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 _____ Лисок М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Изучить основные характеристики и реализовать структуру данных бинарное дерево (англ. *Binary tree*). Создать программу, выполняющую визуализацию заданного арифметическим выражением дерева и выполняющую с ним заданные преобразования, описанные в постановке задачи.

Постановка задачи.

- если в дереве-формуле t терминалами являются только цифры, то вычислить (как целое число) значение дерева-формулы t;
- вычислить производную дерева-формулы t по заданной переменной.
- построить дерево-формулу t_1

Основные теоретические положения.

Двоичное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. Двоичное дерево не является упорядоченным ориентированным деревом.

Существует следующее рекурсивное определение двоичного дерева:

```
<дерево> ::= ( <данные> <дерево> <дерево> ) | nil .
```

Арифметическое выражение с бинарными операциями можно представить в виде бинарного дерева. Пусть, например, дано арифметическое выражение в инфиксной записи: $(a + b) * c \; d / (e + f * g)$. На рис. 1 представлено соответствующее ему бинарное дерево.

Рисунок 1 - Бинарное дерево, представляющее выражение

Тогда три варианта обхода этого дерева порождают три формы записи арифметического выражения:

1) КЛП – префиксную запись

$* + a\ b\ c\ /\ d + e\ * f\ g$;

2) ЛКП – инфиксную запись (без скобок, необходимых для задания последовательности выполнения операций)

$a + b * c\ \ d / e + f * g$;

3) ЛПК – постфиксную запись

$a\ b + c * d\ e\ f\ g * + /$.

Выполнение работы.

Написание работы производилось на базе операционной системы Mac OS в среде разработки QtCreator с использованием фреймворка Qt. Сборка, отладка производились в QtCreator при помощи отладчика gdb 8.3.1 и компилятора g++.

Функции файла infixtoposfix предназначены для перевода, введенного пользователем выражения в постфиксную форму записи, необходимую для построения бинарного дерева.

В классах Node и BinTree описана структура бинарного дерева и непосредственные методы работы с ним, описание которых представлено в табл. 1.

Таблица 1 – методы, описанные в заголовочном файле btree.h

Прототип функции	Действие, которое выполняет функция
struct Node* constructTree(string & postfix);	Метод, который строит дерево-формулу по выражению, представленному в постфиксной форме
int eval(struct Node* root, string &);	Метод, подсчитывающий значения выражения, на основе, которого построено дерево-формула
void calc(char value, Stack<char>& stack, vector<string>&, string&);	Метод, сохранеющий шаги подсчета значения выражения, на основе, которого построено дерево-формула
string Diff (struct Node *root, char var);	Метод, вычисляющий производную дерева-формулы по заданной переменной
void simplifying(string & str);	Метод, упрощающий производную
int countDeep(struct Node *&node);	Метод, подсчитывающий глубину дерева
string inorder(struct Node *t);	Метод, обходящий дерево, как ЛКП
void postorder(struct Node *t, Stack<char>&, vector<string>&,string &);	Метод, обходящий дерево, как ЛПК и вызывающий метод calc()

Файл drawing.cpp содержит функции, выполняющие работу по отрисовке графического представления на QGraphicsScene через данные, получаемые от функциональных алгоритмов. Описание представлено в табл. 2.

Таблица 2 - Функции, связующие графический интерфейс и алгоритмы

Прототип функции	Действие, которое выполняет функция
void graphic(struct Node* tree, QGraphicsScene *&scene);	По заданному бинарному дереву выполняет рисование в объекте QGraphicsScene
int treePainter(QGraphicsScene *&scene, struct 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

Файл Stack.h содержит функции для выполнения различных операций с исходным выражением и построенным на его основе деревом: перевод в постфиксную форму, подсчета значения выражения в дереве – описание представлено в табл. 3.

