Министерство образования и науки РФ
Нижегородский государственный университет имени Н.И. Лобачевского (Национальный исследовательский университет)
Институт Информационных Технологий Математики и Механики

Отчет по лабораторной работе № 3

# Вычисление арифметических выражений

Выполнил:
студент группы 0823-3
Краснов Александр Александрович
Проверил: Козинов Е.А.

г. Нижний Новгород

2016г.

# Содержание

# Введение

Темой моей лабораторной работы является вычисление арифметических выражений с помощью обратной польской записи. За основу взят алгоритм, который был предложен польским математиком Яном Лукашевичем для преобразования выражения в ОПН. Однако первое теоретическое описание и практическое применение этого алгоритма предложил и обосновал нидерландский учёный Эдсгер Дейкстра. Алгоритм получил название «сортировочная станция» за сходство его операций с происходящим на железнодорожных сортировочных станциях.

Исходное выражение вводится в «обычной» для нас инфиксной форме. Инфиксная нотация — это форма математических записей, которую использует большинство людей (например, `3 + 4` или, `3 + 4 * (2 - 1)`). Постфиксная нотация - это форма математических записей, характерной чертой которой является расположение оператора справа от операндов.

В процессе разработки использовалась такая динамическая структура данных, как стек, для хранения операций при переводе выражения в обратную польскую запись и для хранения значений операндов. Напомню, что стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Таким образом, данная программа принимает выражение в инфиксной записи, переводит его в постфиксную запись, подсчитывает результат и выводит его на экран.

При реализации данного проекта применяется язык C++ (с применением классов).

# Постановка задачи

Дано арифметическое выражение, состоящее из вещественных чисел и содержащее операции: «+» (сумма), «-» (разность), «*» (произведение), «/» (деление), «^» (возведение в степень).

Данная программа должна обеспечить следующий ряд действий:

1) Принять вводимое пользователем арифметическое выражение
2) Разбиение арифметического выражения на лексемы (т.е. выделить операнды и операции)
3) Проверка корректности выражения (правильность расстановки скобок, пропущены ли операнды или знаки операций, недопустимые символы)
4) В случае неуспешного выполнения пункта 3 выдать пользователю сообщение о конкретной ошибке
5) В случае успешного выполнения пункта 3 перевести выражение в ОПЗ (обратную польскую запись)
6) Вычисление арифметического выражения в ОПЗ (обратной польской записи)
7) Вывод на экран выражения в инфиксной нотации
8) Вывод на экран выражения в постфиксной нотации
9) Вывод на экран результата вычислений.
10) Проверить работоспособность программы с помощью Google Test-ов.

# Руководство пользователя

Данная программа предлагает пользователю следующий набор возможностей:

1) Ввод арифметического выражения из любых вещественных чисел;
2) Реализация операций: «+» (сумма), «-» (разность), «*» (произведение), «/» (деление), «^» (возведение в степень).
3) Проверка на корректность выражения: отсутствие посторонних символов и повторений операций, правильность расстановки скобок и т.д.
4) Вывод выражения в ОПН (обратной польской нотации).
5) Вывод результата вычислений.

# Руководство программиста

Исключительной особенностью обратной польской записи является то, что все аргументы (или операнды) расположены перед знаком операции. В общем виде запись выглядит следующим образом:

1) Запись набора операций состоит из последовательности операндов и знаков операций. При письменной записи операнды в выражении разделяются пробелами.
2) Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
3) Результатом вычисления выражения становится результат последней вычисленной операции.

Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул. Во-первых, любая формула может быть выражена без скобок. Во-вторых, она удобна для вычисления формул в машинах со стеками. В-третьих, инфиксные операторы имеют приоритеты, которые произвольны и нежелательны. Например, мы знаем, что `ab+c` значит `(ab)+c` , а не `a(b+c)` поскольку произвольно было определено, что умножение имеет приоритет над сложением. Но имеет ли приоритет сдвиг влево перед операцией объединения? Кто знает? Обратная польская запись позволяет устранить такие недоразумения.

Порядок следования переменных в инфиксной и постфиксной записи одинаков. Однако порядок следования операторов не всегда один и тот же. В обратной польской записи операторы появляются в том порядке, в котором они будут выполняться.

