Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №20**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: “ Бинарные деревья.”

Вариант 16

Выполнил:

Студент группы ИВТ-20-2Б Тедеев Алесандр Зурабович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**

1. Сформировать идеально сбалансированное бинарное дерево.
2. Тип информационного поля int. Найти максимальный элемент в дереве.
3. Распечатать полученное дерево.
4. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
5. Преобразовать идеально сбалансированное дерево в дерево поиска.  
   Распечатать полученное дерево.

**Анализ задачи**

**1.** Для решения задач необходимо:

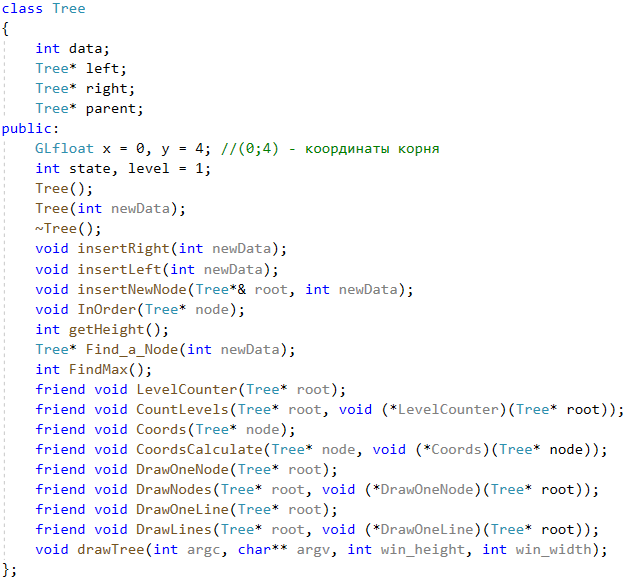
**1.1.** Организовать класс Tree с полями под указатель на правый, левый и родительский узлы, а также на хранящееся значение в узле, координаты узла и уровень, на котором узел находится.

**1.2.** Подключить библиотеку отрисовки glut.h для печати дерева.

**1.3.** Организовать отдельные функции в главном файле для работы с библиотекой OpenGL.

**2.** В ходе работы были использованы следующие типы данных:

**2.1.** Класс Tree с полями под указатель на правый, левый и родительский узлы, а также на хранящееся значение в узле, координаты узла и уровень, на котором узел находится.



**2.2.** Переменные типа int – width, depth, maxim. Width – максимальная ширина дерева (количество узлов на самом низком уровне), depth – глубина дерева (количество уровней), maxim – максимальный по значению элемент дерева.

int depth = 0, width = 0, maxim = 0;

**2.3.** Указатель tree класса Tree для работы с бинарным деревом. Значение, которое передаётся в качестве параметра – значение корня бинарного дерева.

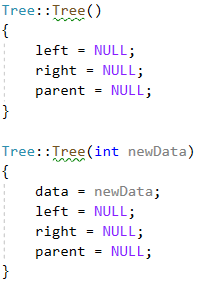
Tree\* tree = new Tree(5);

**2.4.** Вектор arr типа int для хранения данных узлов дерева.

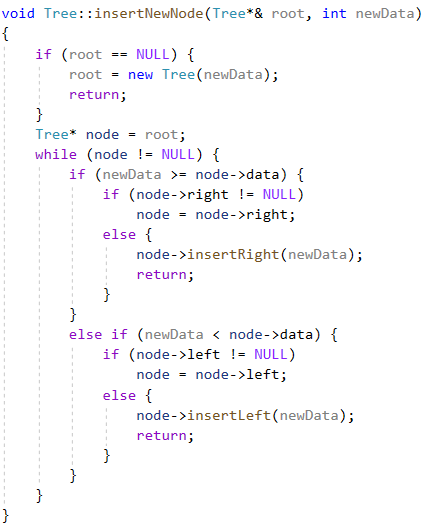
vector<int> arr = { 3,8,6,10,4,7,1,12,-1,3 };

**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Бинарное дерево создаётся через оператор new и конструктор класса Tree.

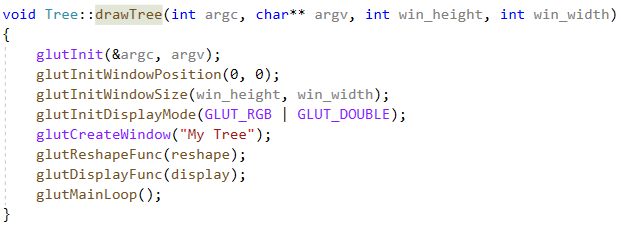


**3.2.** Добавление узлов в дерево обеспечивается методом insetNewNode(). Данный метод также сортирует дерево с помощью дополнительных проверок: значение, большее чем элемент узла отправляется по правой ветке дерева, а иначе по левой.



