Министерство науки и высшего образования РФ

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**

Выполнил:

студент гр. 381806-1

Пронин И.П.

Проверил:

ассистент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2019

Оглавление

[Введение 3](#_Toc27945661)

[Постановка задачи 4](#_Toc27945662)

[Руководство пользователя 5](#_Toc27945663)

[Руководство программиста 7](#_Toc27945664)

[Описание структуры программы 7](#_Toc27945665)

[Описание структур данных 7](#_Toc27945666)

[Класс TStack 7](#_Toc27945667)

[Класс Calculating 8](#_Toc27945668)

[Описание алгоритмов 8](#_Toc27945669)

[Алгоритм работы Stack 8](#_Toc27945670)

[Перевод в постфиксную форму 9](#_Toc27945671)

[Вычисление Арифметического выражения 10](#_Toc27945672)

[Заключение 12](#_Toc27945673)

[Литература 13](#_Toc27945674)

[Приложения 14](#_Toc27945675)

[Приложение 1. Исходный код основной функции 14](#_Toc27945676)

[Приложение 2. Класс TStack 14](#_Toc27945677)

[Приложение 3. Модуль Calculating.h 15](#_Toc27945678)

[Приложение 4. Модуль Calculating.cpp 16](#_Toc27945679)

# Введение

При вычислении произвольных арифметических выражений возникают две основные задачи: проверка корректности введённого выражения и выполнение операций в порядке, определяемом их приоритетами и расстановкой скобок. Существует алгоритм, позволяющий реализовать вычисление произвольного арифметического выражения за один просмотр без хранения промежуточных результатов. Для реализации данного алгоритма выражение должно быть представлено в постфиксной форме.

# Постановка задачи

В рамках лабораторной работы ставится задача реализации программ, обеспечивающих поддержку стеков, и разработки программных средств, производящих обработку арифметических выражений, включая проверку правильности записи выражения, перевод в постфиксную форму и вычисление результата.

# Руководство пользователя

1. Запустить файл **02\_Stack.exe**. Перед вами откроется консольное приложение с запросом ввести инфиксную форму арифметического выражения. (см. Рис. 1)

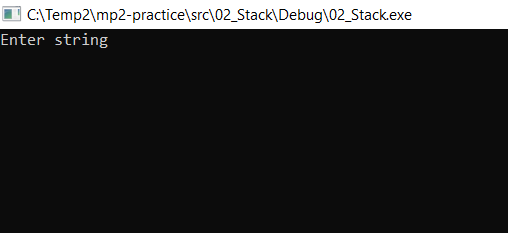


Рис. 1 Запрос на ввод инфиксной формы

1. Программа выведет постфиксную форму записи арифметического выражения и запросит значения переменных, присутствующих в арифметическом выражении. (см. Рис. 2)

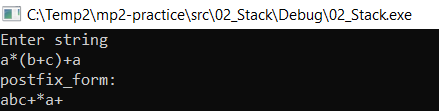


Рис. 2 Вывод постфиксной формы

1. После ввода значений переменных программа выведет результат арифметического выражения. (см. Рис. 3)

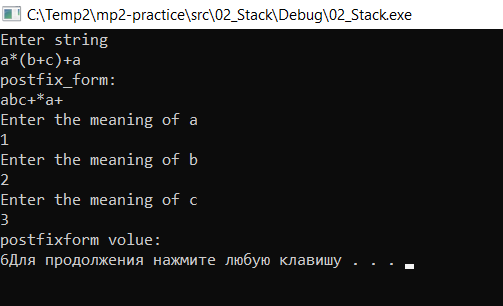


Рис. 3 Ввод значений переменных и вывод результата арифметического выражения

Действия, приводящие к ошибке

1. Не каждой открывающей скобке соответствует закрывающая скобка.
2. Введены два и более подряд операнда.
3. Введены две и более подряд операции.
4. При подсчете результата операции деления, второй операнд ноль.
5. Закрывающая скобка введена раньше, чем открывающая скобка.

Пример обработки ошибок. (см. Рис. 4)

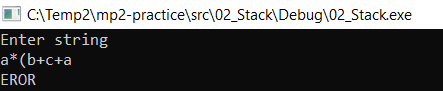


Рис. 4 Обнаружение ошибки программой

# Руководство программиста

## Описание структуры программы

* Stack.h – модуль с классом, содержащий описание и реализацию функций структуры данных Стек;
* Calculating.h, Calculating.cpp – модуль с набором функций и структур, реализующий обработку арифметических выражений;
* Main.cpp – модуль программы тестирования.

## Описание структур данных

### Класс TStack

#pragma once

template <class ValType>

class TStack

{

public:

int size;

ValType \* elements;

int top;

public:

TStack(int \_size = 15);

TStack(const TStack&);

~TStack();

bool IsFull() const;

bool IsEmpty() const;

void Push(ValType);

void Pop();

ValType Top() const;

};

Тип “ValType” описывает тип значения элементов стека.

**Описание методов класса:**

TStack(int)

**Назначение:** Выделение динамической памяти в куче.

**Входные параметры:** Размер стека

**Выходные данные:** Отсутствуют

TStack(const TStack&)

**Назначение:** Инициализирует объекты, копируя значения элементов из объекта

того же типа.

**Входные параметры:** Константная ссылка на объект типа Stack.

**Выходные данные:** Отсутствуют

~TStack()

**Назначение:** Освобождает выделенную память.

**Входные параметры:** Отсутствуют

**Выходные данные:** Отсутствуют

Void Push(ValType)

**Назначение:** Добавление элемента в стек.

**Входные параметры:** Элемент типа ValType

**Выходные данные:** Отсутствуют

ValType Top() const

**Назначение:** Возвращает значение верхнего элемента стека, не изменяя стек

**Входные параметры:** Отсутствуют

**Выходные данные:** Значение верхнего элемента стека

Void Pop()

**Назначение:** Изымает верхний элемент из стека

**Входные параметры:** Отсутствуют

**Выходные данные:** Верхний элемент стека

Bool IsEmpty() const

**Назначение:** Проверка стека на пустоту.

**Входные параметры:** Отсутствуют

**Выходные данные:** Значение типа Boolean

Bool IsFull() const

**Назначение:** Проверка стека на полноту.

**Входные параметры:** Отсутствуют

**Выходные данные:** Значение типа Boolean

### Класс Calculating

#pragma once

#include "Stack.h"

#include <string>

using namespace std;

class Calculating

{

private:

static bool priority(char, char);

static double calculator(double, double, char);

public:

static string create\_postfix(const string);

static void get\_operands(const string , double \*&, char \*&, int &);

static double calculate(const string, const double \*, const char \*, int);

};

**Описание методов класса:**

static string create\_postfix(const string);

**Назначение:** Возвращает строку в постфиксной форме.

**Входные параметры:** Cтрока

**Выходные данные:** Cтрока

static void get\_operands(const string, double \*&, char \*&, int &);

**Назначение:** Заполняет массив уникальных операндов и массив значений уникальных операндов

**Входные параметры:** Строка, указатель на ссылку на массив типа double, указатель на ссылку на массив char, ссылка на переменную int

static double calculate(const string, const double \*, const char \*, int);

**Назначение:** Возвращает значение арифметического выражения, записанного в постфиксной форме

**Входные параметры:** Строка, указатель на массив типа double, указатель на массив типа char, int

**Выходные данные:** объект типа double

## Описание алгоритмов

### Алгоритм работы Stack

Стек (англ. stack), магазин – схема запоминания информации, при которой каждый вновь поступающий ее элемент как бы «проталкивает» вглубь отведенного участка памяти находящиеся там элементы (подобно патрону, помещаемому в магазин винтовки) и занимает крайнее положение (так называемую вершину стека). При выдаче информации из стека выдается элемент, расположенный в вершине стека, а оставшиеся элементы продвигаются к вершине; следовательно, элемент, поступивший последним, выдается первым. В качестве структуры хранения стека предлагается использовать одномерный (одноиндексный) массив, размещаемый в динамической области памяти. Для описания структуры хранения следует использовать следующие переменные: elements – указатель на память, выделенную для хранения стека, size – размер выделенной памяти, top – индекс элемента массива, в котором хранится последнее добавленное значение стека. Для работы со стеком предлагается реализовать следующие операции: Метод Push – добавить элемент; При добавлении элемента в стек необходимо переместить указатель вершина стека, увеличить на 1, записать элемент в соответствующую позицию динамического массива. Метод Pop – удаление элемента; При удалении элемента из стека необходимо переместить указатель вершины стека, уменьшить на 1. Метод Top – просматривает верхний элемент стека; Возвращает значение из динамического массива по индексу вершины стека. Метод IsEmpty – проверить стек на пустоту; Стек пуст, если в нем нет ни одного элемента, т.е. когда size = top. Метод IsFull – проверить стек на полноту; Стек полон при исчерпании всей отведенной под хранение элементов памяти, т.е. когда значение size совпадает со значением top.

### Перевод в постфиксную форму

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Знак операции | “(“ | “)” | “+”“-“ | “\*”“/” |
| Приоритет | 0 | 1 | 2 | 3 |

1. Создаем два стека, Стек №1, Стек №2.
2. Если пришел операнд, то кладем его в Стек №2.
3. Если пришла левая открывающаяся скобка, то кладем ее в Стек №1.
4. Если пришла правая открывающая скобка, то изымаем операции из Стека №1 в Стек №2, пока не встретим левую открывающую скобку. После удаляем ее.
5. Если пришла операция, то смотрим приоритет операции, которая хранится на верхушке Стека №1. Если приоритет пришедшей операции больше либо равен, чем приоритет операции, которая хранится на верху Стека №1, то записываем пришедшую операцию в Стек №1. Иначе перекладываем все операции, приоритет которых больше либо равен приоритету операции на верху Стека №2, из Стека №1 в Стек №2.
6. Если дошли до конца выражения, то необходимо переложить все операции из Стека №1 в Стек №2.

**Пример**: A+B\*(C-D)

1. Пришел операнд ‘A’, кладем его в Стек №2.

|  |  |
| --- | --- |
| Стек№2 | A |
| Стек№1 |  |

1. Пришел оператор ‘+’, кладем его в Стек №1.

|  |  |
| --- | --- |
| Стек№2 | A |
| Стек№1 | + |

1. Пришел операнд ‘B’, кладем его в Стек №2.

|  |  |  |
| --- | --- | --- |
| Стек№2 | A | B |
| Стек№1 | + |  |

1. Пришел оператор ‘\*’, кладем его в Стек №1, проверяя приоритет.

|  |  |  |
| --- | --- | --- |
| Стек№2 | A | B |
| Стек№1 | + | \* |

1. Пришла левая скобка, кладем ее в Стек №1.

|  |  |  |  |
| --- | --- | --- | --- |
| Стек№2 | A | B |  |
| Стек№1 | + | \* | ( |

1. Пришел операнд ‘С’, кладем его в Стек №2.

|  |  |  |  |
| --- | --- | --- | --- |
| Стек№2 | A | B | C |
| Стек№1 | + | \* | ( |

1. Пришел оператор ‘-’, кладем его в Стек №1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Стек№2 | A | B | C |  |
| Стек№1 | + | \* | ( | - |

1. Пришел операнд ‘D’, кладем его в Стек №2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Стек№2 | A | B | C | D |
| Стек№1 | + | \* | ( | - |

1. Пришла правая скобка, изымаем операции из Стека №1 в Стек №2, пока не встретим левую открывающую скобку. Левую скобку удаляем.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Стек№2 | A | B | C | D | - |
| Стек№1 | + | \* |  |  |  |

10.Больше в строке ничего не осталось, перекладываем операции из Стека№1 в Стек№2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Стек№2 | A | B | C | D | - |
| Стек№1 | + | \* | / |  |  |

11.Получаем ответ.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Стек№2 | A | B | C | D | - | \* | + |
| Стек№1 |  |  |  |  |  |  |  |

Получаем строку ABCD-\*+ - это и есть постфиксная форма.

### Вычисление Арифметического выражения

Выражение просматривается посимвольно слева направо. При обнаружении операнда производится перевод в числовую форму и помещение в стек. При обнаружении знака операции происходит извлечение из стека двух значений, которые рассматриваются как операнд1 и операнд2. Выполнение операции происходит в порядке «Опернад2 операция Операнд1». Результат операции помещается в стек. По окончании просмотра всего выражения из стека извлекается окончательный результат.

Пример: ABCD-\*+(В качестве значений переменных, буду считать имя переменных).

1. Пришел операнд ‘A’, кладем его в Стек.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | A | B | C | D | - | \* | + |
|  | ↓ |  |  |  |  |  |  |
| Стек | A |  |  |  |  |  |  |

1. Пришел операнд ‘B’, кладем его в Стек.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение |  | B | C | D | - | \* | + |
|  |  | ↓ |  |  |  |  |  |
| Стек | A | B |  |  |  |  |  |

1. Пришел операнд ‘C’, кладем его в Стек.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение |  |  | C | D | - | \* | + |
|  |  |  | ↓ |  |  |  |  |
| Стек | A | B | C |  |  |  |  |

1. Пришел операнд ‘D’, кладем его в Стек.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение |  |  |  | D | - | \* | + |
|  |  |  |  | ↓ |  |  |  |
| Стек | A | B | C | D |  |  |  |

1. Пришел оператор ‘-’, изымаем 2 операнда из Стека и записываем в Стек C-D.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выражение |  |  |  | \* | + |
|  |  |  |  |  |  |
| Стек | A | B | C-D |  |  |

1. Пришел операнд ‘\*’, изымаем 2 операнда из Стека и записываем в СтекB\*(C-D).

|  |  |  |  |
| --- | --- | --- | --- |
| Выражение |  |  | + |
|  |  |  |  |
| Стек | A | B\*(C-D) |  |

7.Пришел операнд ‘+’, изымаем 2 операнда из Стека и записываем в СтекA+B\*(C-D).

|  |  |
| --- | --- |
| Выражение |  |
|  |  |
| Стек | A+B\*(C-D) |

Получаем ответ. Алгоритм закончен.

# Заключение

Была рассмотрена постфиксная форма, а также перевод арифметического выражения из инфиксной формы в постфиксную, основанный на структуре данный Стек.

# Литература

1. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гегреля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

# Приложения

## Приложение 1. Исходный код основной функции

#include "Stack.h"

#include "Calculating.h"

#include <iostream>

int main()

{

string s;

string postfixform;

double postfixformvolue = 0;

int size = 0;

cout << "Enter string" << endl;

cin >> s;

try

{

postfixform = Calculating::create\_postfix(s);

}

catch (const char\* str)

{

cout << str << endl;

}

cout << "postfix\_form:" << endl;

cout << postfixform << endl;

double \*exchange = new double[postfixform.length()];

char \* operands = new char[postfixform.length()];

try

{

Calculating::get\_operands(postfixform, exchange, operands, size);

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

postfixformvolue = Calculating::calculate(postfixform, exchange, operands, size);

}

catch (const char\* str)

{

cout << str << endl;

}

cout << "postfixform volue:" << endl;

cout << postfixformvolue;

system("pause");

}

## Приложение 2. Класс TStack

#pragma once

template <class ValType>

class TStack

{

public:

int size;

ValType \* elements;

int top;

public:

TStack(int \_size = 15);

TStack(const TStack&);

~TStack();

bool IsFull() const;

bool IsEmpty() const;

void Push(ValType);

void Pop();

ValType Top() const;

};

template <class ValType>

TStack <ValType> ::TStack(int \_size)

{

size = \_size;

top = -1;

elements = new ValType[size];

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

elements[i] = 0;

}

template <class ValType>

TStack <ValType> ::TStack(const TStack& a)

{

size = a.size;

top = a.top;

elements = new ValType[size];

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

elements[i] = a.elements[i];

}

template <class ValType>

TStack<ValType>::~TStack()

{

size = 0;

top = -1;

delete[] elements;

}

template <class ValType>

bool TStack<ValType>::IsFull() const

{

return (top == size);

}

template <class ValType>

bool TStack <ValType> ::IsEmpty() const

{

return (top == -1);

}

template <class ValType>

void TStack <ValType> ::Push(ValType a)

{

if (IsFull())

throw "Full Stack";

elements[++top] = a;

}

template <class ValType>

void TStack <ValType> ::Pop()

{

if (IsEmpty())

throw "Empty Stack";

top--;

}

template <class ValType>

ValType TStack <ValType> ::Top() const

{

if (IsEmpty())

throw "Empty Stack";

return elements[top];

}

## Приложение 3. Модуль Calculating.h

#pragma once

#include "Stack.h"