Таблица 3 – Методы работы с выражением и слоты взаимодействия с UI

Прототип функции	Действие, которое выполняет функция
void on_resultButton_clicked();	Слот, отвечающий за подсчет значения выражения, на основе, которого построено дерево-формула
void on_step-Button_clicked();	Слот, отвечающий за пошаговый режим подсчета значения выражения, на основе, которого построено дерево-формула
void on_deriative-Button_clicked();	Слот, отвечающий за вычисление производной дерева-формулы по заданной переменной

void on_file-Output_clicked();	Слот, отвечающий за запись из поля вывода в файл
void on_fileInput_clicked();	Слот, отвечающий за считывание из файла
void on_draw_clicked();	Слот, отвечающий за отображение дерева
void on_stepDraw_clicked();	Слот, отвечающий за отображение дерева в пошаговом режиме

Для реализации пошагового режима в данной лабораторной работе использовался метод задержки выполнения основного потока программы с возможностью взаимодействия с пользовательским интерфейсом. Данные метода из-за применения только для отрисовки бинарного дерева описаны в заголовочном файле drawing.cpp.

Представленный ниже код позволяет “поставить на паузу” выполнение текущей операции и ожидать от пользователя нажатия кнопки следующего шага (смена состояние происходит при помощи изменения флага состояния):

```
void loopLatency() {
    qDebug() << "Loop latency for step-by-step started" << endl;
    for( ; ; ) {
        QApplication::processEvents();
        if(stepLoopSwitcher == false) break;
    }
    stepLoopSwitcher = true;
}
```

Проверка начального входа в петлю осуществляется за счёт флага stepButStepEnable, переключаемого благодаря выбору соответствующей опции в UI.

Оценка сложности алгоритма

Алгоритмы обхода дерева inorder и postorder являются итеративными: каждый элемент строки обрабатывается один раз, а значит сложность алгоритма можно оценить как $O(N)$.

Алгоритм подсчёт выражения по его дереву с использованием стека также имеет линейную зависимость, так как количество осуществляемых операций является фиксированных для любых арифметических действий. График зависимости представлен на рис. 2.

Алгоритм, используемый в функции simplifying не поддаётся однозначной оценке, так как упрощение математического выражения зависит от его структуры и, например, наличие констант может как усложнять операцию (вынесение за скобки), так и упрощать её (“уничтожение” большой части выражения за счёт умножения на 0).

В связи с использованием вышеописанной функции в алгоритме дифференцирование, он тоже не поддаётся строго оценке сложности своей работы, однако, за исключением упрощения, его сложность аналогично будет линейной из-за фиксированного количества операций для каждого действия с константой или переменной.

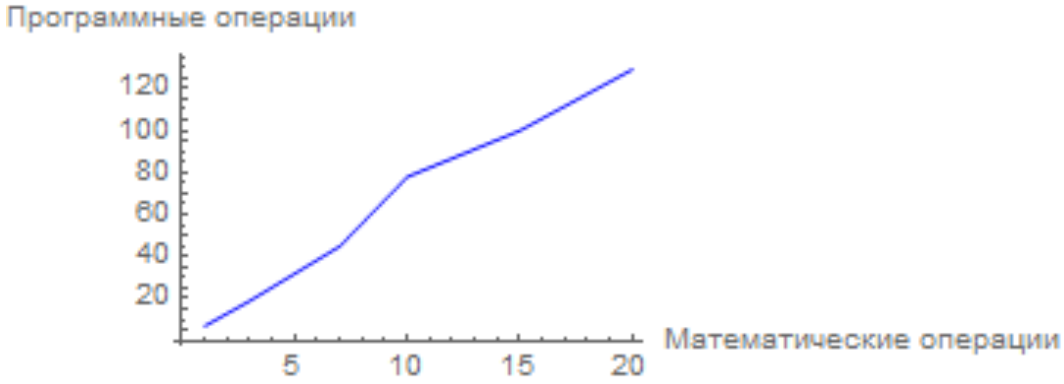

Рисунок 2 – зависимость программных операций от математических

Тестирование.

Результаты тестирования подсчета значения дерева-формулы представлены в табл.4.

Таблица 4 – подсчет значение дерева-формулы *t*

Входное выражение	Результат
(1)	result:1
(2+3)	result:5
(a-2)	Input numbers!
(1-2	check input string
(1++2)	Please input correct expression!!!
(1+)	Please input correct expression!!!
((2*(5-(3-2)))+(9-8))	Result:9

Результаты тестирования подсчета производной дерево-формулы по заданной переменной представлены в табл.5.