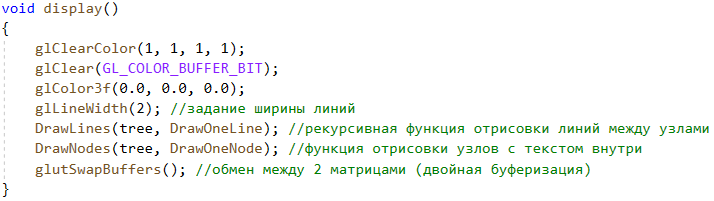
**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для вывода дерева в отдельное окно используется метод drawTree() класса Tree. Используются методы, прописанные в библиотеке glut.h, такие как glutInit() для инициализации glut; glutInitWindowPosition() для указания позиции окна, glutWindowSize() для указания размеров окна в пикселях, glutDisplayMode() для включения двойной буферизации и для отображения графической информации через 3 основных цвета; glutCreateWidnow() для создания самого окна с названием, отправляемым в качестве параметра; glutReshapeFunc() для задания функции обработки изменения размеров окна, в качестве параметра отправляется указатель на функцию; glutDisplayFunc() задаёт функцию рисования изображения; glutMainLoop() – запуск главного цикла.

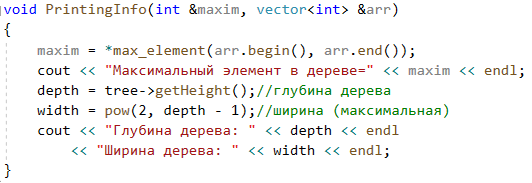


**4.2.** Функция diplay() необходима для отрисовки узлов и связей между ними.

glClearColor() – задание белого фона окна; glClear() – очистка буфера; glColor3f() – задаёт черный цвет для связей; glLineWidth() – задаёт ширину связей; DrawLines() и DrawNodes() – рекурсивные функции для прямого прохода, который необходимы для отрисовки связей и узлов; glutSwapBuffers() – смена экранного буфера для двойной буферизации.

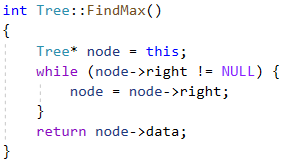


**4.3.** В функции PrintingInfo() происходит печать на консоль данных о глубине, ширине и максимальном элементе дерева.

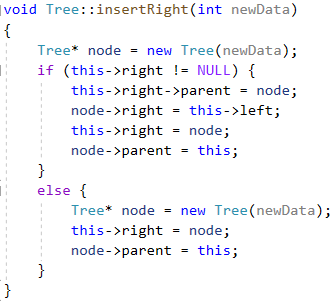
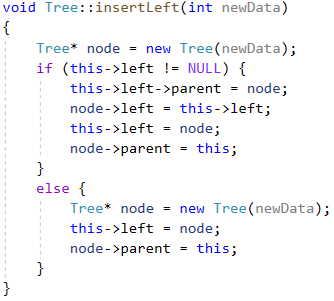


**5.** Поставленные задачи будут решены следующими действиями:

**5.1.** Нахождение максимального элемента осуществимо через метод FindMax().

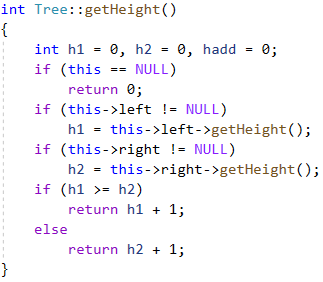


**5.2.** Добавление узлов в дерево обеспечивается методом insertNewNode(), который описан в пункте 3.2.Однако внутри этого метода используются другие два метода: insertRight() и insertLeft() для добаление в правую или левую ветку соответственно, в случае, если узел, на место которого необходимо вставить новый элемент не является пустым (NULL).

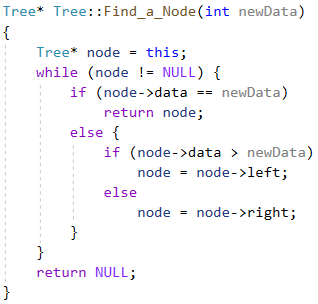
 

В данных методах происходит подмена связей.

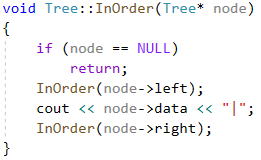
**5.3.** Вычисление высоты дерева происходит в методе getHeight(). Данный метод рекурсивно двигается по правой и по левой выисляет максимально количество по узлов с обеих сторон в переменных h1, h2 и затем возвращает, большее из них увеличенной на 1, чтобы учесть ещё корень дерева.



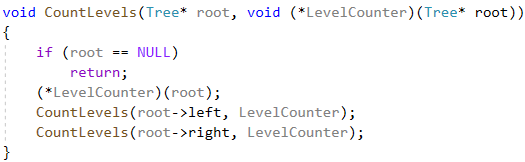
**5.4.** Поиск узла происходит в методе Find\_a\_Node(). В цикле while происходит сравнение значения узла и значения, отправленного параметром в функцию. Если происходит равенство этих величин, то



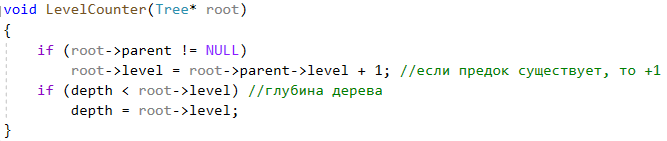
**5.5.** Прямой проход также продемонстрирован методом InOrder(). Метод рекурсивный сначала с проходом по левой стороне, а оптом по правой.



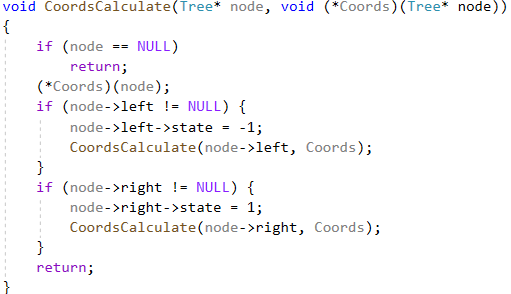
**5.6.** В функции CountLevels() происходит вычисление уровня для каждого узла путём прямого прохода по каждому из элементов. Данная функция в качестве параметра вызывает другую функцию – LevelCounter().



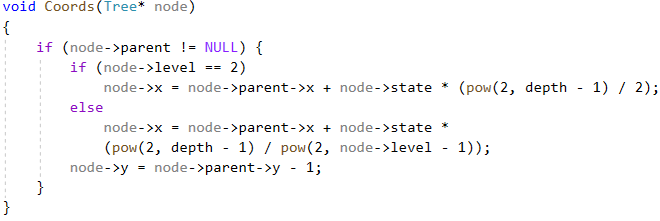
Данная функция проверяет, существует ли у узла предок. Если да, то данный узел получает уровень, увеличенный на 1 от родительского. Также происходит сравнение глубины дерева с данным уровнем (дополнительная подстраховка того, что глубина определена правильно).



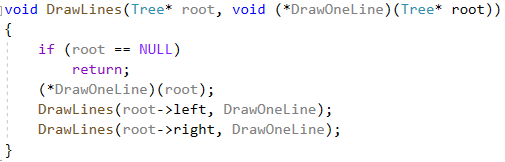
**5.7.** В функции CoordsCalculate() происходит вычисление координат для каждого узла путём прямого прохода по каждому из элементов. Данная функция в качестве параметра принимает другую функцию – Coords(). Кроме того, в ветвления заполняется поле класса Tree state, которое используется как коэффициент для вычисления координат: для узлов справа коэффициент положительный, так как сдвиг идёт от центра вправо. Если переменная слева, то нужно сдвигать координаты узла в отрицательное направление.



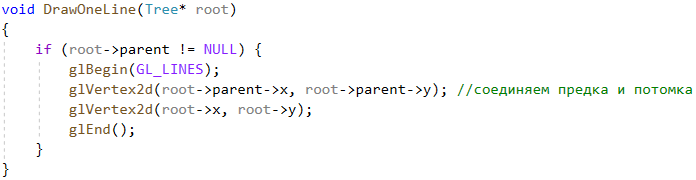
Данная функция проверяет, существует ли у узла предок. Если да, то происходит сравнение уровня узла: если уровень данного узла второй, то координата x вычисляется по одной формуле, если другой, то используется немного другая формула. Для координаты y всё проще: координата наследника меньше на 1 координаты предка.



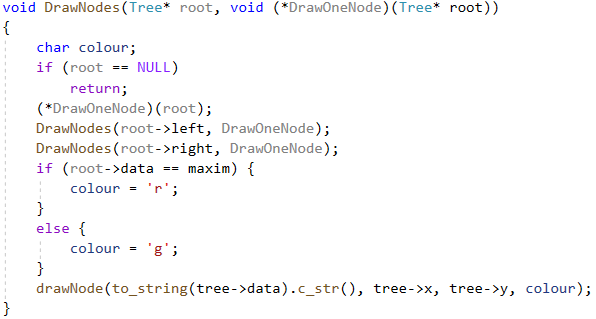
**5.8.** Отрисовка (печать) дерева происходит в методе drawTree, описание которого содержится в пунктах 4.1. и 4.2. Однако в функции display() используются рекурсивные функции DrawLines() и DrawNodes().



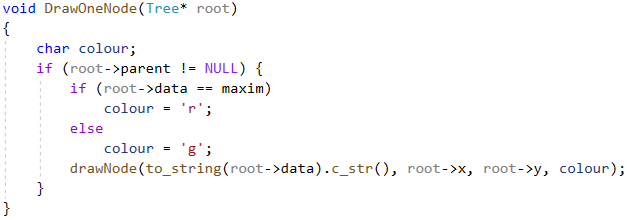
Функция DrawOneLine() проверяет, существует ли у узла предок. Если да, то чертится линия от координат предка до координат потомка.



**5.9** Функция DrawNodes() использует прямой проход по дереву, вызывая функцию DrawOneNode(). Также производится сравнение элемента узла с максимальным значением и если он равен ему, то отправляется буква r параметром в функцию drawNode(), чтобы нарисовать красный узел, в противном случае рисуется зелёный узел.



Функция DrawOneNode() проверяет, существует ли у узла предок. Если да, то производится сравнение элемента узла с максимальным значением и если он равен ему, то отправляется буква r параметром в функцию drawNode(), чтобы нарисовать красный узел, в противном случае рисуется зелёный узел.



**5.10.** В функции drawNode() вычисляются размеры узлов через переменные coefficient, radiusB. Текст передаётся через переменную str в качестве параметра, печатается в окне с помощью цикла for, в котором используется метод glutBitmapChatacter для задания шрифта для текста.

glColor3f(0.0, 0.0, 0.0); // чёрный цвет для текста

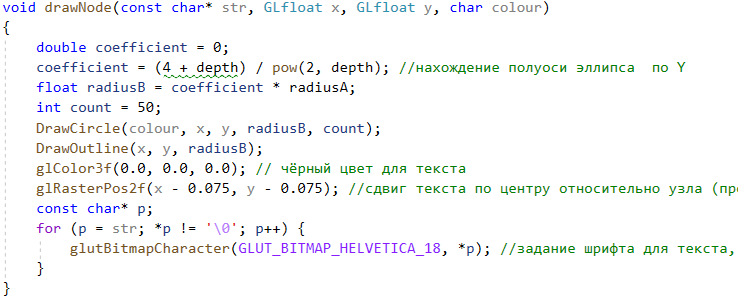
glRasterPos2f(x - 0.075, y - 0.075); //сдвиг текста по центру относительно узла

const char\* p;

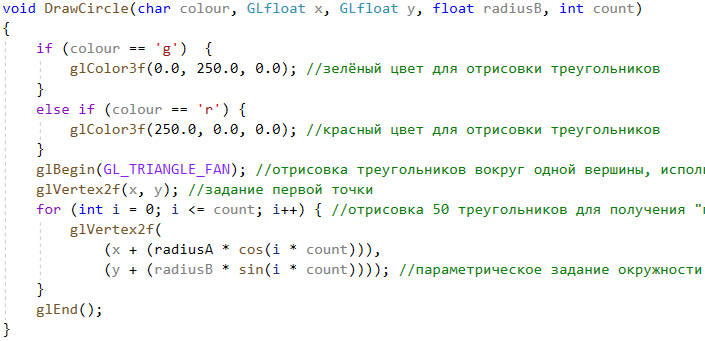
for (p = str; \*p != '\0'; p++) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*p); //задание шрифта для текста, значение

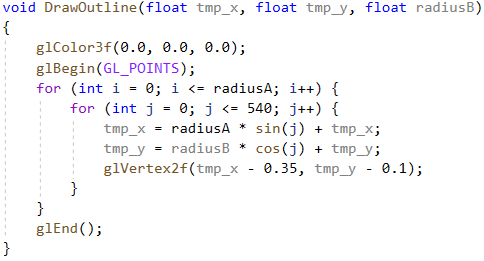
}



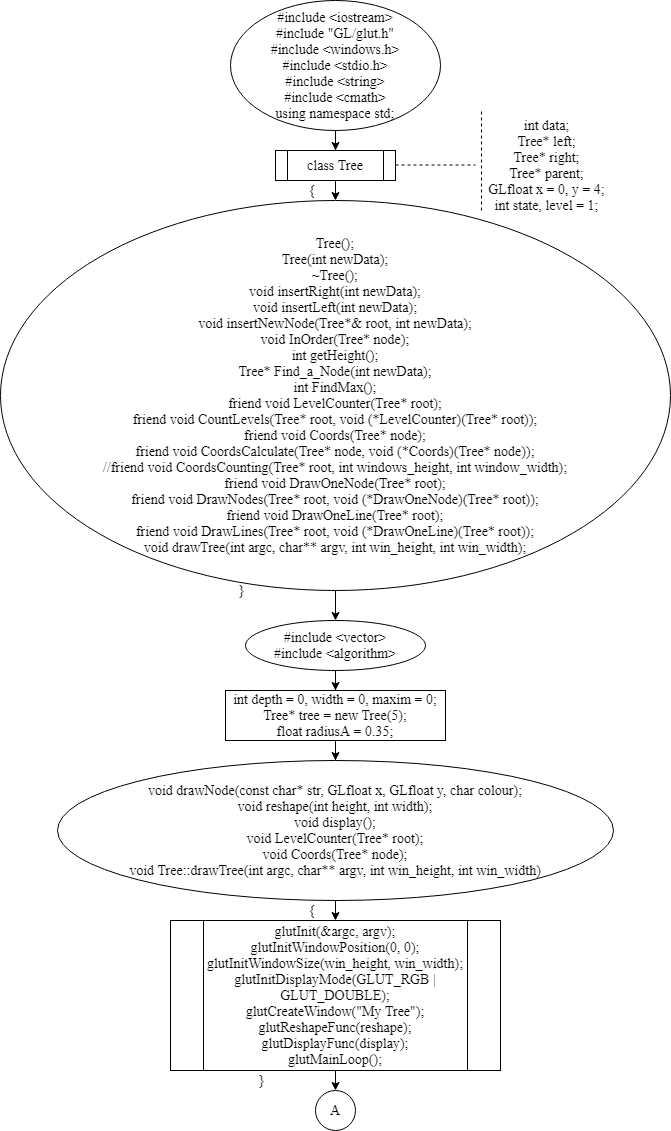
В функции DrawCircle() производится рисование узлов путем рисования 50 маленьких треугольников вокруг одной точки.

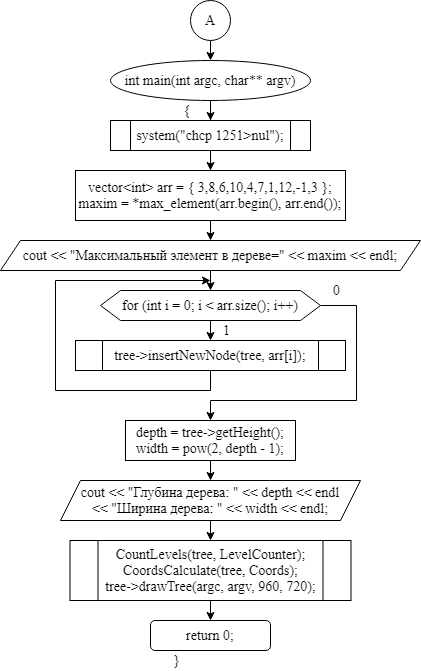


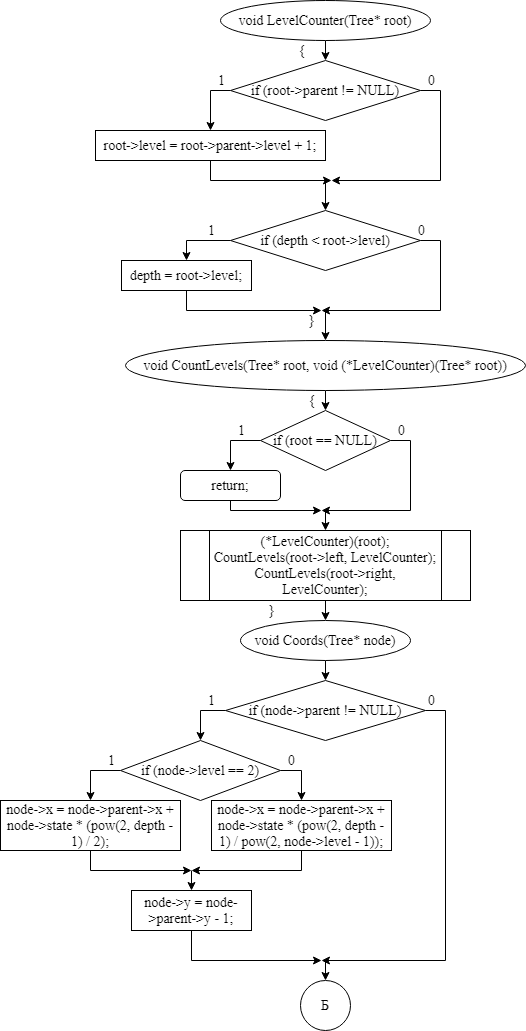
В функции DrawOutline() производится рисование множества точек вокруг одной точки.

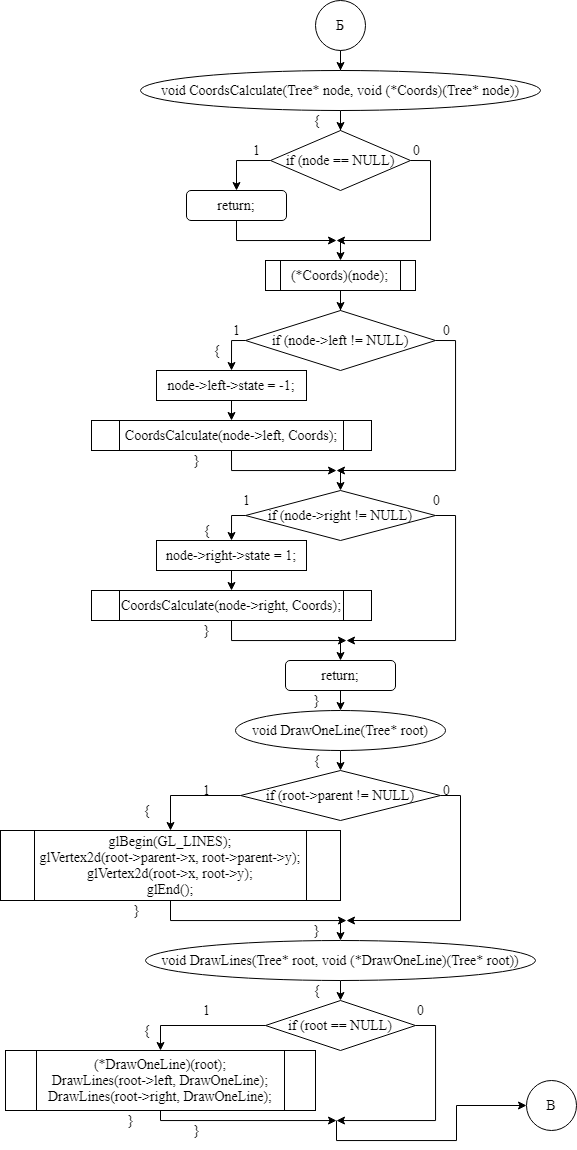


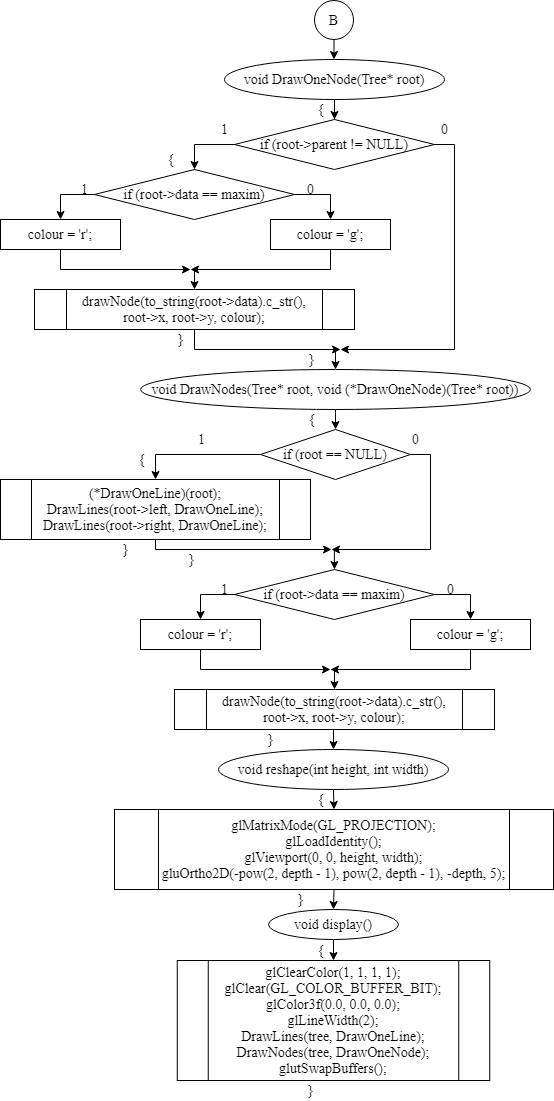
**Блок-схема**

****

****

****

****

****

**Код**

#include "Tree.h"

#include <vector>

#include <algorithm>

int depth = 0, width = 0, maxim = 0;

Tree\* tree = new Tree(5);

float radiusA = 0.35; //радиус окружности

void PrintingInfo(int &maxim, vector<int> &arr);

void DrawCircle(char colour, GLfloat x, GLfloat y, float radiusB, int count);

void DrawOutline(float tmp\_x, float tmp\_y, float radiusB);

void drawNode(const char\* str, GLfloat x, GLfloat y, char colour);

void reshape(int height, int width);

void display();

void LevelCounter(Tree\* root);

void Coords(Tree\* node);

void Tree::drawTree(int argc, char\*\* argv, int win\_height, int win\_width)

{

glutInit(&argc, argv);

glutInitWindowPosition(0, 0);

glutInitWindowSize(win\_height, win\_width);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);

glutCreateWindow("My Tree");

glutReshapeFunc(reshape);

glutDisplayFunc(display);

glutMainLoop();

}

int main(int argc, char\*\* argv)

{

system("chcp 1251>nul");

vector<int> arr = { 3,8,6,10,4,7,1,12,-1,3 };

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

tree->insertNewNode(tree, arr[i]);

}

PrintingInfo(maxim, arr);

CountLevels(tree, LevelCounter);

CoordsCalculate(tree, Coords);

tree->drawTree(argc, argv, 960, 720);

return 0;

}

void LevelCounter(Tree\* root)

{

if (root->parent != NULL)

root->level = root->parent->level + 1; //если предок существует, то +1

if (depth < root->level) //глубина дерева

depth = root->level;

}

void CountLevels(Tree\* root, void (\*LevelCounter)(Tree\* root))

{

if (root == NULL)

return;

(\*LevelCounter)(root);

CountLevels(root->left, LevelCounter);

CountLevels(root->right, LevelCounter);

}

void Coords(Tree\* node)

{

if (node->parent != NULL) {

if (node->level == 2)

node->x = node->parent->x + node->state \* (pow(2, depth - 1) / 2);

else

node->x = node->parent->x + node->state \*

(pow(2, depth - 1) / pow(2, node->level - 1));

node->y = node->parent->y - 1;

}

}

void CoordsCalculate(Tree\* node, void (\*Coords)(Tree\* node))

{

if (node == NULL)

return;

(\*Coords)(node);

if (node->left != NULL) {

node->left->state = -1;

CoordsCalculate(node->left, Coords);

}

if (node->right != NULL) {

node->right->state = 1;

CoordsCalculate(node->right, Coords);

}

return;

}

void DrawCircle(char colour, GLfloat x, GLfloat y, float radiusB, int count)

{

if (colour == 'g') {

glColor3f(0.0, 250.0, 0.0); //зелёный цвет для отрисовки треугольников

}

else if (colour == 'r') {

glColor3f(250.0, 0.0, 0.0); //красный цвет для отрисовки треугольников

}

glBegin(GL\_TRIANGLE\_FAN); //отрисовка треугольников вокруг одной вершины, используется для рисования окружности

glVertex2f(x, y); //задание первой точки

for (int i = 0; i <= count; i++) { //отрисовка 50 треугольников для получения "плавности окружности"

glVertex2f(

(x + (radiusA \* cos(i \* count))),

(y + (radiusB \* sin(i \* count)))); //параметрическое задание окружности

}

glEnd();

}

void DrawOutline(float tmp\_x, float tmp\_y, float radiusB)

{

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POINTS);

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

for (int j = 0; j <= 540; j++) {

tmp\_x = radiusA \* sin(j) + tmp\_x;

tmp\_y = radiusB \* cos(j) + tmp\_y;

glVertex2f(tmp\_x - 0.35, tmp\_y - 0.1);

}

}

glEnd();

}

void drawNode(const char\* str, GLfloat x, GLfloat y, char colour)

{

double coefficient = 0;

coefficient = (4 + depth) / pow(2, depth); //нахождение полуоси эллипса по Y

float radiusB = coefficient \* radiusA;

int count = 50;

DrawCircle(colour, x, y, radiusB, count);

DrawOutline(x, y, radiusB);

glColor3f(0.0, 0.0, 0.0); // чёрный цвет для текста

glRasterPos2f(x - 0.075, y - 0.075); //сдвиг текста по центру относительно узла (простой подбор констант)

const char\* p;

for (p = str; \*p != '\0'; p++) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*p); //задание шрифта для текста, значение

}

}

void DrawOneLine(Tree\* root)

{

if (root->parent != NULL) {

glBegin(GL\_LINES);

glVertex2d(root->parent->x, root->parent->y); //соединяем предка и потомка

glVertex2d(root->x, root->y);

glEnd();

}

}

void DrawLines(Tree\* root, void (\*DrawOneLine)(Tree\* root))

{

if (root == NULL)

return;

(\*DrawOneLine)(root);

DrawLines(root->left, DrawOneLine);

DrawLines(root->right, DrawOneLine);

}

void DrawOneNode(Tree\* root)

{

char colour;

if (root->parent != NULL) {

if (root->data == maxim)

colour = 'r';

else

colour = 'g';

drawNode(to\_string(root->data).c\_str(), root->x, root->y, colour);

}

}

void DrawNodes(Tree\* root, void (\*DrawOneNode)(Tree\* root))

{

char colour;

if (root == NULL)

return;

(\*DrawOneNode)(root);

DrawNodes(root->left, DrawOneNode);

DrawNodes(root->right, DrawOneNode);

if (root->data == maxim) {

colour = 'r';

