Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе №3**

**«Вычисление арифметических выражений (стеки)»**

**Выполнил**:студент группы 381703-2

Безруков Павел Дмитриевич

**Проверил**:

Доцент кафедры МОСТ, к.т.н.

Сысоев А.B.

Нижний Новгород

2018

Содержание

[Введение 3](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962758)

[Постановка задачи 4](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962759)

[Руководство пользователя 5](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962760)

[Руководство программиста 7](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962761)

[Описание структур программы 7](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962762)

[Описание алгоритмов 8](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962763)

[Заключение 10](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962765)

[Литература 11](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962766)

[Приложения 1](file:///D:\Загрузки\Rodionov_Report.docx#_Toc270962767)**2**

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек. С этой целью в лабораторной работе изучаются различные варианты структуры хранения стеков и разрабатываются методы и программы решения ряда задач с использованием стеков. В качестве области приложений выбрана тема вычисления арифметических выражений, возникающей при трансляции программ на языке программирования высокого уровня в исполняемые программы.

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. Рассматриваемые в данной лабораторной работе алгоритмы являются начальным введением в область машинных вычислений.

# 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.

Проверка записи выражения на корректность состоит из:

1) Правильности расстановки скобок,

2) Правильности расстановки знаков операций.

Перевод в постфиксную форму производится только для корректных выражений, а вычисление – для корректных выражений, содержащих числовые операнды, переменные произвольного названия и допустимые знаки операций: сложения, вычитания, умножения и деления.

# Руководство пользователя

Данная программа написана в среде разработки Microsoft Visual Studio 2017 на языке С++.

На первом этапе работы программы пользователю необходимо выбрать пункт меню и ввести соответствующую цифру.



Рис.1. Первый этап выполнения программы.

При выборе первого пункта программа предполагает перевод выражения из инфиксной записи в постфиксную. Необходимо ввести выражение, которое не содержит пробелов и составлено из переменных, целых или дробных чисел и операций +,-,\*,/. Для выделения дробной части необходимо писать ‘.’.

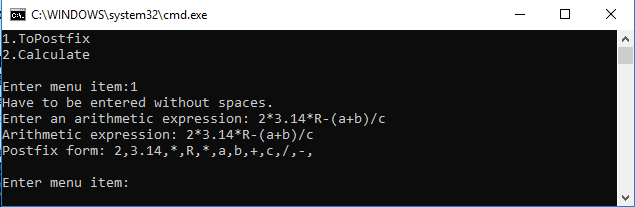


Рис.2. Получение постфиксной формы.

При вводе выражения **2\*3.14\*R-(a+b)/c** на консоль выведется постфиксная форма:

Postfix form: **2,3.14,\*,R,\*,a,b,+,c,/,-.**

И снова пользователю предлагается выбрать пункт меню.

При выборе второго пункта программа предполагает вычисления арифметического выражения. Необходимо ввести выражение, которое не содержит пробелов и состоит из целых или дробных чисел и операций +,-,\*,/.

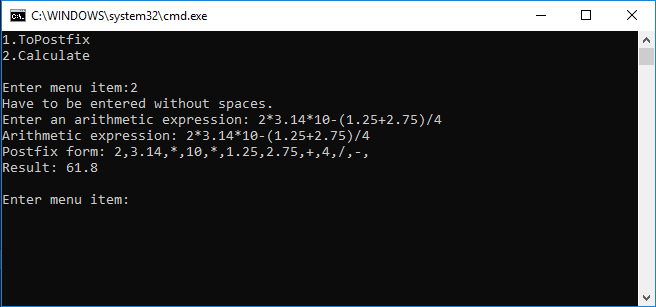


Рис.3. Калькулятор.

При вводе выражения **2\*3.14\*10-(1.25+2.75)/4** на консоль выведется постфиксная форма и результат вычисления:

Postfix form: **2,3.14,\*,10,\*,1.25,2.75,+,4,/,-**.

Result: **61.8**

И снова пользователю предлагается выбрать пункт меню.

При неверном вводе выражения программа сообщит об этом.

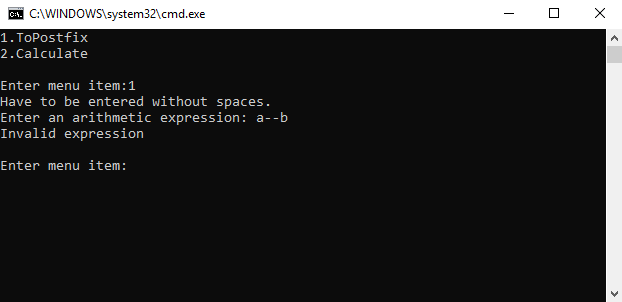


Рис.4. Неверный ввод.

