**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: БДП

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Гречко В.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

## Цель работы.

Ознакомиться с основными характеристиками и особенностями такой структуры данных, как бинарное дерево поиска, изучить особенности ее реализации на языке программирования C++. Разработать программу, использующую бинарное дерево поиска для удаления элемента и её визуализацию.

## Задание.

По заданному файлу F (типа file of Elem), все элементы которого различны, построить структуру данных определённого типа – БДП: рандомизированная пирамида поиска.

Для построенной структуры данных проверить, входит ли в неё элемент е типа Elem, и если входит, то удалить элемент е из структуры данных. Предусмотреть возможность повторного выполнения с другим элементом.

## Основные теоретические положения.

При построении рандомизированного дерева основное исходное положение заключается в том, что любой элемент может с одинаковой вероятностью быть корнем дерева. Причем это справедливо и для поддеревьев рандомизированного дерева. Поэтому при включении нового элемента в дерево, случайным образом выбирается включение элемента в качестве листа, или в качестве корня.

Вероятность появления нового узла в корне дерева или поддерева определяется, как 1/(n+1), где n – число узлов в дереве или поддереве. То есть дерево выглядит так, будто элементы поступали в случайном порядке.

Поскольку принимаемые алгоритмом решения являются случайными, то при каждом выполнении алгоритма при одной и то же последовательности включений будут получаться деревья различной конфигурации. Таким образом, несмотря на увеличение трудоемкости операции включений, за счет рандомизации получается структура дерева, близкая к сбалансированной.

## Выполнение работы.

Написание работы производилось на базе операционной системы Elementary OS в среде разработки QtCreator с использованием фреймворка Qt. Сборка, отладка производились в QtCreator. Исходные коды файлов программы представлены в приложениях А-Ж.

Для реализации программы был разработан графический интерфейс с помощью встроенного в QtCreator UI-редактора. Он представляет из себя поле ввода, кнопку считывания, поле ввода для поиска и удаления элемента, а также поле вывода с возможностью графического отображения результата. Основные слоты для работы графического интерфейса приведены в табл. 1.

Таблица 1 – Слоты класса MainWindow и их назначение

|  |  |
| --- | --- |
| Метод | Назначение |
| on\_printTree\_clicked() | Слот, отвечающий за считывание данных и графического вывода |
| on\_delete\_elem\_clicked() | Слот, отвечающий за поиск и удаление элемента |
| on\_choose\_file\_clicked() | Слот, отвечающий за считывание данных из файла |

Для реализации бинарного дерева были созданы структуры узла Node и самого дерева BinTree, представленные на рис. 2.

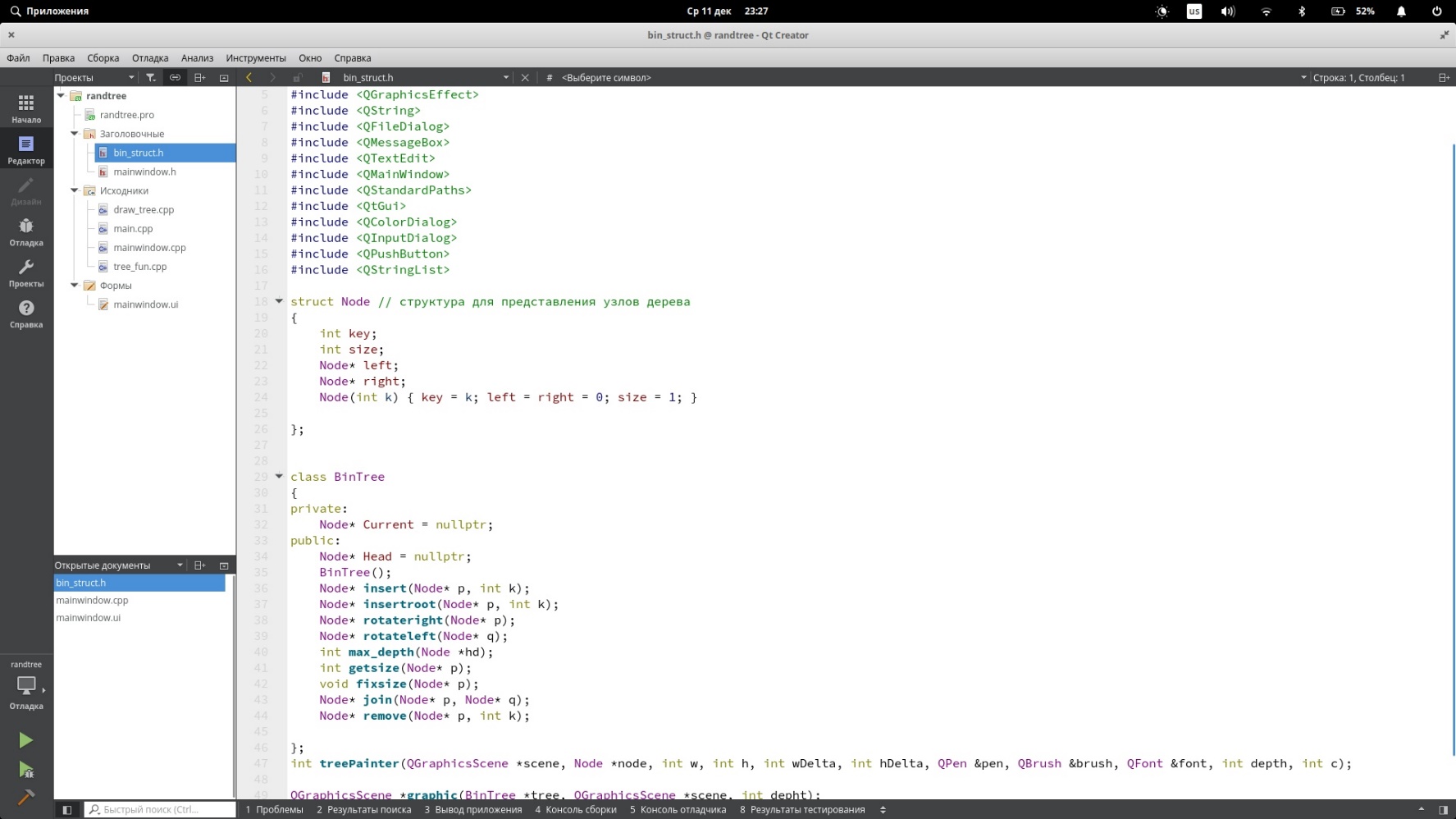


Рисунок 2 – Структуры бинарного дерева и узла

Также были реализованы функции, создающие и изменяющие бинарное дерево, приведенные в табл. 2.

Таблица 2 – Основные функции работы с бинарным деревом

|  |  |
| --- | --- |
| Функция | Назначение |
| void BinTree() | Создает пустое бинарное дерево |
| Node\* insert(Node\* p, int k) | Рандомизированная вставка нового узла с ключом k в дерево p |
| int max\_depth(Node \*hd) | Возвращает максимальную глубину дерева |
| Node\* insertroot(Node\* p, int k) | Вставка нового узла с ключом k в корень дерева p |
| Node\* rotateright(Node\* p) | Правый поворот вокруг узла p |
| Node\* rotateleft(Node\* q) | Левый поворот вокруг узла q |
| int getsize(Node\* p) | Получение размера дерева |
| void fixsize(Node\* p) | Установление корректного размера дерева |
| Node\* join(Node\* p, Node\* q) | Объединение двух деревьев |
| Node\* remove(Node\* p, int k) | Удаление из дерева p первого найденного узла с ключом k |

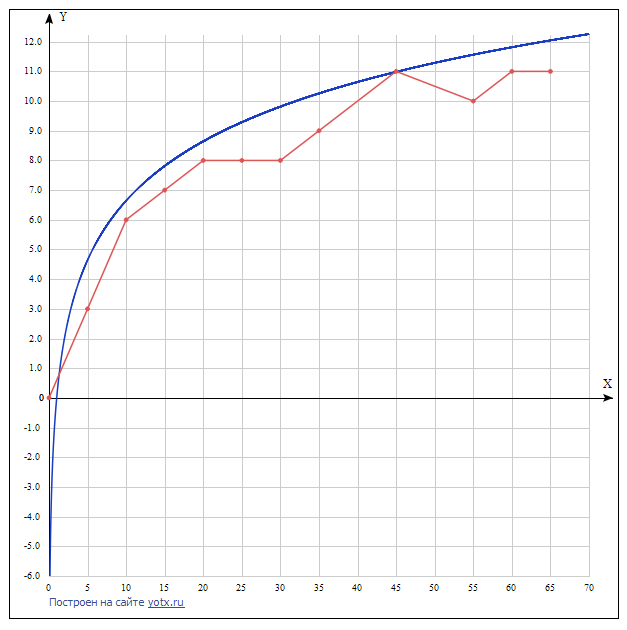
Программа имеет возможность графического отображения полученного бинарного дерева с помощью виджета QGraphicsView. Функции, необходимые для графического представления дерева, представлены в табл. 3.

Таблица 3 – Функции, связующие графический интерфейс и алгоритмы

|  |  |
| --- | --- |
| Функция | Назначение |
| QGraphicsScene \*graphic(BinTree \*tree, QGraphicsScene \*&scene, int depth) | По заданному бинарному дереву выполняет рисование в объекте QGraphicsScene |
| int treePainter(QGraphicsScene \*&scene, Node \*node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth) | Рекурсивный алгоритм обхода дерева и рисования узлов в заданном объекте QGraphicsScene |

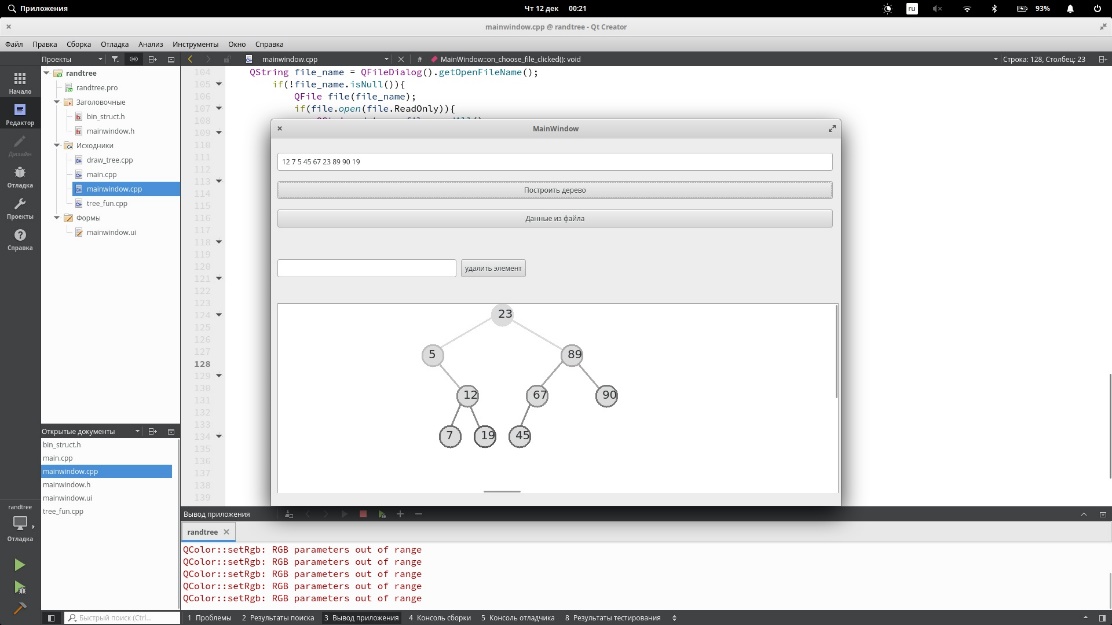
## Оценка сложности алгоритма.

Построенное дерево окажется неплохо сбалансированным: его высота будет порядка 2.

**

## Тестирование программы.

Вид программы после выполнения представлен на рис. 3.



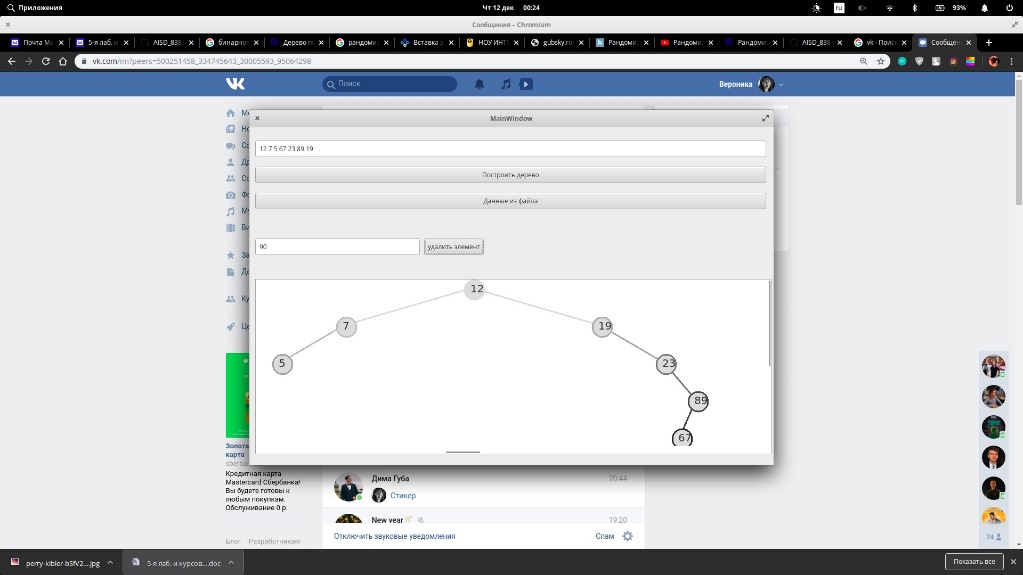
Рисунок 3.1 – Графический интерфейс программы

Рисунок 3.2 – Графический интерфейс программы

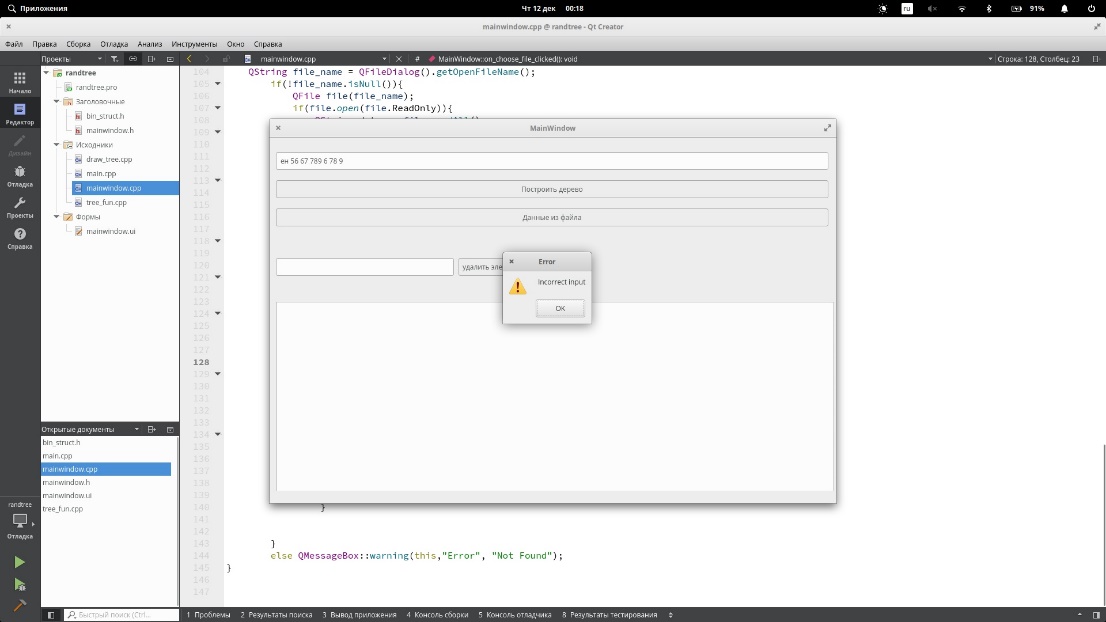
Также был рассмотрен случай некорректно введённых данных на рис. 4.

