МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**Лабораторная работа:**

"Бинарное дерево"

Выполнила: студентка гр.РИС-23-3Б

Мокроусова Ангелина Андреевна

Проверила: доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь 2024

1. Постановка задачи:

1) Сформировать идеально сбалансированное бинарное дерево, тип

информационного поля указан в варианте.

2) Распечатать полученное дерево.

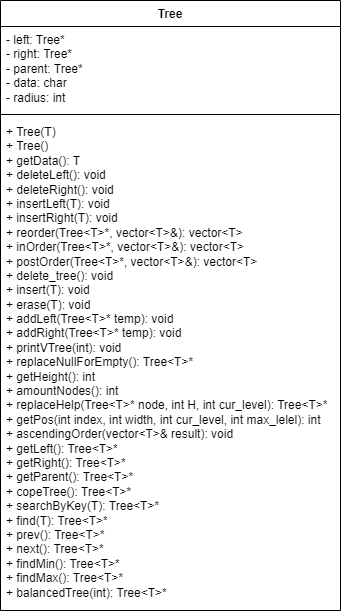
3) Выполнить обработку дерева в соответствии с заданием, вывести полученный

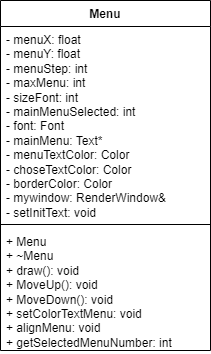
результат.

4) Преобразовать идеально сбалансированное дерево в дерево поиска.

5) Распечатать полученное дерево.

Тип информационного поля char. Найти высоту дерева.

1. Анализ задачи:
2. Класс Tree является шаблоном, что позволяет использовать его с различными типами данных. Шаблонный параметр T определяет тип данных, хранящихся в узлах дерева.   
   - Конструкторы и деструктор: Tree(), Tree(T), ~Tree().  
   - Получение данных: get\_data().  
   - Удаление ветвей: deleteLeft(), deleteRight().  
   - Вставка узлов: insertLeft(T), insertRight(T).  
   - Обходы дерева: reorder(Tree<T> \* , vector<T>&), inOrder(Tree<T> \* , vector<T>&), postOrder(Tree<T> \* , vector<T>&).  
   - Удаление дерева: delete\_tree().  
   - Вставка узла: insert(T).  
   - Полное удаление узла: erase(T).  
   - Установка ветвей: addLeft(Tree<T> \* ), addRight(Tree<T> \* ).  
   - Печать дерева: printVTree(int).  
   - Преобразование дерева в идеальную симметрию: replaceNULLforEmpty().  
   - Определение высоты дерева: get\_height().  
   - Определение количества узлов: amountNodes().  
   - Преобразование дерева из неполного в полное: replaceHelp(Tree<T> \* , int, int).  
   - Определение позиции узла: getPos(int, int, int, int).  
   - Обход дерева в порядке возрастания: ascendingOrder(vector<T>&).  
   - Создание сбалансированного дерева: createBalancedTree(Tree<T> \* ), buildBalanced(const vector<T>&, int, int, Tree<T> \* ).  
   - Получение указателей на левую, правую ветви и родителя: getLeft(), getRight(), getParent().  
   - Копирование дерева: copeTree().  
   - Поиск элемента по ключу: searchByKey(T).  
   - Нахождение узла с указанным значением: find(T).  
   - Получение указателя на предыдущий и следующий узлы: prev(), next().  
   - Нахождение минимального и максимального значения: findMin(), findMax().  
   - Создание сбалансированного дерева заданной высоты: balancedTree(int).
3. Класс Menu представляет собой структуру для создания и управления меню в графическом интерфейсе. Он содержит следующие атрибуты:  
   - menuX и menuY: координаты меню по осям X и Y соответственно.  
   - menuStep: расстояние между пунктами меню.  
   - maxMenu: максимальное количество пунктов меню.  
   - sizeFont: размер шрифта для пунктов меню.  
   - mainMenuSelected: номер текущего выбранного пункта меню.  
   - font: шрифт, используемый для пунктов меню.  
   - mainMenu: динамический массив текстовых объектов, представляющих названия пунктов меню.  
   - menuTextColor, choseTextColor, borderColor: цвета текста, выбранного текста и обводки пунктов меню соответственно.  
   - mywindow: ссылка на графическое окно, в котором будет отображаться меню.  
   Класс предоставляет следующие методы:  
   - setInitText: метод для настройки текста пунктов меню.  
   - Menu: конструктор класса, который инициализирует все атрибуты и создает пункты меню.  
   - ~Menu: деструктор, который освобождает память, выделенную для текстовых объектов пунктов меню.  
   - draw: метод для рисовки меню на экране.  
   - MoveUp и MoveDown: методы для перемещения выбора меню вверх или вниз соответственно.  
   - setColorTextMenu: метод для установки цветов элементов меню.  
   - alignMenu: метод для выравнивания положения меню.  
   - getSelectedMenuNumber: метод для получения номера выбранного элемента меню.
4. Функция drawTreeVertically отвечает за отрисовку вертикального представления дерева. Она создает окно с помощью библиотеки SFML, загружает шрифт и устанавливает фоновый цвет окна. Затем она создает окружности и тексты для каждого узла дерева и рисует линии, соединяющие их. Окно обновляется и отображается на экране. Если пользователь закрывает окно, функция завершает свою работу.
5. Функция drawTreeHorizontally отвечает за отрисовку горизонтального представления дерева. Она создает окно с помощью библиотеки SFML, загружает шрифт и устанавливает фоновый цвет окна. Затем она создает окружности и тексты для каждого узла дерева и рисует линии, соединяющие их. Окно обновляется и отображается на экране. Если пользователь закрывает окно, функция завершает свою работу.
6. Функция FindElementInTree проверяет наличие заданного символа в дереве. Если символ присутствует в дереве или совпадает с глобальной переменной global::key, которая содержит новые добавленные элементы, функция возвращает true.
7. Функция addNode позволяет пользователю ввести элемент, к которому он хочет добавить новый узел. После ввода искомого элемента, функция проверяет его наличие в дереве с помощью FindElementInTree. Если элемент найден, пользователь вводит данные для нового узла, после чего вызывается функция drawNodeAdd, которая добавляет узел и отрисовывает новое дерево.
8. Функция findPosition используется для определения позиции искомого элемента в дереве. Она возвращает индекс, соответствующий символу, введенному пользователем.
9. Функция deleteNode позволяет пользователю удалить элемент из дерева. Сначала пользователь вводит элемент, который хочет удалить, затем функция FindElementInTree проверяет его наличие в дереве. Если элемент найден, функция drawNodeDel вызывается для удаления узла и отрисовки дерева без этого узла.
10. Функция rounds выполняет обходы дерева и отображает результаты этих обходов. Она использует три вспомогательные функции modifyStringStraight, modifyStringBack, и modifyStringSymmetrically для вычисления прямого, обратного и симметричного обходов соответственно. Эти функции модифицируют строку, представляющую дерево, в соответствии с выбранным обходом.
11. Функция findElementForKey ищет элемент в дереве по ключу и отображает его в окне, если элемент найден. Она использует функцию FindElementInTree для проверки наличия элемента в дереве. Если элемент найден, создается новое окно, в котором отображается элемент в виде текста с окружностью.
12. Функция createSearchTree создает окно с помощью библиотеки SFML и отображает в нем меню с функциями для работы с деревом поиска. В меню представлены три пункта: поиск по ключу, получение высоты дерева и выход из меню. При выборе пункта меню вызывается соответствующая функция. Функция createSearchTree также обрабатывает события клавиатуры, такие как нажатие клавиш вверх, вниз и Enter, что позволяет пользователю перемещаться по меню и выбирать пункты.
13. Функция InitText используется для инициализации объекта Text с заданными параметрами. Она устанавливает размер шрифта, позицию, строку, цвет заливки, толщину и цвет обводки. Эта функция может быть полезна для создания текстовых элементов с определенными стилями, например, заголовков или пунктов меню.
14. Объявление глобальных переменных в пространстве имен global позволяет использовать эти переменные в различных частях программы без необходимости объявлять их заново. Переменные sought, key, soughtDel, и soughtForDel используются для хранения информации о поиске и добавлении элементов в дерево.
15. Функция drawNodeAdd отвечает за отрисовку дерева после добавления нового узла. Она создает окно с помощью библиотеки SFML, загружает шрифт и устанавливает фоновый цвет окна. Затем она рисует окружности и тексты для каждого узла дерева и линии, соединяющие их. Если флаг flag установлен в true и global::sought не равен пробелу, функция добавляет новый узел в дерево, рисуя его окружность и текст.
16. Функция drawNodeDel отвечает за отрисовку дерева после удаления узла. Она создает окно с помощью библиотеки SFML, загружает шрифт и устанавливает фоновый цвет окна. Затем функция рисует окружности и тексты для каждого узла дерева и линии, соединяющие их. Если узел для удаления найден, функция пропускает его и рисует остальные узлы. Если узел для удаления не найден, функция рисует все узлы без изменений.
17. Функции drawNodeAddHorizontal и drawNodeDelHorizontal аналогичны функциям drawNodeAdd и drawNodeDel соответственно, но для горизонтальной печати.
18. Функция GetHeight отвечает за отображение высоты дерева в отдельном окне. Она загружает шрифт, создает окно и рисует текст с информацией о высоте дерева.
19. Функция modifyStringStraight модифицирует строку, представляющую дерево, в соответствии с прямым обходом. Она добавляет новый элемент в строку в зависимости от позиции искомого элемента.
20. Функция modifyStringSymmetrically выполняет симметричный обход строки, добавляя новый элемент в соответствующие позиции.
21. Функция modifyStringBack выполняет обратный обход строки, добавляя новый элемент в начало или конец каждого подстроки, соответствующей искомому элементу.
22. Функция main создает окно с помощью библиотеки SFML, загружает фоновое изображение и шрифт для текста. Затем она создает меню с различными пунктами и обрабатывает события клавиатуры для перемещения по меню и выбора пунктов. В зависимости от выбранного пункта меню вызывается соответствующая функция. Когда пользователь выбирает пункт "Выход", окно закрывается. Функция main также получает текущий размер экрана и использует его для создания фона, который покрывает весь экран. Это позволяет отобразить фоновое изображение на весь экран.
23. UML Диаграмма:



1. Код программы:

## Файл Tree.h:

#pragma once

#include <SFML/Graphics.hpp>

#include <iostream>

#include "WorkButton.h"

#include <vector>

#include <cmath>

#include <math.h>

using namespace std;

using namespace sf;

template <class T>

class Tree {

private:

Tree\* left;

Tree\* right;

Tree\* parent;

char data;

int radius;

public:

Tree(T); // конструктор с данными

Tree(); // конструктор по умолчанию

~Tree(); // деструктор

T getData(); // получение данных

void deleteLeft(); // удаление левой ветки

void deleteRight(); // удаление правой ветки

void insertLeft(T); // вставка нового узла в левую ветвь с указанным значением

void insertRight(T); // вставка нового узла в правую ветвь с указанным значением

vector<T> reorder(Tree<T>\*, vector<T>&); // прямой обход дерева

vector<T> inOrder(Tree<T>\*, vector<T>&); // симметричный обход дерева

vector<T> postOrder(Tree<T>\*, vector<T>&); // обратный обход дерева

void delete\_tree() { delete this; }; // удаление дерева

void insert(T); // вставка узла в нужное место

void erase(T); // полное удаление узла с указанным значением

void addLeft(Tree<T>\* temp) { left = temp; }; // установка левой ветви

void addRight(Tree<T>\* temp) { right = temp; }; // установка правой ветви

void printVTree(int);// вертикальная печать

Tree<T>\* replaceNullForEmpty(); // достраивание дерева до идеально симметрии

int getHeight(); // высота дерева

int amountNodes(); // количество узлов в дереве

Tree<T>\* replaceHelp(Tree<T>\* node, int H, int cur\_level); // преобразование дерева из неполного в полное

int getPos(int index, int width, int cur\_level, int max\_lelel); // количество пробелов для данного узла

void ascendingOrder(vector<T>& result); // обход дерева в порядке возрастания и сохранение значений в векторе

