МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Иерархические списки

Студент гр. 8381 Облизанов А.Д.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивной обработки списков, изучить особенности реализации иерархического списка на языке программирования C++. Разработать программу, использующую иерархические списки и их рекурсивную обработку, анализирующую корректность выражения.

Задание.

Пусть логическое выражение представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в постфиксной форме (<аргументы> <операция>)). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (* b (! c))). Необходимо добавить дополнительный логический оператор, который принимает только 2 аргумента и проверить синтаксическую корректность выражения.

Основные теоретические положения.

Согласно рекурсивному определению, иерархический список – такой список, элементами которого могут быть иерархические списки. В случае постфиксного выражения с помощью иерархического списка может быть построена зависимость операций (вложенность скобок).

Для обработки иерархического списка удобно использовать рекурсивные функции, так как он представляет собой множество линейный списков, между которыми установлены связи, иерархия.

Выполнение работы.

Написание работы производилось на базе операционной системы Windows 10 в среде разработки QtCreator с использованием фреймворка Qt. Сборка, отладка и тестирование также производились в QtCreator.

Для реализации программы был разработан графический интерфейс с помощью встроенного в QtCreator UI-редактора. Он представляет из себя поле ввода, кнопку считывания и переноса в поле ввода информации из файла, флаг записи дополнительной информации в вывод, кнопку, запускающую

синтаксический анализатор для текущей строки, поле вывода, а также кнопку выгрузки в файл информации, содержащейся в поле вывода.

Были созданы слоты, обрабатывающие сигналы clicked() кнопок. При нажатии на кнопку считывания из файла, открывается файловый диалог с помощью функции класса QFileDialog – getOpenFileName(). После выбора файла для загрузки, запускается функция getInfo. В ней создается объект класса ifstream для файла, и информация из него считывается в строку, возвращаемую функцией. Далее текст выводится на виджет LineEdit с именем input.

Для реализации иерархического списка была реализована структура HieElem элемента списка, которая состоит из указателей на предыдущий, следующий элемент списка, на родительский элемент, из булевой переменной haveChild, а также объединения union, состоящее из переменной типа char для хранения информации и указателя на дочерний элемент. Кроме того, была реализована структура HieList, состоящая из указателя на головной элемент (head) и на последний обработанный элемент (last).

Функция appendNext() для добавления элемента в иерархический список принимает на вход указатель на структуру HieList и переменную типа char. В функции выделяется динамическая память для нового элемента HieElem, который заполняется в соответствии с входными данными. Если голова поданного на вход списка пуста, то ей присваивается указатель на созданный элемент, иначе он связывается с элементом last списка.

Функция appendChild для добавления дочернего элемента в иерархический список принимает на вход указатель на родительский список, указатель на голову дочернего списка, а также ссылку на переменную ошибки. В функции выделяется динамическая память для нового элемента HieElem, который заполняется в соответствии с входными данными как дочерний элемент.

При нажатии на кнопку запуска анализатора, создается объект структуры HieList, а текст из поля ввода, переменная для отслеживания ошибок и номер

символа для обработки в строке (устанавливается в 1) передаются по ссылке в функцию createHieFromStr().

Функция createHieFromStr() рекурсивно заполняет иерархический список в соответствии с поданной строкой. Функция считывает символы из строки, пока не обнаружит символ конца строки или закрывающей скобки (предполагается, что функции подается индекс символа после открывающей скобки). Все элементы, игнорируя пробелы, функция добавляет в список с помощью функции appendNext(). Если встречена открывающая скобка, вызывается функция appendChild(), которой подаются текущий список и указатель на дочерний список, который создается путем рекурсивного запуска функции createHieFromStr() (см. Приложение А, файл hielist.cpp).

Функция checkPrefix() принимает на вход указатель на головной элемент списка и ссылку на переменную ошибки. В функции проверяется, является ли первый элемент каждого списка оператором, отсутствуют ли списки без операндов и т.д. Функция исследует все элементы списка рекурсивно, вызывая себя для каждого нового элемента, в том числе дочернего. В целом, реализация напоминает обход дерева.

Таблица 1 – Виды ошибок и их коды

Название	**Код**	**Описание ошибки**
NO_CLOSE	1	Отсутствует закрывающая скобка.
EXTRA_SYMB	2	В строке есть корректное выражение, но после него имеются лишние символы.
EMPTY	3	В выражении есть пустой список “( )”.
NO_OPERATOR	4	В одном из списков нет оператора, или он не указан в начале списка (префиксная форма).
NO_OPERAND	5	В одном из списков нет операндов.
BAD_OPERAND	6	Один из операндов не является цифрой или буквой.

