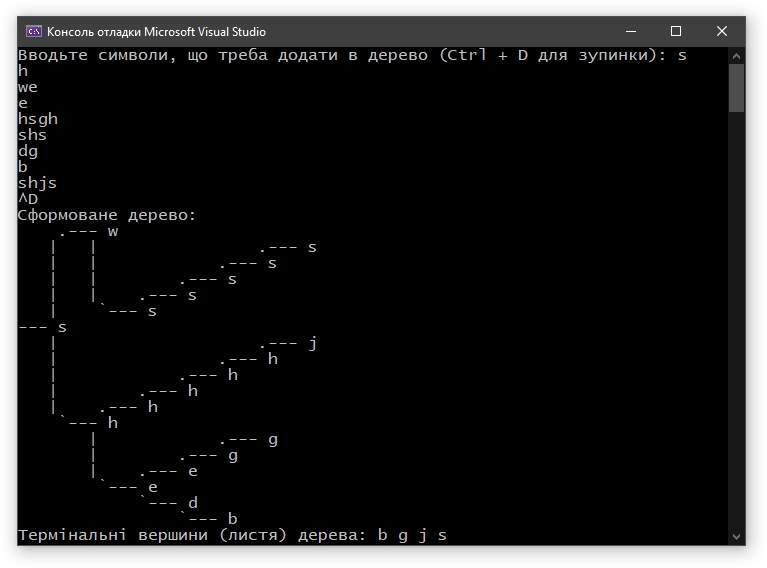
os << \*trunk.m\_prev;

}

return os << trunk.m\_str;

}

Виконання програми мовою C++:



Висновок: під час виконання даної лабораторної роботи я здобув практичних навичок із організації та обробки дерев, вивчив методи обходу дерев, використавши у своїй програмі симетричний обхід, а також навчився визначати термінальні вершини дерев рекурсивним методом.