Tree<T>\* getLeft(); // указатель на левую ветвь

Tree<T>\* getRight(); // указатель на левую ветвь

Tree<T>\* getParent(); // указатель на родителя

Tree<T>\* copeTree(); // копирование текущего дерева

Tree<T>\* searchByKey(T); // поиск элемента по ключу

Tree<T>\* find(T); // нахождение узла с указанным значением

Tree<T>\* prev(); // указатель на предыдущий узел

Tree<T>\* next(); // указатель на следующий узел

Tree<T>\* findMin(); // минимальное значение из дерева

Tree<T>\* findMax(); // максимальное значение из дерева

Tree<T>\* balancedTree(int); // сбалансированное дерево заданной высоты

};

template <class T>

Tree<T>::Tree(T data) { // конструктор со значением

this->data = data;

left = right = parent = nullptr;

}

template <class T>

Tree<T>::Tree() { // конструктор по умолчанию

left = right = parent = nullptr;

}

template <class T>

Tree<T>::~Tree() { // деструктор

deleteLeft();

deleteRight();

}

template <class T>

T Tree<T>::getData() {

return data;

}

template <class T>

Tree<T>\* Tree<T>::getLeft() {

return left;

}

template <class T>

Tree<T>\* Tree<T>::getRight() {

return right;

}

template <class T>

Tree<T>\* Tree<T>::getParent() {

return parent;

}

template <class T>

Tree<T>\* Tree<T>::next() { // указатель на следующий узел

Tree<T>\* current = this; // указатель на текущий узел

if (current->right != nullptr) { // если правая ветвь существует

return current->right->findMin();

}

Tree<T>\* tmp = current->parent;

while (tmp != nullptr && current == tmp->right) {

current = tmp;

tmp = tmp->parent;

}

return tmp;

}

template <class T>

Tree<T>\* Tree<T>::prev() { // указатель на предыдущмй узел

if (left != nullptr) { // если левая ветвь существует

Tree<T>\* tmp = left; // указатель на левую ветвь

while (tmp->right != nullptr) { // пока правая ветвь существует

tmp = tmp->right;

}

return tmp;

}

else {

Tree<T>\* tmp = this; // указатель на текущий узел

while (tmp->parent != nullptr && tmp->parent->left == tmp) {

tmp = tmp->parent;

}

return tmp->parent;

}

}

template <class T>

void Tree<T>::insert(T data) {

Tree<T>\* current = this; // текущий узел

while (current != nullptr) { // пока узел существует

if (data > current->data) { // если данные больше данных в текущем узле

if (current->right != nullptr) {

current = current->right;

}

else { // если правой ветви нет

current->insertRight(data);

return;

}

}

else if (data < current->data) { // если данные меньше данных в текущем узле

if (current->left != nullptr) {

current = current->left;

}

else {

current->insertRight(data);

return;

}

}

}

}

template <class T>

void Tree<T>::insertRight(T data) { // вставка нового узла в правую ветвь с указанным значением

Tree<T>\* new\_node = new Tree(data); // новой дерево с данными

if (this->right != nullptr) {

this->right->parent = new\_node;

new\_node->right = this->right; // правой ветви присваивается новое поддерево

}

this->right = new\_node;

new\_node->parent = this;

}

template <class T>

void Tree<T>::insertLeft(T data) { // вставка нового узла в правую ветвь с указанным значением

Tree<T>\* new\_node = new Tree<T>(data); // новое дерево с данными

if (this->left != nullptr) {

this->left->parent = new\_node;

new\_node->left = this->left; // правой ветви присваивается новое поддерево

}

this->left = new\_node;

new\_node->parent = this;

}

template <class T>

vector<T> Tree<T>::reorder(Tree<T>\* current, vector<T>& vect) { // прямой обход

if (current == nullptr) {

return vect;

}

else {

vect.push\_back(current->getData());

reorder(current->getLeft(), vect);

reorder(current->getRight(), vect);

}

return vect;

}

template <class T>

vector<T> Tree<T>::inOrder(Tree<T>\* tree, vector<T>& vect) { // симметричный обход

if (tree != nullptr) {

inOrder(tree->left, vect);

vect.push\_back(tree->getData());

inOrder(tree->right, vect);

}

return vect;

}

template <class T>

vector<T> Tree<T>::postOrder(Tree<T>\* tree, vector<T>& vect) { // обратный обход

if (tree != nullptr) {

postOrder(tree->left, vect);

postOrder(tree->right, vect);

vect.push\_back(tree->getData());

}

return vect;

}

template <class T>

int Tree<T>::getHeight() { // высота дерева

int h1 = 0, h2 = 0;

if (this == NULL) { // если дерево пустое

return 0;

}

if (this->left != NULL) { // если левая ветвь не пустая

h1 = this->left->getHeight();

}

if (this->right != NULL) { // если левая ветвь не пустая

h2 = this->right->getHeight();

}

if (h1 >= h2) {

return h1 + 1;

}

return h2 + 1;

}

template <class T>

int Tree<T>::amountNodes() { // количество узлов в дереве

if (this == NULL) {

return 0;

}

if ((this->left == NULL) && (this->right == NULL)) {

return 1;

}

int left\_branch = 0, right\_branch = 0;

if (this->left != NULL) { // если левая ветвь существует

left\_branch = this->left->amountNodes();

}

if (this->right != NULL) { // если правая ветвь существует

right\_branch = this->right->amountNodes();

}

return (left\_branch + right\_branch + 1);

}

template <class T>

Tree<T>\* Tree<T>::replaceNullForEmpty() { // преобразование дерева из неполного в полное

Tree<T>\* node = this->copeTree();

int H = node->getHeight(); // вычисление высоты дерева

node = replaceHelp(node, H, 0); // дополнение дерева пустыми узлами

return node;

}

template <class T>

Tree<T>\* Tree<T>::replaceHelp(Tree<T>\* node, int H, int cur\_level) {

if ((node->getLeft() == NULL) && (cur\_level != H - 1)) { // если ветвь пустая и высота не конечная

node->insertLeft(NULL);

}

if ((node->getRight() == NULL) && (cur\_level != H - 1)) {

node->insertRight(NULL);

}

if (node->getLeft() != NULL) { // если левая ветвь существует

node->addLeft(replaceHelp(node->getLeft(), H, cur\_level + 1));

}

if (node->getRight() != NULL) { // если правая ветвь существует

node->addRight(replaceHelp(node->getRight(), H, cur\_level + 1));

}

return node;

}

template <class T>

int Tree<T>::getPos(int index, int width, int cur\_level, int max\_level) {

int x1 = 0, x2 = pow(2, cur\_level) - 1;

int y1 = width / pow(2, cur\_level + 1), y2 = width - pow(2, max\_level - cur\_level);

if (x1 == x2) {

return y1;

}

double k = (y1 - y2) / (x1 - x2);

double m = -x1 \* k + y1;

int y = (int)(k \* index + m);

return y;

}

template <class T>

Tree<T>\* Tree<T>::balancedTree(int count) { // создание сбалансированного дерева

if (count <= 0) {

return nullptr;

}

T data;

cout << "Введите данные для сбалансированного дерева: ";

cin >> data;

Tree<T>\* tmp = new Tree<T>(data);

tmp->addLeft(balancedTree(count / 2));

tmp->addRight(balancedTree(count - count / 2 - 1));

return tmp;

}

template <class T>

void Tree<T>::erase(T data) { // удаление узла по данным

Tree<T>\* to\_erase = this->find(data);

Tree<T>\* to\_parent = to\_erase->parent;

if (to\_erase->left == nullptr && to\_erase->right == nullptr) {

if (to\_parent->left == to\_erase) {

to\_parent->left = nullptr;

}

else {

to\_parent->right = nullptr;

}

delete to\_erase;

}

else if ((to\_erase->left != nullptr && to\_erase->right == nullptr) || (to\_erase->left == nullptr && to\_erase->right

!= nullptr)) {

if (to\_erase->left == nullptr) {

if (to\_erase == to\_parent->left) {

to\_parent->left = to\_erase->right; // устанавливаем новую левую ветвь

}

else {

to\_parent->right = to\_erase->right; // устанавливаем новую правую ветвь

}

to\_parent->right->parent = to\_parent; // устанавливаем нового родителя для правой ветви

}

else {

if (to\_erase == to\_parent->left) {

to\_parent->left = to\_erase->left; // устанавливаем новую левую ветвь

}

else {

to\_parent->right = to\_erase->left; // устанавливаем новую правую ветвь

}

to\_parent->left->parent = to\_parent; // устанавливаем нового родителя для левой ветви

}

}

else {

Tree<T>\* next = to\_erase->next(); // получение следующего узела

to\_erase->data = next->data;

if (next == next->parent->left) {

next->parent->left = next->right;

if (next->right != nullptr) {

next->right->parent = next->parent;

}

}

else {

next->parent->right = next->right;

if (next->right != nullptr) {

next->right->parent = next->parent; // устанавливаем нового родителя для правой ветви

}

}

delete next;

}

}

template <class T>

void Tree<T>::deleteLeft() { // удаление левого узла

if (left != NULL) {

left->deleteLeft();

left->deleteRight();

delete left;

}

}

template <class T>

void Tree<T>::deleteRight() { // удаление правого узла

if (right != NULL) {

right->deleteLeft();

right->deleteRight();

delete right;

}

}

template <class T>

Tree<T>\* Tree<T>::searchByKey(T key) { // поиск по ключу

if (data == key) {

return this;

}

if (left != nullptr) {

Tree<T>\* result = left->searchByKey(key);

if (result != nullptr) {

return result; // возвращение поддерева

}

}

if (right != nullptr) {

Tree<T>\* result = right->searchByKey(key);

if (result != nullptr) {

return result;// возвращение поддерева

}

}

return nullptr;// возвращение поддерева

}

template <class T>

Tree<T>\* Tree<T>::find(T data) { // поиск узла по данным

if (this == nullptr || this->data == data) {

return this;

}

else if (data > this->data) {

return this->right->find(data);

}

else {

return this->left->find(data);

}

}

template <class T>

Tree<T>\* Tree<T>::findMax() { // поиск максимального элемента

if (this->right == nullptr) { // если больше нет правых узлов

return this;

}

return this->right->findMax();

}

template <class T>

Tree<T>\* Tree<T>::findMin() { // поиск минимального элемента

Tree<T>\* min\_node = this;

while (min\_node->left != nullptr) { // если больше нет левых узлов

min\_node = min\_node->left;

}

return min\_node;

}

template <class T>

void Tree<T>::ascendingOrder(vector<T>& result) { // обход дерева в порядке возрастания и сохранение значений в векторе

if (left != nullptr) {

left->ascendingOrder(result);

}

result.push\_back(data);

if (right != nullptr) {

right->ascendingOrder(result);

}

}

template <class T>

Tree<T>\* Tree<T>::copeTree() { // копия дерева

Tree<T>\* new\_tree = new Tree<T>(data);

if (left != nullptr) {

new\_tree->left = left->copeTree();

new\_tree->left->parent = new\_tree;

}

if (right != nullptr) {

new\_tree->right = right->copeTree();

new\_tree->right->parent = new\_tree;

}

return new\_tree;

}

struct pos {

int column;

int row;

};

void printSpaces(int start, int end) {

for (int j = start; j < end; j++) { // пробелы до узла

cout << " ";

}

}

template <class T>

void Tree<T>::printVTree(int k) {

int height = this->getHeight();

int max\_leafs = pow(2, height - 1);

int width = 2 \* max\_leafs - 1; // минимальная ширина дерева

int cur\_level = 0;

int index = 0;

int fact\_spaces = getPos(index, width, cur\_level, height - 1); // число пробелов перед корнем

pos node; // определение структуры для хранения позиции узла

vector<Tree<T>\*> vect\_1;

vector<pos> vect\_2;