BAD_EQUIVALENT	7	Для оператора эквиваленции указаны не два операнда.
BAD_PARENT	11	Один из списков, являющийся родительским, содержит в начале не оператор, а указатель на дочерний список
BAD_CHILD	12	Один из списков, являющийся дочерним, пустой.

Функция printRecursion() выводит все элементы списка в строку. Функция deleteList очищает память, выделенную на элементы списка.

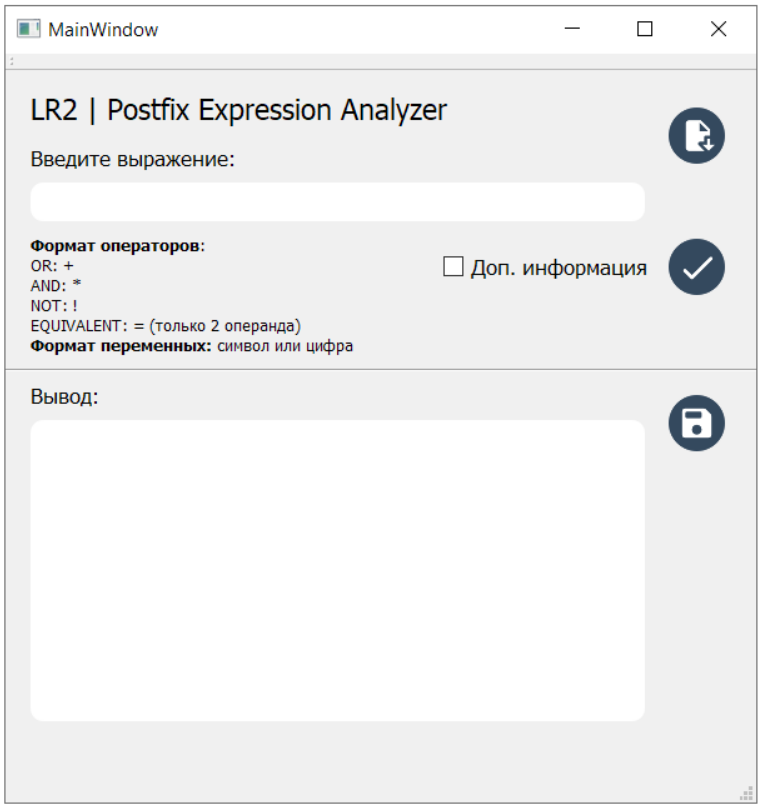
Оценка эффективности алгоритма.

Алгоритм, реализованный в программе, имеет линейную зависимость от длины строки, то есть сложность оценивается как O(n). Рассуждения отталкиваются от того, что функция createHieFromStr() имеет сложность O(1) и запускается не более 1 раза для каждого символа строки, так же как и функция checkPrefix(). Стоит отметить, что из-за выделения памяти для каждого элемента списка (то есть обращения к системе), скорость выполнения программы ниже, чем была бы при реализации через массивы.

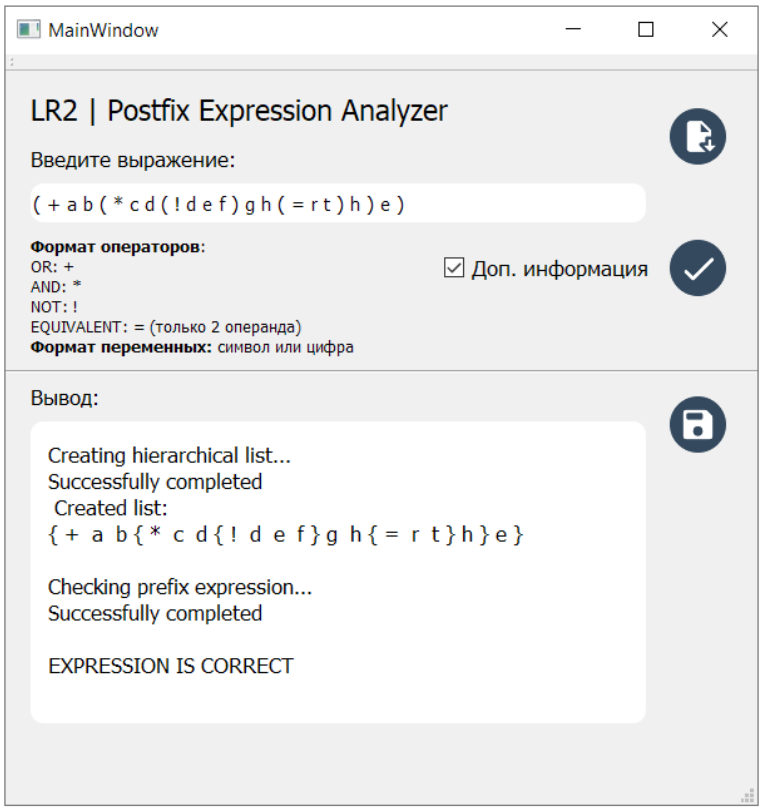
Ввиду рекурсивного алгоритма рост занимаемой памяти растет линейно из-за создаваемых в функциях временных переменных.

