Министерство образования Российской Федерации

Федеральное государственное автономное образовательное учреждение  
профессионального образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

**Вычисление арифметических выражений**

**Выполнил:**

студент института ИТММ гр. 0826-1

Варавин А.О.

**Проверил:**

Доцент кафедры МОСТ

Сысоев. А.В.

Нижний Новгород

2016 г.

Содержание

[Введение 3](#_Toc468653108)

[Постановка задачи 4](#_Toc468653109)

[Руководство пользователя 5](#_Toc468653110)

[Руководство программиста 6](#_Toc468653111)

[Описание структуры программы 6](#_Toc468653112)

[Описание структур данных 6](#_Toc468653113)

[Описание алгоритмов 6](#_Toc468653114)

[Заключение 8](#_Toc468653115)

[Литература 9](#_Toc468653116)

[Приложения 10](#_Toc468653117)

[Код программы 10](#_Toc468653118)

# Введение

Обратная польская запись — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Из-за отсутствия скобок обратная польская запись короче инфиксной. За этот счёт при вычислениях на калькуляторах повышается скорость работы оператора (уменьшается количество нажимаемых клавиш), а в программируемых устройствах сокращается объём тех частей программы, которые описывают вычисления. Последнее может быть немаловажно для портативных и встроенных вычислительных устройств, имеющих жёсткие ограничения на объём памяти.

# Постановка задачи

Разработать и реализовать многофайловую программу на языке С++, состоящую из следующих файлов:

1. Шаблонного класса TStack
2. Класса TPostfix для работы с постфиксной формой.
3. Набора тестов для классов TStack и TPostfix на базе Google Test.
4. Примера выполнения программы для работы с выражением.

# Руководство пользователя

1. При запуске программы будет предложено ввести арифметическое выражение, придерживаясь некоторых правил ввода:

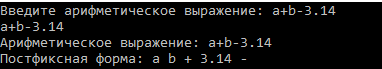
**Доступные операции:**

* +
* -
* /
* \*
* (
* )
* sin()
* cos()
* tg()
* ctg()
* sqrt()

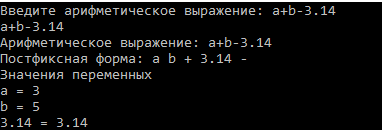
**Символы, доступные при вводе операндов:**

* ‘a’-‘z’
* ‘A’-‘Z’
* ‘0’-‘9’
* ‘.’

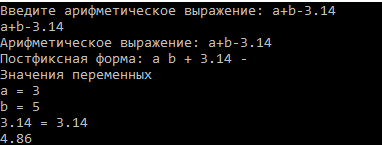
1. После завершения ввода программой будет проведена проверка корректности ввода и, в случае положительного результата, на экран будет выведена постфиксная форма записи введенного арифметического выражения.



1. На экран выводятся все найденные в арифметическом уравнении операнды для ввода их значений. Значение числовых операндов будут заполнены автоматически.



1. Программа завершит подсчёт значения выражения при данных значениях операндов и выведет ответ на экран



# Руководство программиста

## Описание структуры программы

Программа состоит из 4 основных частей:

* Класс TPostfix. Включается в себя 2 файла postfix.h и postfix.cpp
* Шаблонный класс TStack, реализованный в файле stack.h
* Тесты для классов TStack и TPostfix на базе Google Test. Состоит из файлов test\_main.cpp, test\_tpostfix.cpp, test\_track.cpp, а также фреймворка gtest.
* Образец программы, представленный файлом sample\_postfix.cpp

## Описание структур данных

Для хранения данных во время работы методов класса TPostfix разработан шаблонный класс TStack, представляющий собой список элементов, организованных по принципу LIFO. В классе реализованы автоматические проверки на пустоту и заполненность стека, а также автоматическое увеличение размера стека при его заполнении.

## Описание алгоритмов

1. **Преобразование выражения в ОПЗ**

Пока есть элементы:

Читаем очередной символ.