# Руководство программиста

В программе реализованы два класса под названием **TStack** и **TPostfix**.

**Класс TStack:**

Является шаблонным

Private:

**pMem** - указатель на объект шаблонного типа, отвечающий за хранение элементов стека.

**size** - переменная типа **int**, хранящая размер стека.

**Top** – переменная **int**, хранит индекс верхнего элемента стека (если -1, то стек пуст)

Public:

**TStack**(int \_size) Конструктор с одним аргументом.

**TStack**(TStack &s) Конструктор копирования.

**IsEmpty** Метод, проверяющий, пустой ли стек.

**IsFull** Метод, проверяющий, полон ли стек.

**Pop** Метод, который позволяет узнать верхний элемент.

**Top** Метод, удаляющий верхний элемент.

**Push** Метод, добавляющий элемент в стек.

**~TStack** Деструктор.

**Класс TPostfix:**

Private:

**infix** Строка, которая хранит инфиксную форму.

**postfix** Строка, которая хранит постфиксную форму.

**IsOperator** Метод, проверяющий является ли символ оператором

**Priority** Метод, который узнаёт отношение приоритета между двумя операторами

**Operation** Метод, который производит вычисление с одним оператором.

Public:

**TPostfix**(string str = “a+b”) конструктор с аргументом и значением по умолчанию.

**TPostfix** (TPostfix &p) конструктор копирования.

**GenInfix** Метод, возвращающий строку **infix**.

**GetPostfix** Метод, возвращающий строку **postfix**.

**CheckInfix** Метод, который проверяет на корректность ввода выражения.

**ToPostfix** Метод, преобразующий инфиксную форму в постфиксную.

**Calculate** Метод, который вычисляет арифметическое выражение.

**Описание структуры программы**

Программа содержит три проекта:

1. **base**,
2. **base\_test**,
3. **gtest**.

Проект base содержит модули **“stack.h”** с объявлением и реализацией класса **TStack**, “**postfix**.h” с объявлением класса **TPostfix**, **“postfix.cpp”** с реализацией методов класса **TPostfix** и **“sample\_postfix.cpp”** для составления пользовательского меню.

Проект **base\_test** содержит набор необходимых тестов **Google Test**, проверяющих правильность реализации основных классов.

Проект **gtest** содержит необходимую структуру для работы тестов **Google Test**.

Программа построена на использовании меню, который реализован в главной функции **main()** с помощью оператора **switch**. В нём создаётся объект типа **TPostfix** и вызываются его методы.

**Описание алгоритмов**

**Алгоритмы класса TStack:**

**IsEmpty:** метод, который возвращает тип **bool**. **True** если **top** равен -1.

**IsFull:** метод, который возвращает тип **bool**. **True** если **top** равен (**size**-1).

**Pop:** метод, который удаляет верхний элемент стека. Если стек не пуст, возвращает верхний элемент и уменьшает **top** на 1.

**Top:** метод, который возвращает верхний элемент стека, если стек не пуст.

**Push:** метод, который добавляет значение в стек. Если стек не полон, увеличивает top и присваивает элементу массива **pMem** с индексом top переменную, которая пришла в метод в аргументе.

**Алгоритмы класса TPostfix:**

**IsOpetator**: метод, принимающий аргумент типа **char**. Проверяет, является ли символ оператором, Возвращает тип **bool**.

**Priority**: метод, который принимает два символа типа **char**, проверяет их отношение приоритетности, возвращает 1, если первый аргумент приоритетнее второго, 0, если приоритеты равны, -1, если второй приоритетнее первого.

**Operation**: Метод, который принимает на вход две переменных типа **double** и одну переменную типа **char**. С помощью оператора switch проверяет какой символ был передан методу и если он является одним из операторов, то возвращает результат вычисления (1арг операция 2арг) типа **double**.

**GetInfix** и **GetPostfix**: методы, которые возвращают инфиксную и постфиксную форму соответственно, типа **string**.

**Checkinfix**: Метод, проверяющий корректность строки **infix**. Возвращает тип **bool**. Происходит проверка на пустоту строки, является ли первый и последний элементы операторами и проверки на количество скобок и операторов.

**ToPostfix**: метод, преобразующий инфиксную форму в постфиксную и возвращающий строку, содержащую постфиксную форму. Алгоритм перевода в постфиксную запись обрабатывает исходный массив лексем и строит новый массив из тех же лексем, расположенных в другом порядке. Кроме того, необходим еще стек – аналогичный массив, используемый для временного хранения операций.

Алгоритм перевода выражения в постфиксную запись следующий.

1. Константы и переменные кладутся в формируемую запись в порядке их появления в исходном массиве.
2. При появлении операции в исходном массиве:
   1. если в стеке нет операций или верхним элементом стека является открывающая скобка, операции кладётся в стек;
   2. если новая операции имеет больший*[\*](http://natalia.appmat.ru/c&c++/postfisso.html" \l "nota)* приоритет, чем верхняя операции в стеке, то новая операции кладётся в стек;
   3. если новая операция имеет меньший или равный приоритет, чем верхняя операции в стеке, то операции, находящиеся в стеке, до ближайшей открывающей скобки или до операции с приоритетом меньшим, чем у новой операции, перекладываются в формируемую запись, а новая операции кладётся в стек.
3. Открывающая скобка кладётся в стек.
4. Закрывающая скобка выталкивает из стека в формируемую запись все операции до ближайшей открывающей скобки, открывающая скобка удаляется из стека.
5. После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке, перекладываются в формируемое выражение.

**Calculate**: метод, вычисляющий выражение используя постфиксную форму. Возвращает результат вычисления типа **double**.

Алгоритм следующий:

Просматриваем постфиксную запись. Значения констант кладутся в стек. Когда встречается операция, из стека берутся два верхних значения, вычисляется результат применения операции к этим значениям, и результат помещается в стек. Результатом вычисления является элемент, который остался в стеке в конце после просматривания постфиксной записи.

# Заключение

Благодаря данной лабораторной работе я познакомился с такой структурой данных, как стек. Также я узнал о реализации постфиксной формы и способами эффективно управлять памятью с помощью стека, который значительно ускорил и упростил работу.

# 

# Литература

1. Бьерн Страуструп. «Язык программирования С++»,
2. Роберт Лафоре. «Объектно-ориентированное программирование в С++»,
3. Герберт Шилдт. «C++. Руководство для начинающих»,
4. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. «Лабораторный практикум. Учебно-методическое пособие».

# Приложение

**Модуль “stack.h”**

#ifndef \_\_STACK\_H\_\_

#define \_\_STACK\_H\_\_

#include <iostream>

const int MaxStackSize = 100;

template <class T>

class TStack

{

T \*pMem;

int size;

int top;

public:

TStack(int \_size)

{

size = \_size;

top = -1;

if ((size < 1) || (size > MaxStackSize))

throw size;

pMem = new T[size];

}

TStack(TStack &s)

{

size = s.size;

top = s.top;

pMem = new T[size];

for (int i = 0; i < top + 1; i++)

pMem[i] = s.pMem[i];

}

bool IsEmpty()

{

return (top == -1);

}

bool IsFull()

{

return (top == (size - 1));

}

T Pop() //взять

{

if (!IsEmpty())

return pMem[top--];

else

throw "Empty";

}

T Top()//посмотреть

{

if (!IsEmpty())

return pMem[top];

else

throw "Empty";

}

void Push(T v)//положить

{

if (!IsFull())

pMem[++top] = v;

else

throw "Empty";

}

~TStack()

{

delete[] pMem;

}

};

#endif

**Модуль “postfix.h”**

#ifndef \_\_POSTFIX\_H\_\_

#define \_\_POSTFIX\_H\_\_

#include <string>

#include "stack.h"

using namespace std;

class TPostfix

{

string infix;

string postfix;

bool IsOperator(char op);

int Priority(char a, char b);

double Operation(double right, double left, char c);

public:

TPostfix(string str = "a+b")

{

infix = str;

}

TPostfix(TPostfix &p)

{

infix = p.infix;

postfix = p.postfix;

}

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

bool CheckInfix();

string ToPostfix();

double Calculate(); // Вычисление по постфиксной форме

};

#endif

**Модуль “postfix.cpp”**

#include "postfix.h"

#include "stack.h"

#include <string>

using namespace std;

bool TPostfix::IsOperator(char op)

{

if (op == '+' || op == '-' || op == '\*' || op == '/')

return true;

else

return false;