Таблица 5 – подсчета значения производной дерева-формулы *t*

Входное выражение	Заданная переменная	Результат
(a)	a	1
(a+1)	a	1
(a*(a*a))	a	3*(a*a)

((b-c)*(b+c))	b	((b-c)+(b+c))
(1+2)	a	0
((a-c)*(a+(b-(d+(k*(l+a))))))	a	(((a-c)*(1+(0-k)))+(a+(b-(d+(k*(l+a))))))

Визуализация дерева по заданному выражению представлена на рис.3.

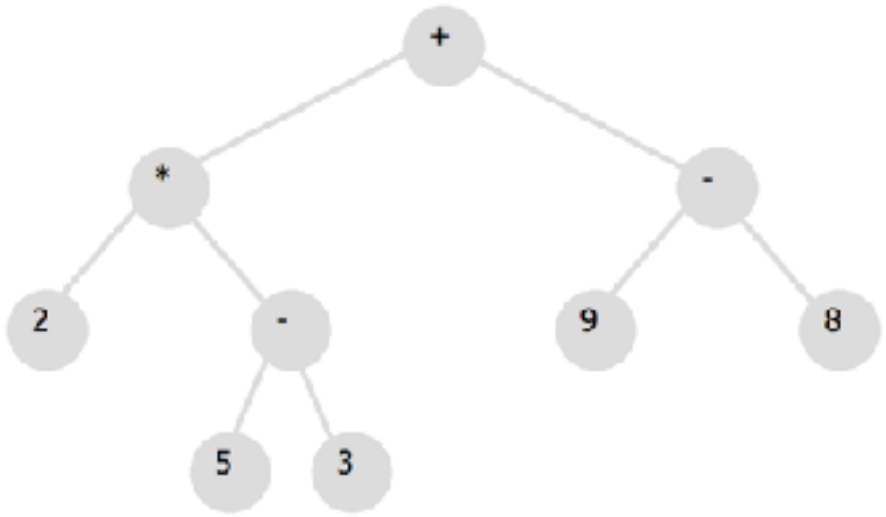

Рисунок 3 – визуализация выражения ((2*(5-3))+(9-8)) в виде дерева-формулы

Вывод.

В ходе выполнения данной работы была изучена и реализована такая структура данных, как бинарное дерево. На её основе была составлена программа для выполнения вычисления численного или дифференцирования сложного арифметического выражения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: allheaders.h

```
#ifndef ALLHEADERS_H
#define ALLHEADERS_H

/* C++ */

#include <iostream>
#include <vector>
#include <map>
#include <fstream>
#include <algorithm>
#include <memory>
#include <cstdint>
#include <cstring>
#include <string>
#include <cstdlib>
#include <unistd.h>
#include <exception>
#include <stdexcept>
#include <cstdio>
#include <cassert>
#include <regex>
#include <cmath>
#include <unistd.h>
#include <iostream>
#include <algorithm>
#include <regex>

/* QT */

#include <QMainWindow>
#include <QApplication>
#include <QObject>
#include <QMessageBox>
#include <QDebug>
#include <QString>
#include <QFileDialog>
#include <QGraphicsItem>
#include <QtGui>
```

```
#include <QDialog>
#include <QColorDialog>
#include <QString>
#include <QDebug>
#include <QPainter>
#include <QComboBox>
#include <QLabel>
#include <QPushButton>
#include <QFile>
#include <QWidget>
#include <QVBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QLineEdit>
#include <QGroupBox>
#include <QRadioButton>
#include <QTextEdit>
#include <QEventLoop>
#include <QTimer>
#include <QColor>
#include <QDebug>
#include <QGraphicsView>
#include <QFormLayout>
#include <QGraphicsEffect>
#include <QApplication>

using namespace std;

#endif // ALLHEADERS_H
```

Название файла: btree.h

```
#ifndef BTREE_H
#define BTREE_H

#include "stack.h"
#include "infixtopostfix.h"

struct Node {
    char value;
    struct Node* left, *right;
};
```

