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 _____ Нгуен Ш. Х.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

Цель работы

Ознакомиться с основными характеристиками и особенностями такой структуры данных, как бинарное дерево, изучить особенности ее реализации на языке программирования C++. Разработать программу, которая строит изображение леса и бинарного дерева.

Задание

Для заданного леса с произвольным типом элементов:

- получить естественное представление леса бинарным деревом;
- вывести изображение бинарного дерева;
- перечислить элементы леса в горизонтальном порядке (в ширину).

Выполнение

Написание работы производилось на базе операционной системы Windows 10 в среде разработки QtCreator с использованием фреймворка Qt. Сборка, отладка производились в QtCreator, запуск программы осуществлялся через командную строку. Исходные коды файлов программы представлены в приложениях А-М.

Для реализации программы был разработан графический интерфейс с помощью встроенного в QtCreator UI-редактора. Он представляет из себя поле ввода, кнопку считывания, поле вывода с возможностью графического отображения результата.

Структура леса :

```
struct Node{
    string info;
    int total;
    Node **Tnode;
};
struct Tree{
    int deep;
    Node *root;
```

};

struct Forest{

T

r

e Также были реализованы функции, создающие лес из входной строки и

подключены к серверной части через файл forest.cpp, использующий

библиотеку forest.h, приведены в табл.1

r	**Функция**	**Назначение**
e	Forest *takeforest(string str, int len);	Создать лес из входной строки.
i e	Node *AppendNode(Node *root,string	Создать новый узел и добавить это
n	str, int &index,int len,int total);	у
t	int CountTree(string str, int len);	Подсчитывает количество деревьев,
c		которые будут создать из входной
o		строки.
u	int CheckErr(string str, int len);	Возвращает значение ошибки
n		FlagErr.
t	int BranchOfNode(string str, int index,	Подсчитывает количество ветвей для
	int len);	узла.
	Forest *createForest(int count);	Создать лес с count деревьев.
	Tree *createTree();	Создать новое дерево.
	Node *createNode(string info, int	Создать новый узел
	total);	

int DeepOfTree(string str, int index, int len);	Возвращает глубину дерева
void takeInfoOfNode(Node *root, string &out);	Получить информацию о дереве через корень root дерева и сохраняются в string out.
string takeString(string str,int &index,int len);	Возвращает string info узла бинарного дерева и index соответствует с местом следующего элемента.

Таблица 1– Основные функции создания лес из входной строки

Далее получим естественное представление леса бинарным деревом через функции, приведены в табл.2 с структурой бинарного дерева :

```
struct BinNode
{
   string info;
   BinNode *left;
   BinNode *right;
};

struct BinTree
{
   BinNode *root;
   int deep;
};
```

Функция	Назначение
BinTree *createBinTree();	Создать новое бинарное дерево.
BinNode *createBinNode(string info);	Создать новый бинарный узел.
BinTree *createBTFromForest(Forest* forest);	Создать бинарное дерево из леса.
BinNode *ConsBT(BinNode *root, Forest *left, Forest *right);	Создать полный бинарный узел.
Tree *Head(Forest *forest);	Возвращает первое дерево леса.
BinNode *Root(Tree* tree);	Возвращает корень root любого дерева.
Forest *Listing(Tree *tree);	Создать лес деревьев из корневых узлов корня root.
Tree *NodeToTree(Node *node);	Превращает структуру узла в дерево.
int CountDeepOfTree(Node* root);	Возвращает глубину дерева.
void takeInfoBT(BinNode *root, string &out);	Возвращает информацию о дереве и сохраняется в string out.
void takeInfoByDeep(Node *root,int deep,string &out);	Возвращает информацию о дереве и хранится в string out в зависимости от глубины дерева deep.

табл.2 - Основные функции создания бинарного дерева

Оценка эффективности алгоритма

Алгоритм создания бинарного дерева по строке является итеративным, каждый элемент строки обрабатывается один раз, а значит сложность алгоритма можно оценить как O(N).