Tree<T>\* tree\_2 = this->copeTree();

tree\_2 = tree\_2->replaceNullForEmpty(); // достраивание дерева до симметрии

Tree<T>\* tree\_3 = tree\_2;

vect\_1.push\_back(tree\_3);

node.column = fact\_spaces; // число пробелов перед корнем

node.row = cur\_level;

vect\_2.push\_back(node);

for (int i = 0; i < tree\_2->amountNodes(); i++) { // перебор всех узлов дерева

if (pow(2, cur\_level) <= index + 1) {

index = 0;

cur\_level++;

}

if (vect\_1.at(i)->left != NULL) { // проверка наличия левого потомка

vect\_1.push\_back(vect\_1.at(i)->left);

fact\_spaces = getPos(index, width, cur\_level, height - 1); // вычисление количества пробелов перед новым корнем

node.column = fact\_spaces; // заполнение позиции нового узла

node.row = cur\_level;

vect\_2.push\_back(node);

index++;

}

if (vect\_1.at(i)->right != NULL) { // проверка наличия правого потомка

vect\_1.push\_back(vect\_1.at(i)->right); // добавление потомка в вектор

fact\_spaces = getPos(index, width, cur\_level, height - 1); // вычисление количества пробелов перед новым узлом

node.column = fact\_spaces; // заполнение позиции нового узла

node.row = cur\_level;

vect\_2.push\_back(node);

index++;

}

}

for (int i = vect\_1.size() - 1; i >= 0; i--) {

if (i != 0) {

if (vect\_2.at(i - 1).row == vect\_2.at(i).row) {

vect\_2.at(i).column = vect\_2.at(i).column - vect\_2.at(i - 1).column - 1;

}

}

}

int flag = 0;

for (int i = 0; i < vect\_1.size(); i++) {

node = vect\_2.at(i);

cur\_level = node.row;

if (flag < cur\_level) {

flag = cur\_level;

cout << endl; // переход на новую строку

}

fact\_spaces = node.column; // число пробелов перед узлом

int real\_spaces = k \* fact\_spaces;

printSpaces(0, real\_spaces);

if (vect\_1.at(i)->getData() == NULL) {

cout << " ";

}

else {

cout << vect\_1.at(i)->getData(); // вывод узла

}

printSpaces(0, k);

}

cout << endl;

}

void printInConsol() { // функция печати дерева вертикально в консоль

Tree<char>\* tree = new Tree<char>('k');

tree->insertLeft('u');

tree->getLeft()->insertLeft('l');

tree->getLeft()->insertRight('c');

tree->insertRight('!');

tree->getRight()->insertRight('!');

tree->getRight()->insertLeft('y');

cout << "Вертикальная печать" << endl;

tree->printVTree(2);

}

## Файл Menu.h:

#pragma once

#include <SFML/Graphics.hpp>

using namespace std;

using namespace sf;

namespace game {

class MyMenu

{

float menu\_X; // Координаты меню по X

float menu\_Y; // Координаты меню по Y

int menu\_Step; // Расстояние между пунктами меню

int max\_menu; // Максимальное количество пунктов меню

int size\_font; // Размер шрифта

int mainMenuSelected; // Номер текущего пункта меню

Font font; // Шрифт меню

vector<Text> mainMenu; // Динамический массив названий меню

Color menu\_text\_color = Color::White; // Цвет меню

Color chose\_text\_color = Color::Yellow; // Цвет выбора меню

Color border\_color = Color::Black; // Цвет обводки текста

void setInitText(Text& text, const String& str, float xpos, float ypos) const; // Настройка текста меню

RenderWindow& mywindow; // Ссылка на графическое окно

public:

MyMenu(RenderWindow& window, float menux, float menuy, int sizeFont, int step, vector<String>& name);

void draw(); // Рисуем меню

void MoveUp(); // Перемещение выбора меню вверх

void MoveDown(); // Перемещение выбора меню вниз

void setColorTextMenu(Color menColor, Color ChoColor, Color BordColor); // Цвет элементов игрового меню

void AlignMenu(int posx); // Выравнивание положения меню (по левому по правому по центру)

int getSelectedMenuNumber() const{ return mainMenuSelected;} // Возвращает номер выбраного элемента

};

}

## Файл Menu.cpp:

#pragma once

#include <SFML/Graphics.hpp>

using namespace sf;

class Menu {

float menuX; // координаты меню по x

float menuY; // координаты меню по y

int menuStep; // расстояние между пунктами меню

int maxMenu; // максимальное количество пунктов меню

int sizeFont; // размер шрифта

int mainMenuSelected; // номер текущего пункта меню

sf::Font font; // шрифт меню

sf::Text\* mainMenu; // динамический массив текстовых объектов названий пунктов меню

sf::Color menuTextColor = sf::Color::White; // цвет пунктов меню

sf::Color choseTextColor = sf::Color::Yellow; // цвет выбора пункта меню

sf::Color borderColor = sf::Color::Black; // цвет обводки текста пунктов меню

sf::RenderWindow& mywindow; // ссылка на графическое окно

void setInitText(sf::Text& text, sf::String str, float xpos, float ypos); // настройка текста пунктов меню

public:

Menu(sf::RenderWindow& window, float menux, float menuy,

int index, sf::String name[], int sizeFont = 60, int step = 80); // конструктор

~Menu() { // деструктор

delete[] mainMenu;

}

void draw(); // рисовка меню

void MoveUp(); // перемещение выбора меню вверх

void MoveDown(); // перемещение выбора меню вниз

void setColorTextMenu(sf::Color menColor, sf::Color ChoColor,

sf::Color BordColor); // цвет элементов игрового меню

void alignMenu(int posx); // выравнивание положения меню

int getSelectedMenuNumber() { // возвращает номер выбранного элемента меню

return mainMenuSelected;

}

};

## Файл WorkButton.h:

#pragma once

#include <SFML/Graphics.hpp>

#include<SFML/Audio.hpp>

#include "Tree.h"

#include <SFML/System.hpp>

#include <SFML/Window.hpp>

#include <iostream>

#include "finishing.h"

#include <string>

#include <sstream>

using namespace std;

using namespace sf;

void drawTreeVertically() {

if (global::soughtForDel != ' ') { drawNodeDel(true); return; } // печать дерева если были удалены элементы

if (global::key != ' ') { cout << global::soughtDel; drawNodeAdd(true); return; } // печать дерева если были добавлены элементы

sf::Color backgroundСolor(600, 600, 600);

sf::RenderWindow window(sf::VideoMode(1280, 720), "Vertical printing"); // создание окна

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) { // установка шрифта

exit(32);

}

char masData[7] = { 'l', 'u', 'c', 'k', 'y', '!', '!' };

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) { // событие для закрытия окна

window.close();

}

}

window.clear(backgroundСolor); // установка фона

sf::Vector2f positions[] = { // позиции для каждой окружности

sf::Vector2f(100.f, 250.f),

sf::Vector2f(200.f, 150.f),

sf::Vector2f(280.f, 250.f),

sf::Vector2f(350.f, 50.f),

sf::Vector2f(400.f, 250.f),

sf::Vector2f(500.f, 150.f),

sf::Vector2f(600.f, 250.f)

};

for (int i = 0; i < 7; ++i) {

sf::CircleShape circle(50.f); // создание окружности

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(150,150,150));

char data = masData[i];

sf::Text text(data, font, 24); // устновка данных и их параметров

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

sf::Vector2f positionsS[] = { // позиции начала линий, соединяющих окружности

sf::Vector2f(150.f, 250.f),

sf::Vector2f(295.f, 220.f),

sf::Vector2f(280.f, 160.f),

sf::Vector2f(450.f, 117.f),

sf::Vector2f(597.f, 220.f),

sf::Vector2f(507.f, 220.f),

};

sf::Vector2f positionsS2[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(205.f, 220.f),

sf::Vector2f(325.f, 250.f),

sf::Vector2f(350.f, 113.f),

sf::Vector2f(510.f, 170.f),

sf::Vector2f(652.f, 250.f),

sf::Vector2f(450.f, 250.f),

};

for (size\_t i = 0; i < 6; ++i) { // добавление линий между окружностями

sf::Vertex line[] = {

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i])

};

window.draw(line, 2, sf::Lines);

}

window.display(); // отображение на экране рисунков

}

}

void drawTreeHorizontally() {

if (global::soughtForDel != ' ') { drawNodeDelHorizontal(true); return; } // печать дерева если были удалены элементы

if (global::key != ' ') { cout << global::key; drawNodeAddHorizontal(true); return; } // печать дерева если были добавлены элементы

sf::Color backgroundСolor(600,600,600);

sf::RenderWindow window(sf::VideoMode(1280, 720), "Horizontal printing"); // создание окна

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) { // загрузка шрифта

exit(32);

}

char masData[7] = { 'l', 'u', 'c', 'k', 'y', '!', '!' };

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) { // событие для закрытия окна

window.close();

}

}

window.clear(backgroundСolor); // установка фона

sf::Vector2f positions[] = { // определение позиций для каждой окружности

sf::Vector2f(310.f, 550.f),

sf::Vector2f(190.f, 460.f),

sf::Vector2f(310.f, 380.f),

sf::Vector2f(70.f, 330.f),

sf::Vector2f(310.f, 240.f),

sf::Vector2f(190.f, 190.f),

sf::Vector2f(310.f, 100.f)

};

for (int i = 0; i < 7; ++i) { // рисовка окружности и текста

sf::CircleShape circle(50.f); // создание окружности

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(150, 150, 150));

char data = masData[i];

sf::Text text(data, font, 24); // установление текста и его параметров

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

sf::Vector2f positionsS[] = { // позиции начала линий для соединения окружностей

sf::Vector2f(310.f, 590.f),

sf::Vector2f(200.f, 478.f),

sf::Vector2f(310.f, 430.f),

sf::Vector2f(145.f, 335.f),

sf::Vector2f(280.f, 270.f),

sf::Vector2f(270.f, 200.f),

};

sf::Vector2f positionsS2[] = { // позиции конца линий для соединения окружностей

sf::Vector2f(270.f, 550.f),

sf::Vector2f(145.f, 420.f),

sf::Vector2f(279.f, 477.f),

sf::Vector2f(200.f, 270.f),

sf::Vector2f(310.f, 296.f),

sf::Vector2f(310.f, 160.f),

};

for (size\_t i = 0; i < 6; ++i) { // добавление линий между окружностями

sf::Vertex line[] =

{

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i])

};

window.draw(line, 2, sf::Lines);

}

window.display(); // отображение рисунков на экране

}

}

bool FindElementInTree(char sought) { // функция для проверки наличия данных в дереве

switch (sought) {

case 'l': return true; break;

case 'u': return true; break;

case 'c': return true; break;

case 'k': return true; break;

case 'y': return true; break;

case '!': return true; break;

default:

break;

}

if (global::soughtForDel != global::key)

if (sought == global::key) return true; // проверка новых добавленных элементов

return false;

}

bool FindElementInTreeForDel(char sought) { // функция для проверки наличия данных в дереве

switch (sought) {

case 'l': return true; break;

case 'u': return true; break;

case 'c': return true; break;

case 'k': return true; break;

case 'y': return true; break;

case '!': return true; break;

default:

break;

}

if (sought == global::key) return true; // проверка новых добавленных элементов

return false;

}

void addNode() { // функция дабвления узла в дерево

cout << "Введите элемент, к которому нужно добавить новый узел: " << endl;

cin >> global::sought;

if (FindElementInTree(global::sought) == true) { // проверка наличия искомого узла

cout << "Введите данные для нового узла: " << endl;

cin >> global::key;

Font font;

drawNodeAdd(true); // добавление узла и отрисовка нового дерева

}

else {

cout << "Такой элемент не найден" << endl;

return;

}

}

int findPosition(char sought) { // найти позицию элемента в дереве

switch (sought) {

case 'l': return 0; break;

case 'u': return 1; break;

case 'c': return 2; break;

case 'k': return 3; break;

case 'y': return 4; break;

case '!': return 5; break;

default:

break;

