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 _____ Сергеев 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, высчитывающую значение выражения.

Задание.

Пусть арифметическое выражение представлено иерархическим списком. В выражение входят константы, которые являются атомами списка. Операции представляются в префиксной форме (<операция> <аргументы>)). Аргументов может быть 1 или 2. Например (в префиксной форме): (+ a (* b (- c))). Также во входной строке может присутствовать операция power(a,b) не в префиксной записи. Необходимо высчитать значение выражения.

Основные теоретические положения.

Согласно рекурсивному определению, иерархический список – такой список, элементами которого могут быть иерархические списки. В случае префиксного 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 Ubuntu, в среде CLion, а также с использованием библиотек qt и среды QTCreator.

Класс AtomicList содержит методы для построения, вывода в файл и вычисления префиксного арифметического выражения, представленного в виде иерархического списка.

Оценка эффективности алгоритма.

Алгоритм имеет линейную сложность, так как иерархический список строится, заполняется и рассчитывается один раз.

Тестирование программы.

Ввод (корректный ввод):

> (* (- (- power(power(c,c),d) v) (* (- d) (+ c w))) (+ (- power(l,c) l) (- k)))

> ((c 2) (v 54) (d 6) (w 1) (o 90) (l 7) (k 14))

Вывод:

< ANSWER IS: 113680

Ввод (некорректно заданное выражение):

> (* (- (- power(power(c , c),d) v) (* (- d) (+ c w))) (+ (- power(l,c) l) (- k)))

> ((c 2) (v 54) (d 6) (w 1) (l 7) (o 90) (k 14))

Вывод:

< INPUT WRONGLY FORMATTED, APPLICATION TERMINATED

< (* (- (- (^ (^ c c) d) v) (* (- d) (+ c w))) (+ (- (^ l c) l) (- k)))

< ^

< Unidentified symbol here.

Ввод (некорректно заданные константы):

> (* (- (- power(power(c,c),d) v) (* (- d) (+ c w))) (+ (- power(l,c) l) (- k)))

> ((c 2) (v 54) (d 6) (w 1) (o 90) (k 14))

Вывод:

> VARIABLES WERE NOT REPLACED WITH NUMBERS BEFORE COUNTING, APPLICATION TERMINATED

> (* (- (- (^ (^ c c) d) v) (* (- d) (+ c w))) (+ (- (^ l c) l) (- k)))

> ^

> Variable «l» caused the exception.

Ввод (некорректный оператор):

> (* (- (! power(power(c,c),d) v) (* (- d) (+ c w))) (+ (- power(l,c) l) (- k)))

> ((c 2) (v 54) (d 6) (w 1) (o 90) (k 14))

Вывод:

> VARIABLES WERE NOT REPLACED WITH NUMBERS BEFORE COUNTING, APPLICATION TERMINATED

> (* (- (- (! (^ c c) d) v) (* (- d) (+ c w))) (+ (- (^ l c) l) (- k)))

> ^

> Operator «!» can not be recognised.

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 префиксной записи арифметического выражения в виде иерархического списка.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

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

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("lab2");
    w.show();

    return a.exec();
}
```

Файл AtomicList.cpp:

```
//
// Created by alex on 9/26/19.
//

#include "AtomicList.h"

using namespace std;

AtomicList::AtomicList(string &expression, unsigned long recursivePosition) {
    int spaceRunner = 0;
    unsigned long lastSpace = 0;
    unsigned long firstPos = 0, secondPos = 0;
    string first, second;

    this->isFilled = false;

    lastSpace = expression.find(' ');
    if (lastSpace == 2) {
        if (isOperator(expression[lastSpace-1])) {
            this->algOperator = expression[lastSpace-1];
        } else {
            throw UnidentifiedOperatorException + expression[lastSpace-1];
        }
    } else {
        throw WrongFormatException + (int) (recursivePosition + lastSpace);
    }

    for (unsigned long i = lastSpace + 1; i < expression.length() - 1; i++) {
        if (expression[i] == '(') spaceRunner++;
        else if (expression[i] == ')') spaceRunner--;
        else if (spaceRunner == 0) {
            if (expression [i] == ' ') {
```