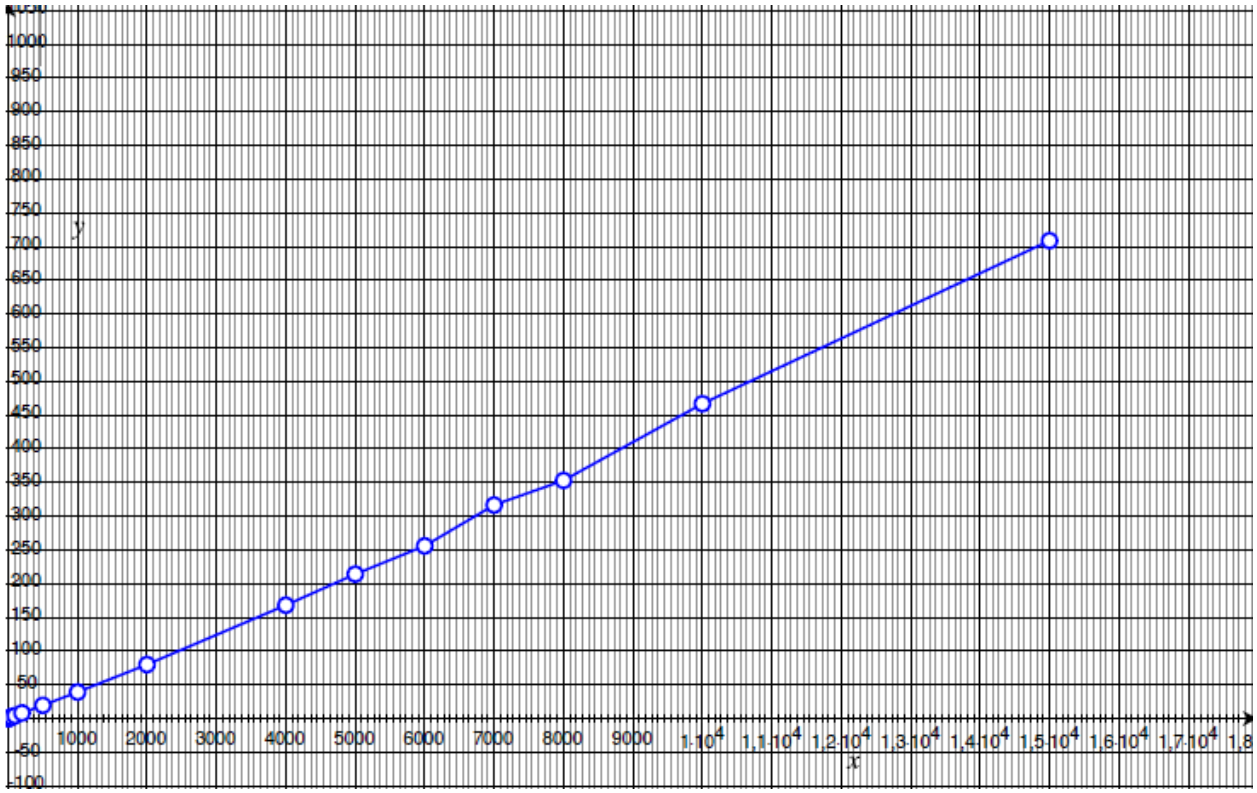


График 1 — Зависимость количества элементов к времени строительства

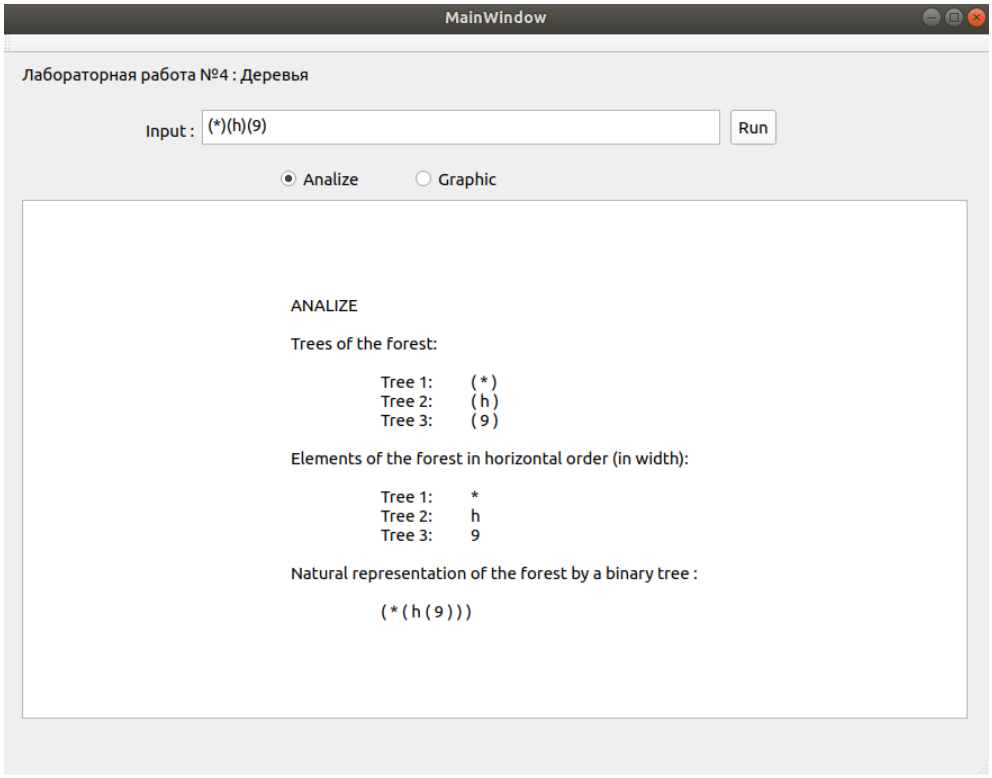
Алгоритм вывода элементов дерева в горизонтальном порядке является рекурсивным, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма также .