}

return - 1;

}

void deleteNode() { // функция удаления элемента из дерева

cout << "Введите элемент, который хотите удалить: " << endl;

cin >> global::soughtForDel;

if (FindElementInTreeForDel(global::soughtForDel) == true) { // проверка существует ли в дереве такой элемент

global::soughtDel = findPosition(global::soughtForDel);

drawNodeDel(true); // удаление и отрисовка дерева без заданного узла

}

else {

cout << "Такой элемент не найден" << endl;

return;

}

}

void rounds() { // обходы дерева

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) { // установка шрифта

exit(32);

}

sf::Color backgroundСolor(600,600,600);

sf::RenderWindow window(sf::VideoMode(800, 600), "Tree traversals");

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) { // событие закрытия окна

if (event.type == sf::Event::Closed) {

window.close();

}

}

window.clear(backgroundСolor); // установление фона

sf::Text text1;

text1.setFont(font);

text1.setString(L"Прямой обход:");

text1.setFillColor(sf::Color::Black);

text1.setCharacterSize(50);

text1.setPosition(30, 10);

window.draw(text1);

string message = "kulc!y!";

message = modifyStringStraight(message); // вычисление прямого обхода с учетом всех удалений/добавлений

String mes(message);

text1.setString(mes);

text1.setPosition(30, 80);

window.draw(text1);

text1.setString(L"Обратный обход: ");

text1.setPosition(30, 150);

window.draw(text1);

message = "lcuy!!k";

message = modifyStringBack(message); // вычисление обратного обхода с учетом всех удалений/добавлений

mes = message;

text1.setString(mes);

text1.setPosition(30, 220);

window.draw(text1);

text1.setString(L"Симметричный обход: ");

text1.setPosition(30, 290);

window.draw(text1);

message = "lucky!!";

message = modifyStringSymmetrically(message); // вычисление симметричного обхода с учетом всех удалений/добавлений

mes = message;

text1.setString(mes);

text1.setPosition(30, 360);

window.draw(text1);

window.display();

}

}

void findElementForKey() { // функция поиска элемента по ключу

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) { // загрузка шрифта

exit(32);

}

sf::Color backgroundСolor(600, 600, 600);

char Key;

cout << "Введите элемент, который хотите найти: " << endl;

cin >> Key;

bool flag = FindElementInTree(Key); // проверка наличия элемента в дереве

if (flag == false) {

cout << "Такой элемент не найден!" << endl;

return;

}

else {

sf::RenderWindow window(sf::VideoMode(1280, 720), "Item found"); // создание окна

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) { // событие закрытия окна

if (event.type == sf::Event::Closed) {

window.close();

}

}

window.clear(backgroundСolor);

sf::CircleShape circle(100.f); // создание окружности

circle.setPosition(310, 200);

circle.setFillColor(sf::Color(150, 150, 150));

sf::Text text(Key, font, 60); // создание текста и его параметров

text.setPosition(390, 260);

text.setFillColor(sf::Color::Black);

window.draw(circle); // отрисовка окружности

window.draw(text); // отрисовка текста

window.display(); // отображение

}

}

}

void createSearchTree(){ // функция для создания отедльного меню с функциями дерева поиска

system("chcp 1251 > Null");

RenderWindow window; // создание окна

window.create(VideoMode(1280, 720), L"Search Tree", sf::Style::Close); // параметры окна

float width = VideoMode::getDesktopMode().width; // получение текущиего размера экрана

float height = VideoMode::getDesktopMode().height; // получение текущиего размера экрана

RectangleShape background(Vector2f(width, height)); // создаем прямоугольник для изображения

Texture texture\_window;

if (!texture\_window.loadFromFile("fon.jpg")) return ; // загрузка в прямоугольник текстуру с изображением

background.setTexture(&texture\_window);

Font font;

if (!font.loadFromFile("ArialRegular.ttf")) exit(32); // установка шрифта

Text Titul;

Titul.setFont(font);

InitText(Titul, 460, 50, L"Бинарное дерево", 70, sf::Color(15,15,15), 0); // параметры для заголовка

String nameMenu[]{ L"Поиск по ключу",L"Получить высоту дерева", L"Назад" };

Menu mymenu(window, 670, 250, 3, nameMenu, 60, 120); // объект меню

mymenu.setColorTextMenu(sf::Color(150,150,150), Color(600,600,600), Color::Black); // установка цвета элементов пунктов меню

mymenu.alignMenu(2); // выравнивание по центру пунктов меню

while (window.isOpen()) {

Event event;

while (window.pollEvent(event)) {

if (event.type == Event::KeyReleased) {

if (event.type == sf::Event::Closed) {

window.close();

}

if (event.key.code == Keyboard::Up) { mymenu.MoveUp(); } // событие нажатия на клавиатуре стрелки вверх

if (event.key.code == Keyboard::Down) { mymenu.MoveDown(); } // событие нажатия на клавиатуре стрелки вниз

if (event.key.code == Keyboard::Return){ // событие нажатия на клавиатуре клавиши Enter

switch (mymenu.getSelectedMenuNumber()) {

case 0: findElementForKey(); break;

case 1: GetHeight(); break;

case 2: window.close();

}

}

}

}

window.draw(background);

window.draw(Titul);

mymenu.draw();

window.display();

}

}

## Файл finishing.h:

#pragma once

#include <SFML/Graphics.hpp>

#include "Tree.h"

#include <SFML/System.hpp>

#include <SFML/Window.hpp>

#include "WorkButton.h"

#include <iostream>

#include <sstream>

using namespace std;

using namespace sf;

namespace global { // объявление глобальных переменных

char sought = ' '; // искомый для добавления

char key = ' '; // новое значение

int soughtDel = -1; // искомый для удаления

char soughtForDel = ' '; // искомый для удаления

}

void InitText(Text& mtext, float xpos, float ypos, String str, int size\_font = 60,

Color menu\_text\_color = Color::White, int bord = 0, Color border\_color = Color::Black) { // установка настроек

mtext.setCharacterSize(size\_font);

mtext.setPosition(xpos, ypos);

mtext.setString(str);

mtext.setFillColor(menu\_text\_color);

mtext.setOutlineThickness(bord);

mtext.setOutlineColor(border\_color);

}

void drawNodeAdd(bool flag) { // отрисовка дерева с новым узлом

sf::RenderWindow window(sf::VideoMode(1280, 720), "Adding successfully");

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) { // установка шрифта

exit(32);

}

sf::Color backgroundСolor(600,600,600);

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) { // событие закрытия окна

if (event.type == sf::Event::Closed) {

window.close();

}

}

window.clear(backgroundСolor);

sf::Vector2f positions[] = { // определение позиций для каждой окружности

sf::Vector2f(100.f, 250.f),

sf::Vector2f(200.f, 150.f),

sf::Vector2f(280.f, 250.f),

sf::Vector2f(350.f, 50.f),

sf::Vector2f(400.f, 250.f),

sf::Vector2f(500.f, 150.f),

sf::Vector2f(600.f, 250.f)

};

char masData[7] = { 'l', 'u', 'c', 'k', 'y', '!', '!' };

for (int i = 0; i < 7; ++i) { // рисование окружностей и данных

sf::CircleShape circle(50.f);

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(150,150,150));

char data = masData[i]; // Замените на нужные данные

sf::Text text(data, font, 24);

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

sf::Vector2f positionsS[] = { // определение позиций для начала линий

sf::Vector2f(150.f, 250.f),

sf::Vector2f(295.f, 220.f),

sf::Vector2f(280.f, 160.f),

sf::Vector2f(450.f, 117.f),

sf::Vector2f(597.f, 220.f),

sf::Vector2f(507.f, 220.f),

};

sf::Vector2f positionsS2[] = { // определение позиций для конца линий

sf::Vector2f(205.f, 220.f),

sf::Vector2f(325.f, 250.f),

sf::Vector2f(350.f, 113.f),

sf::Vector2f(510.f, 170.f),

sf::Vector2f(652.f, 250.f),

sf::Vector2f(450.f, 250.f),

};

for (size\_t i = 0; i < 6; ++i) { // добавление линий между окружностями

sf::Vertex line[] = {

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i])

};

window.draw(line, 2, sf::Lines);

}

if (flag == true && global::sought != ' ') { // добавление нового узла

if (global::sought == 'l' || global::sought == 'u') { // если нужно добавлять к узлу l или u

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(10.f, 350.f));

circle.setFillColor(sf::Color(150,150,150));

char data = global::key;

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(10.f + 30, 350.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] = {

sf::Vertex(sf::Vector2f(55.f , 350.f)),

sf::Vertex(sf::Vector2f(102.f, 320.f))

};

window.draw(line, 2, sf::Lines);

}

else if (global::sought == 'c') { // если нужно добавлять к узлу c

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(350.f, 365.f));

circle.setFillColor(sf::Color(150,150,150));

char data = global::key;

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(350.f + 30, 365.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] = {

sf::Vertex(sf::Vector2f(400.f , 363.f)),

sf::Vertex(sf::Vector2f(370.f, 325.f))

};

window.draw(line, 2, sf::Lines);

}

else if (global::sought == 'k' || global::sought == 'y') { // если нужно добавлять к узлу k или y

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(350.f, 365.f));

circle.setFillColor(sf::Color(600,600,600));

char data = global::key; // Замените на нужные данные

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(350.f + 30, 365.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(400.f, 363.f)),

sf::Vertex(sf::Vector2f(420.f, 340.f))

};

window.draw(line, 2, sf::Lines);

}

else { // если нужно добавлять к узлу !

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(680.f, 365.f));

circle.setFillColor(sf::Color(150,150,150));

char data = global::key; // Замените на нужные данные

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(680.f + 30, 365.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] = {

sf::Vertex(sf::Vector2f(730.f, 365.f)),

sf::Vertex(sf::Vector2f(694.f, 323.f))

};

window.draw(line, 2, sf::Lines);

}

}

window.display();

}

}

void drawNodeDel(bool flag) { // функция добавления и отрисовки дерева после удаления узла

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) { // установка шрифта

exit(32);

}

sf::Color backgroundСolor(600,600,600);

sf::RenderWindow window(sf::VideoMode(1280, 720), "Removal successful"); // создание окна

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) { // событие для закрытия окна

window.close();

}

}

window.clear(backgroundСolor); // установка фона

sf::Vector2f positions[] = { // позиции для каждой окружности

sf::Vector2f(100.f, 400.f),

sf::Vector2f(200.f, 300.f),

sf::Vector2f(280.f, 400.f),

sf::Vector2f(350.f, 150.f),

sf::Vector2f(400.f, 400.f),

sf::Vector2f(500.f, 300.f),

sf::Vector2f(600.f, 400.f)

};

char masData[8] = { 'l', 'u', 'c', 'k', 'y', '!', '!', ' '};

int amount;

if (global::key != ' ') {

amount = 8;

masData[7] = global::key;

}

else amount = 7;

for (int i = 0; i < amount; i++) { // рисование дерева

if (i == global::soughtDel) { // если элемент для удаления найден

if (i != 7) {

for (int j = i + 1; j < amount; j++) { // пропуск заданного элемента

sf::CircleShape circle(50.f); // создание окружности

circle.setPosition(positions[j - 1]);

circle.setFillColor(sf::Color(150,150,150));

char data = masData[j];

sf::Text text(data, font, 24); // устновка данных и их параметров

text.setPosition(positions[j - 1].x + 30, positions[j - 1].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

break;

}

else break;

}

else {

sf::CircleShape circle(50.f); // создание окружности

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(150,150,150));

char data = masData[i];

sf::Text text(data, font, 24); // устновка данных и их параметров

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

}

sf::Vector2f positionsS[] = { // позиции начала линий, соединяющих окружности

sf::Vector2f(150.f, 400.f),

sf::Vector2f(290.f, 380.f),

sf::Vector2f(280.f, 310.f),

sf::Vector2f(440.f, 235.f),

sf::Vector2f(450.f, 400.f),

sf::Vector2f(590.f, 380.f),

};

sf::Vector2f positionsS2[] = { // позиции конца линий, соединяющих окружности

sf::Vector2f(210.f, 380.f),

sf::Vector2f(320.f, 400.f),

sf::Vector2f(360.f, 233.f),

sf::Vector2f(520.f, 310.f),

sf::Vector2f(510.f, 380.f),

sf::Vector2f(655.f, 400.f),

};

for (size\_t i = 0; i < amount-2; ++i) { // добавление линий между окружностями

sf::Vertex line[] = {

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i])

