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

ОТЧЁТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Вычисление постфиксного арифметического выражения с использованием иерархических списков

Студент гр. 8381 _____ Почаев Н.А.
Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы

Формирование практических навыков работы с иерархическими списками и их рекурсивной обработки. Изучение и практическое применение постфиксной записи арифметических выражений.

Основные теоретические положения

Линейный список представляет собой способ организации последовательности однотипных элементов, при котором каждый элемент, кроме первого, имеет одного предшественника (предыдущий элемент) и каждый элемент, кроме последнего, имеет одного преемника (следующий элемент). Доступ к каждому элементу списка можно получить, последовательно продвигаясь по списку от элемента к элементу. Схематично изображённый линейный список представлен на рисунке 1.

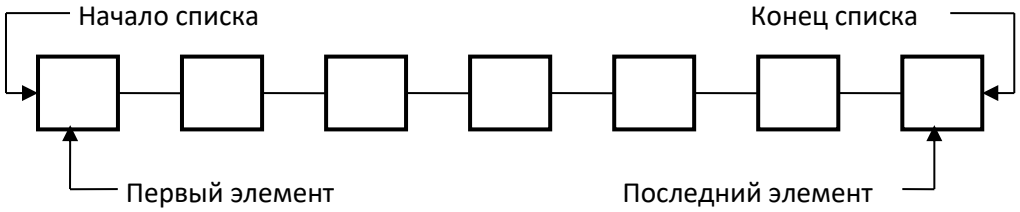

Рис. 1 - Модельное представление списка, обрабатываемого линейно

В классической реализации линейные списки обрабатываются последовательно и идёт чёткое определение текущего элемента, в реализации же, использованной в лабораторной работе, применяется подход рекурсивной обработки списка, который логически представлен внутри, как показано на рисунке 2: список делится на отдельный элемент - голову (первый элемент), а все остальные представляется ”хвостом” .

Кроме того, использование рекурсивный алгоритмов обусловлено использование не стандартного, а иерархического списка, где каждая функция должна выполняться на отдельном уровне ветвления. Главным отличием данной структуры данных является возможность назначения элемента ссылкой на другой

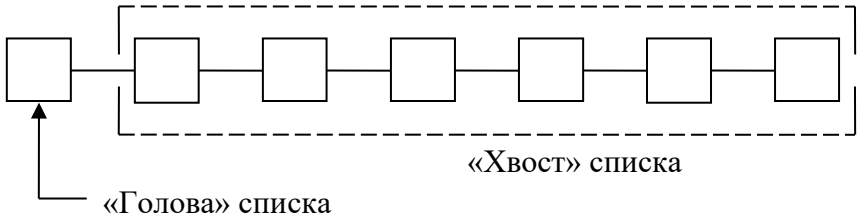

Рис. 2 - Модельное представление списка, обрабатываемого рекурсией

иерархический список, за счёт чего и реализуется нелинейность хранения данных и обработки. Схематическое представление списка `(a (() (b c) d) e)` представлено на рисунке 3.

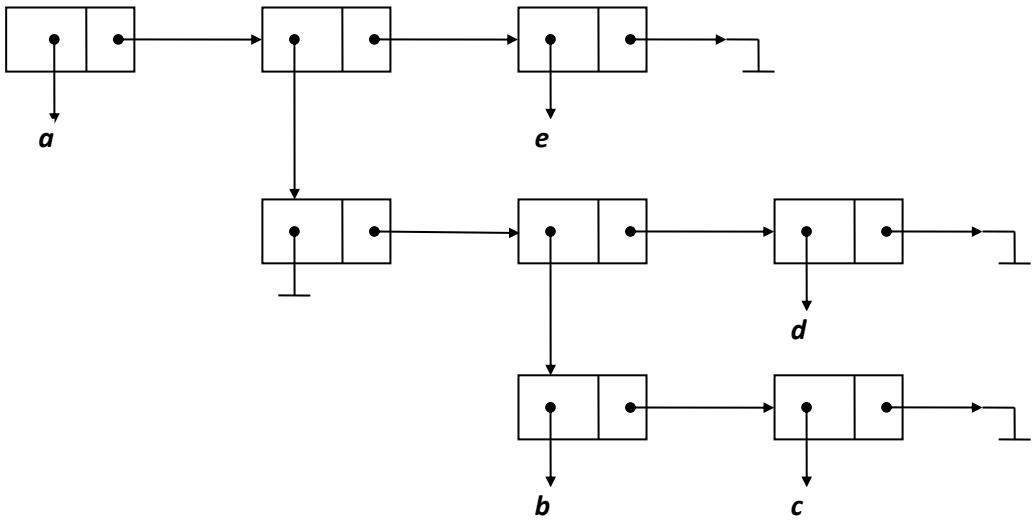

Рис. 3 - Модельное представление списка, обрабатываемого рекурсией

Постановка задачи

Арифметическое (т.е. используются операции +, -, /, *) выражение представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в постфиксной форме (<arguments><operation>). Константами могут быть положительные, отрицательные числа и 0: 5, +5, -5, +0, -0, 0.