}

int TPostfix::Priority(char a, char b)//1 - a приоритетнее b,0 - одинаковы,-1 - b приоритетнее

{

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

return 0;

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

return 1;

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

return -1;

if ((a == '+' || a == '-') && (b == '+' || b == '-'))

return 0;

throw "not operation";

}

double TPostfix::Operation(double left, double right, char c)

{

switch (c)

{

case '+':

{

return left + right;

}

case '-':

{

return left - right;

}

case '\*':

{

return left \* right;

}

case '/':

{

if (right == 0)

throw 0;

return left / right;

}

default:

throw "invalid operation";

}

}

bool TPostfix::CheckInfix()//пустой, первый или последний операторы,два подряд оператора,нет операторов

{

if (infix.empty())

return false;

else

if (IsOperator(infix[0]) || IsOperator(infix[infix.size() - 1]))

return false;

for (int i = 0; i < infix.size() - 1; i++)

{

if (IsOperator(infix[i]) && IsOperator(infix[i + 1]))

return false;

}

int count1 = 0;

int count2 = 0;

int count3 = 0;

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

{

if (infix[i] == ' ')

return false;

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

count1++;

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

count2++;

if (IsOperator(infix[i]))

count3++;

}

if ((count1 != count2) || count3 == 0)

return false;

return true;

}

string TPostfix::ToPostfix()

{

TStack<char> opers(infix.size());

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

{

if (!IsOperator(infix[i]) && infix[i] != '(' && infix[i] != ')')

{

int j = i;

while (!IsOperator(infix[j]) && j != infix.size() && infix[j] != ')')

{

postfix += infix[j];

j++;

}

i = j - 1;

postfix += ',';

}

else

if (opers.IsEmpty())

opers.Push(infix[i]);

else

if (opers.Top() == '(')

opers.Push(infix[i]);

else

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

opers.Push(infix[i]);

else

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

{

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

{

postfix += opers.Pop();

postfix += ',';

}

opers.Pop();

}

else

if (Priority(infix[i], opers.Top()) == 1)

opers.Push(infix[i]);

else

{

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

{

if (Priority(infix[i], opers.Top()) == 1)

break;

postfix += opers.Pop();

postfix += ',';

if (opers.IsEmpty())

break;

}

opers.Push(infix[i]);

}

}

while (!opers.IsEmpty())

{

postfix += opers.Pop();

postfix += ',';

}

return postfix;

}

double TPostfix::Calculate()//элементы разделяются запятыми

{

ToPostfix();

TStack<double> nums(infix.size());

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

{

if (!IsOperator(postfix[i]))

{

string doub;

doub += postfix[i];

int j = i + 1;// добавляем в doub и переходим к следующему эл-ту

while (postfix[j] != ',')

{

doub += postfix[j];

j++;

}

nums.Push(atof(doub.c\_str()));// atof из string в double

i = j;//на место ','

}

else

{

nums.Push(Operation(nums.Pop(), nums.Pop(), postfix[i]));

i++;//на место ','

}

}

return nums.Top();

}

**Модуль “sample\_postfix.cpp”**

#include <iostream>

#include <string>

#include "postfix.h"

using namespace std;

int main()

{

int item = 0;

string expression;

cout << "1.ToPostfix" << endl;

cout << "2.Calculate" << endl;

cout << endl;

while (true)

{

cout << "Enter menu item:";

cin >> item;

switch (item)

{

case 1:

{

cout << "Have to be entered without spaces." << endl;

cout << "Enter an arithmetic expression: ";

cin >> expression;

TPostfix postfix(expression);

if (!postfix.CheckInfix())

{

cout << "Invalid expression" << endl << endl;

break;

}

cout << "Arithmetic expression: " << postfix.GetInfix() << endl;

postfix.ToPostfix();

cout << "Postfix form: " << postfix.GetPostfix() << endl << endl;

break;

}

case 2:

{

cout << "Have to be entered without spaces." << endl;

cout << "Enter an arithmetic expression: ";

cin >> expression;

TPostfix postfix(expression);

if (!postfix.CheckInfix())

{

cout << "Invalid expression" << endl << endl;

break;

}

cout << "Arithmetic expression: " << postfix.GetInfix() << endl;

postfix.ToPostfix();

cout << "Postfix form: " << postfix.GetPostfix() << endl;

cout << "Result: " << postfix.Calculate() << endl << endl;

break;

