**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №4**

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

Тема: Деревья

|  |  |  |
| --- | --- | --- |
| 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, изучить особенности ее реализации на языке программирования C++. Разработать программу, использующую бинарное дерево для обработки формулы.

## Задание.

Для заданного бинарного дерева b типа BT с произвольным типом элементов:

* определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;
* напечатать элементы из всех листьев дерева b;
* подсчитать число узлов на заданном уровне n дерева b (корень считать узлом 1-го уровня);

## Основные теоретические положения.

Дерево – конечное множество Т, состоящее из одного или более узлов, таких, что:

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m ³ 0 попарно не пересекающихся множествах Т1, Т2, ..., Тm, каждое из которых, в свою очередь, является деревом. Деревья Т1, Т2, ..., Тm называются поддеревьями данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое рекурсивное определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

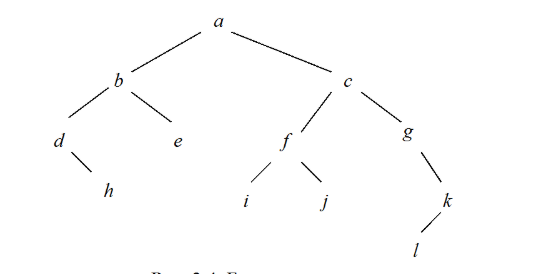


Рисунок 1 - Бинарное дерево

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом

## 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 нахождения дополнительных значений и поле, для их вывода, 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\_onPrint\_clicked() | Слот, отвечающий за считывание данных и графического и текстового выводов |
| on\_onLeavesonLevel\_clicked() | Слот, отвечающий за дополнительную обработку данных |