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

struct Node* constructTree(string & postfix);
int countDeep(struct Node *&node);
string inorder(struct Node *t);
void postorder(struct Node *t, Stack<char>&,
vector<string>&,string &);
int eval(struct Node* root, string &);

void calc(char value, Stack<char>& stack, vector<string>&,
string&);
string Diff (struct Node *root, char var);
void simplifying(string & str);

#endif // BTREE_H
```

Название файла: btree.cpp

```
#include "allheaders.h"
#include "btree.h"

struct Node* newNode(char v)
{
    struct Node *temp = new Node;
    temp->left = nullptr;
    temp->right = nullptr;
    temp->value = v;
    return temp;
};

string inorder(struct Node *root)
{
    if (!root)
            return "";
    string output;

    if (root->left || root->right)
        output += "(";
```

```
        output += inorder(root->left);
        output += root->value;
        output += inorder(root->right);

        if (root->left || root->right)
            output += ")";

        return output;
    }

    void calc(char value, Stack<char>& stack, vector<string>&
steps, string& error){
        if(isOperation(value)){
            if(stack.isEmpty()){
                    error += "check string. it must consist of
numbers\n";
                return;
            }
            int sum = *stack.onTop()-'0';
            stack.pop();
            if (value=='+'){
                string str;
                str+=*stack.onTop();
                str += "+"+to_string(sum)+"=";
                sum+=(*stack.onTop()-'0');
                str+=to_string(sum);
                steps.push_back(str);
            }

            if (value=='-'){
                string str;
                str+=*stack.onTop();
                str += "-"+to_string(sum)+"=";
                sum =(*stack.onTop()-'0')-sum;
                str+=to_string(sum);
                steps.push_back(str);
            }

            if (value=='*'){
                string str;
                str+=*stack.onTop();
```

```
                str += "*"+to_string(sum)+"=";
                sum*=(*stack.onTop()-'0');
                str+=to_string(sum);
                steps.push_back(str);
            }
            stack.pop();
            stack.push(static_cast<char>(sum + '0'));
        }else{
            if(!isdigit(value)){
                      error += "check string. it must consist of
numbers\n";
                   return;
            }
            stack.push(value);
        }
    }

    void   postorder(struct   Node   *t,   Stack<char>&   stack,
vector<string>& steps, string & errors)
    {
        if(t){
            postorder(t->left, stack, steps,errors);
            postorder(t->right, stack, steps, errors);
            if(errors.empty())
                calc(t->value, stack, steps, errors);
        }else{
            return;
        }
    }

    void simplifying(string & str) {
        cout << str << endl;
           if(!str.compare("(1+0)")||  !str.compare("(1-0)")  ||  !
str.compare("(0+1)")){
            str = "1";
            return;
        }
        else if(!str.compare("(0+0)")){
            str = "0";
            return;
        }
        else if(!str.compare("(1+1)")){
```

```
            str = "2";
            return;
        }
        smatch resStr;

        regex reg1("\\((.*)[+-]0\\)");
        if(regex_match(str, resStr, reg1)){
            str = resStr[1] ;
        }

        regex reg2("\\(0\\+(.*)\\)");
        if(regex_match(str, resStr, reg2)){
            str = resStr[1] ;
        }

        regex reg3("\\((.*)\\*1\\)");
        if(regex_match(str, resStr, reg3)){
            str = resStr[1];
        }

        regex reg4("\\((.*)\\*0\\)");
        if(regex_match(str, resStr, reg4)){
            str = "0";
            return;
        }

        regex reg5("\\(0\\*(.*)\\)");
        if(regex_match(str, resStr, reg5)){
            str = "0";
            return;
        }


        regex reg6("\\((.*)-(.*)\\)");
                 if(regex_match(str,   resStr,   reg6)   &&   !
resStr[1].compare(resStr[2])) {
            str = "0";
            return;
        }

          regex reg8("\\(([a-z]\\*){1,}([a-z]\\*[a-z]\\*)*([0-9]+\
\*)[a-z]{1,}(\\*[a-z])*\\)");
```