На вход программе подаётся выражение в виде строки (необходимо реализовать его перевод в список) и набор аргументов в форме:

$$((x_1c_1)(x_2c_2)\dots(x_ic_i)),$$

где x_i - переменная, а c_i - её значение (константа).

Константы и переменные - это атомы. Атом следует считать особым случаем выражения.

Выполнение работы

Реализация шаблона класса стек

Перед началом реализации непосредственного функционала программы был создан шаблон класса стека `template <typename StackType>`, где за счёт использования возможностей стандарта C++17 реализована поддержка любых примитивных и пользовательских типов данных. Аналогично были реализованы базовые методы данного класса, такие как: `push()`, `pop()`, `getStackSize`, `getTop()`, `peek()` (возвращает n-ый элемент от начала стека) и `printStack()`. Для оптимизации работы данные методы имеют `inline` реализацию.

Реализация класса узла

Для работы со стеком в виде его минимальной составляющей части (атома иерархического списка) был реализован класс `Node`, приватные поля которого содержат флаг содержимого данного экземпляра - `isAtom`, указатель на следующий элемент - `next` и непосредственно объединение (`union content`) с тремя типами возможного содержания: `expr` - указатель на ответвлённый иерархический список, `character` - символ, хранящий символ арифметической операции или наименование переменной, `number` - числовое значение.

Для корректного создания каждого нового узла используется перегрузка конструктора класса с изменяемым типом входных параметров. Для решения проблемы совпадения типа данных `char` у знака и переменной используется передача дополнительного логического параметра, `warning` от неиспользования которого в теле метода подавляется при помощи директивы `Q_UNUSED`.

Для взаимодействия с внешними методами класса `HierarchicalList` реа-

лизован ряд публичных (public) методов, большая часть из которых представляет собой реализацию классических геттеров и сеттеров. Примерами таких методов могут служить getContNumb() и getNext(), отдельно реализован метод получения значения флага - getAtomValue().

Реализация класса списка

Для осуществления трансформации входной строки в иерархический список и последующей обработки реализован класс HierarchicalList. Приватными полями которого являются: Stack<Node*> stackNodes - стек с узлами создаваемого списка, head - указатель на "голову", currentPtr - указатель на текущий рабочий элемент (используется для минимизации количества передуваемых параметров при работе со стеком), expression - исходная строка, записанная в пользовательский тип данных ExpStr (описание представлено ниже) и std::map<char, int> varValues - словарь значений переменных, полученных от пользователя.

Для обеспечения корректной работы и переносимости в public методах класса реализованы: ряд перегруженных конструкторов, метод соединения нового узла с предыдущим connectWithPrev(), метод создания словаря из входной строки значений переменных makeValDict(), метод чтения числа (для считывания значений из нескольких цифр) readNumber. Отдельно реализованы перегрузки метода записи логов - printLogMess() и обращения к методам класса ExpStr. Реализация основных методов лабораторной работы по созданию иерархического списка - makeHierarchichalList и проверки выражения параллельно с вычислением - checkExprCorrectionAndCount описаны позднее. Данный класс кроме того предоставляет static метод логирования ошибки и вывода соответствующего сообщения для пользователя - globalProblemMes().

Реализация класса хранения и обработки строки

Для корректной реализации инкапсуляции все функции по обработки входной строки (хранимой в единственном приватном поле str) реализованы в клас-

се `ExpStr`. Таким образом, перед началом основной работы производится удаление пустых скобок (экономия памяти и времени работы на узлах иерархического списка) методов `deleteEmptyBrackets()` и ”лёгкая” проверка выражения на корректность: равное количество открывающих и закрывающих скобок, отсутствие лишних знаков. Разрешённый знаки (арифметических операций) хранятся в `public static` поле `allowableSignes`. Аналогично данный класс предоставляет метод доступа к символу строки по переданному индексу - `getSymbByIndex`. Основываясь на частоте его вызова, он также имеет `inline` организацию.