## Описание алгоритмов:

*Конвертирование*

1) Для операций вводится приоритет:
2) ‘^’ (4),’*’, ’/‘ (3), ‘+’,’-‘ (2), ‘)‘ (1), ‘(’ (0)
3) Для хранения данных используется стек для операций и строка для результата
4) Исходное выражение просматривается слева направо
5) Операнды по мере их появления помещаются в строку
6) Символы операций и левые скобки помещаются в стек
7) При появлении правой скобки последовательно изымаются элементы из стека и переносятся в строку. Данные действия продолжаются либо до опустошения стека, либо до попадания в стеке на левую скобку
8) Если текущая операция, выделенная при обходе выражения, имеет меньший или равный приоритет, чем операция на вершине стека, то такие операции из стека переписываются в строку
9) После завершения обхода выражения, если стек ещё не пуст, выталкиваем оставшиеся операторы в строку.

* Данный алгоритм описан без учета обработки отрицательных чисел. В подобной ситуации ‘-‘ заносится в строку как часть числа, а не в стек как оператор.

*Вычисление результата выражения по постфиксной записи*

1) Для вычисления выражения используется стек и полученное постфиксное выражение
2) При появлении цифры считываем всё число и помещаем в стек
3) При появлении символа операции достаем два последних значения из стека, производим над ними данную операцию, затем результат закидываем в стек
4) При завершении обода постфиксного выражения выводим число с вершины стека.

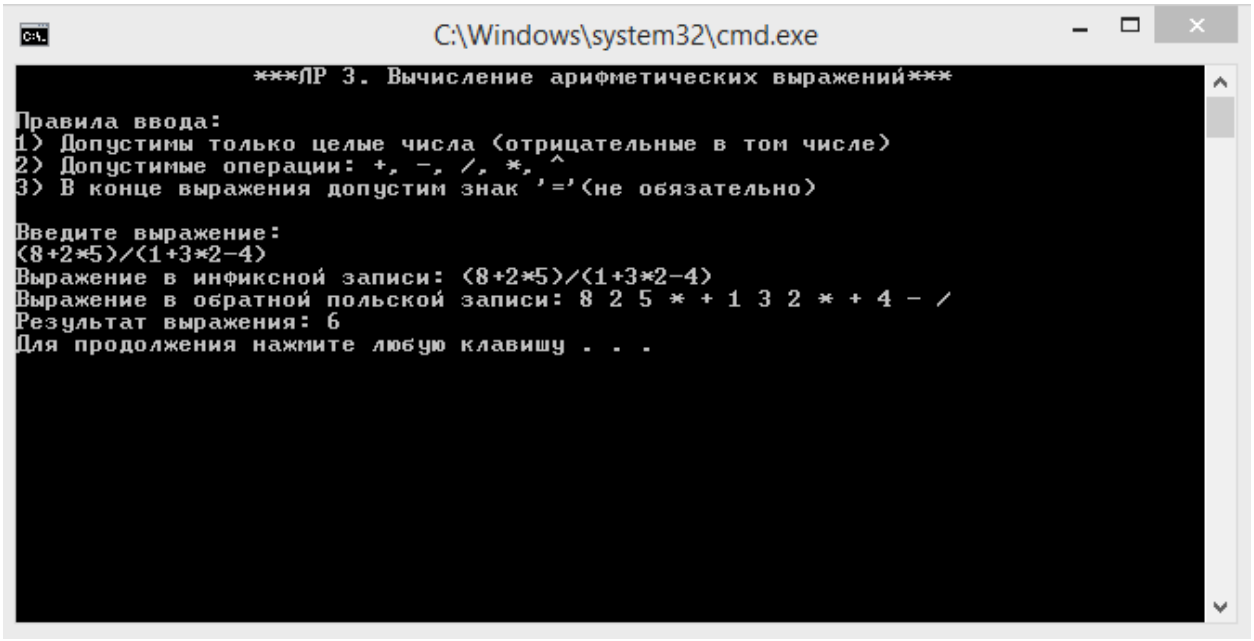
* Данный алгоритм описан без учёта обработки отрицательных чисел. В подобной ситуации ‘-‘ считывается из строки как часть числа, а не как оператор.

## Пример вычисления выражения в обратной польской записи

В качестве примера рассмотрим вычисление следующего выражения: `(8+2*5)/(1+3*2-4)` . Соответствующая формула в обратной польской записи выглядит так: `825*+132*+4-/` .

| Шаг | Оставшаяся цепочка | Стек |
|---|---|---|
| 1 | 8, 2,5, *+1,3,2*+4-/ | 8 |
| 2 | 2,5, *+1,3,2*+4-/ | 8,2 |
| 3 | 5*+1,3,2*+4-/ | 8,2,5 |
| 4 | *+1,3,2*+4-/ | 8,10 |
| 5 | +1,3,2*+4-/ | 18 |
| 6 | 1,3,2*+4-/ | 18,1 |
| 7 | 3,2*+4-/ | 18,1,3 |
| 8 | 2*+4-/ | 18,1,3,2 |
| 9 | *+4-/ | 18,1,6 |
| 10 | +4-/ | 18,7 |
| 11 | 4-/ | 18,7,4 |
| 12 | -/ | 18,3 |
| 13 | / | 6 |

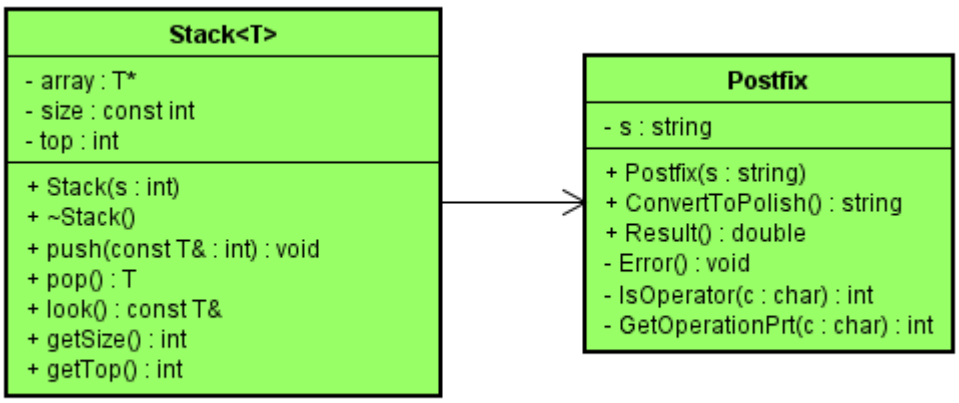
Число на вершине стека – это правый операнд (а не левый). Это очень важно для операций деления, вычитания и возведения в степень, поскольку порядок следования операндов в данном случае имеет значение (в отличие от операций сложения и умножения). Другими словами, операция деления действует следующим образом: сначала в стек помещается числитель, потом знаменатель, и тогда операция даёт правильный результат.




## Структура программы

- stack.h - реализация класса Stack (стек реализован на шаблоне)
- postfix.h – заголовочный файл класса Postfix
- postfix.cpp – файл с реализацией класса Postfix
- sample.cpp – главная функция программы
- test_main.cpp, test_stack_and_postfix.cpp – реализация тестов

## Схема наследования



# Заключение

В результате выполнения данной лабораторной работы было создано программное приложение, позволяющее эффективно вычислять арифметические выражения над вещественными числами.

Разработан алгоритм и составлен и протестирован программный комплекс для решения соответствующей задачи.

Продемонстрированы плюсы подобного подхода.

Итогом работы стали два полноценных класса: класс стека на шаблоне и класс конвертирования инфиксного выражения в постфиксное.

Данную программу можно смело совершенствовать. Например, реализовать выполнение различных функций, скажем, тригонометрических.

Была написана тестовая оболочка (тестовое приложение), в которой можно легко проверить работу данных классов.

Таким образом, задачи, поставленные в начале, были успешно выполнены, а результаты проверены.

# Литература


1) Гергель В.П. Рабочие материалы к учебному курсу «Методы программирования», ННГУ, 2002. – 100 с.
2) Б. Страуструп Язык программирования С++. Специальное издание.
    a. Пер. с англ. – М.: ООО «Бином-Пресс», 2008 г. – 1104 с.
3) Р.Лафоре. Объектно-ориентированное программирование в языке СИ++.-М: «ПИТЕР», 2004 г.-922 с.
4) https://habrahabr.ru/post/100869/
5) https://ru.wikipedia.org/wiki/Обратная_польская_запись

# Приложения

stack.h

```
////////////////////////////////////////////////////////////////////
// stack.h                                                        //
// Вычисление арифметических выражений                            //
// Автор - Краснов А.А., Нижний Новгород, 2016                    //
////////////////////////////////////////////////////////////////////

#pragma once

#define MemSize 50

template <typename T>
class Stack
{
private:
    T *array;
    const int size;
    int top;
public:
    Stack(int s=15);
    ~Stack();
    void push(const T & );
    T pop();
    const T &look() const;
       int getSize() const;
    int getTop() const;
};

template <typename T>
Stack<T>::Stack(int s):size(s), top(-1)
{
       if (size>MemSize) throw "very big size";
       if (size<0) throw "negative size";
    array = new T[size];
}

template <typename T>
Stack<T>::~Stack()
{
    delete [] array;
}
template <typename T>
void Stack<T>::push(const T &val)
{
       if (top+1>=size) throw "Stack is full";
       array[++top] = val;
}
template <typename T>
T Stack<T>::pop()
{
       if (top<0) throw "Stack is empty";
       return array[top--];
}
template <class T>
const T &Stack<T>::look() const
{
       return array[top];
}
```

```
template <typename T>
int Stack<T>::getSize() const
{
    return size;
}

template <typename T>
int Stack<T>::getTop() const
{
    return top;
}
```