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

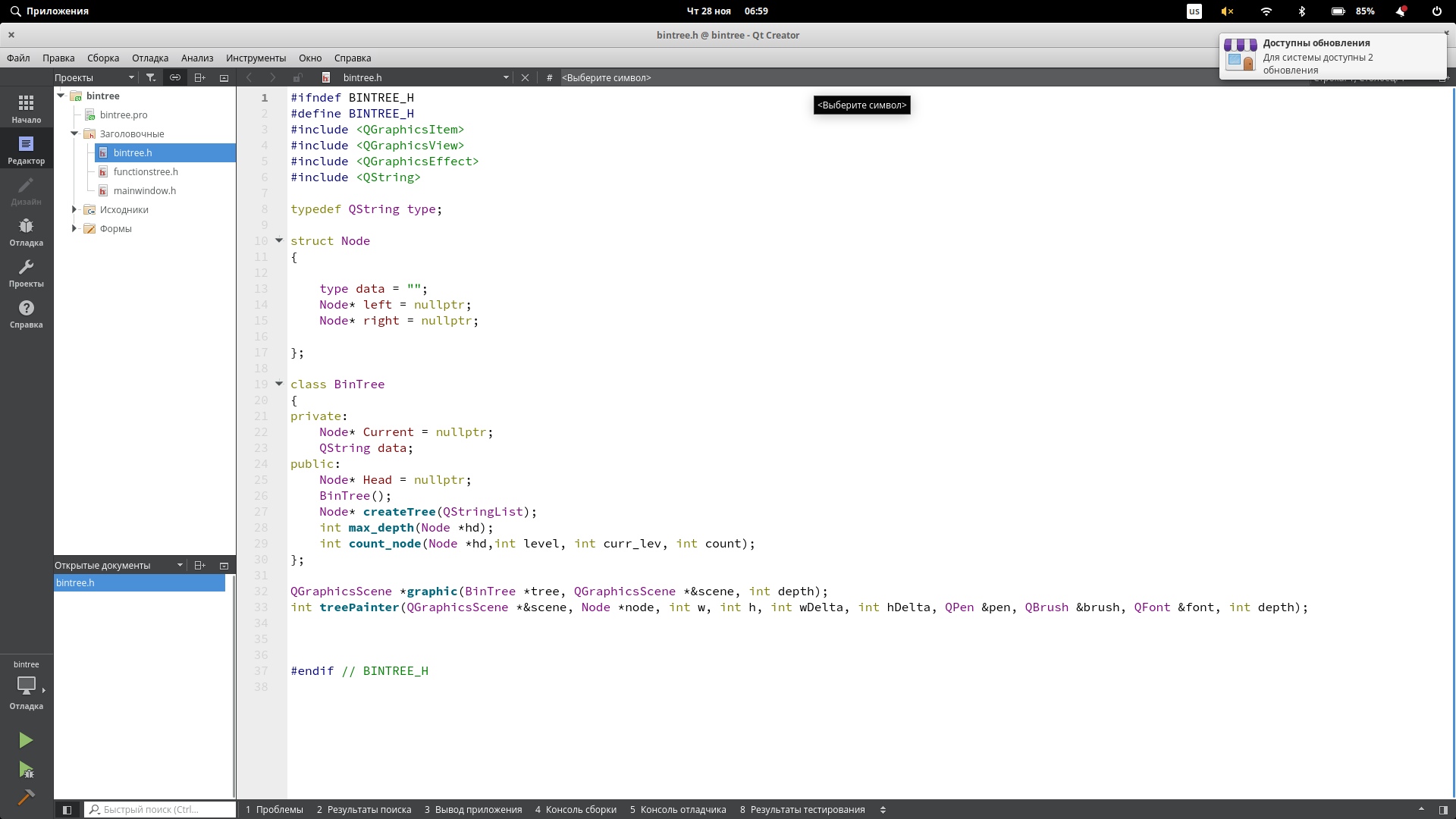


Рисунок 2 – Структуры бинарного дерева и узла

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\* createTree(QStringList,int\* count) | Создает бинарное дерево из массива строк-элементов, полученного из входной строки |
| int max\_depth(Node \*hd) | Возвращает максимальную глубину дерева |
| int count\_node(Node \*hd,int level, int curr\_lev, int count) | Возвращает количество узлов на заданном уровне |

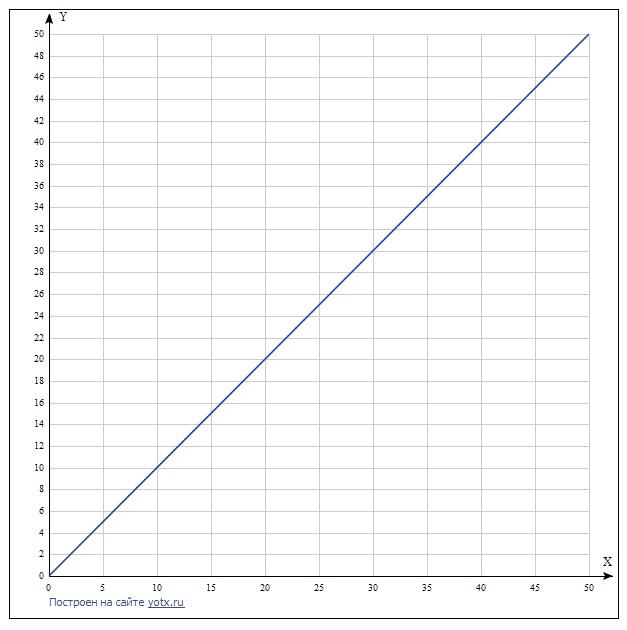
Программа имеет возможность графического отображения полученного бинарного дерева с помощью виджета QGraphicsView. Функции, необходимые для для графического представления дерева, представлены в табл. 3.

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 |
| QStringList mySplit(QString rowInput) | Обработка входной строки и преобразование её к массиву строк для дальнейшего создания бинарного дерева |

## Оценка сложности алгоритма.

Алгоритмы нахождения максимальной глубины и количества улов на заданном уровне являются рекурсивными, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма

**

## Тестирование программы.

Вид программы после выполнения представлен на рис. 3.

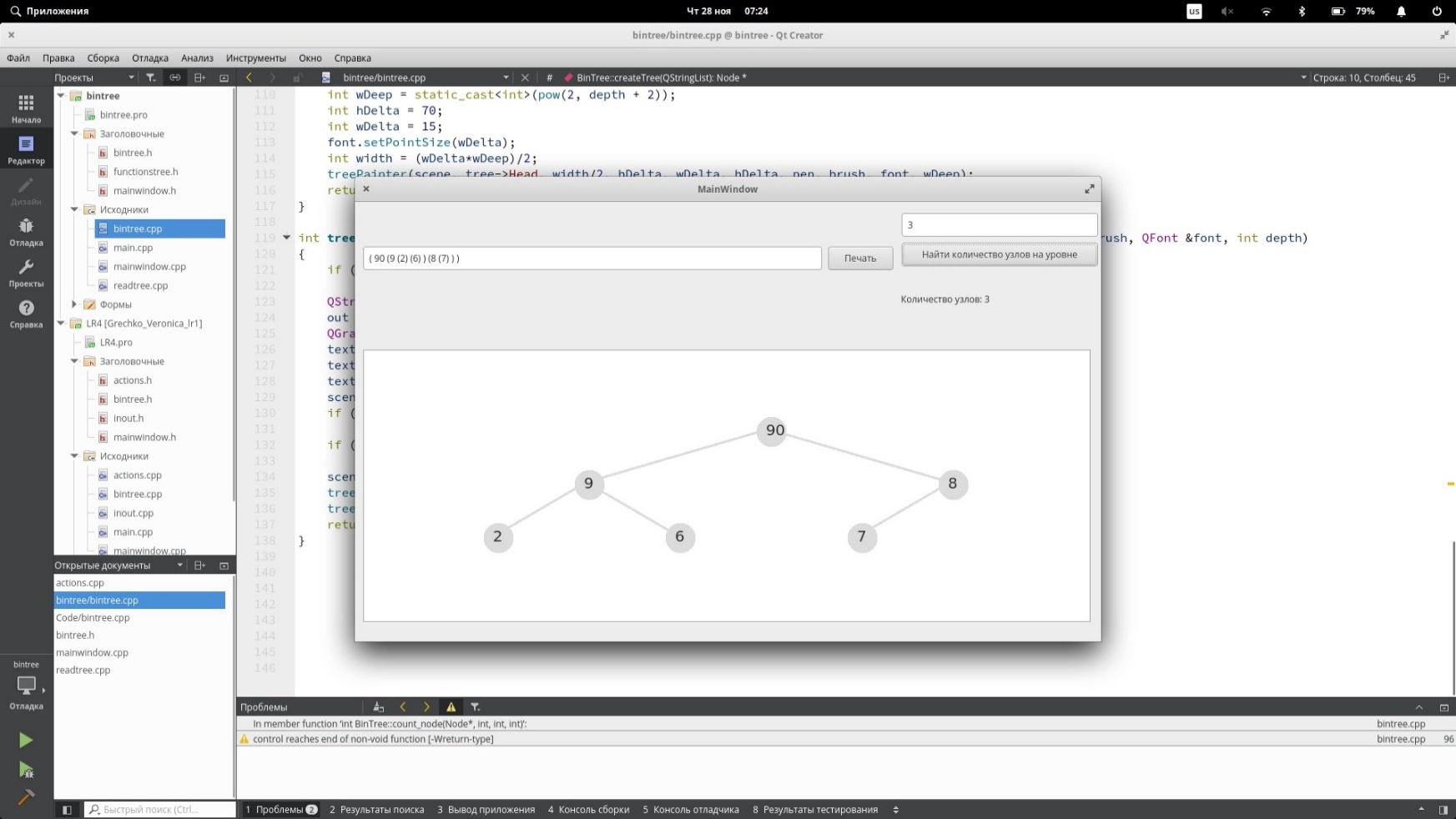


Рисунок 3 – Графический интерфейс программы

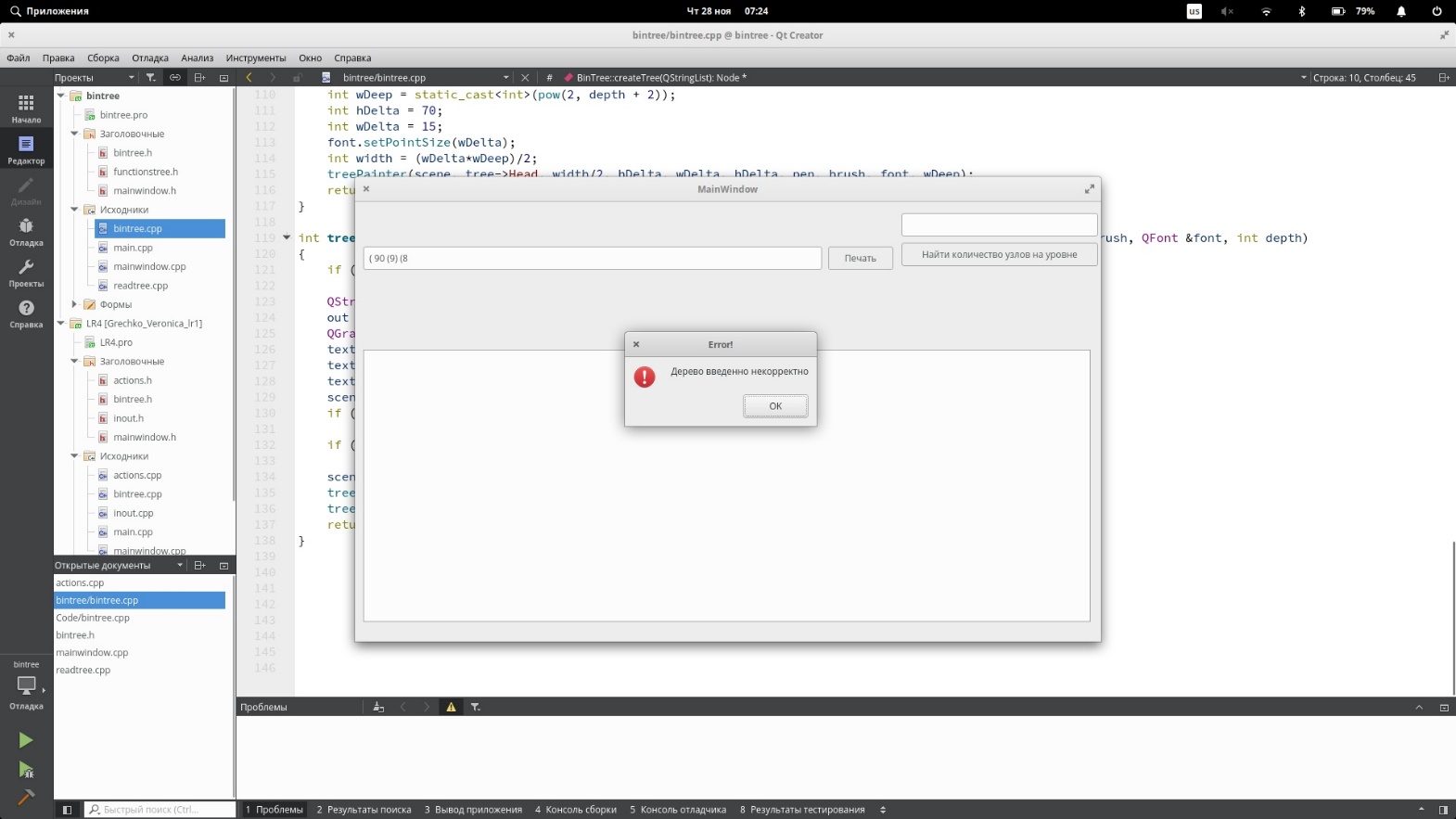
 Также был рассмотрен случай некорректно введённых данных на рис. 4.

Рисунок 4 – Ошибка ввода

Был проведен ряд тестов, проверяющих корректность работы программы. Результаты тестирование приведены в табл. 4.

Таблица 4 – Тестирование программы

|  |  |
| --- | --- |
| Входная строка | Вывод |
| ( 6 ( 4 (5) (9) ) (5 )) | Максимальная глубина: 2 |
| ( 4 (5 ) (5 | Дерево введено некорректно |
| ( 6 ( 4 (5) (9) ) (5 )) | Количество узлов: 2 |
| ( 6 ( 4 (5) (9) ) (5 (7) (9) )) | Количество узлов: 4 |

## Выводы.

В ходе выполнения лабораторной работы была написана программа, создающая бинарное дерево, подсчитывающая его максимальную глубину, а также находящая количество узлов на заданном уровне. Печать бинарного дерева выполняется графически.

# Приложение А Исходный код программы. MAIN.c

#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\_onPrint\_clicked();

void on\_onLeavesonLevel\_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 <functionstree.h>

#include <bintree.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\_onPrint\_clicked()

{

QString rinput = ui->lineEdit->text();

QString input = delSpace(rinput);

if (input != ""){

if (input[0] != '(' || input[input.size() - 1] != ')'){

QMessageBox::critical(this, "Error!", "Р”РµСЂРµРІРѕ РІРІРµРґРµРЅРЅРѕ РЅРµРєРѕСЂСЂРµРєС‚РЅРѕ");

return;

}

}

else {

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

return;

}

BinTree\* BT = new (BinTree);

int\* count = new int;

\*count = 0;

BT->Head = BT->createTree(mySplit(input), count);

QString out;

out.append("Р“Р»СѓР±РёРЅР° РґРµСЂРµРІР°: ");

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

out.append(QString::number(depth - 1));

out.append(" ");

out.append(QString::number(\*count));

ui->label->setText(out);

graphic(BT, scene,depth);

}

void MainWindow::on\_onLeavesonLevel\_clicked()

{

QString rinput = ui->lineEdit->text();

QString level\_i = ui->lineEdit\_2->text();

int level = level\_i.toInt();

QString input = delSpace(rinput);

if (input != ""){

}

else {

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

return;

}

BinTree\* BT = new (BinTree);

// BT->Head = BT->createTree(mySplit(input));

QString out;

out.append("РљРѕР»РёС‡РµСЃС‚РІРѕ СѓР·Р»РѕРІ: ");

int count = BT->count\_node(BT->Head,level, 1, 0);

out.append(QString::number(count));

ui->label->setText(out);

}

# Приложение Г Исходный код программы. FUNCTIONSTREE.H

#ifndef FUNCTIONSTREE\_H

#define FUNCTIONSTREE\_H

#include <bintree.h>

#include <QMessageBox>

#include <QString>

#include <QStringList>

QString delSpace (QString rowInput);

QStringList mySplit(QString rowInput);

#endif // FUNCTIONSTREE\_H

# Приложение Д Исходный код программы. FUNCTIONSTREE.cpp

#include<functionstree.h>

QString delSpace (QString rowInput){

QString out="";

for (auto i = 0;i < rowInput.length();i++){

if (rowInput[i]!=' ' && rowInput[i]!='\n' && rowInput[i]!='\t' )

out.push\_back(rowInput[i]);

}

return out;

}

QStringList mySplit(QString rowInput){

QString st = delSpace(rowInput);

auto i = 0;

QStringList out={};

QString tmp="";

for(;i<st.length();i++){

if (st[i] == ')' || st[i] == '(') out.push\_back(QString(st[i]));

else {

for (; st[i]!=')' && st[i]!='(' ;i++){

tmp.push\_back(st[i]);