Тестирование программы.

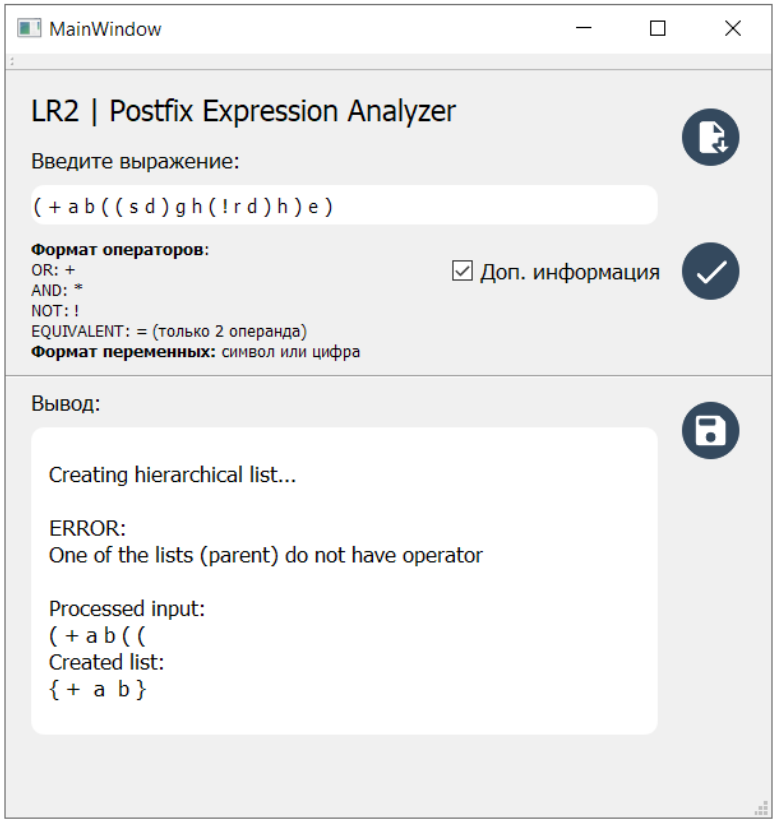
Вид программы после запуска:



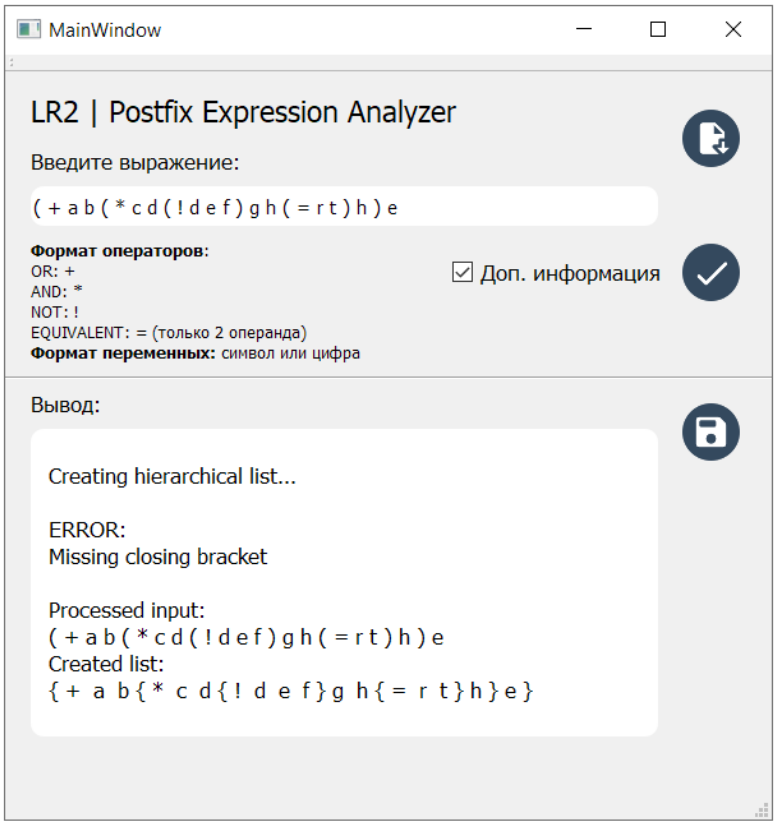
Проверка корректного выражения:



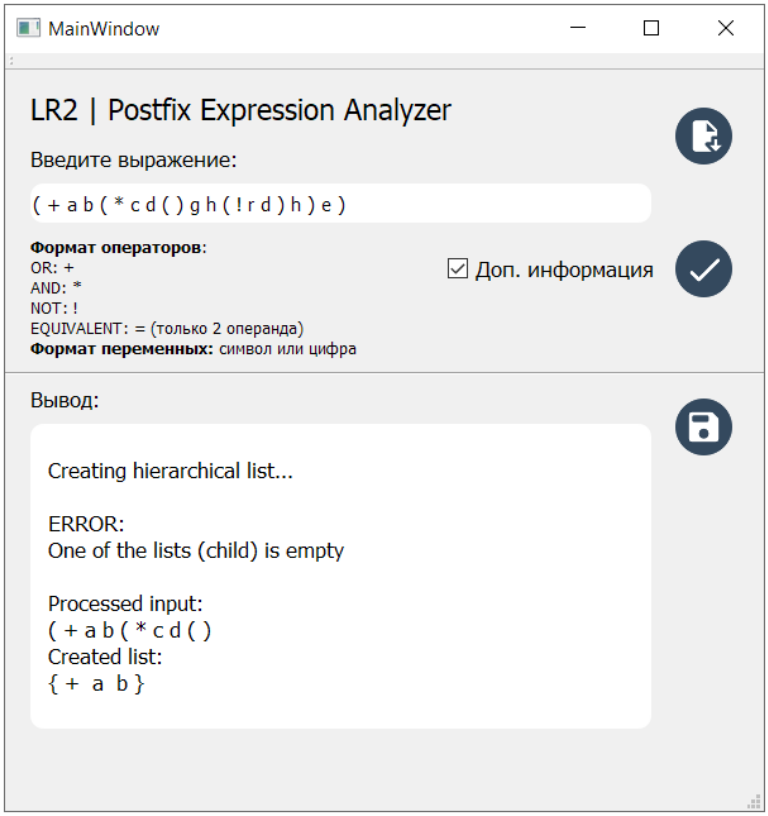
Проверка выражения с ошибкой (нет оператора):



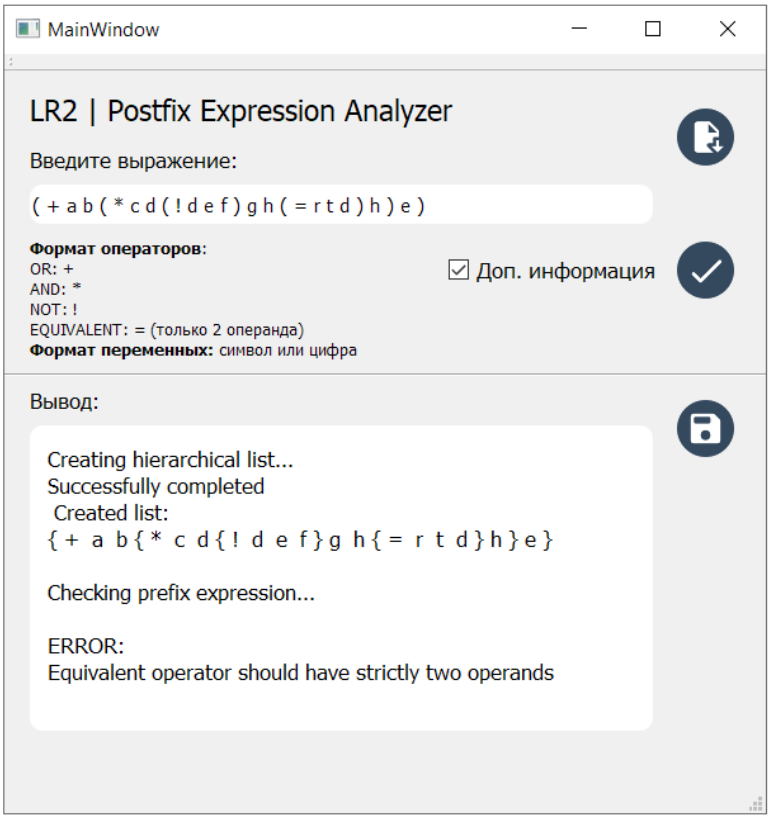
Проверка выражения с ошибкой (отсутствует закрывающая скобка):