};

window.draw(line, 2, sf::Lines);

}

window.display(); // отображение на экране рисунков

}

}

void drawNodeAddHorizontal(bool flag) { // добавление узла и отрисовка дерева горизонтально

sf::Color backgroundСolor(600,600,600);

sf::RenderWindow window(sf::VideoMode(800, 800), "Adding successfully");

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) {

exit(32);

}

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

}

window.clear(backgroundСolor);

sf::Vector2f positions[] = {

sf::Vector2f(310.f, 550.f),

sf::Vector2f(190.f, 460.f),

sf::Vector2f(310.f, 380.f),

sf::Vector2f(70.f, 330.f),

sf::Vector2f(310.f, 240.f),

sf::Vector2f(190.f, 190.f),

sf::Vector2f(310.f, 100.f)

};

for (int i = 0; i < 7; ++i) {

sf::CircleShape circle(50.f);

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(150,150,150));

char masData[7] = { 'l', 'u', 'c', 'k', 'y', '!', '!' };

char data = masData[i]; // Замените на нужные данные

sf::Text text(data, font, 24);

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

sf::Vector2f positionsS[] = {

sf::Vector2f(310.f, 590.f),

sf::Vector2f(200.f, 478.f),

sf::Vector2f(310.f, 430.f),

sf::Vector2f(145.f, 335.f),

sf::Vector2f(280.f, 270.f),

sf::Vector2f(270.f, 200.f),

};

sf::Vector2f positionsS2[] = {

sf::Vector2f(270.f, 550.f),

sf::Vector2f(145.f, 420.f),

sf::Vector2f(279.f, 477.f),

sf::Vector2f(200.f, 270.f),

sf::Vector2f(310.f, 296.f),

sf::Vector2f(310.f, 160.f),

};

for (size\_t i = 0; i < 6; ++i) {

sf::Vertex line[] =

{

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i])

};

window.draw(line, 2, sf::Lines);

}

if (flag == true && global::sought != ' ') {

if (global::sought == 'l' || global::sought == 'u') {

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(440.f, 650.f));

circle.setFillColor(sf::Color(150, 150, 150));

char data = global::key;

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(440.f + 30, 650.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(440.f , 690.f)),

sf::Vertex(sf::Vector2f(385.f, 640.f))

};

window.draw(line, 2, sf::Lines);

}

else if (global::sought == 'c') {

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(440.f, 310.f));

circle.setFillColor(sf::Color(150,150,150));

char data = global::key;

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(440.f + 30, 310.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(440.f , 363.f)),

sf::Vertex(sf::Vector2f(380.f, 390.f))

};

window.draw(line, 2, sf::Lines);

}

else if (global::sought == 'k' || global::sought == 'y') {

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(440.f, 310.f));

circle.setFillColor(sf::Color(150,150,150));

char data = global::key;

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(440.f + 30, 310.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(440.f, 368.f)),

sf::Vertex(sf::Vector2f(386.f, 328.f))

};

window.draw(line, 2, sf::Lines);

}

else {

sf::CircleShape circle(50.f);

circle.setPosition(sf::Vector2f(440.f, 190.f));

circle.setFillColor(sf::Color(150,150,150));

char data = global::key;

sf::Text text(global::key, font, 24);

text.setPosition(sf::Vector2f(440.f + 30, 190.f + 30));

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(440.f, 230.f)),

sf::Vertex(sf::Vector2f(390.f, 185.f))

};

window.draw(line, 2, sf::Lines);

}

}

window.display();

}

}

void drawNodeDelHorizontal(bool flag) { // функция удаления и отрисовки дерева горизонтально

sf::Color backgroundСolor(600,600,600);

sf::RenderWindow window(sf::VideoMode(1280, 720), "Removal succcessful");

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) {

exit(32);

}

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

}

window.clear(backgroundСolor);

sf::Vector2f positions[] = {

sf::Vector2f(310.f, 550.f),

sf::Vector2f(190.f, 460.f),

sf::Vector2f(310.f, 380.f),

sf::Vector2f(70.f, 330.f),

sf::Vector2f(310.f, 240.f),

sf::Vector2f(190.f, 190.f),

sf::Vector2f(310.f, 100.f)

};

char masData[8] = { 'l', 'u', 'c', 'k', 'y', '!', '!', ' ' };

int amount;

if (global::key != ' ') {

amount = 8;

masData[7] = global::key;

}

else amount = 7;

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

if (i == global::soughtDel) {

for (int j = i + 1; j < amount; j++) {

sf::CircleShape circle(50.f);

circle.setPosition(positions[j - 1]);

circle.setFillColor(sf::Color(150,150,150));

char data = masData[j];

sf::Text text(data, font, 24);

text.setPosition(positions[j - 1].x + 30, positions[j - 1].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

break;

}

else {

sf::CircleShape circle(50.f);

circle.setPosition(positions[i]);

circle.setFillColor(sf::Color(150,150,150));

char data = masData[i];

sf::Text text(data, font, 24);

text.setPosition(positions[i].x + 30, positions[i].y + 30);

text.setFillColor(sf::Color::Black);

window.draw(circle);

window.draw(text);

}

}

sf::Vector2f positionsS[] = {

sf::Vector2f(310.f, 590.f),

sf::Vector2f(200.f, 478.f),

sf::Vector2f(310.f, 430.f),

sf::Vector2f(145.f, 335.f),

sf::Vector2f(280.f, 270.f),

sf::Vector2f(270.f, 200.f),

};

sf::Vector2f positionsS2[] = {

sf::Vector2f(270.f, 550.f),

sf::Vector2f(145.f, 420.f),

sf::Vector2f(279.f, 477.f),

sf::Vector2f(200.f, 270.f),

sf::Vector2f(310.f, 296.f),

sf::Vector2f(310.f, 160.f),

};

for (size\_t i = 0; i < amount-2; ++i) {

sf::Vertex line[] =

{

sf::Vertex(positionsS[i]),

sf::Vertex(positionsS2[i])

};

window.draw(line, 2, sf::Lines);

}

window.display();

}

}

int GetHeight() { // функция получения высоты дерева

char height = ' ';

if (global::key != ' ' && global::soughtForDel != ' ') height = '3';

else if (global::key != ' ')

height = '4';

else return '3';

sf::Font font;

if (!font.loadFromFile("ArialRegular.ttf")) { // загрузка шрифта

exit(32);

}

sf::Color backgroundСolor(600,600,600);

sf::RenderWindow window(sf::VideoMode(800, 600), "Tree Height");

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) { // событие закрытия окна

window.close();

}

}

window.clear(backgroundСolor); // установка фона

sf::Text text1;

text1.setFont(font);

text1.setString(L"Высота дерева:");

text1.setFillColor(sf::Color::Black);

text1.setCharacterSize(100);

text1.setPosition(30, 100);

window.draw(text1);

text1.setString(height);

text1.setPosition(360, 250);

window.draw(text1);

window.display();

}

}

string modifyStringStraight(std::string& original) { // функция, создающая итоговый ответ прямого обхода

string line = "";

if (global::sought == 'l' || global::sought == 'u')

original.insert(4, 1, global::key);

if (global::sought == 'c')

original.insert(3, 1, global::key);

if (global::sought == 'k' || global::sought == 'y')

original.insert(6, 1, global::key);

if (global::sought == '!')

original.insert(7, 1, global::key);

if (global::soughtForDel != ' ')

original.erase(std::remove(original.begin(), original.end(), global::soughtForDel), original.end());

return original;

}

string modifyStringSymmetrically(std::string& original) { // функция, создающая итоговый ответ симметричного обхода

string line = "";

if (global::sought == 'l' || global::sought == 'u')

original.insert(1, 1, global::key);

if (global::sought == 'c')

original.insert(3, 1, global::key);

if (global::sought == 'k' || global::sought == 'y')

original.insert(5, 1, global::key);

if (global::sought == '!')

original.insert(7, 1, global::key);

if (global::soughtForDel != ' ')

original.erase(std::remove(original.begin(), original.end(), global::soughtForDel), original.end());

return original;

}

string modifyStringBack(std::string& original) { // функция, создающая итоговый ответ обратного обхода

string line = "";

if (global::sought == 'l' || global::sought == 'u')

original.insert(0, 1, global::key);

if (global::sought == 'c')

original.insert(1, 1, global::key);

if (global::sought == 'k' || global::sought == 'y')

original.insert(3, 1, global::key);

if (global::sought == '!')

original.insert(4, 1, global::key);

if (global::soughtForDel != ' ')

original.erase(std::remove(original.begin(), original.end(), global::soughtForDel), original.end());

return original;

}

## Файл main.cpp:

#include "Menu.h"

#include "Tree.h"

#include "WorkButton.h"

#include <vector>

#include "finishing.h"

using namespace sf;

using namespace std;

int main() {

system("chcp 1251 > Null");

RenderWindow window; // создание окна windows

window.create(VideoMode(1280,720), L"Меню", sf::Style::Close); // параметры окна

float width = VideoMode::getDesktopMode().width; // получение текущего размера экрана

float height = VideoMode::getDesktopMode().height; // получение текущего размера экрана

// Загрузка иконки

Image icon;

icon.loadFromFile("Image/icon1.png");

window.setIcon(1200, 1084, icon.getPixelsPtr());

window.setMouseCursorVisible(false);// Отключение видимости курсора

Music music;// Вщзпроизведение музыкальной дороожки

music.openFromFile("music.ogg");

music.play();

music.setLoop(true);

music.setVolume(2.f);

// Создание движущегося фона

Texture textureFon;

textureFon.loadFromFile("Image/fon.jpg");

RectangleShape fon(Vector2f(1280, 720));

fon.setTexture(&textureFon);

RectangleShape fon2(Vector2f(1280, 720));

fon2.setTexture(&textureFon);

fon2.setPosition(Vector2f(1280, 0));

Vector2f pos;// Координаты объектов

Clock clock; // Таймер

float time;

RectangleShape background(Vector2f(width, height)); // создаем прямоугольник для изображения

Texture texture\_window;

if (!texture\_window.loadFromFile("fon.jpg")) return 4; // загрузка в прямоугольник текстуру с изображением

background.setTexture(&texture\_window);

Font font;

if (!font.loadFromFile("ArialRegular.ttf")) return 5; // установка шрифта для названия

Text Titul;

Titul.setFont(font);

InitText(Titul, 410, 50, L"БИНАРНОЕ ДЕРЕВО:", 70, Color(15, 15, 15), 0); // параметры для заголовка

String nameMenu[]{ L"Вертикальная печать дерева", L"Горизонтальная печать дерева", L"Обходы дерева", L"Добавить узел", L"Удалить узел", L"Сделать деревом поиска", L"Вертикальная печать исходного дерева в консоль", L"Выход"};

Menu mymenu(window, 670, 170, 8, nameMenu, 45, 90); // объект меню

mymenu.setColorTextMenu(sf::Color(150, 150, 150), Color(1, 1, 1), Color::Black); // установка цвета элементов пунктов меню

mymenu.alignMenu(2); // выравнивание по центру пунктов меню

while (window.isOpen()) {

Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) //Для закрытия через крестик

window.close();

if (event.type == Event::KeyReleased) {

if (event.key.code == Keyboard::Up) { mymenu.MoveUp(); } // событие нажатия на клавиатуре стрелки вверх

if (event.key.code == Keyboard::Down) { mymenu.MoveDown(); } // событие нажатия на клавиатуре стрелки вниз

if (event.key.code == Keyboard::Return) { // событие нажатия на клавиатуре клавиши Enter

switch (mymenu.getSelectedMenuNumber()) {

case 0: drawTreeVertically(); break;

case 1: drawTreeHorizontally(); break;

case 2: rounds(); break;

case 3: addNode(); break;

case 4: deleteNode(); break;

case 5: createSearchTree(); break;

case 6: printInConsol(); break;

case 7: window.close(); break;