Реализация метода `makeHierarchichalList()`

Данный метод рекурсивно выполняет парсинг входной строки в иерархический список, возвращаемое значение является ссылкой типа `Node+` на голову нового созданного ответвления (в том числе основного). В теле метода в цикле осуществляется проверка каждого нового символа в строке и в зависимости от его типа либо совершается рекурсивный вызов метода (открывающая скобка ’(’), либо подъём на уровень выше (т.е. завершение построение ответвления при встрече закрывающей скобки ’)’). В случае совпадения текущего символа с цифрой осуществляется чтение числа и создание узла с соответствующим значением, аналогичное происходит при встрече переменной. При чтение знака операции происходит дополнительная проверка, что знак ’-’ не принадлежит числу, иначе оно обрабатывается по своей схеме создания узла.

Реализация метода `checkExprCorrectionAndCount()`

Данный метод выполняет рекурсивное вычисление каждого ответвления иерархического списка по упрощённой польской нотации (согласно условию, атом списка - однословное выражение арифметической операции). Для каждого конкретного случае создаётся отдельный стек с типом данных `double`, куда в зависимости от типа текущего узла помещается значения (число), извлекается два верхних элемента и выполняется операция (знак). Также отдельно происходит проверка на наличе необходимого количества чисел в стеке, а в случае с

переменными на наличе их значений в словаре, и при успешном нахождении соответствующая подстановка. В конце всех операций, определение которого происходит по отсутствию нового элемента в следующем узле, проверяется наличие одного числа в стеке - результата и логической переменной `check` на наличие различных ошибок при вычислении (связанных с корректностью исходной строки). Результирующее число в результате хранится в переменной `res` в теле главной рабочей функции программы (она передаётся в метод по ссылке).

Дополнительные особенности реализации

В коде программы к данной лабораторной работе реализован графический интерфейс с двумя входными строками для пользователя, информирование его об успехе операции и различных ошибках осуществляется через открытие диалоговых окон. Данный функционал реализован на базе фреймворка Qt. Также внутри программы осуществляется отключаемое логирование в файл `log.txt` и вывод конечного результата. Аналогичный вывод осуществляется в консоль.

Тестирование

Результаты тестирования программы приведены в таблице

Входная строка	**Входные переменные**	**Результат**
Только числовые значения		
2 2 +	-	4
(2 3 +) 5 -	-	0
((3 6 *) 25 -) 9 /	-	1,28
(2 3 +) (4 9 -) *	-	25
((2 3 +)(4 5 /)+)(9 3 *) -	-	20,75
-5 6 +	-	1
7 -9 +	-	2

Переменные		
a b +	((a 2) (b 3))	5
((3 6 *) 25 -) x /	((x 9))	1,28
Некорректные входные данные		
3 + 3	-	Something wrong with expression!
12 (1 1 1 +) -11 ()	-	Something wrong with expression!
a b -	((a 3))	Error: variable with no value was found!

Таблица 1 - Результаты тестирования

Исходя из результатов тестирования можно сделать вывод, что программа корректно производит вычисления, несмотря на уровень вложенности иерархического списка и на тип операций. Аналогично происходит обработка ошибок входных данных: сообщение об этом выводится пользователю и программа продолжает работать.

Выводы

В процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы по построению и рекурсивной обработке иерархического списка. Аналогично были созданы необходимые классы и методы для вычисления входного выражения в постфиксной записи.

Написание работы производилось на базе системы Ubuntu 18.04 в IDE QtCreator с отдельнном настроенным файлом конфигурации для использования стандарта C++17. Графическое оформление реализовано при помощи библиотеки ColinDuqesnoy/QDarkStyleSheet по открытой лицензии и графических

иконок из свободных источников.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: expstr.h

```
#ifndef EXPSTR_H
#define EXPSTR_H

#include ”basicheaders.h”

class ExpStr
{
private:
    std::string str;

public:
    ExpStr();
    ExpStr(std::string expression);
    ~ExpStr();
    void deleteEmptyBrackets();
    static const std::string allowableSignes;
    int checkExpCorrectionLight();  // 0 - , 1 -
    inline char getSymbByIndex(int index)
    {
        return str[index];
    }
    inline int getStrLength()
    {
        return str.length();
    }
    std::string getSubStr(int beginPos, int CurrentPos);
};

#endif // EXPSTR_H
```

Название файла: node.h

```
#ifndef NODE_H
#define NODE_H


class Node
{
```

```
public:
    Node();
    Node(char sign);
    Node(int number);
    Node(bool flag, char var);
    Node(Node* expr);
    int getAtomValue();
    int getContNumb();
    char getContChar();
    Node* getNext();
    void setNext(Node* next);
    void setContExprNext(Node* next);
    Node* getContExprVal();

private:
    int isAtom;  // 0 - ; 1 - ; 2 - ; 3 -
    Node* next;
    union {
        Node* expr;
        char character;
        int number;
    } content;
};

#endif // NODE_H
```

