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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Стек и очередь

Студентка гр. 8381 _____ Лисок М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с методами вычисления выражения в постфиксной форме и реализовать на практике.

Задание.

Рассматривается выражение следующего вида:

< выражение >::= < терм > | < терм > + < выражение > |
< терм > < выражение >

< терм > ::= < множитель > | < множитель > * < терм >

< множитель > ::= < число > | < переменная > | (< выражение >) |
< множитель > ^ < число >

< число > ::= < цифра >

< переменная > ::= < буква >

Такая форма записи выражения называется ***инфиксной***.

Постфиксной (префиксной) формой записи выражения *aDb* называется запись, в которой знак операции размещен за (перед) операндами: *abD* (*Dab*).

Примеры

Инфиксная	***Постфиксная***	***Префиксная***
ab	*ab*	*ab*
*a*b+c*	*ab*c+*	*+*abc*
a(b+c)*	*abc+**	**a+bc*
*a+b^c^d*e*	*abc^d^e*+*	*+a*^b^cde*.

Отметим, что постфиксная и префиксная формы записи выражений не содержат скобок.

Требуется:

а) вычислить как целое число значение выражения (без переменных), записанного в постфиксной форме в заданном текстовом файле *postfix*;

Выполнение работы.

В ходе данной работы была разработана дополнительная структура данных стек, а так же все необходимые методы для работы с ним, исходный код представлен в приложении А.

Дополнительно были написаны методы для подсчета постфиксного выражения. Названия методов и их назначение представлены в табл.1.

Таблица 1 - Методы и их назначения

Метод	Назначение
bool **isOperation**(char symbol)	Определяет является ли входной параметр знаком операции +, *, ^
void **step**(char, Stack<long long int>&, string&,string&)	Достает два значение из стека, если он не пуст, производит вычисления согласно знаку операции +, *, ^, в противном случае сообщает об ошибке
long long int **calculation**(const string&, string&,string&)	Проходит по всем операндам и операциям, вызвая метод **onestep,** возражает конечный результат
void **onestep**(char, Stack<long long int>&, string &,string&)	Определяет является ли входной символ знаком операции, если да вызывает функцию **step,** если нет, определяет является ли символ числом, если да, то кладет элемент в стек, если нет сообщает об ошибке

Были разработаны консольный и графический интерфейсы. В зависимости от передаваемых параметров в функцию main() работал продолжается в консоли либо открывается графический интерфейс. Данные работы можно сохранять в файл, а так же считывать из него.

Оценка эффективности алгоритма.

Алгоритм вычисления значения арифметического выражения, записанного в постфиксной форме, имеет линейную сложность O(n). Алгоритм использует стек. При чтении выражения слева направо в вершину стека помещаются операнды. Как только при чтении встречается знак арифметической операции, из стека извлекаются два последних операнда, к ним применяется текущая операция, и результат записывается обратно в вершину стека. По завершении работы алгоритма в стеке оказывается один элемент – значение арифметического выражения.

Из-за точного соответствия числа операндов количеству чисел, которые должны находится в стеке для корректного завершения процесса вычисления, и пренебрежения ко времени выполнения различных арифметических операций в памяти компьютера из-за сильной зависимости от косвенных факторов в виде типа процессора, загруженности системы и т.п., оценка алгоритма производится по системе: количество элементов - количество операций.

В связи с оговоренной выше независимостью от типа и порядка операций, а также логическому соответствию каждого элемента свои 3-м операциям push(), pop() и onTop() (за исключением последнего в вычислениях) для тестирования были сгенерированы тесты на $n \in \{5, 10, 50, 100, 500, 1000, 5000, 10000\}$ элементов, представляющие из себя последовательность n чисел, а затем n - 1 знаков арифметической операции сложения. На основе промежуточных выводов через qDebug() в ходе выполнения программы и последующего анализа выходной последовательности операций был построен график, представленный на рис. 1. На основе его анализа можно сделать вывод, что алгоритм вычисления постфиксного арифметического выражения обладает линейной сложностью.

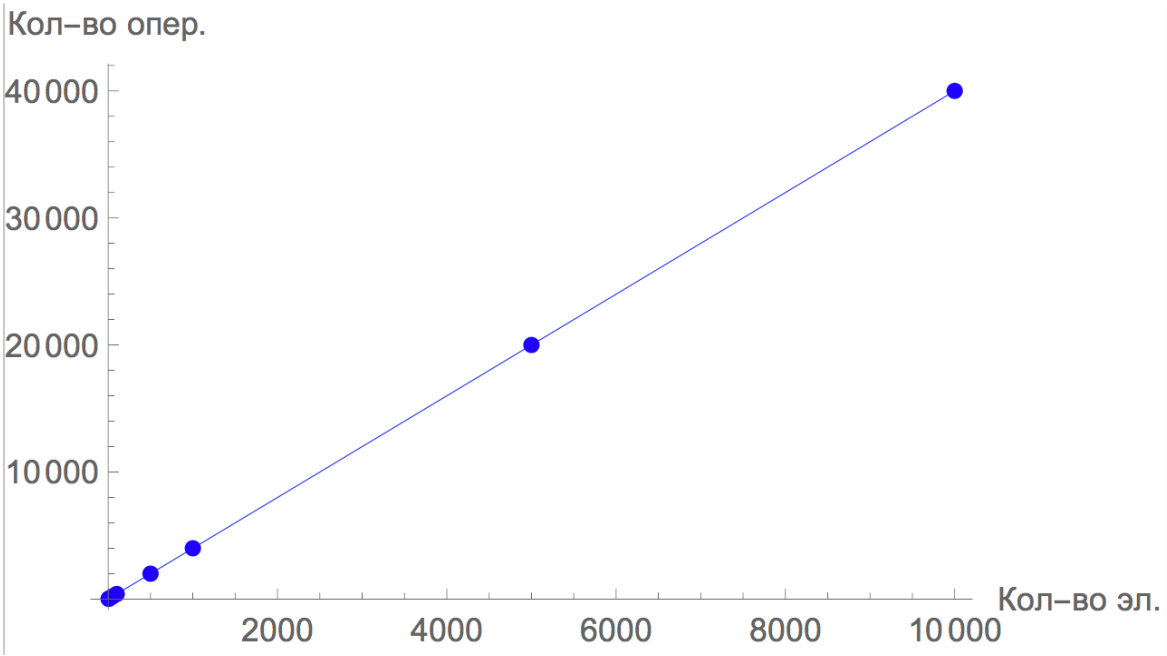

Рисунок 1 - зависимости количества операций от количества элементов

Тестирование программы.

Результаты тестирования программы представлены в табл.2.

Таблиаца 2 - Результаты тестирования программы

Входное выражение	Результат
9	9
2+	Uncorrect test. Check your string!
23	Uncorrect test. There are too much symbols in you string!
ab	Uncorrect test. Input numbers!
234+*56^+86*7^+	587068357911

Выводы.

В ходе лабораторной работы был изучен способ вычисления выражения в постфисной форме. А также реализован пошаговый режим выполнения алгоритма, консольный и графический интерфейсы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#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>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_resultButton_clicked();

    void on_stepButton_clicked();

    void on_fileButton_clicked();

    void on_saveButton_clicked();
```

```
private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Название файла: methods.h

```
#ifndef METHODS_H
#define METHODS_H

#include "stack.h"
#include <math.h>
bool isOperation(char);
void step(char, Stack<long long int>&, string&,string&);
long long int calculation(const string&, string&,string&);
void onestep(char, Stack<long long int>&, string &,string&);
#endif
```

Название файла: methods.cpp

```
#include "methods.h"

bool isOperation(char symbol){
        return ((symbol=='+' || symbol=='*' || symbol=='^')?
true: false);
}
void step(char a, Stack<long long int>& stack, string &
output, string & err){
    long result=0;
    if(!stack.isEmpty() && stack.length()>=2){
        switch(a){
            case '+':
                result+=*stack.onTop();
                output.append(to_string(*stack.onTop())+"+");
                stack.pop();
                result+=*stack.onTop();
                        output.append(to_string(*stack.onTop())
+"="+to_string(result)+"\n");
                stack.pop();
                stack.push(result);
```

```
                    break;
                case '*':
                    result=1;
                    result*=*stack.onTop();
                    output.append(to_string(*stack.onTop())+"*");
                    stack.pop();
                    result*=*stack.onTop();
                           output.append(to_string(*stack.onTop())
+"="+to_string(result)+"\n");
                    stack.pop();
                    stack.push(result);
                    break;
                case '^':
                    long power=*stack.onTop();
                    stack.pop();
                    result=pow(*stack.onTop(),power);
                           output.append(to_string(*stack.onTop())
+"^"+to_string(power)+"="+to_string(result)+"\n");
                    stack.pop();
                    stack.push(result);
                    break;
            }
        }
        else{
            err.append("Uncorrect test. Check your string!\n");
        }
    }
    void onestep(char data, Stack<long long int>& stack, string &
output, string & err){
        if(isOperation(data)){
            step(data, stack, output, err);
            if(!output.empty())
                return;
        }else{
            if(isdigit(data)){
                stack.push(data-'0');
                output.append("push "+to_string(data-'0')+"\n");
            }
            else{
                err.append("Uncorrect test. Input numbers!\n");
            }
        }
```

```
    }
    long long int calculation(const string& str, string & output,
string & err){
        Stack<long long int> stack{};
        for(long long int i=0; i<str.length(); i++){
            onestep(str[i], stack, output, err);
        }
        long long int result = *stack.onTop();
        stack.pop();
        if(stack.isEmpty())
            return result;
         err.append("Uncorrect test. There are too much symbols in
you string!\n");
        stack.clear();
        return 0;
    }
```