}

}

}

}

// Обновление таймера

time = clock.getElapsedTime().asMicroseconds();

time = time / 15000;

clock.restart();

// Движение фона

fon.move(-0.2 \* time, 0);

pos = fon.getPosition();

if (pos.x < -1280) fon.setPosition(1280, pos.y);

fon2.move(-0.2 \* time, 0);

pos = fon2.getPosition();

if (pos.x < -1280) fon2.setPosition(1280, pos.y);

window.draw(fon);// Отрисовка элементов игры

window.draw(fon2);

window.draw(background);

window.draw(Titul);

mymenu.draw();

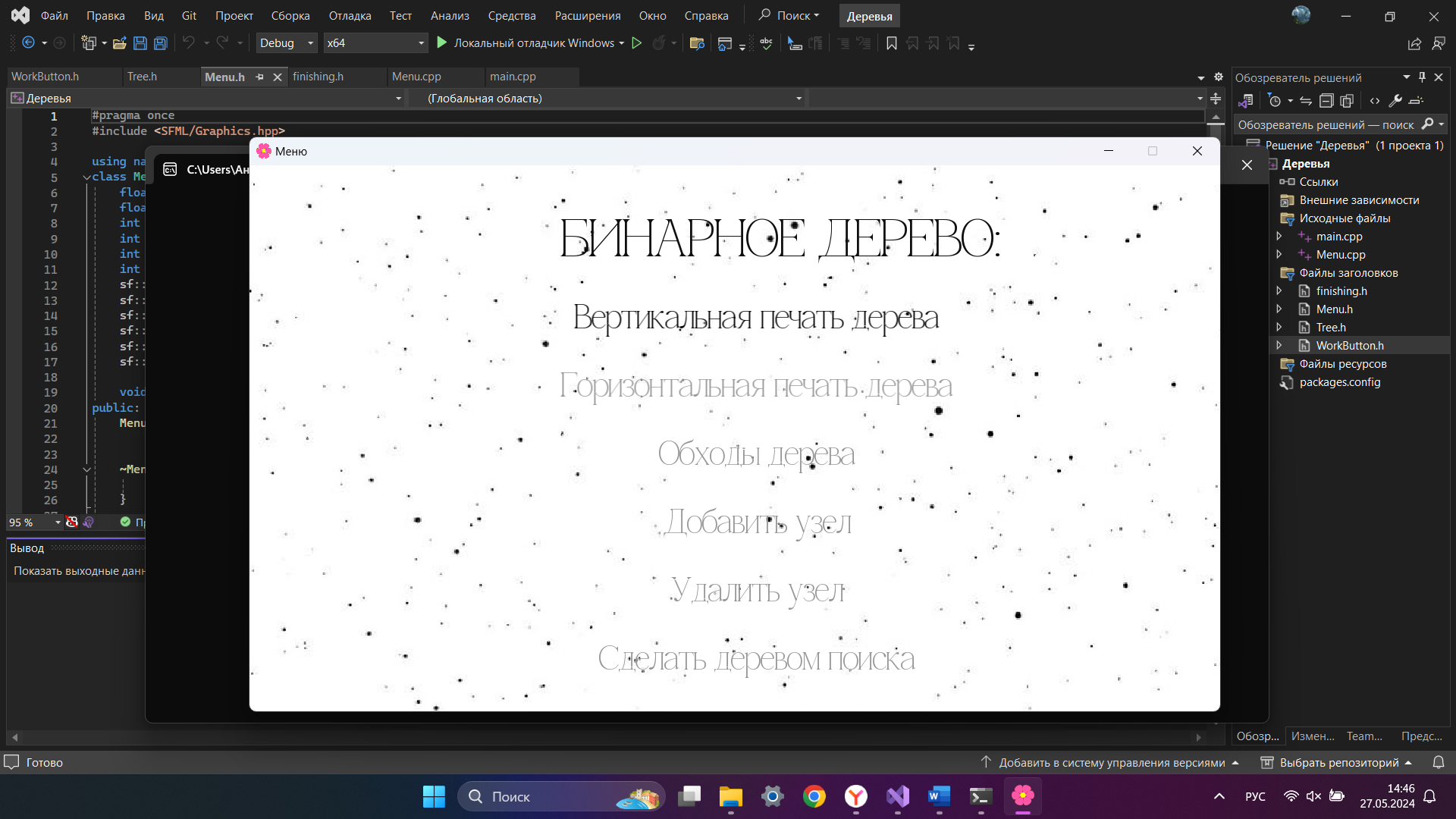
window.display();

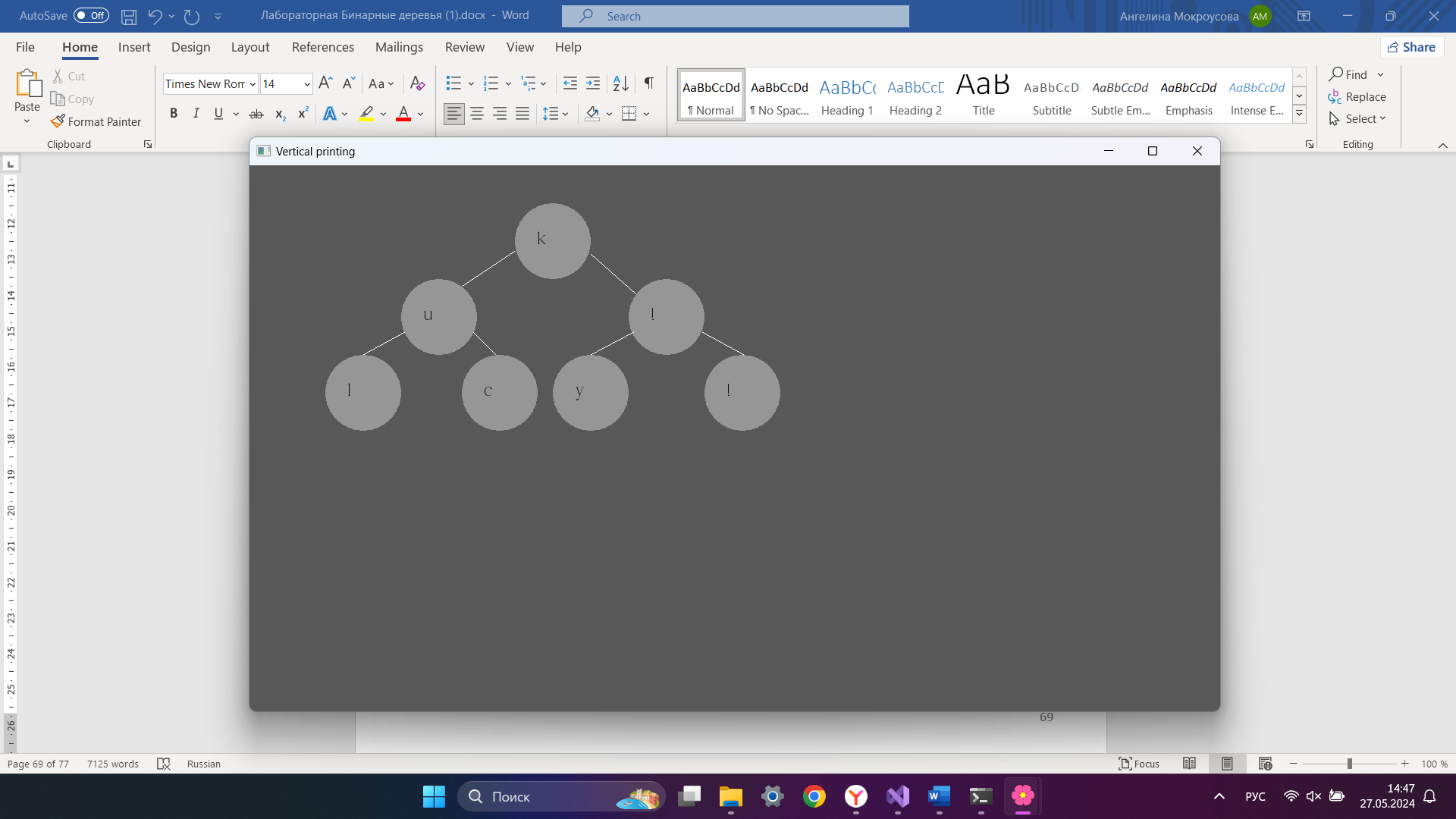
}

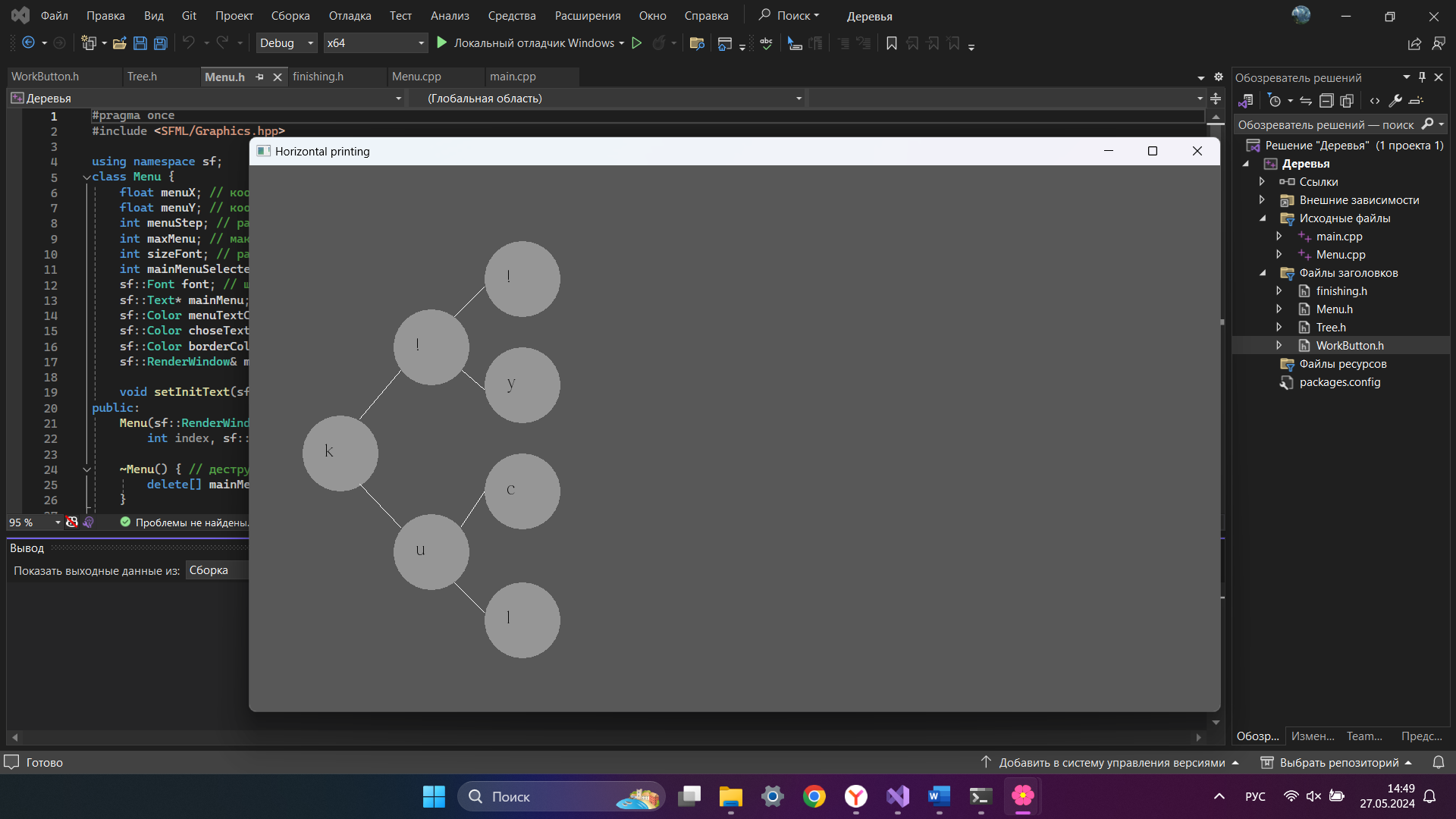
return 0;