Рисунок 4 – Ошибка ввода

## Выводы.

В ходе выполнения лабораторной работы была написана программа, создающая бинарное дерево поиска и удаляющая заданный элемент. Печать бинарного дерева выполняется графически.

# Приложение А Исходный код программы. MAIN.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

# Приложение Б Исходный код программы. mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QGraphicsItem>

#include <QGraphicsView>

#include <QGraphicsEffect>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

private slots:

void on\_printTree\_clicked();

void on\_delete\_elem\_clicked();

void on\_choose\_file\_clicked();

private:

Ui::MainWindow \*ui;

QGraphicsScene \*scene;

};

#endif // MAINWINDOW\_H

# Приложение В Исходный код программы. mainwindow.cPP

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include<bin\_struct.h>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

scene = new QGraphicsScene;

ui->graphicsView->setScene(scene);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_printTree\_clicked()

{

QString data = ui->input\_tree->text();

QStringList abc = data.split(' ');

int\* mas = new int[100];

int i =0;

bool go = true;

for (auto x:abc){

bool convertOK;

x.toInt(&convertOK);

if(convertOK == false){

go = false;

}

else{

mas[i] = x.toInt();

i++;

}

}

if(!go){

QMessageBox::warning(this,"Error", "Incorrect input");

return;

}

BinTree\* BT = new (BinTree);

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

BT->Head = BT->insert(BT->Head, mas[j]);

}

int depth = BT->max\_depth(BT->Head);

graphic(BT, scene, depth);

}

void MainWindow::on\_delete\_elem\_clicked()

{

QString data = ui->input\_tree->text();

QStringList abc = data.split(' ');

int\* mas = new int[100];

int i =0;

bool go = true;

for (auto x:abc){

bool convertOK;

x.toInt(&convertOK);

if(convertOK == false){

go = false;

}

else{

mas[i] = x.toInt();

i++;

}

}

if(!go){

QMessageBox::warning(this,"Error", "Incorrect input");

return;

}

BinTree\* BT = new (BinTree);

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

BT->Head = BT->insert(BT->Head, mas[j]);

}

QString elem\_ = ui->input\_del->text();

bool convert\_;

int elem = elem\_.toInt(&convert\_);

if(!convert\_){

QMessageBox::warning(this,"Error", "Incorrect input");

return;

}

BT->Head = BT->remove(BT->Head, elem);

int depth = BT->max\_depth(BT->Head);

QString out;

for(int l = 0; l < i - 1; l++){

if (mas[l] == elem){

for(int k = l; k < i - 1; k++){

mas[k] = mas[k + 1];

}

}

out.append(QString::number(mas[l]));

if(l != i -2) out.append(" ");

}

ui->input\_tree->setText(out);

graphic(BT, scene, depth);

}

void MainWindow::on\_choose\_file\_clicked()

{

QString file\_name = QFileDialog().getOpenFileName();

if(!file\_name.isNull()){

QFile file(file\_name);

if(file.open(file.ReadOnly)){

QString data = file.readAll();

if (data == "") {

QMessageBox::critical(this, "Error!", "Р’РІРµРґРёС‚Рµ РґРµСЂРµРІРѕ");

return;

}

else{

QStringList abc = data.split(' ');

int\* mas = new int[100];

int i =0;

bool go = true;

for (auto x:abc){

bool convertOK;

x.toInt(&convertOK);

if(convertOK == false){

go = false;

}

else{

mas[i] = x.toInt();

i++;

}

}

if(!go){

QMessageBox::warning(this,"Error", "Incorrect input");

return;

}

BinTree\* BT = new (BinTree);

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

BT->Head = BT->insert(BT->Head, mas[j]);

}

int depth = BT->max\_depth(BT->Head);

graphic(BT, scene, depth);

}

}

}

else QMessageBox::warning(this,"Error", "Not Found");

}

# Приложение Г Исходный код программы. bin\_struct.H

#ifndef BIN\_STRUCT\_H

#define BIN\_STRUCT\_H

#include <QGraphicsItem>

#include <QGraphicsView>

#include <QGraphicsEffect>

#include <QString>

#include <QFileDialog>

#include <QMessageBox>

#include <QTextEdit>

#include <QMainWindow>

#include <QStandardPaths>

#include <QtGui>

#include <QColorDialog>

#include <QInputDialog>

#include <QPushButton>

#include <QStringList>

struct Node // СЃС‚СЂСѓРєС‚СѓСЂР° РґР»СЏ РїСЂРµРґСЃС‚Р°РІР»РµРЅРёСЏ СѓР·Р»РѕРІ РґРµСЂРµРІР°

{

int key;

int size;

Node\* left;

Node\* right;

Node(int k) { key = k; left = right = 0; size = 1; }

};

class BinTree

{

private:

Node\* Current = nullptr;

public:

Node\* Head = nullptr;

BinTree();

Node\* insert(Node\* p, int k);

Node\* insertroot(Node\* p, int k);

Node\* rotateright(Node\* p);

Node\* rotateleft(Node\* q);

int max\_depth(Node \*hd);

int getsize(Node\* p);

void fixsize(Node\* p);

Node\* join(Node\* p, Node\* q);

Node\* remove(Node\* p, int k);

};

int treePainter(QGraphicsScene \*scene, Node \*node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth, int c);

QGraphicsScene \*graphic(BinTree \*tree, QGraphicsScene \*scene, int depht);

#endif // BIN\_STRUCT\_H

# Приложение Д Исходный код программы. TREE\_fun.cpp

#include <bin\_struct.h>

BinTree::BinTree(){

Head = nullptr;

Current = Head;

}

int BinTree::max\_depth(Node \*hd){

if (hd == NULL) return 0;

else{

int lDepth = max\_depth(hd->left);

int rDepth = max\_depth(hd->right);

if (lDepth > rDepth) return(lDepth + 1);

else return(rDepth + 1);

}

}

int BinTree::getsize(Node\* p) // РѕР±РµСЂС‚РєР° РґР»СЏ РїРѕР»СЏ size, СЂР°Р±РѕС‚Р°РµС‚ СЃ РїСѓСЃС‚С‹РјРё РґРµСЂРµРІСЊСЏРјРё (t=NULL)

{

if( !p ) return 0;

return p->size;

}

void BinTree::fixsize(Node\* p) // СѓСЃС‚Р°РЅРѕРІР»РµРЅРёРµ РєРѕСЂСЂРµРєС‚РЅРѕРіРѕ СЂР°Р·РјРµСЂР° РґРµСЂРµРІР°

{

p->size = getsize(p->left)+getsize(p->right)+1;

}

Node\* BinTree::rotateright(Node\* p) // РїСЂР°РІС‹Р№ РїРѕРІРѕСЂРѕС‚ РІРѕРєСЂСѓРі СѓР·Р»Р° p

{

Node\* q = p->left;

if( !q ) return p;

p->left = q->right;

q->right = p;

q->size = p->size;

fixsize(p);

return q;

}

Node\* BinTree::rotateleft(Node\* q) // Р»РµРІС‹Р№ РїРѕРІРѕСЂРѕС‚ РІРѕРєСЂСѓРі СѓР·Р»Р° q

{

Node\* p = q->right;

if( !p ) return q;

q->right = p->left;

p->left = q;

p->size = q->size;

fixsize(q);

return p;

}

Node\* BinTree::insertroot(Node\* p, int k) // РІСЃС‚Р°РІРєР° РЅРѕРІРѕРіРѕ СѓР·Р»Р° СЃ РєР»СЋС‡РѕРј k РІ РєРѕСЂРµРЅСЊ РґРµСЂРµРІР° p

{

if( !p ) return new Node(k);

if( k < p->key )

{

p->left = insertroot(p->left,k);

return rotateright(p);

