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 постфиксной формы арифметических выражений (стеки)»**

**Выполнил**:студент группы 381703-2

Баранов А.А.

**Проверил**:

Доцент кафедры МОСТ, к.т.н.

Сысоев А.B.

Нижний Новгород

2018

**Содержание**

[**Введение**](#_30j0zll) **3**

[**Постановка задачи**](#_1fob9te) **5**

[**Руководство пользователя**](#_3znysh7) **6**

[**Руководство программиста**](#_kvzkkgablhm8) **7**

[Описание структуры программы](#_2et92p0) 7

[Описание алгоритмов](#_iulrxd67w54p) 8

[**Заключение**](#_d2l6mmjsv1gs) **10**

[**Литература**](#_3dy6vkm) **11**

[**Приложение**](#_11oqarw513eq) **12**

# 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

Требуется разработать алгоритм переводящий арифметическое выражение(заданное в формуле с переменными) в постфиксную форму, с использованием стека.

Само вычисление выражение будет производится последовательным вводом аргументов, учавствующих в выражении.

# Руководство пользователя

На первом этапе работы программы пользователю необходимо ввести арифметическое (инфиксное) выражение.Пользователю требуется ввести именно формулу, без использования числовых значений.



Рис.1. Старт программы

После этого пользователю выведут постфиксную форму выражения и предложат ввести числовые аргументы

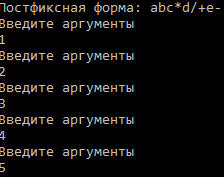


Рис.3. Вывод постфиксной формулы и ввод переменных

И соответственно выведет результат вычислений и предложит пользователю закончить программу.



Рис.4.Вывод результата.

# Руководство программиста

В программе реализованы два класса под названием «Стек» и «Постфикс».

**Класс «Стек» - шаблонный, имеющий три приватных поля:**

1. pMem: указатель на объект шаблонного типа, отвечающий за хранение элементов стека,
2. size: переменная типа int, отвечающая за размер стека,
3. top: переменная типа int, отвечающая за индекс последнего элемента в стеке (инициализирована значением -1, если стек пуст).

Помимо этого, в шаблонном классе «Стек» имеются следующие методы:

1. Все необходимые служебные методы,
2. IsEmpty: метод, который проверяет стек на пустоту и возвращает тип bool,
3. IsFull: метод, который проверяет стек на полноту и возвращает тип bool,
4. Pop: метод, который удаляет последний элемент в стеке ,
5. Put: метод, который записывает элемент в стек,
6. Get: метод, который возвращает последний элемент записанный в стек.

**Класс «Постфикс» имеет следующие приватные поля:**

1. infix: переменная типа string, хранит в себе исходное арифметическое выражение,
2. postfix: переменная типа string, хранит в себе преобразованное выражение в постфиксную форму,
3. count\_of\_args: переменная типа int,хранит в себе количество чисел в выражении
4. Все необходимые служебные методы,
5. GetInfix и GetPostfix: методы, возвращающие поля infix и postfix соответственно,
6. ToPostfix: метод, получающий из infix значение postfix,
7. Calculate: метод, в который заносит числовые аргументы и вычисляющий результат выражения.

Кроме того, программа содержит в себе следующие вспомогательные функции:

1. IsOperation: функция, которая проверяет переданный символ на операцию,
2. Priority: функция, возвращающая приоритет переданной операции,
3. GetArgs: функция, записывающая числа для подстановки в выражение и возвращающее это число,
4. CalculateCountOfArgs: функция, которая проверяет повторяющиеся переменные.

## Описание структуры программы

Программа содержит три основных проекта:

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

Проект base содержит модули “stack.h” с объявлением и реализацией класса «Стек», “postfix.h” с объявлением класса «Постфикс», “postfix.cpp” с реализацией методов класса «Постфикс» и с вспомогательными функциями 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.

## Описание алгоритмов

**Опишем алгоритмы реализации методов класса «Стек»:**

1. IsEmpty: метод, который проверяет стек на пустоту и возвращает тип bool

*Метод сравнивает значение поля класса top со значением -1.*

1. IsFull: метод, который проверяет стек на полноту и возвращает тип bool

*Метод сравнивает значение поля класса top со значением (size – 1).*

1. Pop: метод, который удаляет последний элемент в стеке

*Метод уменьшает значение переменной top на единицу,но сначала проверяет не пустой ли стек.*

1. Put: метод, который записывает элемент в стек

*Метод записывает элемент в стек и увеличивает значение переменной top на единицу.*

**Опишем алгоритмы реализации методов класса «Постфикс»:**

1. GetInfix и GetPostfix: методы, возвращающие поля infix и postfix соответственно

*Методы возвращают значения соответствующих значения переменных.*

1. ToPostfix: метод, получающий из infix значение postfix

*Входная строка просматривается посимвольно слева направо до достижения конца строки. Функция isalpha определяет является ли символ буквой. Буквы по мере их появления переписываются в postfix. При появлении во входной строке операции, происходит вычисление приоритета данной операции. Знак данной операции помещается в стек, если:*