```
                firstPos = lastSpace + 1;
                first = expression.substr(firstPos, i - lastSpace - 1);
                secondPos = i + 1;
                second = expression.substr(secondPos, expression.length() - i - 2);
                lastSpace = i;
                break;
            }
        }
    }

    if (lastSpace == 2) {
        this->isUni = true;
        firstPos = lastSpace + 1;
        first = expression.substr(firstPos, expression.length() - lastSpace - 2);
    }

    if (first[0] != '(') {
        this->isFirstNum = true;
        if (first.length() > 1) throw (int) (WrongFormatException + firstPos + recursivePosition
+ 2);
        this->firstOperand.operand.variable = first[0];
    } else {
        this->isFirstNum = false;
        this->firstOperand.child = new AtomicList(first, recursivePosition + firstPos);
    }

    if (second[0] != '(') {
        this->isSecondNum = true;
            if (second.length() > 1) throw (int) (WrongFormatException + secondPos +
recursivePosition + 2);
        this->secondOperand.operand.variable = second[0];
    } else {
        this->isSecondNum = false;
        this->secondOperand.child = new AtomicList(second, recursivePosition + secondPos);
    }
}

AtomicList::~AtomicList() {
    free(this->firstOperand.child);
    free(this->secondOperand.child);
}

bool AtomicList::isOperator(char c) {
    return (c == '+') || (c == '-') || (c == '*') || (c == '^');
}

double AtomicList::bindVariable(string &data, char c) {
    unsigned long varPos = data.find(c);
    unsigned long numEnd = data.find(')', varPos);
    string number = data.substr(varPos + 2, numEnd - varPos - 2);
```

```
        char* end;
            if ((number[0] != '0') && (strtod(number.c_str(), &end)) == 0) throw
VariableCountingException + c;
        return strtod(number.c_str(), &end);
    }

    void AtomicList::fill(string &data) {
        this->isFilled = true;

        if (this->isFirstNum) {
                this->firstOperand.operand.num = bindVariable(data, this-
>firstOperand.operand.variable);
        } else {
            this->firstOperand.child->fill(data);
        }
        if (!this->isUni) {
            if (this->isSecondNum) {
                this->secondOperand.operand.num = bindVariable(data, this-
>secondOperand.operand.variable);
            } else {
                this->secondOperand.child->fill(data);
            }
        }
    }

    void AtomicList::toFile(string &fileSign, int recurrentCounter) {
        fileSign.append("\n");

        for (int i = 0; i < recurrentCounter; ++i) {
            fileSign.append("   ");
        }

        fileSign.append("( ").append(1, this->algOperator).append(" ");
        if (this->isFirstNum) {
            if (this->isFilled) {
                fileSign.append(to_string(this->firstOperand.operand.num));
            } else {
                fileSign.append(1, this->firstOperand.operand.variable);
            }
        } else {
            fileSign.append("(...)");
        }
        if (!this->isUni) {
            fileSign.append(" ");
            if (this->isSecondNum) {
                if (this->isFilled) {
                    fileSign.append(to_string(this->secondOperand.operand.num));
                } else {
                    fileSign.append(1, this->secondOperand.operand.variable);
                }
            } else {
```

```
                fileSign.append("(...)");
            }
        }
        fileSign.append(" )");

        if (!this->isFirstNum) this->firstOperand.child->toFile(fileSign, recurrentCounter + 1);
          if (!this->isUni && !this->isSecondNum) this->secondOperand.child->toFile(fileSign,
recurrentCounter + 1);
    }

    double AtomicList::count(string &fileSign, int recurrentCounter) {
                    if (!this->isFilled) throw VariableCountingException + this-
>firstOperand.operand.variable;

        double firstNum = 0, secondNum = 0;
        if (!this->isFirstNum) {
            firstNum = this->firstOperand.child->count(fileSign, recurrentCounter + 1);
        } else {
            firstNum = this->firstOperand.operand.num;
        }
        if (!this->isUni) {
            if (!this->isSecondNum) {
                secondNum = this->secondOperand.child->count(fileSign, recurrentCounter + 1);
            } else {
                secondNum = this->secondOperand.operand.num;
            }
        }

        double result = 0;
        if (!this->isUni) {
            switch (this->algOperator) {
                case '+':
                    result = firstNum + secondNum;
                    break;
                case '-':
                    result = firstNum - secondNum;
                    break;
                case '*':
                    result = firstNum * secondNum;
                    break;
                case '^':
                    result = pow(firstNum, secondNum);
                    break;
                default:
                    throw UnidentifiedOperatorException + this->algOperator;
            }
        } else if (this->algOperator == '-') {
            result = -firstNum;
        } else {
            cout << "LOL" << endl;
            throw UnidentifiedOperatorException + this->algOperator;
```

```
        }

        fileSign.append("\n");

        for (int i = 0; i < recurrentCounter; ++i) {
            fileSign.append("   ");
        }

        fileSign.append("( ").append(1, this->algOperator).append(" ");
        fileSign.append(to_string(firstNum));
        if (!this->isUni) {
            fileSign.append(" ");
            fileSign.append(to_string(secondNum));
        }
        fileSign.append(" )");

        fileSign.append(" = ").append(to_string(result));

        return result;
}
```

Файл AtomicList.h:

```
//
// Created by alex on 9/26/19.
//

#ifndef ALG_LAB1_MAIN_H
#define ALG_LAB1_MAIN_H

#include <cstdlib>
#include <bits/stdc++.h>

#define WrongFormatException 11000
#define UnidentifiedOperatorException 12000
#define VariableCountingException 13000

class AtomicList;

using namespace std;

union Content {
    AtomicList* child;
    struct Operand {
        char variable;
        double num;
    } operand;
};

class AtomicList {
```

```
    private:
        char algOperator;
        bool isUni;
        bool isFilled;

        bool isFirstNum;
        Content firstOperand;

        bool isSecondNum;
        Content secondOperand;

        double bindVariable(string &data, char c);

        bool isOperator(char c);

    public:
        explicit AtomicList(string &expression, unsigned long recursivePosition = 0);

        virtual ~AtomicList();

        void fill(string &data);

        void toFile(string &fileSign, int recurrentCounter = 0);

        double count(string &fileSign, int recurrentCounter = 0);
    };


    #endif //ALG_LAB1_MAIN_H
```

Файл MainWindow.cpp:

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

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

        connect(ui->buildButton, SIGNAL (clicked()), this, SLOT (build()));
        connect(ui->fillButton, SIGNAL (clicked()), this, SLOT (fill()));
        connect(ui->outputButton, SIGNAL (clicked()), this, SLOT (calculate()));
    }

    void MainWindow::build() {
        if (ui->buildInput->text().toStdString().length() == 0) return;

        ui->fillButton->setText(FILL_TEXT);
        ui->fillInput->setText("");
        ui->answer->setText("");
```

```
        ui->answer->setEnabled(false);
        ui->fillButton->setEnabled(true);
        ui->outputButton->setEnabled(false);
        ui->output->setText("");

        try {
            output = "";
            this->input = ui->buildInput->text().toStdString();
            if (input.length() > MAX_CHARS) {
                string msg = "WARNING! DUE TO 1000+ CHAR LENGTH OF INPUT, ERROR
DESCRIPTION WILL BE INCORRECT";
                this->printError(input, MAX_CHARS, msg);
            }
            convertPower(input);

            this->list = new AtomicList(input);
            this->output = "LISTED:";
            list->toFile(output);
        } catch (int e) {
            ui->fillButton->setEnabled(false);
            catchErrors(e);
        }

        ui->output->setText(QString::fromStdString(output));
    }

    void MainWindow::fill() {
        if (ui->fillInput->text().toStdString().length() == 0) return;

        ui->fillButton->setText(REFILL_TEXT);
        ui->answer->setText("");
        ui->answer->setEnabled(false);
        ui->outputButton->setEnabled(true);

        try {
            output = "";
            this->data = ui->fillInput->text().toStdString();
            this->list->fill(data);
            this->output = "FILLED:";
            list->toFile(output);
        } catch (int e) {
            catchErrors(e);
        }

        ui->output->setText(QString::fromStdString(output));
    }

    void MainWindow::calculate() {
        ui->answer->setEnabled(true);

        string answer = "ANSWER IS: ";
```

```
        try {
            output = "";
            answer.append(to_string(this->list->count(output)));
            this->output = "CALCULATED:";
        } catch (int e) {
            catchErrors(e);
        }

        ui->output->setText(QString::fromStdString(output));
        ui->answer->setText(QString::fromStdString(answer));
    }

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

    void MainWindow::catchErrors(int e) {
        string msg;
        int pure = e / MAX_CHARS;
        int info = e % MAX_CHARS;
        switch (pure * MAX_CHARS) {
            case WrongFormatException:
                output.append("INPUT WRONGLY FORMATTED, APPLICATION TERMINATED\n");
                msg = "Unidentified symbol here.";
                printError(input, info, msg);
                break;
            case UnidentifiedOperatorException:
                output.append("UNDEFINED OPERATOR FOUND, APPLICATION TERMINATED\n");
                msg = "Operator \"";
                msg += (char) info;
                msg += "\" can not be recognised.";
                printError(input, (int) input.find((char) info) + 1, msg);
                break;
            case VariableCountingException:
                output.append("VARIABLES WERE NOT REPLACED WITH NUMBERS BEFORE CALCULATION, APPLICATION TERMINATED\n");
                msg = "Variable \"";
                msg += (char) info;
                msg += "\" caused the exception.";
                printError(input, (int) input.find((char) info) + 1, msg);
                break;
            default:
                output.append("SOMETHING TERRIBLE HAPPENED, APPLICATION TERMINATED\n");
                break;
            }
    }

    void MainWindow::printError(string &errorString, int pos, string& message) {
```

```
    output.append(errorString).append("\n");
    for (int i = 0; i < pos - 1; ++i) {
        output.append(" ");
    }
    output.append("^\n").append(message);
}

void MainWindow::convertPower(string &expr) {
    unsigned long powerPos = expr.find("power(");
    while (powerPos != string::npos) {
        expr.replace(powerPos, 6, "(^ ");
        powerPos = expr.find("power(");
    }
    powerPos = expr.find(',');
    while (powerPos != string::npos) {
        expr[powerPos] = ' ';
        powerPos = expr.find(',');
    }
}
```