}

out.push\_back(tmp);

out.push\_back(QString(st[i]));

tmp.clear();

}

}

return out;

}

# Приложение Е Исходный код программы. bintree.h

#ifndef BINTREE\_H

#define BINTREE\_H

#include <QGraphicsItem>

#include <QGraphicsView>

#include <QGraphicsEffect>

#include <QString>

typedef QString type;

struct Node

{

type data = "";

Node\* left = nullptr;

Node\* right = nullptr;

};

class BinTree

{

private:

Node\* Current = nullptr;

QString data;

public:

Node\* Head = nullptr;

BinTree();

Node\* createTree(QStringList,int\* count);

int max\_depth(Node \*hd);

int count\_node(Node \*hd,int level, int curr\_lev, int count);

};

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

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

#endif // BINTREE\_H

# Приложение Ж Исходный код программы. BInTree.cpp

#include<bintree.h>

#include<functionstree.h>

#include<cmath>

BinTree::BinTree(){

Head = new Node;

Head->data = "";

Current = Head;

}

Node\* BinTree::createTree(QStringList tokens, int\* count){

Node\* finalNode = new Node;

if(tokens.size()==2) return finalNode;

int i =1;

QString ltree = "";

QString rtree = "";

finalNode->data = tokens[i++];

int index\_i = i; /\* РРЅРґРµРєСЃ РѕС‚РєСЂС‹РІР°СЋС‰РµР№ СЃРєРѕР±РєРё Р»РµРІРѕРіРѕ РїРѕРґРґРµСЂРµРІР° \*/

\*count++;

if(tokens[i] == "("){

auto openBrackets = 1;

auto closeBrackets = 0;

while (openBrackets != closeBrackets) {

i++;

if (tokens[i] == "("){

openBrackets++;

}

else if (tokens[i] == ")"){

closeBrackets++;

}

}

for (;index\_i<=i; index\_i++){

ltree.append(tokens[index\_i]);

}

finalNode->left = createTree(mySplit(ltree), count);

i++;

if (tokens[i] == ")"){ /\* Р•СЃР»Рё РїСЂР°РІРѕРіРѕ РїРѕРґРґРµСЂРµРІР° РЅРµС‚ (РґРѕСЃС‚РёРіРЅСѓС‚ РєРѕРЅРµС† СЃС‚СЂРѕРєРё РїРѕСЃР»Рµ СЃС‚СЂСѓРєС‚СѓСЂС‹ Р»РµРІРѕРіРѕ РїРѕРґРґРµСЂРµРІР°\*/

return finalNode;

}

int index\_j = i; /\* РРЅРґРµРєСЃ РѕС‚РєСЂС‹РІР°СЋС‰РµР№ СЃРєРѕР±РєРё Р»РµРІРѕРіРѕ РїРѕРґРґРµСЂРµРІР° \*/

if(tokens[i] == "("){

auto openBrackets = 1;

auto closeBrackets = 0;

while (openBrackets != closeBrackets) {

i++;

if (tokens[i] == "("){

openBrackets++;

}

else if (tokens[i] == ")"){

closeBrackets++;

}

}

for (;index\_j<=i; index\_j++){

rtree.append(tokens[index\_j]);

}

finalNode->right = createTree(mySplit(rtree), count);

}

}

return finalNode;

}

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

if((hd == NULL) || (hd->data == '^')) 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::count\_node(Node \*hd, int depth, int cur, int count){

if(cur == depth){

if(hd->data == '^') return 0;

else return 1;

}

else {

if(hd->left == nullptr && hd->right != nullptr )

return count\_node(hd->right, depth, cur + 1, count);

else if (hd->right == nullptr && hd->left != nullptr )

return count\_node(hd->left, depth, cur + 1, count);

else if(hd->right != nullptr && hd->left != nullptr)

return count\_node(hd->left, depth, cur + 1, count) + count\_node(hd->right, depth, cur + 1, count);

}

}

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;

}