* *Стек пуст,*
* *Приоритет операции строго больше приоритета операции, лежащей на вершине стека,*

*В противном случае из стека извлекаются все знаки операций с приоритетом больше или равным приоритету текущей операции. Они переписываются в выходную строку, после чего знак текущей операции помещается в стек. После просмотра всей входной строки происходит последовательное извлечение всех элементов стека с одновременной записью знаков операций, извлекаемых из стека, в выходную строку.*

1. GetArgs: функция, записывающая числа для подстановки в выражение и возвращающее это число,

*Метод с помощью цикла for считывает всевозможные переменные в арифметическом выражении, после чего в нужной последовательности вписывает их в массив .*

1. CalculateCountOfArgs: функция, которая проверяет повторяющиеся переменные,

*Метод с помощью цикла for считывает всевозможные переменные в переменную count\_of\_args и в случае повторного использования вычитает 1 .*

1. Calculate: метод, вычисляющий результат выражения

*Выражение просматривается посимвольно слева направо. При обнаружении операнда производится добавление в стек операндов соответствующего аргумента. При обнаружении знака операции происходит извлечение из стека двух значений, которые рассматриваются как операнд2 и операнд1 соответственно, и над ними производится обрабатываемая операция. Результат этой операции помещается в стек. По окончании просмотра всего выражения из стека извлекается окончательный результат.*

# Заключение

Текущая лабораторная работа познакомила нас с такой схемой запоминания информации, как стек, без которой было бы намного сложнее реализовать данную программу. Стек позволяет управлять памятью наиболее эффективным образом, что значительно упрощает некоторые алгоритмы.

# 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\_\_

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 is not correct";

pMem = new T[size];

}

~TStack()

{

delete [] pMem;

}

void Put(const T &elem)

{

if (!IsFull())

{

top += 1;

pMem[top] = elem;

}

else

throw ("cant put");

}

void Pop()

{

if (!IsEmpty())

top-=1;

else

throw "cant pop";

}

T Get()

{

return pMem[top];

}

bool IsEmpty()

{

return top == -1;

}

bool IsFull()

{

return top == (size-1);

}

};

#endif

**Модуль “postfix.h”**

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

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

#include <string>

#include <cctype>

#include "stack.h"

using namespace std;

class TPostfix

{

string infix;

string postfix;

bool OperationIs(char inf\_elem);

int Priority(char sym);

int count\_of\_args=0;

public:

TPostfix()

{

infix = "a + b";

count\_of\_args = 2;

}

TPostfix(string \_infix,int count)

{

infix = \_infix;

count\_of\_args = count;

}

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

double\* GetArgs(double \*arguments);

void ToPostfix();

void CalculateCountOfArgs();

int GetCountOfArgs() { return count\_of\_args; }

double Calculate(int count,double \*arguments);

};

#endif

**Модуль “postfix.cpp”**

#include "postfix.h"

#include "stack.h"

void TPostfix::ToPostfix()

{

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

postfix = "";

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

{

if (isalpha(infix[i]))

{

postfix += infix[i];

}

else if (OperationIs(infix[i]))

{

if (operations.IsEmpty())

operations.Put(infix[i]);

else if (Priority(infix[i]) <= Priority(operations.Get()))

{

postfix += operations.Get();

operations.Pop();

if (Priority(infix[i]) == Priority(operations.Get()))

{

postfix += operations.Get();

operations.Pop();

}

operations.Put(infix[i]);

}

else

operations.Put(infix[i]);

}

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

{

TStack<char> operations1(infix.size());

while (infix[i] != ')')

{

if (isalpha(infix[i]))

{

postfix += infix[i];

}

else if (OperationIs(infix[i]))

{

if (operations1.IsEmpty())

operations1.Put(infix[i]);

else if (Priority(infix[i]) <= Priority(operations1.Get()))

{

postfix += operations1.Get();

operations1.Pop();

if (Priority(infix[i]) == Priority(operations1.Get()))

{

postfix += operations1.Get();

operations1.Pop();

}

operations1.Put(infix[i]);

}

else

operations1.Put(infix[i]);

}

i++;

}

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

{

while (!operations1.IsEmpty())

{

postfix += operations1.Get();//abc\*+

operations1.Pop();

}

}

}

}

while (!operations.IsEmpty())

{

postfix += operations.Get();//abc\*+

operations.Pop();

}

}

bool TPostfix::OperationIs(char inf\_elem)

{

if (inf\_elem == '+' || inf\_elem == '-' || inf\_elem== '\*' || inf\_elem == '/')

return true;

return false;

}

double\* TPostfix::GetArgs(double \*arguments)//12

{

string tmp=postfix;//ab+a

string args;

double param[100];

int b=0;

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

{

if (isalpha(tmp[i]))//ab

{

if (b == GetCountOfArgs())

b = 0;

param[i] = arguments[b];//a=1 b=2

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

{

if (tmp[i] == tmp[j])//j=4

param[j] = arguments[b];//1

else

continue;