```
        string temp = str;
        if(regex_search(str, resStr, reg8)) {
                                    temp.erase(str.find(resStr[3]),
static_cast<unsigned long>(resStr[3].length()));
                temp.insert(str.find(resStr[0]), resStr[3]);
                str = temp;
                cout << "reg8";
        }

         regex reg9("\\(([0-9]*)\\*?\\((.*)\\)([+-])([0-9]*)\\*?\\
((.*)\\)\\)");
        if(regex_search(str, resStr, reg9)) {
            if(!resStr[2].compare(resStr[5])) {
                int factor1=1, factor2=1;
                if(resStr[1].length()) {
                     factor1 = stoi(resStr[1]);
                }
                if(resStr[4].length()) {
                     factor2 = stoi(resStr[4]);
                }
                if(resStr[3]=="+") {
                    factor1 += factor2;
                }else{
                    factor1 -= factor2;
                }
                string buf = to_string(factor1)+"*";
                if(resStr[2].length()>1){
                    buf += "(";
                    buf += resStr[2];
                    buf += ")";
                }
                str = buf;
            }
            return;
        }

        regex reg10("\\(([0-9]*)([+-*])([0-9]*)\\)");
        if(regex_match(str, resStr, reg10)) {
            int factor1, factor2;
            factor1 = stoi(resStr[1]);
            factor2 = stoi(resStr[3]);
            if(resStr[2]=="+") {
```

```
                factor1 += factor2;
            }else if(resStr[2]=="-"){
                factor1 -= factor2;
            }else if(resStr[2]=="*"){
                factor1 *= factor2;
            }

            str = to_string(factor1);
            return;
        }

        regex reg7("\\((.*)\\+(.*)\\)");
                 if(regex_match(str,   resStr,   reg7)   &&   !
resStr[1].compare(resStr[2])) {
            string result = "2*";
            result += resStr[1];
            str = result;
            return;
        }

    }

    string Diff (struct Node *root, char var)
    {
        if (!root)
            return "";
        if (root->value) {
            if (root->value == var)
                return "1";
            else if(root->value == '+'){
                 string str = "(" + Diff(root->left, var) + "+" +
Diff(root->right, var) + ")";
                simplifying(str);
                return str;
            }
            else if(root->value == '-'){
                 string str = "(" + Diff(root->left, var) + "-" +
Diff(root->right, var) + ")";
                simplifying(str);
                return str;
            }
            else if(root->value == '*'){
```

```
            string str1 = inorder(root->left);
            if(str1.length()!=1){
                //str1 = "("+str1+")";
                simplifying(str1);
            }

            string str2= inorder(root->right);
            if(str2.length()!=1){
                //str2 = "("+str2+")";
                simplifying(str2);
            }

            string str3 = Diff(root->right, var);
            string arg1 = "("+str1+"*"+str3+")";
            simplifying(arg1);
            string str4 =  Diff(root->left, var);
            string arg2 = "("+str2+"*"+str4+")";
            simplifying(arg2);
            string result = "("+arg1+"+"+arg2+")";
            simplifying(result);
            return result;
        }else
            return "0";
    }

    return "0";
}

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

int eval(struct Node* root, string & err) {

    if (!root)
        return 0;

    if (!root->left && !root->right){
```

```
        if(isdigit(root->value))
            return root->value - '0';
        err = "Input numbers!";
        return 0;
    }


    int l_val = eval(root->left, err);


    int r_val = eval(root->right, err);


    if (root->value=='+')
        return l_val+r_val;

    if (root->value=='-')
        return l_val-r_val;

    if (root->value=='*')
        return l_val*r_val;

    return 0;
}

struct Node* constructTree(string & postfix) {
    Stack<struct Node *> st;
    struct Node *t, *t1, *t2;

    for (unsigned long i=0; i<postfix.length(); i++)
    {

        if (!isOperation(postfix[i]))
        {
            t = newNode(postfix[i]);
            st.push(t);
        }
        else
        {
            t = newNode(postfix[i]);
            t1 = *st.onTop();
            st.pop();
```