Название файла: hierarchicallist.h

```
#ifndef HIERARCHICALLIST_H
#define HIERARCHICALLIST_H

#define DEBUG 1

#include ”basicheaders.h”
#include ”basicqtheader.h”
#include ”stackList.h”
#include ”node.h”
#include ”expstr.h”

class HierarchicalList
{
private:
    Stack<Node*> stackNodes;    //
```

```
    Node *head;
    Node *currentPtr;           //
    ExpStr *expression;         //
    std::map<char, int> varValues;

public:
    HierarchicalList();
    HierarchicalList(std::string str);
    HierarchicalList(std::string str, std::string vars);
    ~HierarchicalList();

    /**
     * @brief globalProblemMes :
     * @param message :
     */
    static void globalProblemMes(std::string message);

    /**
     * @brief connectWithPrev :
     * 1.     .,
     *     ;
     * 2.
     *  ,
     *   ””   .
     * 3.      ””
     *   -  ””,    ,
     *    next.
     * @param prevEl :
     * @param nextEl :
     */
    void connectWithPrev(Node* prevEl, Node* nextEl);

    std::map<char, int> makeValDict(std::string varInputLine);

    int readNumber(int& currentSymbNumb);

    Node* makeHierarchichalList(int deep, int& currPos);

    void exprDelEmptBr();

    int exprLightCheck();  // 0 - , 1 -
```

```
    void printLogMess(std::string message);

    void printLogMess(std::string message, char character);

    void printLogMess(std::string message, int numb);

    void printLogMess(std::string message, double numb);

    bool printList(Node* head, int deep, int nodeNumber);

    void setNewHead(Node* head);

    Node* getHead();

    double checkExprCorrectionAndCount(bool& check);

    void setCurrNodeToHead();
};

#endif // HIERARCHICALLIST_H
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsView>
#include <QLabel>

#include ”basicqtheader.h”

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
```

```
private slots:
    void on_fileChoose_clicked();

    void on_inputString_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

Название файла: stackList.h

```
#ifndef STACKLIST_H
#define STACKLIST_H

#include ”basicheaders.h”
#include ”expstr.h”

template <typename StackType>
class Stack
{
private:
    size_t top;                     //
    std::vector<StackType> data;    //

public:
    Stack();
    ~Stack();

    inline void push(const StackType newEl);
    inline StackType pop();
    inline int getStackSize() const;
    inline int getTop() const;
    inline const StackType &peek(int) const;  // n-
    inline void printStack();
};

/**
 *   ,
 *    (   ),
 *    ,
```

```
 *     .
 */

template <typename StackType>
Stack<StackType>::Stack()
{
    top = 0;
}

template <class StackType>
Stack<StackType>::~Stack()
{
    data.clear();
    top = 0;
}

template <typename StackType>
inline void Stack<StackType>::push(StackType newEl)
{
    data.push_back(newEl);
    top++;
}

template <typename StackType>
inline StackType Stack<StackType>::pop()
{
    assert(top > 0);
    StackType tmp = data[getTop() - 1];
    data.erase(data.begin() + (--top));

    return tmp;
}

template <typename StackType>
inline const StackType &Stack<StackType>::peek(int num) const
{
    assert (num <= top);

    return data[top - num];
}

template <typename StackType>
```

```
inline void Stack<StackType>::printStack()
{
    for (int ix = top - 1; ix >= 0; ix--)
        std::cout << '|' << data[ix] << std::endl;
}

template <typename StackType>
inline int Stack<StackType>::getStackSize() const
{
    return data.size();
}

template <typename StackType>
inline int Stack<StackType>::getTop() const
{
    return top;
}

#endif // STACKLIST_H
```

Название файла: workingfunc.h

```
#ifndef WORKINGFUNC_H
#define WORKINGFUNC_H

void mainWorkFunc(std::string expression, std::string varValues);

#endif // WORKINGFUNC_H
```

Название файла: expstr.cpp

```
#include "expstr.h"
#include "hierarchicallist.h"

const std::string ExpStr::allowableSignes = "+-*/";

ExpStr::ExpStr()
{
    str.clear();
    str = "\0";
}

ExpStr::ExpStr(std::string str): ExpStr()
{
```

```
    this->str = str;
}

ExpStr::~ExpStr()
{
    str = nullptr;
}

void ExpStr::deleteEmptyBrackets()
{
    static const std::regex rgx_pattern("\\(\\s*\\)");
    std::string res;
    res.reserve(str.size());
    std::regex_replace(std::back_insert_iterator<std::string>(res), \
                       str.cbegin(), str.cend(), rgx_pattern, "");
    str = res;
}

int ExpStr::checkExpCorrectionLight()
{
    if (str[0] == '\0') {
        HierarchicalList::globalProblemMes("The expression is empty!\n");
        return 1;
    }

    int openBrackets = 0, closeBrackets = 0;
    for (std::string::size_type i = 0; i < str.length(); i++) {
        if (str[i] == '(') {
            openBrackets++;
            continue;
        }
        if (str[i] == ')'){
            closeBrackets++;
            continue;
        }
        if (allowableSignes.find(str[i]) == std::string::npos && \
                !isdigit(str[i]) && !isalpha(str[i]) && str[i] != ' ') {
            HierarchicalList::globalProblemMes("Error: Incorrect input symbol!");
            return 1;
        }
    }
```