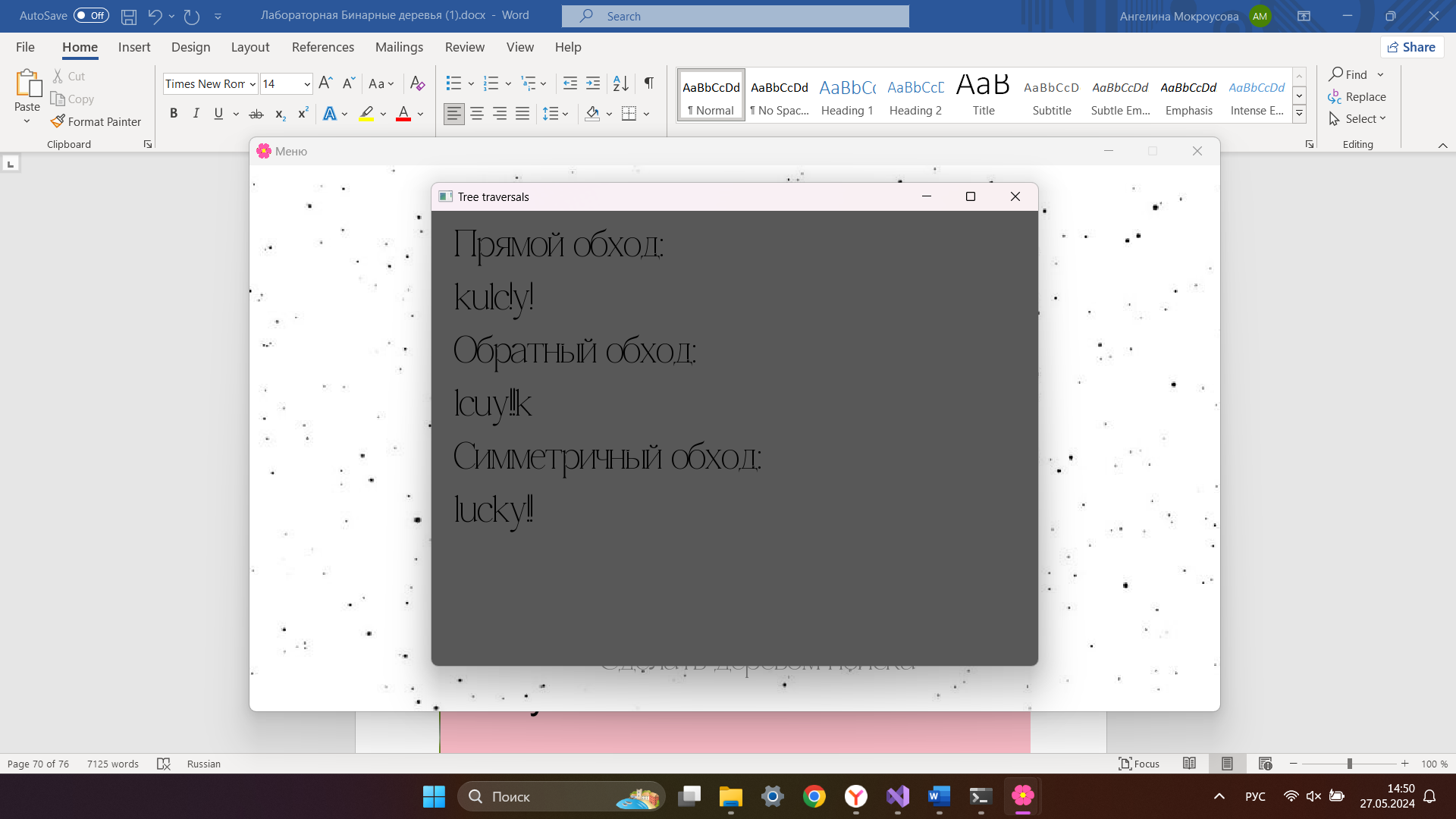
}

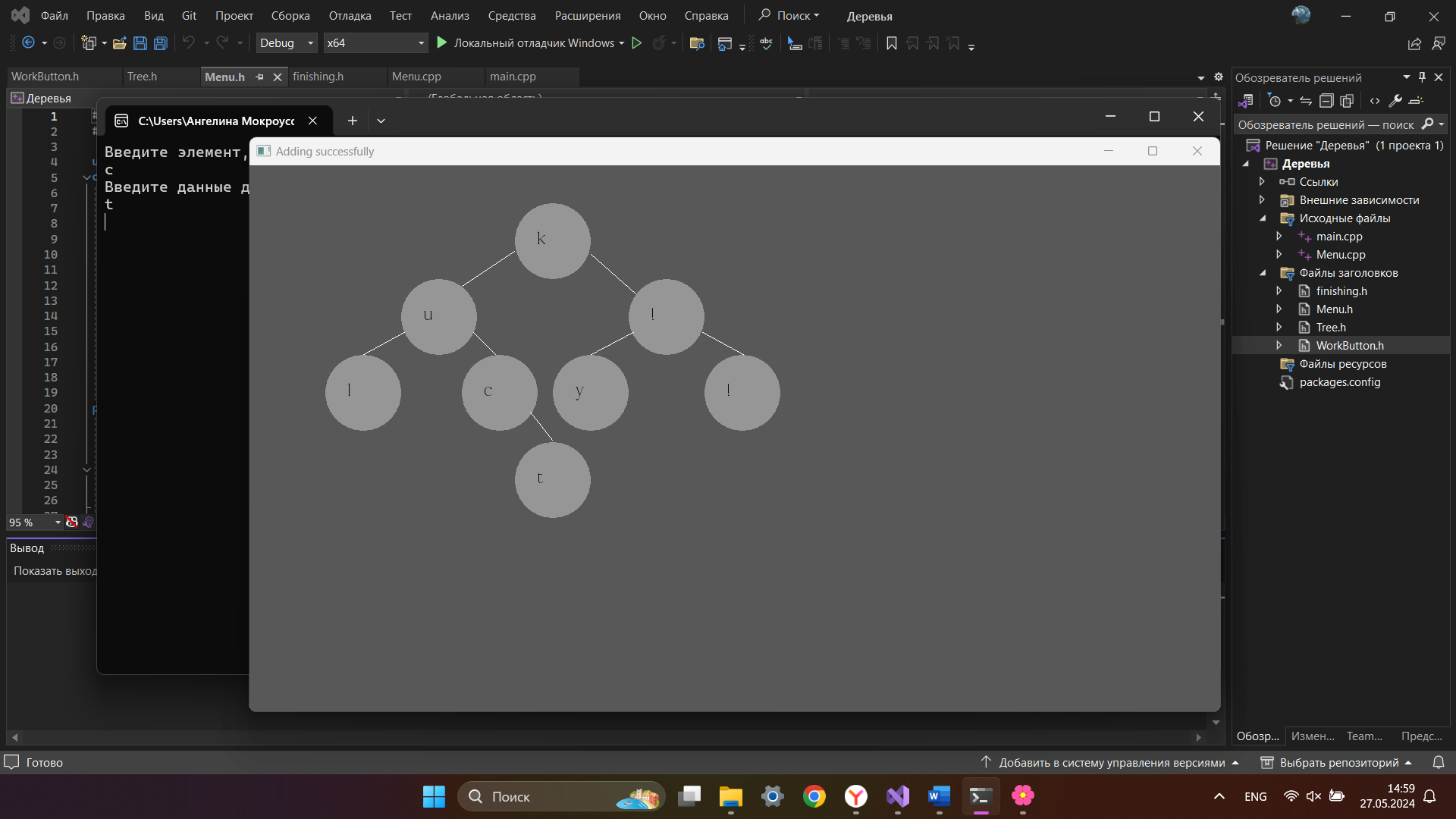
## Результат работы программы:

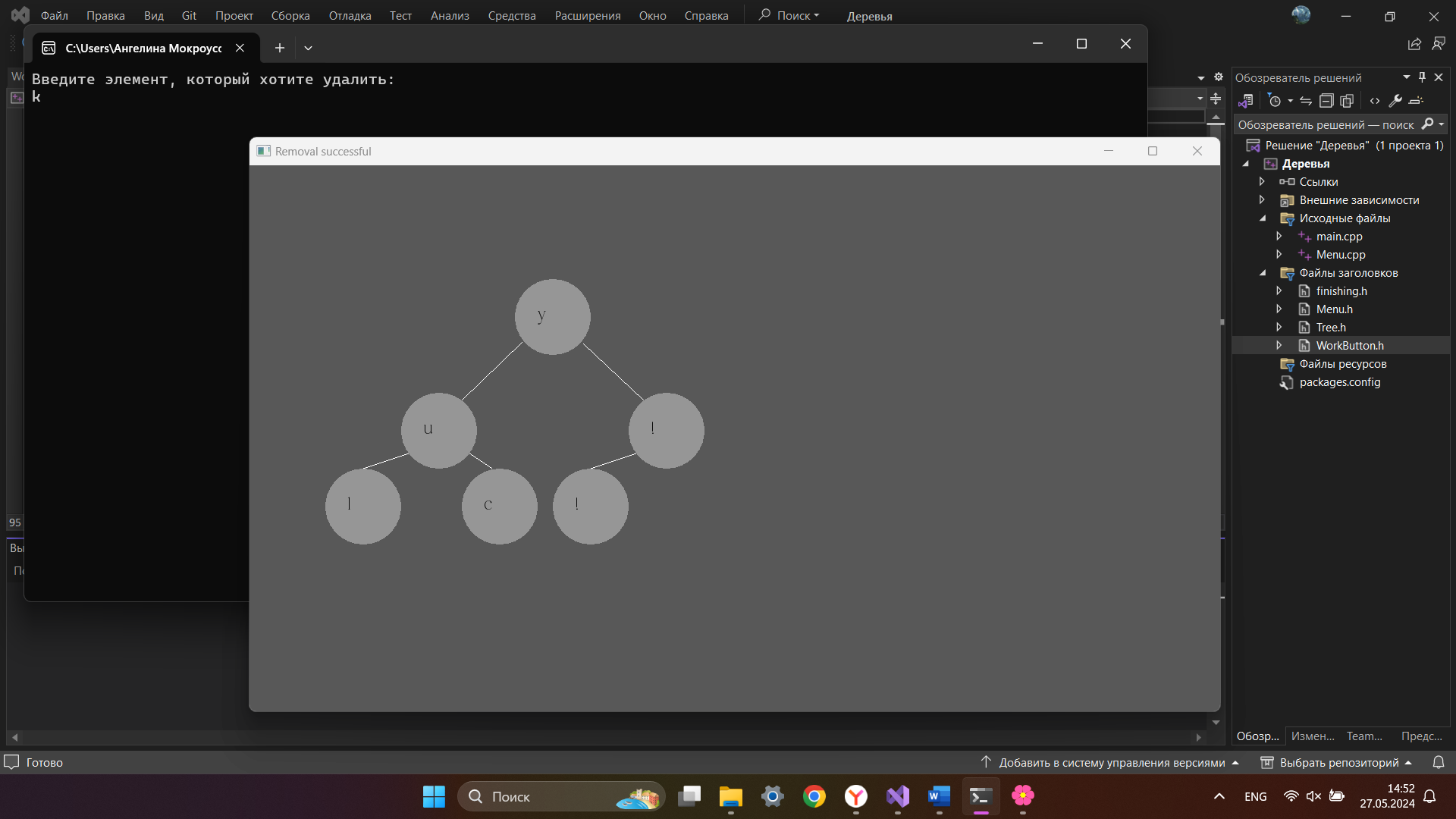


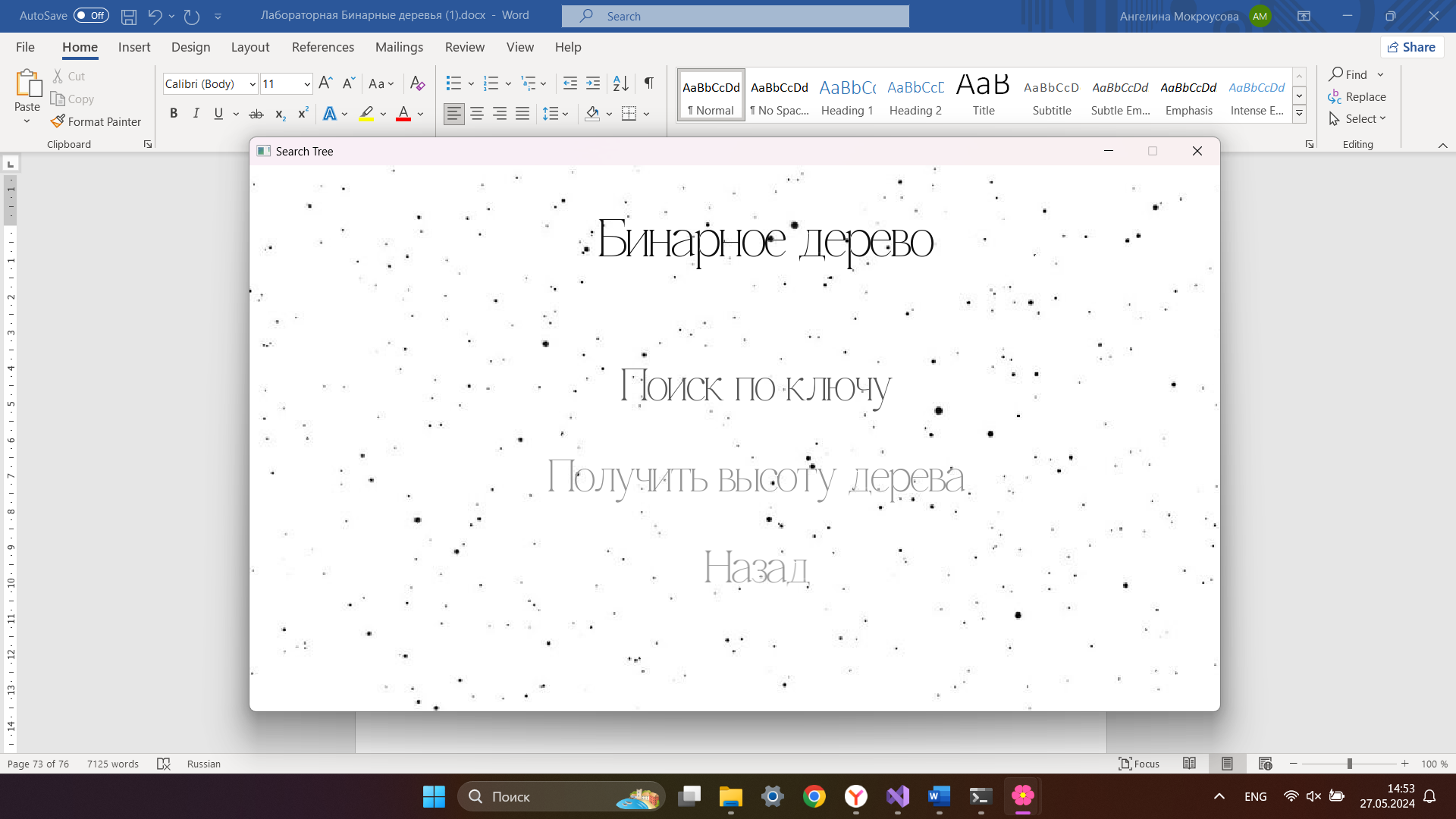


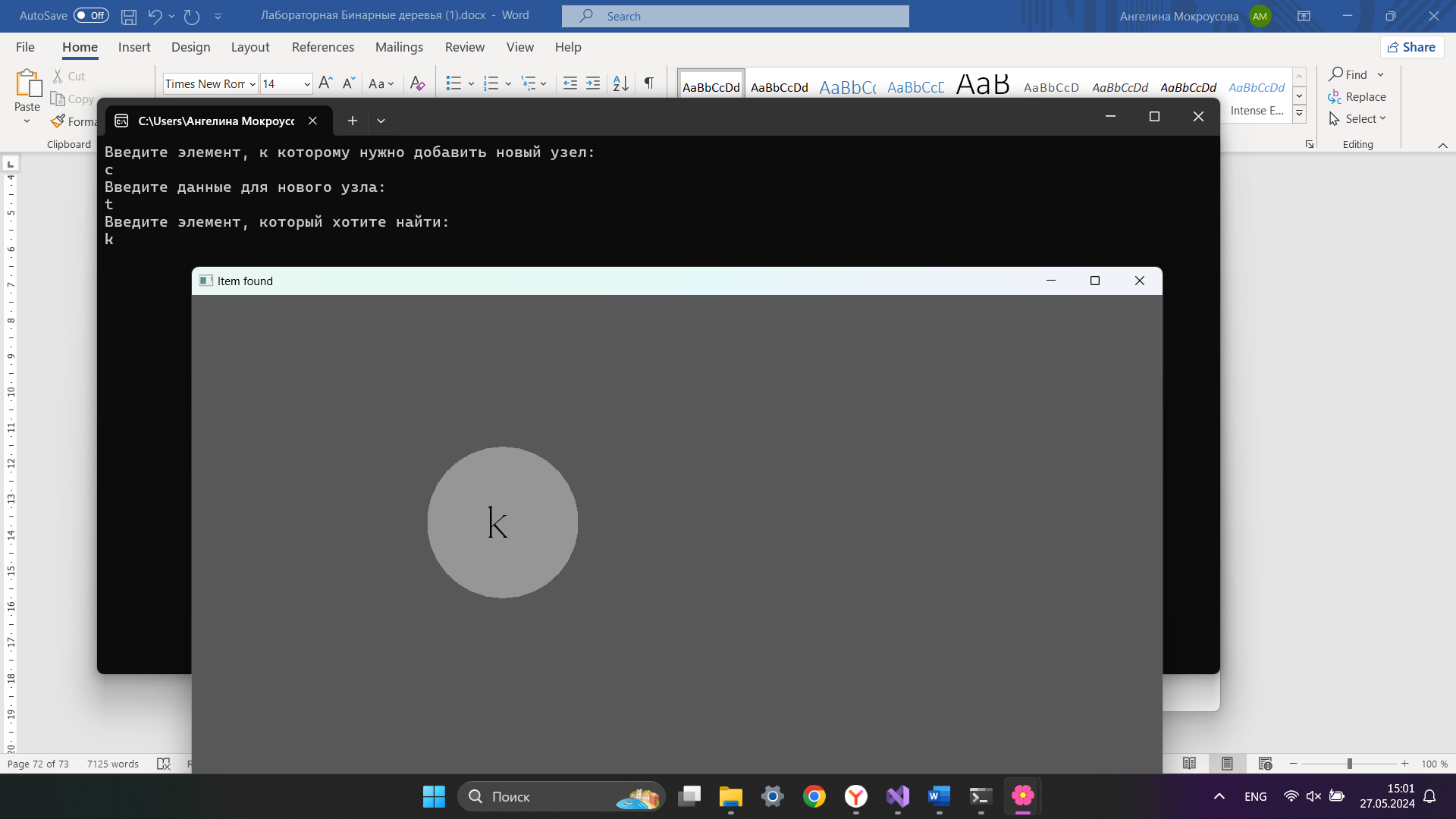


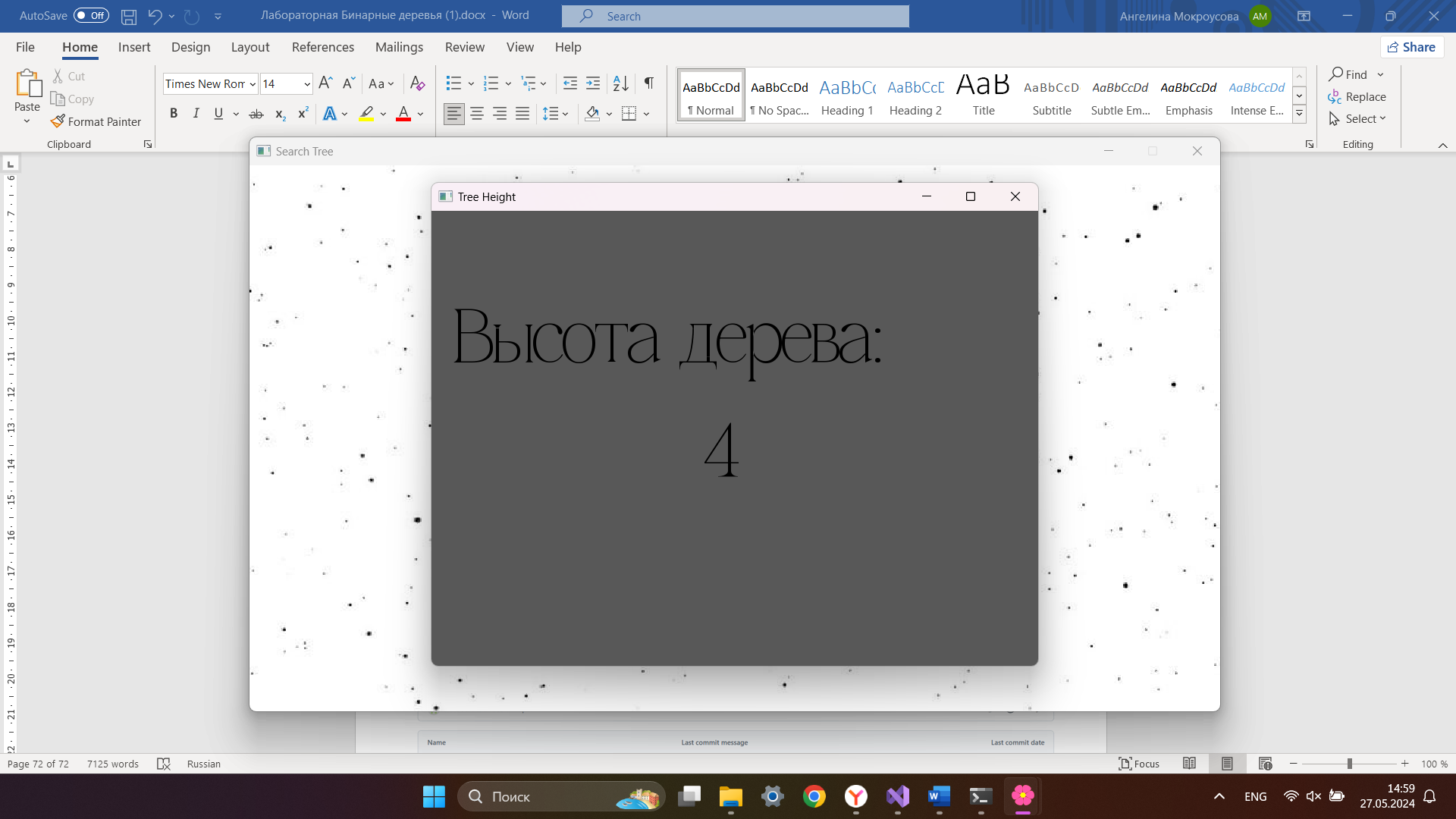












Вывод: В данной лабораторной работе был реализован класс бинарного дерева поиска с использованием шаблонов в языке программирования C++. Для визуализации работы программы использовалась библиотека SFML, которая позволила создать простой графический интерфейс пользователя. Данная лабораторная работа помогла углубить знания в области структур данных и алгоритмов, а также дала практический опыт работы с графическим интерфейсом пользователя.

Ссылка на работу в GitHub: