Министерство науки и высшего образования Российской Федерации   
Федеральное государственное автономное образовательное учреждение   
высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчет по учебной практике

**Вычисление арифметических выражений(стеки)**

Выполнил:

Студент группы 381806-3

Макарычев С.Д.

Проверил:

Доцент кафедры МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г.

**Содержание**

[Введение 4](#_Toc26212150)

[1 Постановка задачи 5](#_Toc26212151)

[2 Руководство пользователя 6](#_Toc26212152)

[3 Руководство программиста 8](#_Toc26212153)

[3.1 Структура программы 8](#_Toc26212154)

[3.2 Описание алгоритмов 8](#_Toc26212155)

[3.2.1 Алгоритм преобразования инфиксной формы в постфиксную 8](#_Toc26212156)

[3.2.2 Алгоритм вычисления выражения по постфиксной форме при определенных значениях операндов. 9](#_Toc26212157)

[4 Описание функций 10](#_Toc26212158)

[4.1 Функции класса TStack 10](#_Toc26212159)

[4.2 Функции класса Postfix 10](#_Toc26212160)

[4.3 Функции класса Exception 11](#_Toc26212161)

[5 Вывод 12](#_Toc26212162)

[6 Приложение 13](#_Toc26212163)

# Введение

Постфиксная запись представляет собой такую запись арифметического выражения, в которой сначала записываются операнды, а затем – знаки операции. Постфиксная форма позволяет избавиться от скобок, присутствующих в инфиксной форме, тем самым помогая избежать неоднозначности выполнения действий.

В данной работе представлена реализация алгоритма, позволяющего преобразовать выражение в постфиксную форму и вычислить полученное в постфиксной форме выражение.

# Постановка задачи

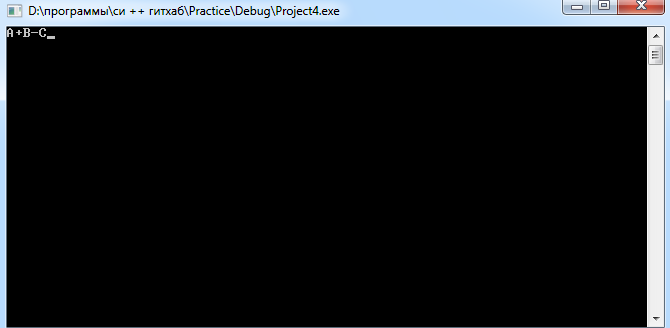
В данной работе необходимо, используя структуру данных “Стек”, реализовать программу, позволяющую получать из выражения в инфиксной форме выражение в постфиксной форме и вычислять значение этого выражения

Входные данные: выражение в инфиксной форме; значения переменных, входящих в это выражение.

Выходные данные: выражение в постфиксной форме; значение выражения при определенных значениях переменных.

# Руководство пользователя

После запуска программы пользователю необходимо заглавными буквами латинского алфавита ввести выражение, которое он хочет преобразовать. Вводим в поле наше выражение и нажимаем Enter.(рис.1)



1. Ввод выражения

Если выражение введено корректно, то в консоли появится выражение в постфиксной форме.(рис.2)

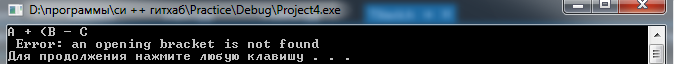


1. Вывод выражения в постфиксной форме

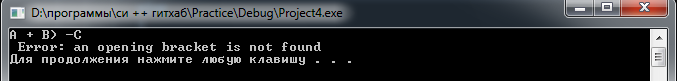
Если же выражение введено неправильно, то сработает одна из исключительных ситуаций. (рис.3), (рис.4), (рис.5), (рис.6)



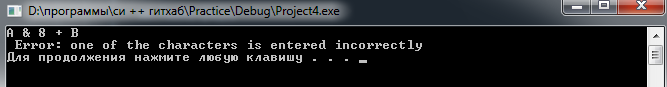
1. Исключительная ситуация: несколько переменных стоят рядом друг с другом



1. Исключительная ситуация: в выражении не найдена закрывающаяся скобка

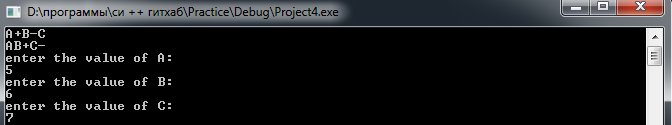


1. Исключительная ситуация: в выражении не найдена открывающаяся скобка



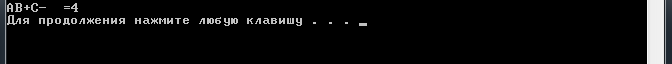
1. Исключительная ситуация: в выражении присутствует недопустимый символ