}

default:

break;

}

}

return 0;

}

**Google Test**

**Модуль “test\_tpostfix.cpp”**

#include "postfix.h"

#include "postfix.cpp"

#include <gtest.h>

#include <string>

TEST(TPostfix, can\_create\_postfix)

{

ASSERT\_NO\_THROW(TPostfix p);

}

TEST(TStack, can\_create\_copied\_postfix)

{

TPostfix p;

ASSERT\_NO\_THROW(TPostfix p1(p));

}

TEST(TPostfix, can\_get\_infix)

{

TPostfix p("a+b");

EXPECT\_EQ("a+b", p.GetInfix());

}

TEST(TPostfix, can\_create\_postfix\_from\_infix1)

{

TPostfix p("a+b");

p.ToPostfix();

EXPECT\_EQ("a,b,+,", p.GetPostfix());

}

TEST(TPostfix, can\_create\_postfix\_from\_infix2)

{

TPostfix p("a+(b-c)\*d-e+f");

p.ToPostfix();

EXPECT\_EQ("a,b,c,-,d,\*,+,e,-,f,+,", p.GetPostfix());

}

TEST(TPostfix, CheckInfix\_return\_true\_when\_infix\_is\_correct)

{

TPostfix p("a+b");

EXPECT\_TRUE(p.CheckInfix());

}

TEST(TPostfix, CheckInfix\_return\_false\_when\_infix\_is\_invalid)

{

TPostfix p("(a+-b");

EXPECT\_FALSE(p.CheckInfix());

}

TEST(TStack, throw\_when\_division\_by\_zero)

{

TPostfix p("2+2/0");

ASSERT\_ANY\_THROW(p.Calculate());

}

TEST(TPostfix, can\_calculate\_expression1)

{

TPostfix p("2+2\*(2+2)");

EXPECT\_EQ(10,p.Calculate());

}

TEST(TPostfix, can\_calculate\_expression2)

{

TPostfix p("(1+2)-3/6\*2");

EXPECT\_EQ(2, p.Calculate());

}

TEST(TPostfix, can\_calculate\_expression3)

{

TPostfix p("(0.5+0.5)/(0.25+0.25)");

EXPECT\_EQ(2, p.Calculate());

}

TEST(TPostfix, can\_calculate\_expression4)

{

TPostfix p("0.1\*(10\*0.1)\*(10/10)\*100");

EXPECT\_EQ(10, p.Calculate());

}

**Модуль “test\_tstack.cpp”**

#include "stack.h"

#include <gtest.h>

TEST(TStack, can\_create\_stack\_with\_positive\_length)

{

ASSERT\_NO\_THROW(TStack<int> st(5));

}

TEST(TStack, cant\_create\_stack\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TStack<int> st(-5));

}

TEST(TStack, cant\_create\_stack\_with\_too\_large\_length)

{

ASSERT\_ANY\_THROW(TStack<int> st(MaxStackSize + 1));

}

TEST(TStack, can\_create\_copied\_stack)

{

TStack<int> st(1);

ASSERT\_NO\_THROW(TStack<int> st1(st));

}

TEST(TStack, IsEmpty\_return\_true\_when\_stack\_is\_empty)

{

TStack<int> st(1);

EXPECT\_TRUE(st.IsEmpty());

}

TEST(TStack, IsFull\_return\_true\_when\_stack\_is\_full)

{

TStack<int> st(1);

st.Push(1);

EXPECT\_TRUE(st.IsFull());

}

TEST(TStack, can\_add\_value\_to\_stack\_and\_view\_top\_value)

{

TStack<int> st(1);

st.Push(1);

EXPECT\_EQ(1, st.Top());

}

TEST(TStack, can\_delete\_value\_from\_stack)

{

TStack<int> st(2);

st.Push(1);

st.Push(2);

st.Pop();

EXPECT\_TRUE(st.Top(),1);

}

TEST(TStack, throw\_when\_delete\_from\_empty\_stack)

{

TStack<int> st(1);

ASSERT\_ANY\_THROW(st.Pop());

}

TEST(TStack, throw\_when\_add\_to\_full\_stack)

{

TStack<int> st(1);

st.Push(1);

ASSERT\_ANY\_THROW(st.Push(1));

}

TEST(TStack, throw\_when\_view\_top\_value\_stack)

{

TStack<int> st(1);

st.Push(1);

ASSERT\_ANY\_THROW(st.Push(1));

}