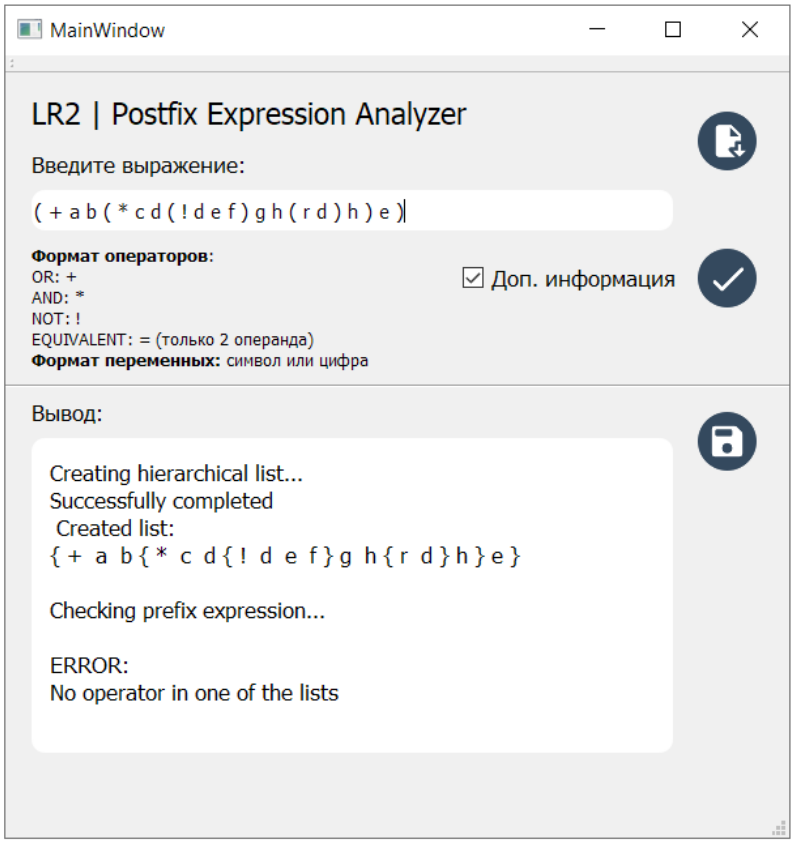
Проверка выражения с ошибкой (пустой список):



Проверка выражения с ошибкой (не 2 операнда эквиваленции)



Проверка выражения с ошибкой (нет оператора):



Выводы.

В ходе выполнения лабораторной работы была изучена такая структура данных как иерархические списки, а также рекурсивные методы ее обработки. Была реализована программа на C++, использующая иерархические списки, которая анализирует строку, определяя ее соответствие постфиксной записи логического выражения в виде иерархического списка.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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

    return a.exec();
}
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsEffect>
#include <QGraphicsView>
#include <QFileDialog>
#include <QStandardPaths>
#include <QtGui>
#include <QLabel>
#include <QColorDialog>
#include <QInputDialog>
#include <QMainWindow>
#include <QPushButton>
#include <QMessageBox>
#include <iostream>
#include <fstream>
using namespace std;


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    string getInfo(string filePath);
    int checkExp(string &expIn, unsigned long &pos, int &funcNum);
    int checkString(string &expIn, unsigned long &pos, int &funcNum);

private slots:
    void on_start_clicked();
```

```
    void on_file_clicked();

    void on_save_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

Название файла: hielist.h

```
#include "mainwindow.h"
#define BAD_PARENT 11
#define BAD_CHILD 12
#define NO_CLOSE 1
#define EXTRA_SYMB 2
#define EMPTY 3
#define NO_OPERATOR 4
#define NO_OPERAND 5
#define BAD_OPERAND 6
#define BAD_EQUIVALENT 7

struct HieElem
{
    bool haveChild;
    HieElem *prev;
    HieElem *next;
    union
    {
        char info;
        HieElem *child;
    };
    HieElem *parent;
    int depth;
};

struct HieList
{
    HieElem *head;
    HieElem *last;
};

int createHieList(HieList *list);
int appendNext(HieList *list, char in);
int appendChild(HieList *list, HieElem *child, int &err);
HieElem *createHieFromStr(string &in, unsigned long long &ptr, int
&err);

int checkPrefix(HieElem *head, int &err);

int printRecursion(HieElem *temp, string &out);
string getList(HieElem *head);
int deleteList(HieElem *head);
```

Название файла: hielist.cpp

```
#include "hielist.h"


int createHieList(HieList *list)
{
    list->head = nullptr;
    list->last = nullptr;
    return 0;
}

int appendNext(HieList *list, char in)
{
    HieElem *next = new HieElem;
    next->haveChild = 0;
    next->next = nullptr;
    next->parent = nullptr;
    next->info = in;
    if (list->head == nullptr)
    {
        list->head = next;
        list->last = next;
        next->prev = nullptr;
    }
    else
    {
        list->last->next = next;
        next->prev = list->last;
        list->last = next;
    }
    return 0;
}

int appendChild(HieList *list, HieElem *child, int &err)
{
    if (err)
        return 1;
    if (child == nullptr)
    {
        err = BAD_CHILD;
        return 1;
    }
    HieElem *next = new HieElem;
    next->haveChild = 1;
    next->next = nullptr;
    next->parent = nullptr;
    list->last->next = next;
    next->prev = list->last;
    next->child = child;
    child->parent = next;
    list->last = next;
    return 0;
}

HieElem *createHieFromStr(string &in, unsigned long long &ptr, int
&err)
{
```

```
    HieList list;
    createHieList(&list);
    while (in[ptr] != ')' && in[ptr] != '\0')
    {
        if (in[ptr] == ' ')
        {
            ptr++;
            continue;
        }
        if (in[ptr] == '(')
        {
            ptr++;
            if (list.head == nullptr)
            {
                err = BAD_PARENT;
                return list.head;
            }
            appendChild(&list, createHieFromStr(in, ptr, err), err);
            if (err)
            {
                return list.head;
            }
        }
        else
        {
            appendNext(&list, in[ptr]);
            ptr++;
        }
    }
    if (in[ptr] == '\0')
    {
        err = NO_CLOSE;
    }
    if (in[ptr] == ')')
        ptr++;
    return list.head;
}

int printRecursion(HieElem *temp, string &out)
{
    if (temp != nullptr)
    {
        if (temp->haveChild == 0)
        {
            out += " ";
            out += temp->info;
            out += " ";
        }
        else
        {
            out += "{";
            printRecursion(temp->child, out);
            out += "}";
        }
        printRecursion(temp->next, out);
    }
```

```
        return 0;
    }

    int checkPrefix(HieElem *head, int &err)
    {
        bool infVar = false;
        HieElem *temp = head;
        if (temp == nullptr)
        {
            err = EMPTY;
            return 1;
        }
        if (temp->haveChild == 1)
        {
            err = BAD_PARENT;
            return 1;
        }
        if (temp->info == '=')
        {
            infVar = true;
        }
        else if (temp->info != '+' && temp->info != '!' && temp->info !=
'*')
        {
            err = NO_OPERATOR;
            return 1;
        }

        if (temp->next == nullptr)
        {
            err = NO_OPERAND;
            return 1;
        }
        temp = temp->next;
        int numVar = 0;
        while (temp != nullptr)
        {
            if (temp->haveChild)
            {
                if (checkPrefix(temp->child, err))
                    return err;
                numVar++;
            }
            else if (isalnum(temp->info) == 0)
            {
                err = BAD_OPERAND;
                return 1;
            }
            else
                numVar++;
            temp = temp->next;
        }
        if (infVar && numVar != 2)
        {
            err = BAD_EQUIVALENT;
            return 1;
```

```
    }
    return 0;
}

string getList(HieElem *head)
{
    string out;
    out += "{";
    printRecursion(head, out);
    out += "}";
    return out;
}

int deleteList(HieElem *head)
{
    if (head == nullptr)
        return 0;
    HieElem *temp = head;
    while (temp != nullptr)
    {
        temp = temp->next;
        delete head;
        head = temp;
    }
    return 0;
}
```

Название файла: mainwindow.cpp

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->output->setWordWrap(true);
}

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

void MainWindow::on_start_clicked()
{
    string expIn = qPrintable(ui->input->text());
    bool logs = ui->logs->isChecked();
    string output;
    string temp;
    int err;
    unsigned long long ptr = 1;
    HieList *list = new HieList;
    createHieList(list);
    if (logs)
```

```
            output += "Creating hierarchical list...\n";
        list->head = createHieFromStr(expIn, ptr, err);
        switch (err) {
        case NO_CLOSE:
            output += "\nERROR:\nMissing closing bracket\n";
            break;
        case BAD_CHILD:
            output += "\nERROR:\nOne of the lists (child) is empty\n";
            break;
        case BAD_PARENT:
            output += "\nERROR:\nOne of the lists (parent) do not have
operator\n";
            break;
        }
        if (err && logs)
        {
            output += "\nProcessed input:\n";
            output += expIn.substr(0, ptr);
            output += "\nCreated list:\n";
            output += getList(list->head);
        }
        else
        {
            if (logs)
            {
                output += "Successfully completed\n Created list:\n";
                output += getList(list->head);
                output += "\n\nChecking prefix expression...\n";
            }
            checkPrefix(list->head, err);
            if (ptr != expIn.length() && !err)
                err = EXTRA_SYMB;
            switch (err) {
            case EXTRA_SYMB:
                output += "\nERROR:\nExtra symbols after expression\n";
                break;
            case EMPTY:
                output  +=  "\nERROR:\nThere  is  an  empty  list  in
expression\n";
                break;
            case NO_OPERATOR:
                output += "\nERROR:\nNo operator in one of the lists\n";
                break;
            case NO_OPERAND:
                output += "\nERROR:\nNo operands in one of the lists\n";
                break;
            case BAD_OPERAND:
                output  +=  "\nERROR:  \nOperands  should  be  digits  or
letters\n";
                break;
            case BAD_EQUIVALENT:
                output  +=  "\nERROR:\nEquivalent  operator  should  have
strictly two operands\n";
                break;
            case 0:
                if (logs)
```

```
                    output += "Successfully completed\n";
                output += "\nEXPRESSION IS CORRECT\n";
            }
        }
        ui->output->setText(QString::fromStdString(output));
        deleteList(list->head);
        delete list;
    }

    string MainWindow::getInfo(string filePath)
    {

        ifstream fin(filePath);
        string out;
        char buff[1010];
        fin.getline(buff, 1000);
        fin.close();
        out.append(buff);
        return out;
    }

    void MainWindow::on_file_clicked()
    {
        this->resize(450, 450);
        QString fileName = QFileDialog::getOpenFileName(this, tr("Choose
input file (TXT)"), QDir::homePath(), tr("*.txt"));
        if (QString::compare(fileName, QString()) != 0)
        {
            string expIn = getInfo(qPrintable(fileName));
            ui->input->setText(QString::fromStdString(expIn));
        }
    }

    void MainWindow::on_save_clicked()
    {
        QString filePath = QFileDialog::getSaveFileName(this, tr("Save
to TXT file"), QDir::homePath(), tr("*.txt"));
        if (QString::compare(filePath, QString()) != 0)
        {
            ofstream fout(qPrintable(filePath));
            fout << qPrintable(ui->output->text());
            fout.close();
        }
    }
```

Название файла: mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>600</width>
    <height>600</height>
```

```
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <widget class="QPushButton" name="start">
    <property name="geometry">
     <rect>
      <x>530</x>
      <y>135</y>
      <width>45</width>
      <height>45</height>
     </rect>
    </property>
    <property name="whatsThis">
     <string>Проверить</string>
    </property>
    <property name="styleSheet">
     <string notr="true">border-image: url(:/ok.png);</string>
    </property>
    <property name="text">
     <string/>
    </property>
   </widget>
   <widget class="QLineEdit" name="input">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>90</y>
      <width>491</width>
      <height>31</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">border-radius: 10px;
font: 9pt &amp;quot;Tahoma&amp;quot;;</string>
    </property>
   </widget>
   <widget class="QLabel" name="name">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>10</y>
      <width>341</width>
      <height>41</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>Tahoma</family>
      <pointsize>14</pointsize>
      <weight>50</weight>
      <italic>false</italic>
      <bold>false</bold>
     </font>
```

```
    </property>
    <property name="styleSheet">
     <string notr="true">font: 14pt &amp;quot;Tahoma&amp;quot;;</string>
    </property>
    <property name="text">
     <string>LR2 | Postfix Expression Analyzer</string>
    </property>
   </widget>
   <widget class="QLabel" name="output">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>280</y>
      <width>491</width>
      <height>241</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">background-color: white;
border-radius: 10px;
padding: 10px;
font: 10pt &amp;quot;Tahoma&amp;quot;;</string>
    </property>
    <property name="text">
     <string/>
    </property>
   </widget>
   <widget class="QLabel" name="message">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>60</y>
      <width>241</width>
      <height>21</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>Tahoma</family>
      <pointsize>10</pointsize>
      <weight>50</weight>
      <italic>false</italic>
      <bold>false</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">font: 10pt &amp;quot;Tahoma&amp;quot;;</string>
    </property>
    <property name="text">
     <string>Введите выражение:</string>
    </property>
   </widget>
   <widget class="Line" name="line">
    <property name="geometry">
     <rect>
      <x>0</x>
```

```
    <y>230</y>
    <width>611</width>
    <height>20</height>
   </rect>
  </property>
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
 </widget>
 <widget class="QPushButton" name="file">
  <property name="geometry">
   <rect>
    <x>530</x>
    <y>30</y>
    <width>45</width>
    <height>45</height>
   </rect>
  </property>
  <property name="whatsThis">
   <string>Проверить</string>
  </property>
  <property name="styleSheet">
   <string notr="true">border-image: url(:/open.png);</string>
  </property>
  <property name="text">
   <string/>
  </property>
 </widget>
 <widget class="QPushButton" name="save">
  <property name="geometry">
   <rect>
    <x>530</x>
    <y>260</y>
    <width>45</width>
    <height>45</height>
   </rect>
  </property>
  <property name="whatsThis">
   <string>Проверить</string>
  </property>
  <property name="styleSheet">
   <string notr="true">border-image: url(:/savef.png);</string>
  </property>
  <property name="text">
   <string/>
  </property>
 </widget>
 <widget class="QLabel" name="message_2">
  <property name="geometry">
   <rect>
    <x>20</x>
    <y>250</y>
    <width>241</width>
    <height>21</height>
   </rect>
  </property>
```

```
          <property name="font">
           <font>
            <family>Tahoma</family>
            <pointsize>10</pointsize>
            <weight>50</weight>
            <italic>false</italic>
            <bold>false</bold>
           </font>
          </property>
          <property name="styleSheet">
           <string notr="true">font: 10pt &quot;Tahoma&quot;;</string>
          </property>
          <property name="text">
           <string>Вывод:</string>
          </property>
         </widget>
         <widget class="QCheckBox" name="logs">
          <property name="geometry">
           <rect>
            <x>350</x>
            <y>135</y>
            <width>171</width>
            <height>45</height>
           </rect>
          </property>
          <property name="styleSheet">
           <string notr="true">font: 10pt &quot;Tahoma&quot;;</string>
          </property>
          <property name="text">
           <string>Доп. информация</string>
          </property>
         </widget>
         <widget class="QLabel" name="label">
          <property name="geometry">
           <rect>
            <x>20</x>
            <y>130</y>
            <width>271</width>
            <height>101</height>
           </rect>
          </property>
          <property name="text">
           <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
style=&quot;  font-weight:600;&quot;&gt;Формат  операторов&lt;/span&gt;:
&lt;br/&gt;OR:           +&lt;span           style=&quot;           font-
weight:600;&quot;&gt;&lt;br/&gt;&lt;/span&gt;AND:  *&lt;span  style=&quot;
font-weight:600;&quot;&gt;&lt;br/&gt;&lt;/span&gt;NOT:          !&lt;span
style=&quot; font-weight:600;&quot;&gt;&lt;br/&gt;&lt;/span&gt;EQUIVALENT:
=      (только      2      операнда)&lt;span      style=&quot;      font-
weight:600;&quot;&gt;&lt;br/&gt;Формат  переменных:&lt;/span&gt;  символ
или цифра&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
          </property>
         </widget>
        </widget>
        <widget class="QMenuBar" name="menuBar">
         <property name="geometry">
```

```
    <rect>
     <x>0</x>
     <y>0</y>
     <width>600</width>
     <height>26</height>
    </rect>
   </property>
  </widget>
  <widget class="QToolBar" name="mainToolBar">
   <attribute name="toolBarArea">
    <enum>TopToolBarArea</enum>
   </attribute>
   <attribute name="toolBarBreak">
    <bool>false</bool>
   </attribute>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```