Далее после вывода выражения в постфиксной форме пользователю нужно ввести значения переменных, входящих в выражение.(рис.7)



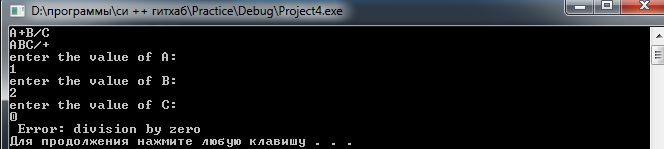
1. Ввод значений переменных.

После этого в консоль выведется значение исходного выражения. (рис.8)



1. Вывод значения выражения при заданных параметрах.

Если в исходном выражении присутствует частное, и делитель в нем равен нулю, то срабатывает исключительная ситуация: деление на ноль.



1. Исключительная ситуация: деление на ноль.

# Руководство программиста

## Структура программы

Данная программа состоит из 5 файлов: TStack.h, в котором описан шаблонный класс, реализующий структуру данных “Стек”; файла Postfix.h, в котором описан класс со статическими методами, выполняющими алгоритмы преобразования; файла Postfix.cpp с реализациями статических функций класса; файла Exception.h с классом для обработки исключений; файла main.cpp, в котором реализуется основная программа.

## Описание алгоритмов

### Алгоритм преобразования инфиксной формы в постфиксную

Алгоритм представлен в виде функции, принимающей на вход строку с выражением в инфиксной форме и возвращающей строку в постфиксной форме.

Сначала создается 2 стека: в первом стеке будут храниться операнды, во втором – операции. Далее всем операциям присваивается приоритет: “\*” и “/” – приоритет 3; “+” и “-“ – приоритет 2; “(“ – приоритет 1.

Поступившая на вход строка начинает просматриваться слева направо с первого элемента. Если пришел операнд, то мы кладём его в первый стек. Если пришла операция, то если второй стек пуст, кладем операцию в данный стек. Если же во втором стеке уже имеются элементы, то сравниваем приоритет верхнего элемента стека и операции, которую мы хотим добавить. Если приоритет верхнего элемента стека больше или равен приоритета рассматриваемой операции, то пока это условие выполняется, мы удаляем элементы из второго стека и вставляем в первый стек.

Если пришла открывающаяся круглая скобка, то мы кладем ее во второй стек. Если пришла закрывающаяся круглая скобка, то мы начинаем перекладывать элементы из второго стека в первый, пока не встретится открывающаяся круглая скобка, ее мы из второго стека удаляем.

Когда дошли до конца исходной строки, начинаем перекладывать элементы из второго стека, начиная с самого верхнего и заканчивая самым нижним элементом, в первый. Из второго стека перекладываем элементы, начиная с самого нижнего, в новую строку. Таким образом, в новой строке содержится выражение в постфиксной форме.

1. Пример пошагового выполнения алгоритма для выражения: (A+B)\*C

|  |  |  |
| --- | --- | --- |
| Обрабатываемая лексема | Стек 1 | Стек 2 |
| ( |  | ( |
| A | A | ( |
| + | A | ( + |
| B | A B | ( + |
| ) | A B + | ( |
| ) | A B + |  |
| \* | A B + | \* |
| C | A B + C | \* |
|  | A B + C \* |  |

### Алгоритм вычисления выражения по постфиксной форме при определенных значениях операндов.

Алгоритм представлен в виде функции, принимающей на вход строку с выражением в постфиксной форме, множество операндов, входящих в это выражение, множество значений этих операндов и возвращающей значение выражения.

Сначала создается вспомогательный стек для хранения значений некоторой части выражения.

Постфиксная форма просматривается слева направо. Если пришел операнд, то выполняется поиск значения этого операнда, и значение кладется в стек. Если пришла операция, то вынимаем с вершины стека 2 значения и выполняем операцию в порядке 2, 1, где 1-верхнее значение из стека, 2-значение идущее после верхнего элемента стека. Результат операции кладется в стек.

Если дошли до конца выражения, то в стеке хранится результат вычисления выражения для заданных значений операндов.