```
    if(openBrackets != closeBrackets) {
        HierarchicalList::globalProblemMes(”ERROR: Invalid number of brackets!”);
        return 1;
    }

    return 0;
}

std::string ExpStr::getSubStr(int beginPos, int CurrentPos)
{
    return str.substr(beginPos, \
                      CurrentPos - beginPos);
}
```

Название файла: hierarchicallist.cpp

```
#include ”hierarchicallist.h”
#include ”basicheaders.h”
#include ”basicqtheader.h”

HierarchicalList::HierarchicalList()
{
    head = nullptr;
    currentPtr = nullptr;
    expression = new ExpStr();
}

HierarchicalList::HierarchicalList(std::string str): \
    HierarchicalList()
{
    expression = new ExpStr(str);
}

HierarchicalList::HierarchicalList(std::string str, std::string val): \
    HierarchicalList()
{
    expression = new ExpStr(str);
    varValues = makeValDict(val);
}

HierarchicalList::~HierarchicalList()
{
    head = nullptr;
```

```
    currentPtr = nullptr;
    /**  ,     ,
     *  expression
     *  , ..    .
     * delete  , ..   .
     */
}

void HierarchicalList::globalProblemMes(std::string message)
{
    QFile log("../log.txt");
    log.open(QIODevice::Append | QIODevice::Text);
    QTextStream logStream(&log);

    std:: cout << '\n' << message << '\n';
    logStream << '\n' << QString::fromStdString(message) << '\n';
    QMessageBox::warning(0, "Mistake", QString::fromStdString(message));

    log.close();
}

void HierarchicalList::connectWithPrev(Node* prevEl, Node* nextEl)
{
    if (!prevEl) {
        head = nextEl;
    } else {
        if (prevEl->getAtomValue()) {
            prevEl->setNext(nextEl);
        } else if (!prevEl->getContExprVal()) {
            prevEl->setContExprNext(nextEl);
        } else {
            prevEl->setNext(nextEl);
        }
    }
}

int HierarchicalList::readNumber(int& currentSymbNumb)
{
    unsigned int beginPos = currentSymbNumb;
    while (expression->getSymbByIndex(currentSymbNumb) >= '0' && \
           expression->getSymbByIndex(currentSymbNumb) <= '9') {
        currentSymbNumb++;
```

```
    }

    std::string tmpStrNumb = expression->getSubStr(beginPos, currentSymbNumb);
    tmpStrNumb[tmpStrNumb.length() + 1] = '\0';

    return std::stoi(tmpStrNumb);
}

std::map<char, int> HierarchicalList::makeValDict(std::string varInputLine)
{
    std::map<char, int> valDict;
    const std::regex reg(R"(\s*\(\s*([a-z])\s*([1-9][0-9]*)\s*\)\s*)");
    std::smatch match;
    for (std::sregex_iterator i = std::sregex_iterator( \
              varInputLine.begin(), varInputLine.end(), reg); \
         i != std::sregex_iterator(); ++i) {
        std::smatch match = *i;
        valDict.insert(std::pair<char, int>(match[1].str()[0], \
                       std::stoi(match[2].str().c_str())));
    }

    return valDict;
}

void HierarchicalList::printLogMess(std::string message)
{
    QFile log("../log.txt");
    log.open(QIODevice::Append | QIODevice::Text);
    QTextStream logStream(&log);

    std::cout << message;
    logStream << QString::fromStdString(message);

    log.close();
}

void HierarchicalList::printLogMess(std::string message, char character)
{
    QFile log("../log.txt");
    log.open(QIODevice::Append | QIODevice::Text);
    QTextStream logStream(&log);
```

```
    std::cout << message << character << '\n';
    logStream << QString::fromStdString(message) << character << '\n';

    log.close();
}

void HierarchicalList::printLogMess(std::string message, int numb)
{
    QFile log("../log.txt");
    log.open(QIODevice::Append | QIODevice::Text);
    QTextStream logStream(&log);

    std::cout << message << numb << '\n';
    logStream << QString::fromStdString(message) << numb << '\n';

    log.close();
}

void HierarchicalList::printLogMess(std::string message, double numb)
{
    QFile log("../log.txt");
    log.open(QIODevice::Append | QIODevice::Text);
    QTextStream logStream(&log);

    std::cout << message << numb << '\n';
    logStream << QString::fromStdString(message) << numb << '\n';

    log.close();
}

bool HierarchicalList::printList(Node* head, int deep, int nodeNumber)
{
    bool isCorrect = 1;
    bool zeroFound = false;
    QFile log("../log.txt");
    log.open(QIODevice::Append | QIODevice::Text);
    QTextStream logStream(&log);
    for (int i = 0; i < deep; i++) {
        std::cout << "   ";
        logStream << QString::fromStdString("   ");
    }
    if (nodeNumber) {
```

```
    std::cout << "Node: " << nodeNumber << " ";
    logStream << QString::fromStdString("Node: ") << nodeNumber << \
                QString::fromStdString(" ");
}
Node* ListHead = nullptr;
Node* curList = nullptr;
Node* ptr = head;
static int NN = 1;
int minNN = NN;
while (ptr) {
    if (ptr->getAtomValue() == 1) {
        std::cout << ptr->getContNumb() << " ";
        logStream << ptr->getContNumb() << QString::fromStdString(" ");
        if (!ptr->getContNumb() && ptr != head) zeroFound = true;
        if (!ptr->getNext()) isCorrect = 0;
    } else if (ptr->getAtomValue() == 2) {
        std::cout << ptr->getContChar() << " ";
        logStream << ptr->getContChar() << QString::fromStdString(" ");
        if (zeroFound && ptr->getContChar() == '/') isCorrect = 0;
        if (ptr->getNext()) isCorrect = 0;
    } else {
        std::cout << "(Node ) -> " << NN;
        logStream << QString::fromStdString("(Node ) -> ") << NN++;
        Node* tmp = new Node();
        tmp->setContExprNext(ptr);
        if (!curList)
            ListHead = tmp;
        else
            curList->setContExprNext(tmp);
        curList = tmp;
    }
    ptr = ptr->getNext();
}
std::cout << "NULL";
logStream << QString::fromStdString("NULL");
if (zeroFound) {
    std::cout << "Zero is found!\n";
    logStream << QString::fromStdString("Zero was found!\n");
}
std::cout << "\n\n";
logStream << QString::fromStdString("\n\n");
curList = ListHead;
```

```
    bool isCorrectRec = 1;
    log.close();
    while (curList) {
        isCorrectRec &= printList(curList->getContExprVal()->getContExprVal(), \
                                  deep + 1, minNN);
        curList = curList->getNext();
        minNN++;
    }
    curList = ListHead;
    while(curList) {
        ListHead = curList;
        curList = curList->getNext();
        delete curList;
    }
    return isCorrect && isCorrectRec;
}

void HierarchicalList::exprDelEmptBr()
{
    expression->deleteEmptyBrackets();
}

int HierarchicalList::exprLightCheck()
{
    if (expression->checkExpCorrectionLight())
        return 1;
    else
        return 0;
}

void HierarchicalList::setCurrNodeToHead()
{
    currentPtr = head;
}

void HierarchicalList::setNewHead(Node *head)
{
    this->head = head;
}

Node* HierarchicalList::getHead()
{
```

```
    return head;
}

Node* HierarchicalList::makeHierarchichalList(int deep, int& currPosStr)
{
    Node* currHead = nullptr;

    while(expression->getStrLength() != currPosStr) {
        if (expression->getSymbByIndex(currPosStr) == '(')
        {
            currPosStr++;
            if (DEBUG) {printLogMess("New node was created!\n");};
            Node* tmp = currentPtr;
            Node* newNode = new Node(makeHierarchichalList(deep++, \
                                                           currPosStr));

            if (!currHead)
                currHead = newNode;
            currentPtr = tmp;
            connectWithPrev(currentPtr, newNode);
            stackNodes.push(newNode);
            currentPtr = newNode;
        } else if ((expression->allowableSignes.find(expression->\
                    getSymbByIndex(currPosStr)) != std::string::npos))
        {
            if (DEBUG) {printLogMess("Sign is found: ", \
                                     expression->getSymbByIndex(currPosStr));}
            //
            if (expression->getSymbByIndex(currPosStr) == '-' && \
                    isdigit(expression->getSymbByIndex(currPosStr+1)))
            {
                        currPosStr++;
                        int tmpNumber = readNumber(currPosStr);
                        tmpNumber *= -1;
                        Node* newNode = new Node(tmpNumber);
                        if (!currHead)
                            currHead = newNode;
                        if (DEBUG) { printLogMess("Number is found: ", tmpNumber);}
                        connectWithPrev(currentPtr, newNode);
                        currentPtr = newNode;
            } else
            {
```