```
                t2 = *st.onTop();
                st.pop();
                t->left = t1;
                t->right = t2;
                st.push(t);
            }
        }
        t = *st.onTop();
        st.pop();

        return t;
    }
```

Название файла: stack.h

```
    #ifndef stack_h
    #define stack_h

    #include "allheaders.h"

    template<class T>
    class Stack
    {
        public:
            Stack(size_t n=50)
            : SIZE{ n }, top{}, items{ new T[SIZE]{} }
            {}
            ~Stack() {delete[] items;}
            void push(T);
            void pop();
            T* onTop() const;
            bool isFull() const{return SIZE == top;}
            bool isEmpty() const{return top==0;}
            size_t size() const {return SIZE;}
            size_t length() const {return top;}
            void clear(){top=0;}
              void setSize(size_t n){SIZE=n; items=new T[SIZE]{};
top=0;}
        private:
            size_t SIZE;
            size_t top;
```

```
        T *items;
};

template<class T>
void Stack<T>::push(T item)
{
    if(!isFull()){
        items[top++] = item;
    }
    else
        cout<<"Стек полон\n";
}

template<class T>
void Stack<T>::pop()
{
    if(!isEmpty()){
        top--;
    }
    else
        cout<<"Стек пуст\n";
}

template<class T>
T *Stack<T>::onTop() const
{
    if(!isEmpty())
        return &items[top-1];
    else{
        cout<<"Стек пуст\n";
        return nullptr;
    }
}
#endif
```

Название файла: drawing.h

```
#ifndef DRAWING_H
#define DRAWING_H

#include "allheaders.h"
#include "btree.h"
```

```
    void graphic(struct Node* tree, QGraphicsScene *&scene);

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

    void loopLatency();

    void changeStepByStepMode();

    void changeLoopState();

    void setDisableStepMode();

    #endif // DRAWING_H
```

Название файла: drawing.cpp

```
    #include "allheaders.h"
    #include "mainwindow.h"
    #include "btree.h"
    #include "drawing.h"

    static bool stepLoopSwitcher = false;
    static bool stepButStepEnable = false;

    void graphic(struct Node* tree, QGraphicsScene *&scene) {
        if (tree == nullptr)
            return;
        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, countDeep(tree))+2);
        int hDelta = 70;
        int wDelta = 15;
```

```
        font.setPointSize(wDelta);
        int width = (wDelta*wDeep)/2;
         treePainter(scene, tree, width/2, hDelta, wDelta, hDelta,
pen, brush, font, wDeep);
    }

    int treePainter(QGraphicsScene *&scene, struct Node* node,
int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush,
QFont &font, int depth) {
        if (node == nullptr)
            return 0;
        QString out;
        out += node->value;
        if(stepButStepEnable) loopLatency();
        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)
                  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;
    }

    // STEP BY STEP
    void loopLatency() {
         qDebug() << "Loop latency for step-by-step started" <<
endl;
        for( ; ; ) {
            QApplication::processEvents();
            if(stepLoopSwitcher == false) break;
        }
```

```
        stepLoopSwitcher = true;
    }

    void changeLoopState() {
        stepLoopSwitcher = false;
    }

    void changeStepByStepMode() {
        stepButStepEnable = true;
    }

    void setDisableStepMode() {
        stepButStepEnable = false;
    }
```

Название файла: infixtopostfix.h

```
#ifndef INFIXTOPREFIX_H
#define INFIXTOPREFIX_H

#include "allheaders.h"
#include "stack.h"

bool isOperation(char buf);
void infixToPrefix(string &, string&, string &);

#endif // INFIXTOPREFIX_H
```

Название файла: infixtopostfix.cpp

```
#include "allheaders.h"
#include "infixtopostfix.h"

bool isOperation(char buf)
{
      return (buf == '*' || buf=='-' || buf=='+')?  true:
false;
}

void infixToPrefix(string & str, string & postfixForm, string
& err)
{
```

```
        reverse(str.begin(), str.end());

        Stack<char> stack;
        for(unsigned long i=0; i<str.length(); i++){
            if(isOperation(str[i])){
                     if(((i+1)!=str.length() && str[i+1]==')') ||
stack.isEmpty() || *stack.onTop()==')')
                    stack.push(str[i]);
                else{
                    if(!stack.isEmpty()){
                        postfixForm+=*stack.onTop();
                        stack.pop();
                        stack.push(str[i]);
                    }else{
                        err = "check input string\n";
                        return;
                    }
                }
            }else if(str[i]==')') {
                stack.push(str[i]);
            }else if(str[i]=='(') {
                char buf;
                do{
                    if(!stack.isEmpty()){
                        buf = *stack.onTop();
                        stack.pop();
                        if(buf != ')')
                            postfixForm+=buf;
                    }else{
                        err = "check input string\n";
                        return;
                    }
                }while(buf != ')');