1. Пример пошагового выполнения алгоритма для постфиксного выражения A B + C \* при A = 5, B = 4, C = 3.

|  |  |  |
| --- | --- | --- |
| Обрабатываемая лексема | Стек | Выражение |
| A | 5 |  |
| B | 5 4 |  |
| + | 9 | 4 + 5 |
| C | 9 3 |  |
| \* | 27 | 3 \* 9 |

# Описание функций

## Функции класса TStack

**Функция “**IsEmpty”

Описание: проверяет стек на пустоту.

Ничего не принимает. Возвращает 1, если стек пуст, 0, если в стеке имеются элементы.

**Функция “**IsFull”

Описание: проверяет стек на переполнение.

Ничего не принимает. Возвращает 1, если стек полон, 0, если стек не полон.

**Функция “**Push”

Описание: выполняет вставку элемента наверх стека.

Принимает элемент, который нужно вставить в стек. Ничего не возвращает.

**Функция “**Pop”

Описание: удаляет верхний элемент из стека.

Ничего не принимает. Возвращает удаленный элемент.

**Функция “**Top”

Описание: позволяет узнать верхний элемент стека.

Ничего не принимает. Возвращает верхний элемент стека.

## Функции класса Postfix

**Статическая Функция “**PostfixForm”

Описание: преобразует инфиксное выражение в постфиксное.

Принимает строку, содержащую выражение в инфиксной форме. Возвращает строку, содержащую выражение в постфиксной форме.

**Статическая Функция “**Calculating”

Описание: вычисляет значение выражения по постфиксной форме.

Принимает константную ссылку на строку, содержащую выражение в постфиксной форме, ссылку на массив операндов этого выражения, ссылку на массив значений этих операндов, ссылку на количество неповторяющихся символов в строке. Возвращает значение выражения.

**Статическая Функция “** GetOperandsValues”

Описание: заполняет массив неповторяющимися операндами из строки в постфиксной форме, считает количество этих операндов, заполняет массив значений этих операндов.

Принимает константную ссылку на строку, содержащую выражение в постфиксной форме, ссылку на массив операндов этого выражения, ссылку на массив значений этих операндов, ссылку на количество неповторяющихся символов в строке. Возвращает значение выражения.

**Статическая Функция “**Prioritet”

Описание: присваивает приоритет операциям.

Принимает операцию из строки. Возвращает приоритет операции.

## Функции класса Exception

**Функция “what”**

Описание: переопределяет функцию класса потомка exception, определяет сработавшее исключение.

Ничего не принимает. Возвращает строку с ошибкой.

# Вывод

Выполняя данную работу, я ознакомился со структурой данных “стек”. А также с алгоритмами вычисления арифметических выражений.

# Приложение

Файл TStack.h

#pragma once

#include <string>

#include "Exсeption.h"

using namespace std;

template <class ValueType>

class TStack

{

private:

int size;

ValueType\* elems;

int top;

public:

TStack(int size);

TStack(const TStack &s);

~TStack();

bool IsEmpty()const;

bool IsFull()const;

void Push(ValueType e);

void Pop();

ValueType Top() const;

};

template <class ValueType>

ValueType TStack<ValueType>::Top() const

{

return (elems[top]);

}

template <class ValueType>

TStack<ValueType>::TStack(int size)

{

this->top = -1;

this->size = size;

elems = new ValueType[size];

}

template <class ValueType>

TStack<ValueType> ::TStack(const TStack<ValueType> &s)

{

this->size = s.size;

this->top = s.top;

for (int i = 0; i < top; i++)

this->elems[i] = s.elems[i];

}

template<typename ValueType>

TStack<ValueType>::~TStack()

{

delete[] elems;

}

template <class ValueType>

bool TStack<ValueType> ::IsEmpty()const

{

return (top == -1);

}

template <class ValueType>

bool TStack<ValueType> ::IsFull()const

{

return(top == size - 1);

}

template <class ValueType>

void TStack<ValueType> ::Push(ValueType e)

{

if (IsFull())

throw M\_Exeption("Stack is full");

elems[++top] = e;

}

template <class ValueType>

void TStack<ValueType>::Pop()

{

if (IsEmpty())

throw M\_Exeption("Stack is empty");

top--;

}

Файл Postfix.h

#pragma once

#include "TStack.h"

#include <string>

#include "Exсeption.h"

using namespace std;

class Postfix

{

private:

static int Prioritet(char v);

public:

static string PostfixForm(const string &v);

static void GetOperandsValues(const string& postfix, char\*& operands, float\*& values, int& count);

static float Calculating(const string &v, char\* operands, float\* values, int count);

};

Файл Postfix.cpp

#include "Postfix.h"