```
                Node* newNode = new Node(expression->getSymbByIndex(currPosStr));
                if (!currHead)
                    currHead = newNode;
                currPosStr++;
                connectWithPrev(currentPtr, newNode);
                currentPtr = newNode;
            }
        } else if (isdigit(expression->getSymbByIndex(currPosStr)))
        {
            int tmpNumber = readNumber(currPosStr);
            if (DEBUG) { printLogMess("Number is found: ", tmpNumber);}
            Node* newNode = new Node(tmpNumber);
            if (!currHead)
                currHead = newNode;
            connectWithPrev(currentPtr, newNode);
            currentPtr = newNode;
        } else if (isalpha(expression->getSymbByIndex(currPosStr)))
        {
            if (DEBUG) { printLogMess("Variable is found: ", \
                        expression->getSymbByIndex(currPosStr));}
            Node* newNode = new Node(true, expression->getSymbByIndex(currPosStr));
            if (!currHead)
                currHead = newNode;
            currPosStr++;
            connectWithPrev(currentPtr, newNode);
            currentPtr = newNode;
        } else if (expression->getSymbByIndex(currPosStr) == ')') {
            if (DEBUG) {printLogMess("The end of the current node!\n");}
            currPosStr++;
            return currHead;
        } else currPosStr++;
    }
    if (expression->getSymbByIndex(currPosStr) == ')') {
        if (DEBUG) {printLogMess("The end of the current node!\n");}
        currentPtr->setNext(nullptr);
        currPosStr++;
        return currHead;
    }

    return currHead;
}
```

```
// 0 - ; 1 - ; 2 - ; 3 -   //

double HierarchicalList::checkExprCorrectionAndCount(bool& check)
{

    Stack<double> resStack;
    double a = 0;
    double b = 0;

    while(currentPtr) {
        if (currentPtr->getAtomValue() == 1) {
            resStack.push(currentPtr->getContNumb());
            currentPtr = currentPtr->getNext();
        } else
        if (currentPtr->getAtomValue() == 2) {
            if (resStack.getTop() != 2) {
                    check = false;
                    return 0;
                }
            switch(currentPtr->getContChar()) {
                case '-':
                    a = resStack.pop();
                    b = resStack.pop();
                    resStack.push(a - b);
                    break;
                case '+':
                    a = resStack.pop();
                    b = resStack.pop();
                    resStack.push(a + b);
                    break;
                case '*':
                    a = resStack.pop();
                    b = resStack.pop();
                    resStack.push(a * b);
                    break;
                case '/':
                    a = resStack.pop();
                    b = resStack.pop();
                    if (b == 0) {
                        check = false;
                        return 0;
                    }
```

```
                    resStack.push(((a * 1.0) / b));
                    break;
                }
                currentPtr = currentPtr->getNext();
            } else
            if (currentPtr->getAtomValue() == 3) {
                if (varValues.find(currentPtr->getContChar()) == varValues.end()) {
                    globalProblemMes("Error: variable with no value was found!\n");
                    check = false;
                    return 0;
                } else {
                    resStack.push(varValues[currentPtr->getContChar()]);
                    currentPtr = currentPtr->getNext();
                }
            } else
            if (currentPtr->getAtomValue() == 0)
            {
                Node* tmpPtr = currentPtr;
                currentPtr = currentPtr->getContExprVal();
                double tmp = checkExprCorrectionAndCount(check);
                if (check == false) {
                    return 0;
                }
                resStack.push(tmp);
                currentPtr = tmpPtr->getNext();
            }
        }

        if (check == false || resStack.getStackSize() != 1)
            return 0;
        else
            return resStack.pop();
    }
```

Название файла: main.cpp

```
#include "mainwindow.h"
#include "visualfunc.h"
#include "basicqtheader.h"
#include "basicheaders.h"
#include <QApplication>

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

```
{
    QApplication a(argc, argv);
    MainWindow w;
    connectVisualLib();
    w.show();

    return a.exec();
}
```