}

else {

colour = 'g';

}

drawNode(to\_string(tree->data).c\_str(), tree->x, tree->y, colour);

}

void reshape(int height, int width)

{

glMatrixMode(GL\_PROJECTION); //все последующие изменения будут применяться к проекционной матрице

glLoadIdentity();

glViewport(0, 0, height, width); //область просмотра (c точки 0:0) на ширину и длину окна

gluOrtho2D(-pow(2, depth - 1), pow(2, depth - 1), -depth, 5); //ортогональная проекция

}

void display()

{

glClearColor(1, 1, 1, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

glLineWidth(2); //задание ширины линий

DrawLines(tree, DrawOneLine); //рекурсивная функция отрисовки линий между узлами

DrawNodes(tree, DrawOneNode); //функция отрисовки узлов с текстом внутри

glutSwapBuffers(); //обмен между 2 матрицами (двойная буферизация)

}

void PrintingInfo(int &maxim, vector<int> &arr)

{

maxim = \*max\_element(arr.begin(), arr.end());

cout << "Максимальный элемент в дереве=" << maxim << endl;

depth = tree->getHeight();//глубина дерева

width = pow(2, depth - 1);//ширина (максимальная)

cout << "Глубина дерева: " << depth << endl

<< "Ширина дерева: " << width << endl;

}

#pragma once

#include <iostream>

#include "GL/glut.h"

#include <windows.h>

#include <stdio.h>

#include <string>

#include <cmath>

using namespace std;

class Tree

{

int data;

Tree\* left;

Tree\* right;

Tree\* parent;

public:

GLfloat x = 0, y = 4; //(0;4) - координаты корня

int state, level = 1;

Tree();

Tree(int newData);

~Tree();

void insertRight(int newData);

void insertLeft(int newData);

void insertNewNode(Tree\*& root, int newData);

void InOrder(Tree\* node);

int getHeight();

Tree\* Find\_a\_Node(int newData);

int FindMax();

friend void LevelCounter(Tree\* root);

friend void CountLevels(Tree\* root, void (\*LevelCounter)(Tree\* root));

friend void Coords(Tree\* node);

friend void CoordsCalculate(Tree\* node, void (\*Coords)(Tree\* node));

friend void DrawOneNode(Tree\* root);

friend void DrawNodes(Tree\* root, void (\*DrawOneNode)(Tree\* root));

friend void DrawOneLine(Tree\* root);

friend void DrawLines(Tree\* root, void (\*DrawOneLine)(Tree\* root));

void drawTree(int argc, char\*\* argv, int win\_height, int win\_width);

};

#include "Tree.h"

Tree::Tree()

{

left = NULL;

right = NULL;

parent = NULL;

}

Tree::Tree(int newData)

{

data = newData;

left = NULL;

right = NULL;

parent = NULL;

}

Tree::~Tree()

{

if (this != NULL)

delete this;

}

void Tree::insertRight(int newData)

{

Tree\* node = new Tree(newData);

if (this->right != NULL) {

this->right->parent = node;

node->right = this->left;

this->right = node;

node->parent = this;

}

else {

Tree\* node = new Tree(newData);

this->right = node;

node->parent = this;

}

}

void Tree::insertLeft(int newData)

{

Tree\* node = new Tree(newData);

if (this->left != NULL) {

this->left->parent = node;

node->left = this->left;

this->left = node;

node->parent = this;

}

else {

Tree\* node = new Tree(newData);

this->left = node;

node->parent = this;

}

}

void Tree::insertNewNode(Tree\*& root, int newData)

{

if (root == NULL) {

root = new Tree(newData);

return;

}

Tree\* node = root;

while (node != NULL) {

if (newData >= node->data) {

if (node->right != NULL)

node = node->right;

else {

node->insertRight(newData);

return;

}

}

else if (newData < node->data) {

if (node->left != NULL)

node = node->left;

else {

node->insertLeft(newData);

return;

}

}

}

}

void Tree::InOrder(Tree\* node)

{

if (node == NULL)

return;

InOrder(node->left);

cout << node->data << "|";

InOrder(node->right);

}

int Tree::getHeight()

{

int h1 = 0, h2 = 0, hadd = 0;

if (this == NULL)

return 0;

if (this->left != NULL)

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

if (this->right != NULL)

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

if (h1 >= h2)

return h1 + 1;

else

return h2 + 1;

}

Tree\* Tree::Find\_a\_Node(int newData)

{

Tree\* node = this;

while (node != NULL) {

if (node->data == newData)

return node;

else {

if (node->data > newData)

node = node->left;

else

node = node->right;

}

}

return NULL;

}

int Tree::FindMax()

{

Tree\* node = this;

while (node->right != NULL) {

node = node->right;

}

return node->data;

}

**Скриншоты**