#include "TStack.h"

using namespace std;

string Postfix::PostfixForm(const string &v)

{

TStack<char> s1(v.length());

TStack<char> s2(v.length());

if((v[0] == 42) || (v[0] == 43) || (v[0] == 45) || (v[0] == 47))

throw M\_Exeption("an expression cannot start with operands");

for (int i = 0; i < v.length(); i++)

{

if ((v[i] >= 65) && (v[i] <= 90))

{

if (i > 0)

{

if ((v[i - 1] >= 65) && (v[i - 1] <= 90))

throw M\_Exeption("two letters stand side by side");

}

s1.Push(v[i]);

}

else if ((v[i] == 42) || (v[i] == 43) || (v[i] == 45) || (v[i] == 47))

{

if ((v[i-1] == 42) || (v[i-1] == 43) || (v[i-1] == 45) || (v[i-1] == 47))

throw M\_Exeption("2 operands stand side by side");

if (s2.IsEmpty())

s2.Push(v[i]);

else

{

if (Prioritet(v[i]) <= Prioritet(s2.Top()))

{

while ((Prioritet(v[i]) <= Prioritet(s2.Top())) && (!s2.IsEmpty()))

{

s1.Push(s2.Top());

s2.Pop();

}

s2.Push(v[i]);

}

else

{

s2.Push(v[i]);

}

}

}

else if (v[i] == '(')

s2.Push(v[i]);

else if (v[i] == ')')

{

while ((s2.Top()) != '(')

{

if (s2.IsEmpty())

throw M\_Exeption("an opening bracket is not found");

s1.Push(s2.Top());

s2.Pop();

}

s2.Pop();

}

else if (v[i] == ' ')

continue;

else

throw M\_Exeption("one of the characters is entered incorrectly");

}

while (!s2.IsEmpty())

{

s1.Push(s2.Top());

s2.Pop();

}

string q;

while (!s1.IsEmpty())

{

q = q + s1.Top();

s1.Pop();

}

string q1;

int len = q.length();

for (int i = 0; i < len; i++)

{

if(q[len - (i + 1)] == '(')

throw M\_Exeption("a closing bracket is not found");

q1 = q1 + q[len - (i + 1)];

}

return q1;

}

float Postfix::Calculating(const string& v, char\* operands, float\* values, int count)

{

TStack<float> s(v.length());

float a, b;

for (int i = 0; i < v.length(); i++)

{

if ((v[i] >= 65) && (v[i] <= 90))

{

for (int k = 0; k < count; k++)

{

if (v[i] == operands[k])

{

s.Push(values[k]);

break;

}

}

}

else if ((v[i] == 42) || (v[i] == 43) || (v[i] == 45) || (v[i] == 47))

{

a = s.Top();

s.Pop();

b = s.Top();

s.Pop();

switch (v[i])

{

case '\*':

s.Push(b \* a);

break;

case '/':

if (a == 0)

throw M\_Exeption("division by zero");

s.Push(b / a);

break;

case '+':

s.Push(b + a);

break;

case '-':

s.Push(b - a);

break;

}

}

else

throw "Incorrect symbol";

}

return (s.Top());

}

void Postfix::GetOperandsValues(const string& postfix, char\*& operands, float\*& values, int& count)

{

count = 0;

for (int i = 0; i < postfix.length(); i++)

{

if ((postfix[i] >= 65) && (postfix[i] <= 90))

{

count++;

}

}

operands = new char[count];

values = new float[count];

int n = 0;

for (int i = 0; i < postfix.length(); i++)

{

if ((postfix[i] >= 65) && (postfix[i] <= 90))

{

int f = 0;

for (int j = 0; j < n; j++)

{

if (postfix[i] == operands[j])

f = 1;

}

if (f == 0)

{

operands[n] = postfix[i];

cout << "enter the value of " << operands[n] << ": " << endl;

cin >> values[n];

n++;

}

}

}

}

int Postfix::Prioritet(char v)

{

{

{

switch (v)

{

case '\*':

return 3;

break;

case '/':

return 3;

break;

case '+':

return 2;

break;

case '-':

return 2;

break;

case '(':

return 1;

break;

}

}

}

}

Файл Exception.h

#pragma once

#include <exception>

#include <string>

#include <iostream>

using namespace std;

class M\_Exeption:public exception

{

private:

string m\_error;

public:

M\_Exeption(string error) : m\_error(error)

{

}

const char\* what() const noexcept

{

return m\_error.c\_str();

}

};