#include <string>

using namespace std;

class Calculating

{

private:

static bool priority(char, char);

static double calculator(double, double, char);

public:

static string create\_postfix(const string);

static void get\_operands(const string , double \*&, char \*&, int &);

static double calculate(const string, const double \*, const char \*, int);

};

## Приложение 4. Модуль Calculating.cpp

#include <iostream>

#include "Calculating.h"

bool Calculating::priority(char a, char b)

{

if (((a == '+' || a == '-') && (b == '\*' || b == '/' || b == '+' || b == '-')) || ((a == '\*' || a == '/') && (b == '\*' || b == '/')))

return false;

return true;

}

double Calculating::calculator(double a, double b, char c)

{

if (c == '+')

return a + b;

if (c == '-')

return a - b;

if (c == '\*')

return a \* b;

if (c == '/')

{

if (b == 0)

throw"EROR";

return a / b;

}

}

string Calculating::create\_postfix(const string s)

{

int flag = 0, flag1 = 0;

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

{

if (s[i] == '(')

flag++;

if (s[i] == ')')

flag--;

}

if (flag != 0)

throw "EROR";

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

{

if (s[i] == ')')

{

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

{

if (s[k] == '(')

flag1 = 1;

}

if (flag1 == 0)

throw"EROR";

}

}

for(int i = 1; i < s.length(); i++)

if (((s[i] == '+') || (s[i] == '-') || (s[i] == '\*') || (s[i] == '/')) && ((s[i-1] == '+') || (s[i-1] == '-') || (s[i-1] == '\*') || (s[i-1] == '/')))

throw "EROR";

for (int i = 1; i < s.length(); i++)

if (((s[i] != '+') && (s[i] != '-') && (s[i] != '\*') && (s[i] != '/')) && ((s[i - 1] != '+') && (s[i - 1] != '-') && (s[i - 1] != '\*') && (s[i - 1] != '/')))

throw "EROR";

TStack<char> operators;

TStack<char> operands;

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

{

if ((s[i] == '+') || (s[i] == '-') || (s[i] == '\*') || (s[i] == '/') || (s[i] == ')') || (s[i] == '('))

{

if (operators.IsEmpty()) operators.Push(s[i]);

else

{

if (s[i] == '(') operators.Push(s[i]);

else

{

if (s[i] == ')')

{

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

{

operands.Push(operators.Top());

operators.Pop();

}

operators.Pop();

}

else

{

if (priority(s[i], operators.Top()) == true) {

operators.Push(s[i]);

}

else

{

while ((operators.top != 0) && (operators.Top() != '(') && (priority(s[i], operators.Top()) != true))

{

operands.Push(operators.Top());

operators.Pop();

}

if (priority(s[i], operators.Top()) != true)

{

operands.Push(operators.Top());

operators.Pop();

}

operators.Push(s[i]);

}

}

}

}

}

else

operands.Push(s[i]);

}

while (operators.top != -1)

{

operands.Push(operators.Top());

operators.Pop();

}

string postfixform(operands.elements);

return postfixform;

}

void Calculating::get\_operands(const string postfixform, double \*&exchange, char \*&operands, int & size)

{

int j = 0;

double a, flag = 0;

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

{

if ((postfixform[i] != '+') && (postfixform[i] != '-') && (postfixform[i] != '\*') && (postfixform[i] != '/'))

{

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

if (postfixform[i] == postfixform[k])

{

flag = 1;

}

if (flag == 0)

{

operands[j] = postfixform[i];

cout << "Enter the meaning of " << postfixform[i] << endl;

cin >> a;

exchange[j] = a;

j++;

size++;

}

flag = 0;

}

}

}

double Calculating::calculate(const string postfixform, const double \*exchange, const char \*operands, int size)

{

double b;

TStack<double> finalvolue;

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

{

if ((postfixform[i] == '+') || (postfixform[i] == '-') || (postfixform[i] == '\*') || (postfixform[i] == '/'))

{

b = calculator(finalvolue.elements[finalvolue.top - 1], finalvolue.elements[finalvolue.top], postfixform[i]);

finalvolue.Pop();

finalvolue.Pop();

finalvolue.Push(b);

}

else

{

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

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

{

finalvolue.Push(exchange[k]);

break;

}

}

}

return finalvolue.elements[0];

}