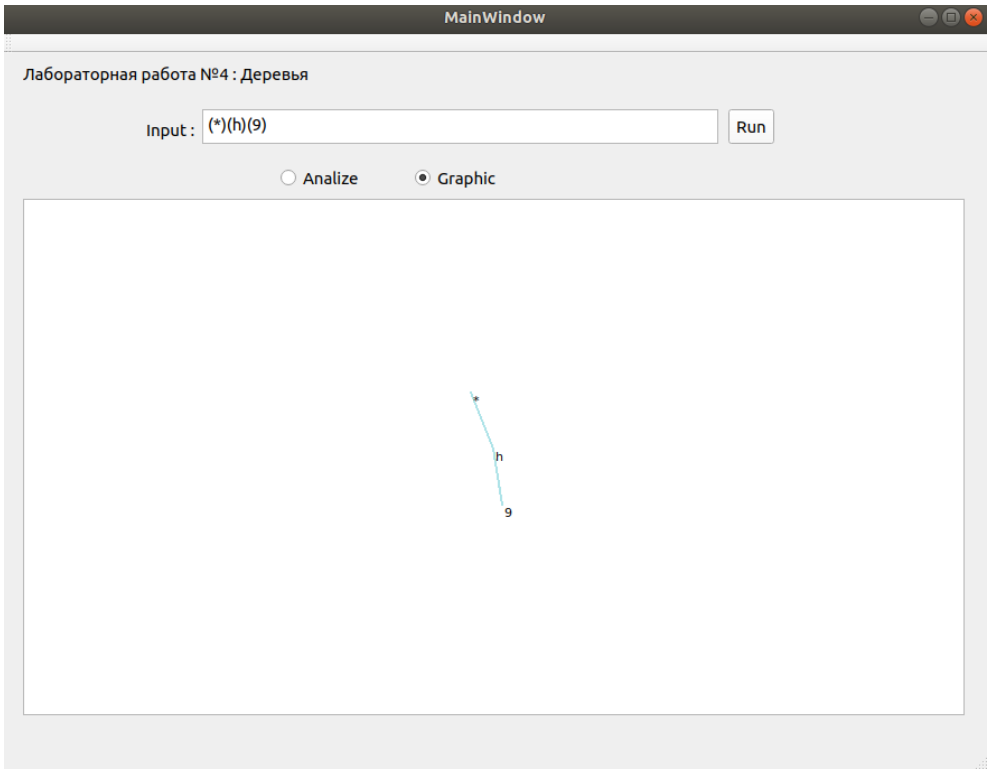
Выводы

В ходе выполнения лабораторной работы была написана программа, создающая естественное представление леса бинарным деревом, таже изображение бинарного дерева и перечислить элементы леса в горизонтальном порядке (в ширину).

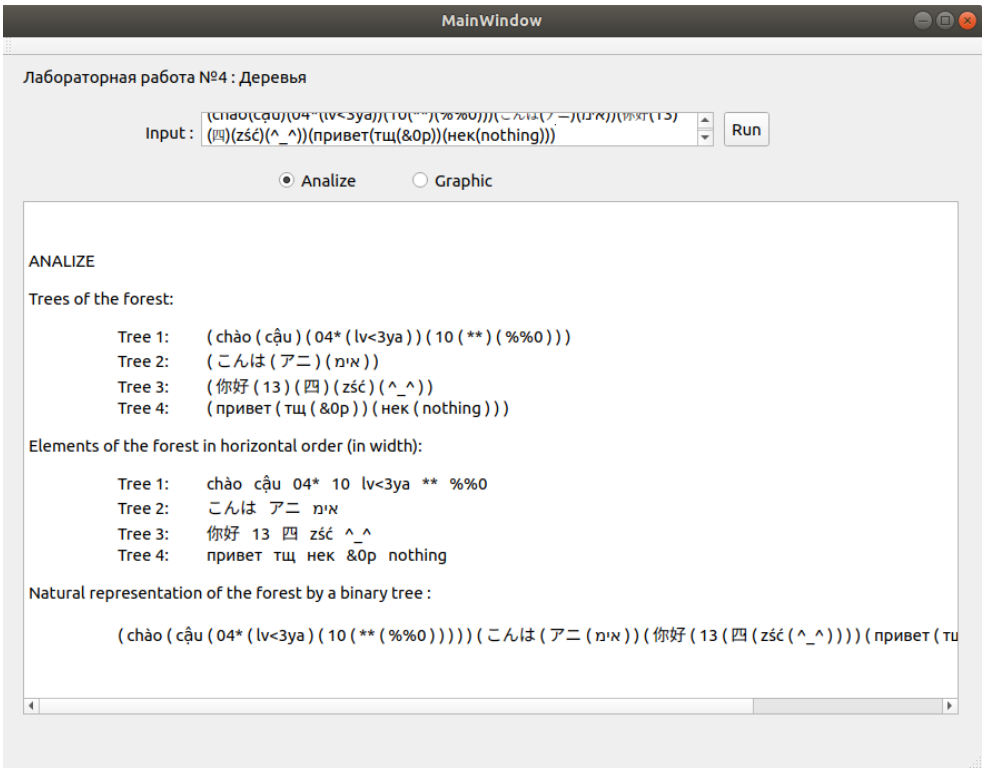
Тестирование программы

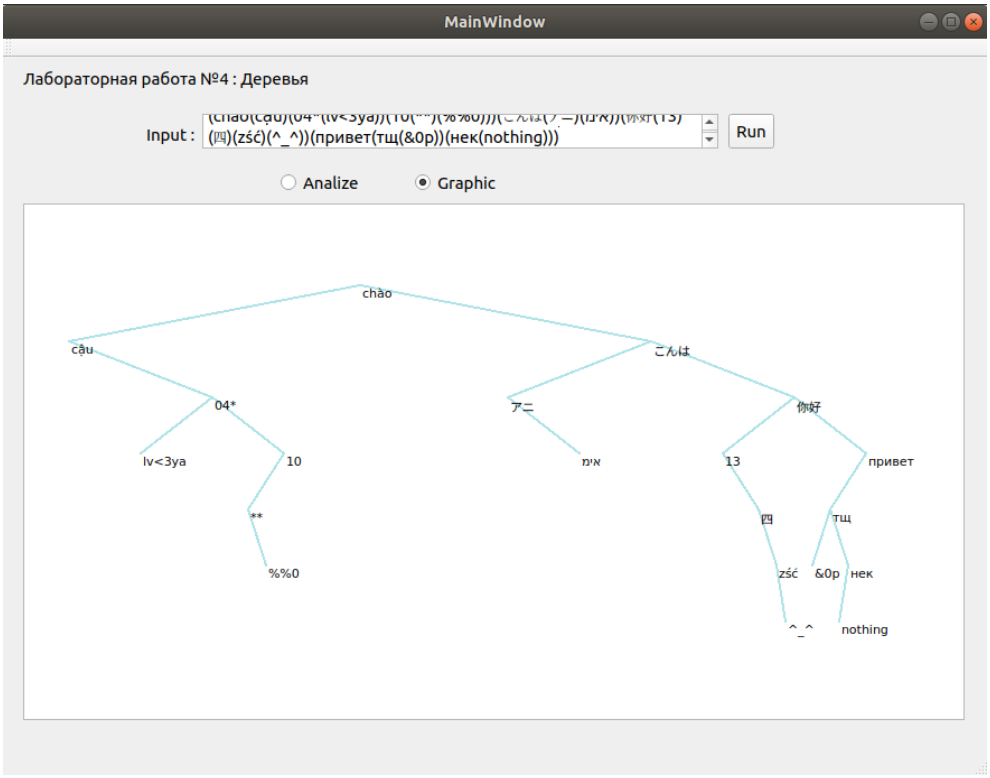
1. Лес : (*)(h)(9)



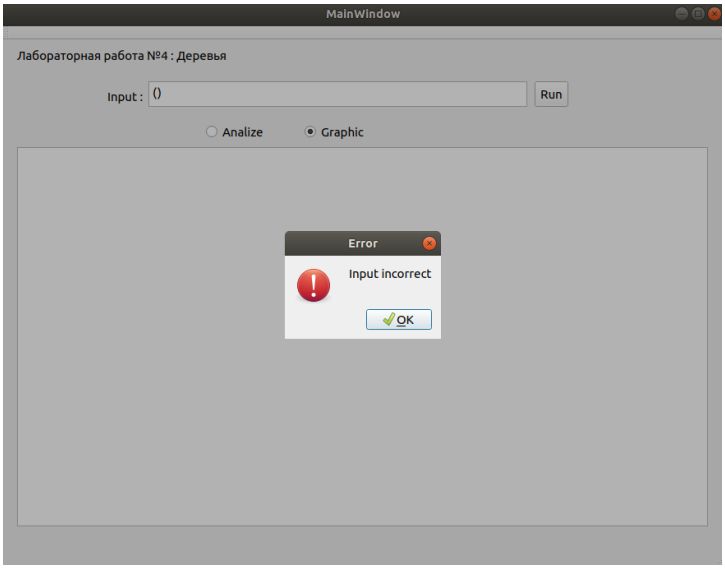


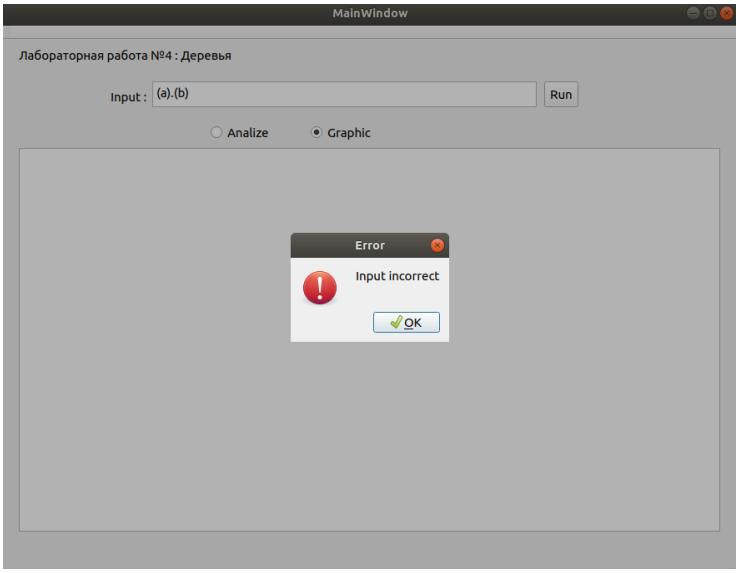
2. Лес : (chào(cậu)(04*(lv<3ya))(10(**)(%%0)))(こんは(アニ)(אימ))(你好(13)(四)(zść)(^_^))(привет(тщ(&0p))(нек(nothing)))

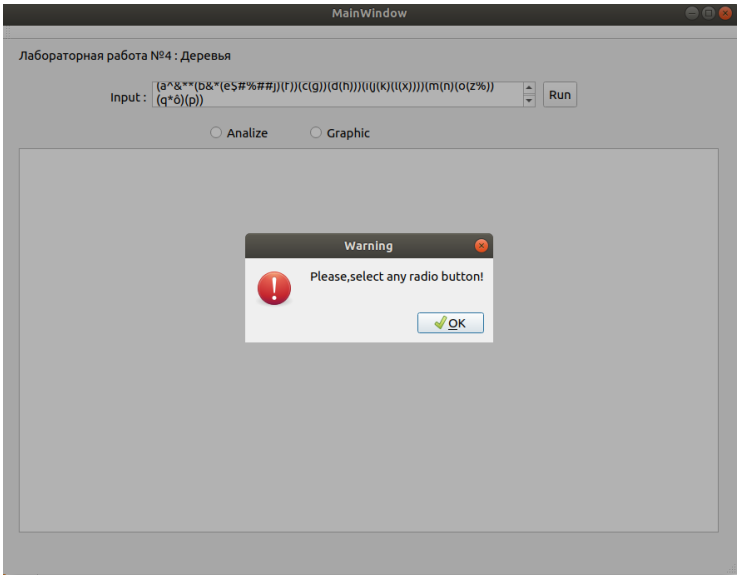




3. Ошибки


MainWindow
Лабораторная работа №4 : Деревья
Input :
()
Run
Analize
Graphic
Error
Input incorrect
OK


MainWindow
Лабораторная работа №4 : Деревья
Input :
(a).(b)
Run
Analize
Graphic
Error
Input incorrect
OK


MainWindow
Лабораторная работа №4 : Деревья
Input :
((a^&**(b&*(eş#%##j)(r))(c(g))(d(h)))(i(j(k)(l(x))))(m(n)(o(z%))
(q*ô)(p))
Run
Analize
Graphic
Warning
Please,select any radio button!
OK

Приложение А

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Файл Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <console.h>
#include <QMessageBox>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    console *Console;
```

```
    bool flagCase_1 = false;
    bool flagCase_2 = false;
    int flagErr;

private slots:
    void on_pushButton_clicked();


    void on_radioButton_2_clicked(bool checked);

    void on_radioButton_clicked(bool checked);


private:
    Ui::MainWindow *ui;
    QGraphicsView *scene;
};

#endif // MAINWINDOW_H
```

Файл Mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    Console = new console;

}

MainWindow::~MainWindow()
```

```
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{
    QString data = ui->textEdit->toPlainText();
    QGraphicsScene *scene = new QGraphicsScene(ui->graphicsView);
    scene->clear();
    flagErr = Console->checkErr(data);

    if(flagErr == 1){
        QMessageBox::warning(this,"Error","Input incorrect");
    }
    else if(flagErr == 2){
        QMessageBox::warning(this,"Error","              Input incorrect.\nCannot be a space between the brackets.");
    }
    else if(flagErr == 3){
        QMessageBox::warning(this,"Error","Input is not alpharic");
    }

    else if (flagErr == 0) {
        if(flagCase_1 == false && flagCase_2 == false)
            QMessageBox::warning(this,"Warning","Please,select any radio button!");
        else if(flagCase_2 == true){
            scene = Console->Console( data);
            ui->graphicsView->setScene(scene);
        }
        else if(flagCase_1 == true){
            scene = Console->Analize(data);
            ui->graphicsView->setScene(scene);
        }
    }
```

```
}


void MainWindow::on_radioButton_2_clicked(bool checked)
{
    flagCase_2 = checked;
    flagCase_1 = false;
}

void MainWindow::on_radioButton_clicked(bool checked)
{
    flagCase_2 = false;
    flagCase_1 = checked;
}
```

Файл forest.h

```
#include <iostream>
#include <string>
#include <ctype.h>
#include <fstream>
#include <cstring>

using namespace std;


struct Node{
        string info;
        int total;
        Node **Tnode;
};
struct Tree{
        int deep;
```

```
        Node *root;
};

struct Forest{
        int count;
        Tree **tree;
};


Forest *takeforest(string str, int len);
Forest *createForest(int count);
Tree *createTree();
Node *AppendNode(Node *root,string str, int &index,int len,int total);
Node *createNode(string info, int total);
string takeString(string str,int &index,int len);
int BranchOfNode(string str, int index, int len);
int DeepOfTree(string str, int index, int len);
int CountTree(string str, int len);
int CheckErr(string str, int len);
void takeInfoByDeep(Node *root,int deep);
void takeInfoOfNode(Node *root, string &out);
```

Файл forest.cpp

```
#include "forest.h"

Forest *takeforest(string str, int len){
    int countTree = CountTree(str,len);
    Forest *forest = createForest(countTree);//создать пустый лес
    int index = 0;
    for(int i = 0; i < forest->count; i++){
        int deep = DeepOfTree(str,index,len);
        string rootInfo = takeString(str,index,len);
        int total;
        if(deep == 0)
```

```
            total = 0;// Neu deepTree = 0 thi sau khi doc ham rootInfo index = vi tri dau tien cua Tree tiep theo
        else
            total = BranchOfNode(str,index,len);

        Node *root = createNode(rootInfo,total);
        AppendNode(root,str,index,len,total);
        forest->tree[i]->root = root;
        forest->tree[i]->deep = deep;
    }
    return forest;
}

Node *AppendNode(Node *root,string str, int &index,int len,int total){// index = vi tri dau tien cua cay con
    if(!total)
        return root;
    for(int i = 0; i < total; i++){
        int deep = DeepOfTree(str,index,len);
            string rootInfo = takeString(str,index,len);
            int total2;
            if(deep == 0)
                total2 = 0;
            else
                total2 = BranchOfNode(str,index,len);
            Node *temp = createNode(rootInfo,total2);
        AppendNode(temp,str,index,len,total2);
        root->Tnode[i] = temp;
        if(i == total -1)
            index++;
    }
    return root;
}
```

```
Node *createNode(string info, int total){
    Node *node = new Node;
    node->info = info;
    node->total = total;
    if(!total)
        node->Tnode = nullptr;
    node->Tnode = new (Node*);
    for(int j = 0; j < total; j++){
            node->Tnode[j] = new Node;
            node->Tnode[j] = nullptr;
        }
        return node;
}

int BranchOfNode(string str, int index, int len){//index = vi tri dau tien cua cay con
    int Delta = 0;
    int branch = 0;
    int i;
    for(i = index; i <len; i++){
        if(str[i] == '('
            Delta++;
        else if(str[i] == ')')
            Delta--;
        if(Delta == 0)
            branch++;
        else if(Delta < 0)
            return branch;
    }
}

string takeString(string str,int &index,int len){
    string temp;
    int flag = index;
    for(index; index < len; index++){
```

```
        if(str[index] == '('){
            if(flag == index)
                continue;
            else
                        return temp;
        }
        else if(str[index] == ')'){ //  Lưu ý rằng đã kiểm tra lỗi empty :()
            index++;
            return temp;
        }
        else{
            temp.append(1,str[index]);
        }
    }
}

/*checked*/
int DeepOfTree(string str, int index, int len){// index = vi tri dau tien cua cay
        int Delta = 0;
        int deep = 0;
        for(int i = index; i < len; i++){
                if(str[i] == '('){
                        Delta += 1;
                        deep = Delta>deep?Delta:deep; // độ sâu  = max của delta - 1
                }
                else if(str[i] == ')')
                        Delta -= 1;
                if(Delta == 0)
                        return deep-1;
        }
}

Forest *createForest(int count){
    Forest *forest = new Forest;
```

```
        forest->count = count;
        forest->tree = new (Tree*);
        for(int i = 0; i<count; i++){
                forest->tree[i] = createTree();
        }
        return forest;
}

Tree *createTree(){
        Tree *tree = new Tree;
        tree->root = nullptr;
        tree->deep = 0;
        return tree;
}


int CheckErr(string str, int len){
    int flagErr = 0;
    int Delta = 0;
    bool flagindex = false;

    for(int i = 0; i < len-2; i++){
        if( (str[i] == '(' || str[i] == ')') && str[i+1] == ' ' && (str[i+2] == '(' || str[i+2] == ')'))
            return 2;
    }

    for(int i = 0; i < len; i++){
        if(str[i] == '('
            Delta++;
        else if (str[i] == ')') {
            Delta--;
        }
        if(Delta<0)
            return 1;// thua hoac thieu dau ( va )
```

```
        if(Delta > 0)
            flagindex = true;
    }
    if(Delta != 0 || flagindex == false)
        return 1;

    for(int i = 0; i < len-1; i++){
        if( (str[i] == '(' && str[i+1] == ')')
         || (str[i] == ')' && (str[i+1] != ')' && str[i+1] != '(')))
            return 1;
    }

    return flagErr;
}


/*checked*/
int CountTree(string str, int len){
    int countTree = 0;
    int Delta = 0;// Hieu cua '(' va ')'
    for(int i = 0; i < len; i++){
        if(str[i] == '(')
            Delta++;
        else if(str[i] == ')')
            Delta--;
        if(Delta == 0)
            countTree++;
    }
    return countTree;
}

void takeInfoOfNode(Node *root, string &out){
        out += "( ";
        out += root->info;
```

```
        out+=" ";
        for(int i = 0; i < root->total; i++){
                takeInfoOfNode(root->Tnode[i],out);
        }
        out += ") ";
}
```

Файл bintree.h

```
#include "forest.h"


struct BinNode
{
   string info;
   BinNode *left;
   BinNode *right;
};

struct BinTree
{
   BinNode *root;
   int deep;
};

BinTree *createBinTree();
BinNode *createBinNode(string info);
BinTree *createBTFromForest(Forest* forest);

BinNode *ConsBT(BinNode *root, Forest *left, Forest *right);
Tree *Head(Forest *forest);
Forest *Tail(Forest *forest);
BinNode *Root(Tree* tree);
Forest *Listing(Tree *tree);
Tree *NodeToTree(Node *node);
```

```
int CountDeep(BinNode *&node);
int CountDeepOfTree(Node* root);
void takeInfoBT(BinNode *root, string &out);
void takeInfoByDeep(Node *root,int deep,string &out);
```

Файл bintree.cpp

```
#include "bintree.h"



BinTree *createBinTree()
{
   BinTree *tree = new BinTree;
   tree->root = nullptr;
   tree->deep = 0;
   return tree;
}

BinNode *createBinNode(string info)
{
   BinNode *node = new BinNode;
   node->info = info;
   node->left = nullptr;
   node->right = nullptr;
   return node;
}

BinTree *createBTFromForest(Forest* forest){
      BinTree *bintree = createBinTree();// checked
      bintree->root = ConsBT(Root(Head(forest)),Listing(Head(forest)),Tail(forest));
      bintree->deep = CountDeep(bintree->root);
      return bintree;
}

Tree *Head(Forest *forest){
```

```
        if(forest == nullptr)// co the dong nay ko can vi forest luon != null
            return nullptr;
       if(forest->count == 0)
        return nullptr;
        Tree *tree = new Tree;
        tree = forest->tree[0];// da bao gom deep
        return tree;
    }

    Forest *Tail(Forest *forest){
        if(forest->count == 1 || forest == nullptr)
            return nullptr;
        Forest *temp = createForest((forest->count)-1); // đã khởi tạo forest cùng
forest->count
        for(int i = 0; i < (forest->count)-1; i++){
            temp->tree[i] = forest->tree[i+1];
        }
        return temp;
    }

    BinNode *Root(Tree* tree){// create Binnode from tree;
        if(tree == nullptr)
            return nullptr;
        BinNode *root = createBinNode(tree->root->info);
        return root;
    }

    Forest *Listing(Tree *tree){
        if(tree->deep == 0 || tree == nullptr) //
            return nullptr;
        Forest *forest = createForest((tree->root->total));
        for(int i = 0; i < forest->count; i++){
            forest->tree[i] = NodeToTree(tree->root->Tnode[i]);//da tao deepoftree
        }
```

```
        return forest;
    }

    Tree *NodeToTree(Node *node){
        Tree *tree = createTree();
        tree->root = node;
        tree->deep = CountDeepOfTree(node) - 1;
        return tree;
    }

    BinNode *ConsBT(BinNode *root, Forest *left, Forest *right){//tạo binnote* root  từ
treenote thong qua ham createbinnode;
      if(left == nullptr && right == nullptr){
        return root;
      }
      if(root == nullptr)
        return root;

        BinNode *binnode = root;
        if(left == nullptr){
            binnode->left = nullptr;
      }
        else{
            binnode->left = ConsBT(Root(Head(left)),Listing(Head(left)),Tail(left));
      }

        if(right == nullptr){
            binnode->right = nullptr;
      }

        else{
            binnode->right = ConsBT(Root(Head(right)),Listing(Head(right)),Tail(right));
      }
```

```
        return binnode;
}

int CountDeep(BinNode *&node)
{
    if (node == nullptr)
        return 0;
    int cl = CountDeep(node->left);
    int cr = CountDeep(node->right);
    return 1 + ((cl>cr)?cl:cr);
}

/*checked*/
int CountDeepOfTree(Node* root){
        if(root == nullptr)
                return 0;
        int max = 0;
        for(int i = 0; i < root->total; i++){
                int count = CountDeepOfTree(root->Tnode[i]);
                max = count>max?count:max;
        }
        return max+1;
}


void takeInfoBT(BinNode *root, string &out){
    if(root == nullptr)
        return;
    out += "( ";
    out += root->info;
    out+=" ";
    takeInfoBT(root->left,out);
    takeInfoBT(root->right,out);
    out += ") ";
```

```
}

void takeInfoByDeep(Node *root,int deep,string &out){// dieu kien la deep <=
DeepOfTree
    if(deep == 0){
        out += root->info;
        out+= "   ";
        return;
    }
    if(root->total == 0)
        return;

    for(int i = 0; i<root->total; i++){
        takeInfoByDeep(root->Tnode[i],deep-1,out);
    }
    return;
}
```