            }else if(str[i]==' '){
                continue;
            }else{
                postfixForm+=str[i];
            }
        }
        if(postfixForm=="" || postfixForm.empty()){
            err = "check input string\n";
```

```
        return;
    }


}
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "allheaders.h"
#include "btree.h"

namespace Ui { class MainWindow; }

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_resultButton_clicked();
    void on_stepButton_clicked();
    void on_deriativeButton_clicked();
    void on_fileOutput_clicked();
    void on_fileInput_clicked();
    void on_stepDraw_clicked();
    void on_drawButton_clicked();

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

struct Node* createTree(string & checkInput, string&);

#endif // MAINWINDOW_H
```

Название файла: mainwindow.cpp

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

static Stack<char> workStack;
static vector<string> steps;
static string str;
static string errors;
static int i = 1;

MainWindow::MainWindow(QWidget *parent):
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->resultWindow->setReadOnly(true);
    scene = new QGraphicsScene;
    ui->graphicsView->setScene(scene);
    ui->stepDraw->setDisabled(true);
    // запрещаем вводить что-то кроме одной буквы в диапазоне
[a-z]
                                ui->variableInput->setValidator(new
QRegExpValidator(QRegExp("[a-z]"), ui->variableInput));
    // оставляем для вводной строки только допустимые символы
                                    ui->inputLine->setValidator(new
QRegExpValidator(QRegExp("[a-z0-9+\\-*()]*"), ui->inputLine));
}

MainWindow::~MainWindow() {
    delete ui;
}

struct Node* createTree(string & checkInput, string & err) {
    string postfixForm;
    infixToPrefix(checkInput, postfixForm, err);
    if(!err.empty()){
        return nullptr;
    }
    return constructTree(postfixForm);
}
```

```
    void MainWindow::on_resultButton_clicked()
    {
        string result="result:", err;
        string str = ui->inputLine->text().toUtf8().constData();

         // проверка на отсутсвие двухначных чисел и двух знаков
подряд
        regex reg(".*[1-9]{2}.*");
        regex reg2(".*[-+*]{2}.*");
        regex reg3(".*[+\\-\\*][)+\\-\\*]+");
          if(regex_search(str, reg) || regex_search(str, reg2)||
regex_search(str, reg3)) {
            err = "Please input correct expression!!!\n";
                                                ui->resultWindow-
>setText(QString::fromStdString(err));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }

        struct Node* tree = createTree(str, err);
        if(!errors.empty()){
                                                ui->resultWindow-
>setText(QString::fromStdString(errors));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }
        int k = eval(tree, err);
        if(!err.empty()){
                                                ui->resultWindow-
>setText(QString::fromStdString(err));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }
        result+=to_string(k);
                                                ui->resultWindow-
>setText(QString::fromStdString(result));
        this->resize(1081, 720);
        this->resize(1080, 720);
```

```
    }

    void MainWindow::on_stepButton_clicked() {
                                    if(str.compare(ui->inputLine-
>text().toUtf8().constData())!=0){
                str.clear();
                                        str.append(ui->inputLine-
>text().toUtf8().constData());

                string err;
                  // проверка на отсутсвие двухначных чисел и двух
знаков подряд
                regex reg(".*[1-9]{2}.*");
                regex reg2(".*[-+*]{2}.*");
                regex reg3(".*[+\\-\\*][)+\\-\\*]+");
                    if(regex_search(str, reg) || regex_search(str,
reg2)||regex_search(str, reg3)) {
                    err = "Please input correct expression!!!\n";
                                                ui->resultWindow-
>setText(QString::fromStdString(err));
                    this->resize(1081, 720);
                    this->resize(1080, 720);
                    return;
                }

                                string  str1  =  ui->inputLine-
>text().toUtf8().constData();
                i = 1;
                steps.clear();
                workStack.clear();
                errors.clear();
                ui->resultWindow->clear();
                struct Node* root = createTree(str1, errors);
                if(!errors.empty()){
                                                ui->resultWindow-