}

b++;

}

else

param[i] = 0;

}

return param;

}

void TPostfix::CalculateCountOfArgs()

{

string tmp = infix;

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

{

if (isalpha(tmp[i]))

{

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

{

if (tmp[i] == tmp[j])

{

tmp[j] = ' ';

count\_of\_args--;

}

}

}

}

}

int TPostfix::Priority(char sym)

{

if (sym == '\*' || sym == '/')

return 2;

if (sym == '+' || sym == '-')

return 1;

}

double TPostfix::Calculate(int count,double \*arguments)// пользователь передает колличесво аргументов их значения

{

TStack<double> res(99);

double tmp1;

double tmp2;

double \*args = GetArgs(arguments);

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

{

if (isalpha(postfix[i]))

{

res.Put(args[i]);

}

else if(OperationIs(postfix[i]))

{

tmp1 = res.Get();

res.Pop();

tmp2 = res.Get();

res.Pop();

if (postfix[i] == '+')

res.Put(tmp1 + tmp2);

if (postfix[i] == '-')

res.Put(tmp2 - tmp1);

if (postfix[i] == '/')

res.Put(tmp2 / tmp1);

if (postfix[i] == '\*')

res.Put(tmp2 \* tmp1);

}

}

return res.Get();

}

**Модуль “sample\_postfix.cpp”**

#include <iostream>

#include <string>

#include <cctype>

#include "postfix.h"

using namespace std;

int main()

{

string expression;

string tmp;

double res;

int count\_args=0;

setlocale(LC\_ALL, "Russian");

cout << "Введите формулу арифметического выражения: ";

cin >> expression;

for (int j = 0; j < expression.size(); j++)

{

if (isalpha(expression[j]))

count\_args++;

}

TPostfix postfix(expression,count\_args);

cout << "Арифметическое выражение: " << postfix.GetInfix() << endl;

postfix.ToPostfix();

cout << "Постфиксная форма: " << postfix.GetPostfix() << endl;

postfix.CalculateCountOfArgs();

double \*args;

args = new double[postfix.GetCountOfArgs()];

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

{

cout << "Введите аргументы" << endl;

cin >> args[i];

}

res = postfix.Calculate(postfix.GetCountOfArgs(),args);

/\* try

{

res=postfix.GetCountOfArgs();

}

catch (...)

{

cout << res;

}\*/

cout <<"Результат выражения "<< res << endl;

return 0;

}

**Google Test**

**Модуль “test\_tpostfix.cpp”**

#include "postfix.h"

#include <gtest.h>

TEST(TPostfix, can\_create\_postfix)

{

ASSERT\_NO\_THROW(TPostfix p);

}

TEST(TPostfix, can\_set\_infix)

{

ASSERT\_NO\_THROW(TPostfix p("a+b",2));

}

TEST(TPostfix, can\_get\_infix)

{

TPostfix p("a+b",2);

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

}

TEST(TPostfix, can\_get\_postfix\_of\_two\_arguments)

{

TPostfix p("a+b",2);

p.ToPostfix();

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

}

TEST(TPostfix, can\_get\_postfix\_of\_three\_arguments)

{

TPostfix p("a+b\*c",3);

p.ToPostfix();

EXPECT\_EQ("abc\*+", p.GetPostfix());

}

TEST(TPostfix, can\_get\_postfix\_with\_parentheses)

{

TPostfix p("a+(b-c)",3);

p.ToPostfix();

EXPECT\_EQ("abc-+", p.GetPostfix());

}

TEST(TPostfix, can\_calculate\_postfix)

{

TPostfix p("a+b",2);

p.ToPostfix();

double tmp[2] = { 2,2 };

EXPECT\_EQ(4, p.Calculate(2,tmp));

}

**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\_too\_large\_length)

{

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

}

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

{

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

}

TEST(TStack,check\_IsEmpty\_with\_empty\_stack)

{

TStack<int> st(5);

EXPECT\_TRUE(st.IsEmpty());

}

TEST(TStack, check\_IsEmpty\_with\_not\_empty\_stack)

{

TStack<int> st(5);

st.Put(1);

EXPECT\_FALSE(st.IsEmpty());

}

TEST(TStack, check\_IsFull\_with\_full\_stack)

{

TStack<int> st(5);

while(!st.IsFull())

st.Put(1);

EXPECT\_TRUE(st.IsFull());

}

TEST(TStack, check\_IsFull\_with\_not\_full\_stack)

{

TStack<int> st(5);

st.Put(1);

EXPECT\_FALSE(st.IsFull());

}

TEST(TStack, throws\_when\_Pop\_stack\_IsEmty)

{

TStack<int> st(5);

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

}

TEST(TStack, throws\_when\_Push\_stack\_IsFull)

{

TStack<int> st(5);

while (!st.IsFull())

st.Put(1);

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

}

TEST(TStack, can\_get\_top\_element\_from\_stack)

{

TStack<int> st(5);

st.Put(1);

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

}