Файл console.h

```
#ifndef CONSOLE_H
#define CONSOLE_H

#include "bintree.h"
#include <QGraphicsScene>
#include <QGraphicsView>
#include <cmath>
#include <QGraphicsTextItem>
#include <QString>
#include <QMessageBox>
#include <QLineEdit>


class console
{
```

```
public:
    console();
    QGraphicsScene *Console(QString data);
    QGraphicsScene *Analize(QString data);
    int checkErr(QString data);
    void treePainter(QGraphicsScene *&scene, BinNode *binnode, int w, int h, int wDelta,
int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);

};

#endif // CONSOLE_H
```

Файл console.cpp

```
#include "console.h"

console::console()
{

}
int console::checkErr(QString data){
    int flagErr;
    string input = data.toStdString();
    flagErr = CheckErr(input,input.length());
    return flagErr;
}
QGraphicsScene* console::Analize(QString data){
    QGraphicsScene *scene = new QGraphicsScene;

    string input = data.toStdString();
    Forest *forest = takeforest(input,input.length());
    BinTree *bintree = createBTFromForest(forest);
    string out;
    out += "ANALIZE";
```

```
    out += "\n\nTrees of the forest:\n";
    for(int i = 0; i <forest->count; i++){
        out += "\n\tTree " + std::to_string(i+1)+":\t";
        takeInfoOfNode(forest->tree[i]->root,out);



    }
    out += "\n\nElements of the forest in horizontal order (in width):\n\t";
    for(int i = 0; i <forest->count; i++){
        out += "\n\tTree " + std::to_string(i+1)+":\t";
        for(int j = 0; j <= forest->tree[i]->deep; j++){
            takeInfoByDeep(forest->tree[i]->root,j,out);
        }
    }
    out+= "\n\nNatural representation of the forest by a binary tree :\n\n\t";

    takeInfoBT(bintree->root,out);
    scene->addText(QString::fromStdString(out));
    return scene;
}

QGraphicsScene* console::Console(QString data){
    QGraphicsScene *scene = new QGraphicsScene;

    string input = data.toStdString();



    Forest *forest = takeforest(input,input.length()
                                );
    BinTree *bintree = createBTFromForest(forest);
    scene->clear();
    QPen pen;
    QColor color;
    color.setRgb(174, 227, 232);
```

```
            pen.setColor(color);
            QBrush brush (color);
            QFont font("Helvetica [Cronyx]", 8, 10,false);
            pen.setWidth(2);


            int wDeep = static_cast<int>(pow(2, bintree->deep)+2);
            int hDelta = 50;
            int wDelta = 4;
            int width = (wDelta*wDeep)/2;
            treePainter(scene, bintree->root, width/2, hDelta, wDelta, hDelta, pen, brush, font,
wDeep);

            return scene;


        }


        void console::treePainter(QGraphicsScene *&scene, BinNode *binnode, int w, int h, int
wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)
        {

            if (binnode == nullptr)
                return ;
            string out;
            out += binnode->info;
            QGraphicsTextItem *textItem = new QGraphicsTextItem;
            textItem->setPos(w, h);// set toa do (x;y) cua nut
            textItem->setPlainText(QString::fromStdString(out));
            textItem->setFont(font);
            //scene->addEllipse(w-wDelta/2, h, wDelta*5/2, wDelta*5/2, pen, brush); // Tạo hình
tròn của các nút
            if (binnode->left != nullptr)
```

```
            scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
        if (binnode->right != nullptr)
            scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
        scene->addItem(textItem);
        treePainter(scene, binnode->left, w-(depth/2)*wDelta, h+hDelta, wDelta, hDelta, pen,
brush, font, depth/2);
        treePainter(scene, binnode->right, w+(depth/2)*wDelta, h+hDelta, wDelta, hDelta,
pen, brush, font, depth/2);
        return ;
    }
```