Файл MainWindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "AtomicList.h"

#define FILL_TEXT "Fill"
#define REFILL_TEXT "Refill"
#define MAX_CHARS 1000

using namespace std;

namespace Ui {
    class MainWindow;
}

class MainWindow : public QMainWindow {
    Q_OBJECT

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

private slots:
    void build();
    void fill();
    void calculate();

private:
    Ui::MainWindow *ui;
```

```
    void printError(string &errorString, int pos, string& message);
    void convertPower(string &expr);
    void catchErrors(int e);

    string input, data, output;
    AtomicList *list;
};

#endif // MAINWINDOW_H
```

Файл 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>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <property name="sizePolicy">
    <sizepolicy hsizetype="Maximum" vsizetype="Maximum">
     <horstretch>0</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <widget class="QWidget" name="verticalLayoutWidget">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>801</width>
      <height>601</height>
     </rect>
    </property>
    <layout class="QVBoxLayout" name="verticalLayout">
     <property name="sizeConstraint">
      <enum>QLayout::SetMinAndMaxSize</enum>
     </property>
     <property name="leftMargin">
      <number>6</number>
     </property>
     <property name="topMargin">
```

```
 <number>6</number>
</property>
<property name="rightMargin">
 <number>6</number>
</property>
<property name="bottomMargin">
 <number>6</number>
</property>
<item>
 <widget class="QLabel" name="appName">
  <property name="font">
   <font>
    <pointsize>16</pointsize>
   </font>
  </property>
  <property name="text">
   <string>Prefix expression calculation</string>
  </property>
  <property name="alignment">
   <set>Qt::AlignCenter</set>
  </property>
  <property name="margin">
   <number>6</number>
  </property>
 </widget>
</item>
<item>
 <layout class="QHBoxLayout" name="buildLayout">
  <property name="sizeConstraint">
   <enum>QLayout::SetMaximumSize</enum>
  </property>
  <item>
   <widget class="QPushButton" name="buildButton">
    <property name="text">
     <string>Build</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLineEdit" name="buildInput">
    <property name="font">
     <font>
      <family>Ubuntu Mono</family>
     </font>
    </property>
    <property name="placeholderText">
     <string>Expression e.g.: (* (+ a b) power(c,d))</string>
    </property>
   </widget>
  </item>
 </layout>
```

```
</item>
<item>
 <layout class="QHBoxLayout" name="fillLayout">
  <property name="sizeConstraint">
   <enum>QLayout::SetMaximumSize</enum>
  </property>
  <item>
   <widget class="QPushButton" name="fillButton">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="text">
     <string>Fill</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLineEdit" name="fillInput">
    <property name="font">
     <font>
      <family>Ubuntu Mono</family>
     </font>
    </property>
    <property name="placeholderText">
     <string>Variables e.g.: ((a 2) (b 5) (c 3) (d 7))</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <widget class="QPushButton" name="outputButton">
  <property name="enabled">
   <bool>false</bool>
  </property>
  <property name="text">
   <string>Calculate!</string>
  </property>
 </widget>
</item>
<item>
 <widget class="QLabel" name="output">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Preferred" vsizetype="Expanding">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="font">
   <font>
    <family>Ubuntu Mono</family>
```

```
        </font>
       </property>
       <property name="text">
        <string/>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QLabel" name="answer">
       <property name="text">
        <string/>
       </property>
       <property name="alignment">
        <set>Qt::AlignCenter</set>
       </property>
      </widget>
     </item>
    </layout>
   </widget>
  </widget>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```

Файл AtomicList.h:

```
# Project created by QtCreator 2019-10-07T03:24:58
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = lab2
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0
```

```
SOURCES += \
        main.cpp \
        mainwindow.cpp \
        AtomicList.cpp

HEADERS += \
        mainwindow.h \
        AtomicList.h

FORMS += \
        mainwindow.ui
```