>setText(QString::fromStdString(errors));
                    this->resize(1081, 720);
                    this->resize(1080, 720);
                    return;
                }
                postorder(root, workStack, steps, errors);
```

```
        }
        if(!errors.empty()){
                                                    ui->resultWindow-
>setText(QString::fromStdString(errors));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }
        string buf="";
           for(unsigned long j = 0; j < static_cast<unsigned
long>(i) && j < steps.size(); j++){
            buf += steps[j] + "\n";
        }
        ui->resultWindow->setText(QString::fromStdString(buf));
        i++;
        this->resize(1081, 720);
        this->resize(1080, 720);
    }

    void MainWindow::on_deriativeButton_clicked()
    {
         string str = ui->inputLine->text().toUtf8().constData(),
err;
        if(ui->variableInput->text().size() == 0){
            string err = "Please input variable!!!\n";
                                                    ui->resultWindow-
>setText(QString::fromStdString(err));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }

         // проверка на отсутсвие двухначных чисел и двух знаков
подряд
        regex reg(".*[1-9]{2}.*");
        regex reg2(".*[-+*]{2}.*");
        regex reg3(".*[+\\-\\*][)+\\-\\*]+");
          if(regex_search(str, reg) || regex_search(str, reg2)||
regex_search(str, reg3)) {
            err = "Please input correct expression!!!\n";
                                                    ui->resultWindow-
>setText(QString::fromStdString(err));
```

```
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }

        struct Node* tree = createTree(str, err);
        if(!err.empty()){
                                                ui->resultWindow-
>setText(QString::fromStdString(err));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }
          char  k  =  ui->variableInput->text().toUtf8().constData()
[0];
        string result = Diff(tree, k);
                                                ui->resultWindow-
>setText(QString::fromStdString(result));
        this->resize(1081, 720);
        this->resize(1080, 720);
    }

    void MainWindow::on_fileOutput_clicked()
    {
        QString fileName = QFileDialog::getOpenFileName(this,
                tr("Open TXT File"),  QDir::homePath(),
                tr("TXT text (*.txt);;All Files (*)"));
        ofstream sourceFile(fileName.toUtf8().constData());
                        sourceFile      <<      ui->resultWindow-
>toPlainText().toUtf8().constData();
    }

    void MainWindow::on_fileInput_clicked()
    {
        QString fileName = QFileDialog::getOpenFileName(this,
                tr("Open TXT File"),  QDir::homePath(),
                tr("TXT text (*.txt);;All Files (*)"));
        ifstream sourceFile(fileName.toUtf8().constData());
        string input;
        sourceFile >> input;
        ui->inputLine->setText(QString::fromStdString(input));
    }
```

```
    void MainWindow::on_stepDraw_clicked()
    {
        changeLoopState();
    }

    void MainWindow::on_drawButton_clicked()
    {
        if(ui->checkBox->isChecked()) {
            ui->stepDraw->setDisabled(false);
            changeStepByStepMode();
        }
        else {
            ui->stepDraw->setDisabled(true);
            changeLoopState();
            setDisableStepMode();
        }
         string str = ui->inputLine->text().toUtf8().constData(),
err;
        regex reg(".*[1-9]{2}.*");
        regex reg2(".*[-+*]{2}.*");
        regex reg3(".*[+\\-\\*][)+\\-\\*]+");
          if(regex_search(str, reg) || regex_search(str, reg2)||
regex_search(str, reg3)) {
            err = "Please input correct expression!!!\n";
                                                ui->resultWindow-
>setText(QString::fromStdString(err));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }
        if(str.empty()){
                                                ui->resultWindow-
>setText(QString::fromStdString("Input expression"));
            this->resize(1081, 720);
            this->resize(1080, 720);
            return;
        }
        struct Node* tree = createTree(str, err);
        if(!err.empty()){
                                                ui->resultWindow-
>setText(QString::fromStdString(err));
```

```
        this->resize(1081, 720);
        this->resize(1080, 720);
        return;
    }
    ui->resultWindow->setText(QString::fromStdString(""));
    this->resize(1081, 720);
    this->resize(1080, 720);
    graphic(tree, scene);
}
```