Название файла: stack.h

```
    #ifndef STACK_H
    #define STACK_H
    #include<iostream>

    using namespace std;

    template<class T>
    class Stack{
    public:
        Stack(size_t n=50)
        : SIZE{ n }, items{ new T[SIZE]{} }, top{}
        {}
        ~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: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "methods.h"
#include <cstring>
#include <fstream>
```

```
#include <QFileDialog>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}
Stack<long long int> stack{};
long long int i = 0;
string answer{};
string inputTest{};
string err{};
MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_resultButton_clicked()
{
        string checkInput = ui->inputLine->text().toUtf8().constData();
    if(checkInput.empty()){
                                    ui->resultWindow->setText(QString::fromStdString("Uncorrect test. Input numbers! \n"));
        return;
    }
    string result;
    string error;

    long answer = calculation(checkInput, result, error);
    if(!error.empty()){
                                    ui->resultWindow->setText(QString::fromStdString(error));
    }else{
        string str = "Результат вычислений: ";
        str+=to_string(answer);
                                    ui->resultWindow->setText(QString::fromStdString(str));
    }
    this->resize(439, 572);
```

```
        this->resize(438, 571);
    }

    void MainWindow::on_stepButton_clicked()
    {
        QString str = ui->inputLine->text();
        if(inputTest.compare(str.toUtf8().constData())!=0){
            i=0;
            answer.clear();
            err.clear();
            inputTest.clear();
            inputTest.append(str.toUtf8().constData());
            ui->resultWindow->clear();
            stack.clear();
            stack.setSize(inputTest.length());
        }
        if(!err.empty()){
            return;
        }
        if(i>=inputTest.length() && stack.length()==1){
            i=0;
            stack.pop();
            answer.clear();
            err.clear();
        }
        onestep(inputTest[i], stack, answer, err);
        i++;
        if(answer.empty()){
                                        ui->resultWindow-
>setText(QString::fromStdString(err));
        }
        else if(err.empty()){
                                        ui->resultWindow-
>setText(QString::fromStdString(answer));
        }
        else{
                                        ui->resultWindow-
>setText(QString::fromStdString(answer)
+QString::fromStdString(err));
        }
        this->resize(439, 572);
        this->resize(438, 571);
```

```
}

void MainWindow::on_fileButton_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_saveButton_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();
}
```

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>438</width>
    <height>571</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QLabel" name="title">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>20</y>
      <width>241</width>
      <height>16</height>
     </rect>
```

```
 </property>
 <property name="text">
  <string>Постфиксный калькулятор</string>
 </property>
</widget>
<widget class="QLineEdit" name="inputLine">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>50</y>
   <width>251</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QTextEdit" name="resultWindow">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>150</y>
   <width>261</width>
   <height>341</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="resultButton">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>90</y>
   <width>113</width>
   <height>32</height>
  </rect>
 </property>
 <property name="text">
  <string>result</string>
 </property>
</widget>
<widget class="QPushButton" name="stepButton">
 <property name="geometry">
  <rect>
   <x>150</x>
   <y>90</y>
   <width>113</width>
   <height>32</height>
  </rect>
 </property>
 <property name="text">
  <string>step by step</string>
 </property>
</widget>
<widget class="QPushButton" name="fileButton">
 <property name="geometry">
  <rect>
   <x>300</x>
   <y>50</y>
   <width>113</width>
   <height>32</height>
  </rect>
```

```
    </property>
    <property name="text">
     <string>file</string>
    </property>
   </widget>
   <widget class="QLabel" name="resultTitle">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>130</y>
      <width>101</width>
      <height>16</height>
     </rect>
    </property>
    <property name="text">
     <string>Результат</string>
    </property>
   </widget>
   <widget class="QPushButton" name="saveButton">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>500</y>
      <width>113</width>
      <height>32</height>
     </rect>
    </property>
    <property name="text">
     <string>save</string>
    </property>
   </widget>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```