Название файла: mainwindow.cpp

```
#include ”mainwindow.h”
#include ”ui_mainwindow.h”
#include ”basicqtheader.h”
#include ”workingfunc.h”

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

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

void MainWindow::on_fileChoose_clicked()
{
    QString fileName = QFileDialog::getOpenFileName(this,
    tr(”Open TXT File”),  QDir::homePath(),
    tr(”TXT text (*.txt);;All Files (*)”));
    if (fileName == nullptr)
    {
        QMessageBox::warning(this,
        ”Warning!”,
        ”Please choose text file for work.”);
        return;
    }

    QString logFilePath = ”/”;
```

```
    QFile file(fileName);
    QString fileString;

    //    log
    QFile log("log.txt");
    log.open(QIODevice::ReadWrite);
    QTextStream logStream(&log);
    logStream << "************\n";
    logStream << "* LOG FILE *\n";
    logStream << "************\n\n";
    log.close();

    //
    if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        fileString = stream.readLine();
    }

    file.close();

    std::string eqInputLine = fileString.toUtf8().constData();

    // TODO: CALLFUNC
}

void MainWindow::on_inputString_clicked()
{
    // convert QString to std::string, using UTF-8
    std::string eqInputLine = \
            ui->eqInputLine->text().toUtf8().constData();
    if (eqInputLine.empty()) {
        QMessageBox::warning(this,
        "Warning!",
        "Please input the expression.");
        return;
    }

    std::string varInputLine = \
            ui->variableInputLine->text().toUtf8().constData();

    QFile log("../log.txt");
    log.open(QIODevice::ReadWrite | \
```

```
            QIODevice::Truncate | QIODevice::Text);
    QTextStream logStream(&log);
    logStream << ”************\n”;
    logStream << ”* LOG FILE *\n”;
    logStream << ”************\n\n”;
    log.close();

    mainWorkFunc(eqInputLine, varInputLine);
}
```

Название файла: node.cpp

```
#include ”node.h”
#include ”basicheaders.h”
#include ”basicqtheader.h”

/**
 * @brief Node::Node :
 */
Node::Node()
{
    next = nullptr;
    isAtom = 0;
    content.expr = nullptr;
    content.character = ’\0’;
    content.number = 0;
}

Node::Node(int number): Node()
{
    isAtom = 1;
    content.number = number;
}

Node::Node(char sign): Node()
{
    isAtom = 2;
    content.character = sign;
}

Node::Node(bool flag, char var): Node()
{
    /** ,
```

```
     *
     */
    Q_UNUSED(flag);
    isAtom = 3;
    content.character = var;
}

Node::Node(Node* expr): Node()
{
    content.expr = expr;
}

int Node::getAtomValue()
{
    return isAtom;
}

int Node::getContNumb()
{
    return content.number;
}

char Node::getContChar()
{
    return content.character;
}

Node* Node::getNext()
{
    return next;
}

void Node::setNext(Node* next)
{
    this->next = next;
}

void Node::setContExprNext(Node* next)
{
    this->content.expr = next;
}
```

```
Node* Node::getContExprVal()
{
    return this->content.expr;
}
```

Название файла: visualfunc.cpp

```
#ifndef VISUALFUNC_CPP
#define VISUALFUNC_CPP
#include <QFile>
#include <QTextStream>
#include <QApplication>
#include <iostream>

void connectVisualLib()
{
    QFile f(":qdarkstyle/style.qss");
    if (!f.exists())
    {
        std::cout << "Unable to set stylesheet, file not found\n";
    }
    else
    {
        f.open(QFile::ReadOnly | QFile::Text);
        QTextStream ts(&f);
        qApp->setStyleSheet(ts.readAll());
    }
}
#endif // VISUALFUNC_CPP
```

Название файла: workingfunc.cpp

```
#include "basicheaders.h"
#include "basicqtheader.h"
#include "hierarchicallist.h"

void mainWorkFunc(std::string expression, std::string varValues)
{
    HierarchicalList* expList = new HierarchicalList(expression, varValues);

    expList->exprDelEmptBr();
    if (expList->exprLightCheck())
        return;
```

```
    int currPos = 0;
    expList->setNewHead(expList->makeHierarchichalList(0, currPos));
    if(!expList->getHead()) {
        expList->globalProblemMes("Error: not enought numbers!\n");
        return;
    }

    expList->setCurrNodeToHead();
    bool check = true;
    double res = expList->checkExprCorrectionAndCount(check);
    if (check == true) {
        expList->printLogMess("Expression correct and res is: ", res);
        std::stringstream resStr;
        resStr << "Expression correct and res is: " << res;
        QMessageBox::information(0,\
                                 "Success", \
                                 QString::fromStdString(resStr.str()));
    } else {
        expList->globalProblemMes("Something wrong with expression!\n");
    }
}
```

Название файла: basicheaders.h

```
#ifndef BASICHEADERS_H
#define BASICHEADERS_H

#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>
```

```
#endif // BASICHEADERS_H
```

Название файла: basicqtheader.h

```
#ifndef UNIVERSALQTHEADER_H
#define UNIVERSALQTHEADER_H

#include ”mainwindow.h”
#include ”ui_mainwindow.h”

#include <experimental/filesystem>
#include <cmath>
#include <unistd.h>
#include <iostream>

#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>

#endif // UNIVERSALQTHEADER_H
```