* Если элемент является операндом, добавляем его в постфиксную форму (также единственный раз добавляется в вектор операндов).
* Если элемент является открывающей скобкой, помещаем его в стек.
* Если символ является оператором о1, тогда:

1) пока приоритет текущего оператора меньше или равен приоритету оператора, находящегося на вершине стека

… выталкиваем верхний элемент стека в выходную строку

2) иначе помещаем оператор o1 в стек.

* Если символ является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется.

Извлечение из стека оставшихся операций.

1. **Проверка введённой строки на корректность**

Проверка выражения на корректность происходит в несколько этапов:

* Строка проверяется на наличие недопустимых символов
* Проверка на недопустимые сочетания символов
* Сравнение количества открывающих и закрывающих скобок
* Проверка на соответствие количества операций и операндов

1. **Подсчёт значения выражения**
2. Ввод значения переменных:

Для каждого операнда, не являющегося числом, предлагается ввести его значение.

1. Поэлементное считывание из постфиксной формы записи.

Если элемент является числом, помещение его в стек

Иначе выполнение операции, используя необходимое количество операндов, взятых с вершины стека. Результат выполнения операции помещается на вершину стека

1. Единственный оставшийся в стеке элемент является значением выражения при указанных значениях переменных.

# Заключение

В результате выполнения лабораторной работы были разработаны классы TStack и TPostfix, соответствующие поставленной задачи. Были разработаны и реализованы тесты для классов TStack и TPostfix на базе Google Test Framework. Для ручного тестирования классов реализован пример.

# Литература

1. [http://www.cplusplus.com/reference/string/basic\_string/](https://msdn.microsoft.com/ru-ru/library/ydd54a6t.aspx%20wikipedia.org) - справочный материал по базовому классу string.
2. [http://www.cplusplus.com/reference/vector/vector/?kw=vector](http://www.cplusplus.com/reference/vector/vector/?kw=vector%20) - справочный материал по базовому классу vector.
3. <https://ru.wikipedia.org/wiki/Обратная_польская_запись/> – обратная польская запись.

\

# Приложения

## Код программы

***Класс TStack***

template <class T>

class TStack

{

T \*pMem;

int Size;

int top;

void AddSize(); //увеличение размера стека

public:

TStack();

TStack(int s);

~TStack();

bool IsEmpty(); //проверка на пустоту

bool IsFull(); //проверка на заполненность

void PutIn(T val); //добавление элемента в стек

T PutOut(); //извлечение элемента из стека

T GetValue(); //получить значение последнего в стеке элемента без извлечения

};

template <class T>

TStack<T>::TStack()

{

pMem = new T[START\_SIZE];

if (pMem == nullptr)

throw("Memory allocation error");

Size = START\_SIZE;

top = -1;

}

template <class T>

TStack<T>::TStack(int s)

{

if (s < 1 || s > MAX\_SIZE)

throw("Invalid size of the stack");

pMem = new T[s];

if (pMem == nullptr)

throw("Memory allocation error");

Size = s;

top = -1;

}

template <class T>

TStack<T>::~TStack()

{

delete[] pMem;

}

template <class T>

void TStack<T>::AddSize()

{

int tmpSize = 0;

if (Size \* 2 > MAX\_SIZE){

if (Size < MAX\_SIZE)

tmpSize = MAX\_SIZE;

else

throw("Exceeding maximum size of the stack");

}

else

tmpSize = Size \* 2;

T\* tmp = new T[tmpSize];

if (tmp == nullptr)

throw("Memory allocation error");

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

tmp[i] = pMem[i];

delete[] pMem;

pMem = tmp;

Size = tmpSize;

}

template <class T> //пустота

bool TStack<T>::IsEmpty()

{

return top == -1;

}

template <class T> //заполненность

bool TStack<T>::IsFull()

{

return top == Size - 1;

}

template <class T> //добавление

void TStack<T>::PutIn(T val)

{

if (IsFull() )

AddSize();

pMem[++top] = val;

}

template <class T> //извлечение

T TStack<T>::PutOut()

{

if (IsEmpty() )

throw("Extraction error. The stack is empty");

return pMem[top--];

}

template <class T> //получение значения

T TStack<T>::GetValue()

{

if (IsEmpty())

throw("Extraction error. The stack is empty");

return pMem[top];

}

***Класс TPostfix***

class TPostfix

{

string infix;

string postfix;

vector<string> OpTable; //вектор переменных

//служебные методы

bool IsNumber(string &str); //является ли операнд числом

bool IsCorrect(string &str); //проверка на корректность введённой строки

void ToOpTable(string &str);

void ToStack(TStack<string> &stack, string &str); //работа со стеком операций

void ToPostfix();

int TheTable(const string &str, int IsNeed); //таблица операций, их приоритетов, количество операндов

public:

TPostfix() { infix = "String is empty"; };

TPostfix& operator=(string &str);

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

double Calculate(); // Ввод переменных, вычисление по постфиксной форме

};

const int OPER\_IN\_ALL = 11; //всего операций

const int PROH\_ALL = 25; //всего проверок

bool TPostfix::IsCorrect(string &str)

{

if (!(str[0] == 40 || str[0] == 41 || 65 <= str[0] <= 90 || 97 <= str[0] <= 122)) // '(' ')' 'A'-'Z' 'a'-'z'

return 0;

int a = (str.length() - 1);

if (!(str[a] == 40 || str[a] == 41 || 65 <= str[a] <= 90 || 97 <= str[a] <= 122))

return 0;

int left = 0;

int right = 0;

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

{

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

left++;

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

right++;

}

if (left != right)

return 0;

string proh[PROH\_ALL];

proh[0] = "(+";

proh[1] = "(-";

proh[2] = "(\*";

proh[3] = "(/";

proh[4] = "()";

proh[5] = "++";

proh[6] = "+-";

proh[7] = "+\*";

proh[8] = "+/";

proh[9] = "+)";

proh[10] = "-+";

proh[11] = "--";

proh[12] = "-\*";

proh[13] = "-/";

proh[14] = "-)";

proh[15] = "\*+";

proh[16] = "\*-";

proh[17] = "\*\*";

proh[18] = "\*/";

proh[19] = "\*)";

proh[20] = "/+";

proh[21] = "/-";

proh[22] = "/\*";

proh[23] = "//";

proh[24] = "/)";

for (int i = 0; i < PROH\_ALL; i++)

if (str.find(proh[i]) != -1)

return 0;

return 1;

}

bool TPostfix::IsNumber(string &str)

{

int dots = 0;

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

{

if (str[i] == '.')

dots++;

if (!(str[i] >= '.' && str[i] <= '9'))

return 0;

}

if (dots > 1)

return 0;

return 1;

}

int TPostfix::TheTable(const string &str, int IsNeed) {

/\* Таблица операций

IsNeed = 0 - является ли переименной, 1 - приоритет \*/

if (str == "")

return 0;

string TOperations[OPER\_IN\_ALL]; //операции

TOperations[0] = '+';

TOperations[1] = '-';

TOperations[2] = '\*';

TOperations[3] = '/';

TOperations[4] = '(';

TOperations[5] = ')';

TOperations[6] = "sin";

TOperations[7] = "cos";

TOperations[8] = "tg";

TOperations[9] = "ctg";

TOperations[10] = "sqrt";

int TPriority[OPER\_IN\_ALL]; //приоритет

TPriority[0] = 1;

TPriority[1] = 1;

TPriority[2] = 2;

TPriority[3] = 2;

TPriority[4] = 0;

TPriority[5] = -1;

TPriority[6] = 3;

TPriority[7] = 3;

TPriority[8] = 3;

TPriority[9] = 3;

TPriority[10] = 3;

switch ( IsNeed )

{

case 0:

for (int i = 0; i < OPER\_IN\_ALL; i++)

if (str.compare(TOperations[i]) == 0)

return 1;

return 0;

break;

case 1:

for (int i = 0; i < OPER\_IN\_ALL; i++)

if (str.compare(TOperations[i]) == 0)

return TPriority[i];

return -1; //не является операцией

break;

default:

throw("Invalid second variable in the function TheTable"); //assert???

break;

}

}

TPostfix& TPostfix::operator=(string &str) {

if (!IsCorrect(str))

throw("String is not correct");

infix = str;

ToPostfix();

}

void TPostfix::ToOpTable(string &str)

{

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

if (OpTable[i].compare(str) == 0)

return;

OpTable.push\_back(str);

}

void TPostfix::ToStack(TStack<string> &stack, string &str) //стек операций

{

if (str[0] == '(')

{

stack.PutIn(str);

return;

}

if (stack.IsEmpty() == 1)

{

stack.PutIn(str);

return;

}

string LastOp = stack.GetValue();

if (TheTable(str, 1) <= TheTable(LastOp, 1))

{

while (TheTable(str, 1) <= TheTable(LastOp, 1))

{

if (LastOp[0] == '(')

return;

postfix = postfix + LastOp + " ";

stack.PutOut();

if (!stack.IsEmpty())

LastOp = stack.GetValue();

else

{

break;

}

}

stack.PutIn(str);

}

else

{

stack.PutIn(str);

return;

}

}

void TPostfix::ToPostfix()

{

TStack<string> stack;

string str, tmp;

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

{

tmp = infix[i];

if (TheTable(str, 0) == 1)

{

ToStack(stack, str);

str.clear();

}

if (TheTable(tmp, 0) == 1)

{

if (str.length() != 0)

{

ToOpTable(str);

postfix = postfix + str + " ";

str.clear();

}

ToStack(stack, tmp);

}

else

str = str + tmp;

tmp.clear();

}

if (str.length() != 0)

{

if (TheTable(str, 0) == 1)

throw("Invalid string");

else

{

ToOpTable(str);

postfix = postfix + str + " ";

}

}

while (!stack.IsEmpty())

{

if (stack.GetValue()[0] == '(')

{

stack.PutOut();

continue;

}

postfix = postfix + stack.PutOut() + " ";

}

}

double TPostfix::Calculate()

{

double\* OpValue = new double[OpTable.size()];

cout << "Значения переменных" << endl;

setlocale(LC\_ALL, "C");

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

{

if (IsNumber(OpTable[i]))

{

cout << OpTable[i] << " = ";

cout << stod(OpTable[i]) << endl;

OpValue[i] = stod(OpTable[i]);

}

else

{

cout << OpTable[i] << " = ";

cin >> OpValue[i];

}

}

setlocale(LC\_ALL, "Russian");

string str;

TStack<double> calc;

for (int i = 0; i < postfix.length() - 1; i++)

{

str.clear();

while (postfix[i] != ' ')

str = str + postfix[i++];

if (!TheTable(str, 0))

{

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

if (OpTable[j] == str)

{

calc.PutIn(OpValue[j]);

break;

}

}

else

{

if (calc.IsEmpty())

throw("String is not correct");

if (str == "+")

{

calc.PutIn(calc.PutOut() + calc.PutOut());

continue;

}

if (str == "-")

{

double right = calc.PutOut();

double left = calc.PutOut();

calc.PutIn(left - right);

continue;

}

if (str == "\*")

{

calc.PutIn(calc.PutOut() \* calc.PutOut());

continue;

}

if (str == "/")

{

double right = calc.PutOut();

double left = calc.PutOut();

calc.PutIn(left / right);

continue;

}

if (str == "sin")

{

calc.PutIn(sin(calc.PutOut()));

continue;

}

if (str == "cos")

{

calc.PutIn(cos(calc.PutOut()));

continue;

}

if (str == "tg")

{

calc.PutIn(sin(calc.GetValue()) / cos(calc.PutOut()));

continue;

}

if (str == "ctg")

{

calc.PutIn(cos(calc.GetValue()) / sin(calc.PutOut()));

continue;

}

if (str == "sqrt")

{

calc.PutIn(sqrt(calc.PutOut()));

continue;

}

}

}

delete[] OpValue;

double res = calc.PutOut();

if (!calc.IsEmpty())

throw("String is not correct");

return res;

}