}

else

{

p->right = insertroot(p->right,k);

return rotateleft(p);

}

}

Node\* BinTree::insert(Node\* p, int k) // СЂР°РЅРґРѕРјРёР·РёСЂРѕРІР°РЅРЅР°СЏ РІСЃС‚Р°РІРєР° РЅРѕРІРѕРіРѕ СѓР·Р»Р° СЃ РєР»СЋС‡РѕРј k РІ РґРµСЂРµРІРѕ p

{

if( !p ) return new Node(k);

if( rand()%(p->size+1)==0 )

return insertroot(p,k);

if( p->key>k )

p->left = insert(p->left,k);

else

p->right = insert(p->right,k);

fixsize(p);

return p;

}

Node\*BinTree:: join(Node\* p, Node\* q) // РѕР±СЉРµРґРёРЅРµРЅРёРµ РґРІСѓС… РґРµСЂРµРІСЊРµРІ

{

if( !p ) return q;

if( !q ) return p;

if( rand()%(p->size+q->size) < p->size )

{

p->right = join(p->right,q);

fixsize(p);

return p;

}

else

{

q->left = join(p,q->left);

fixsize(q);

return q;

}

}

Node\*BinTree::remove(Node\* p, int k) // СѓРґР°Р»РµРЅРёРµ РёР· РґРµСЂРµРІР° p РїРµСЂРІРѕРіРѕ РЅР°Р№РґРµРЅРЅРѕРіРѕ СѓР·Р»Р° СЃ РєР»СЋС‡РѕРј k

{

if( !p ) return p;

if( p->key==k )

{

Node\* q = join(p->left,p->right);

delete p;

return q;

}

else if( k<p->key )

p->left = remove(p->left,k);

else

p->right = remove(p->right,k);

return p;

}

# Приложение Е Исходный код программы. main\_fun.h

#ifndef MAIN\_FUN\_H

#define MAIN\_FUN\_H

#include <bin\_struct.h>

int treePainter(QGraphicsScene \*scene, Node \*node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);

QGraphicsScene \*graphic(BinTree \*tree, QGraphicsScene \*scene, int depht);

#endif // MAIN\_FUN\_H

# Приложение Ж Исходный код программы. draw\_Tree.cpp

#include<bintree.h>

#include<functionstree.h>

#include<cmath>

QGraphicsScene \*graphic(BinTree \*tree, QGraphicsScene \*&scene, int depth)

{

if (tree == nullptr)

return scene;

scene->clear();

QPen pen;

QColor color;

color.setRgb(220, 220, 220);

pen.setColor(color);

QBrush brush (color);

QFont font;

font.setFamily("Tahoma");

pen.setWidth(3);

int wDeep = static\_cast<int>(pow(2, depth + 2));

int hDelta = 70;

int wDelta = 15;

font.setPointSize(wDelta);

int width = (wDelta\*wDeep)/2;

treePainter(scene, tree->Head, width/2, hDelta, wDelta, hDelta, pen, brush, font, wDeep);

return scene;

}

int treePainter(QGraphicsScene \*&scene, Node \*node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)

{

if ((node == nullptr) || (node->data == '^'))

return 0;

QString out;

out += node->data;

QGraphicsTextItem \*textItem = new QGraphicsTextItem;

textItem->setPos(w, h);

textItem->setPlainText(out);

textItem->setFont(font);

scene->addEllipse(w-wDelta/2, h, wDelta\*5/2, wDelta\*5/2, pen, brush);

if ((node->left != nullptr) && (node->left->data != '^') )

scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)\*wDelta+wDelta/2, h+hDelta+wDelta, pen);

if (node->right != nullptr)

scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)\*wDelta+wDelta/2, h+hDelta+wDelta, pen);

scene->addItem(textItem);

treePainter(scene, node->left, w-(depth/2)\*wDelta, h+hDelta, wDelta, hDelta, pen, brush, font, depth/2);

treePainter(scene, node->right, w+(depth/2)\*wDelta, h+hDelta, wDelta, hDelta, pen, brush, font, depth/2);

return 0;

}