## postfix.h

```
////////////////////////////////////////////////////////////////////////
// postfix.h                                                          //
// Вычисление арифметических выражений                                //
// Автор - Краснов А.А., Нижний Новгород, 2016                        //
////////////////////////////////////////////////////////////////////////

#include "stack.h"
#include <cstring>
#include <sstream>
#include <locale>
#include <iostream>

using namespace std;

class Postfix
{
       string s;
public:
       Postfix(string = "");
       string ConvertToPolish();      // перевод инфиксной формы в постфиксную
    double Result();               // подсчёт результата
private:
       void Error();                  // проверка на правильность выражения
       int IsOperator(char c);        // проверка на операцию
       int GetOperationPrt(char c);   // определение приоритета операции
   };
```

## postfix.cpp

```
////////////////////////////////////////////////////////////////////////
// postfix.cpp                                                        //
// Вычисление арифметических выражений                                //
// Автор - Краснов А.А., Нижний Новгород, 2016                        //
////////////////////////////////////////////////////////////////////////

#include "postfix.h"

Postfix::Postfix(string str)
{
       s=str;
       Error();
}

int Postfix::IsOperator(char c)      // проверка на операцию
{
       if (c=='+' || c=='-' || c=='*' || c=='/' || c=='^' || c=='('|| c==')') return 1;
```

```
	else if (c==' ' || c=='=') return 2;
	else return 0;
}

int Postfix::GetOperationPrt(char c)  // определение приоритета операции
{
	switch (c)
	{
	case '(': return 0;
	case ')': return 1;
	case '+': case '-': return 2;
	case '*': case '/': return 3;
	case '^': return 4;
	default: return 5;
	}
}

void Postfix::Error()
{
	string infix = this->s;
	int len = infix.length(), count1(0),count2(0);
	bool flag = true;
	for (int i(0);i<len;i++)
	{
		// Проверяем есть ли посторонние символы
		if ((!isdigit(infix[i])) && (!IsOperator(infix[i])) && (!(infix[i]=='.')))
			throw "Ошибка! Недопустимый символ" ;
		// Проверяем: выражение не может быть без цифр
		if (isdigit(infix[i])) flag = false;
		// Проверяем: кол-во '(' и ')' должно быть равно
		if ((infix[i])=='(') count1++;
		if ((infix[i])==')') count2++;
		// Проверяем: орерации не могут быть друг за другом
		if (strchr("+-/*^", infix[i])!=NULL && strchr("+-/*^", infix[i+1])!=NULL &&
i<len-1)
			throw "Ошибка! Операции не согласованы" ;
	}
	if (flag == true) throw "Ошибка! Выражение не содержит цифр" ;
	if (count1!=count2) throw "Ошибка! Не согласовано кол-во скобок" ;
}

string Postfix::ConvertToPolish()
{
	string infix = this->s;
	int len = infix.length();
	string polish;                    // Строка для обратной польской записи
	Stack<char> OperationStack(len); // Стек для операторов

	for (int i(0);i<len;i++)
	{
		if ((i==1) && (infix[0]=='-')) polish="-"; // Если в начале минус
		if ((IsOperator(infix[i]))==2) continue;   // Если пробел или равно,
пропускаем
		if (isdigit(infix[i])|| infix[i]=='.')                    // Если цифра,
		{
			while (!IsOperator(infix[i]))          // считываем всё число
			{
				polish+=infix[i++];
				if (i==len) break;
			}
			polish.push_back(' ');                 // добавляем пробел после
числа
			i--;
		}
		if ((IsOperator(infix[i]))==1)             // если операция или скобки
```

```
                        if (infix[i] == '(' && infix[i+1] == '-' && i<len-2)  // Если '-'
после '('
                        {
                                i=i+2;
                                if (isdigit(infix[i]) || infix[i]=='.')
                                {
                                        polish.push_back('-');
                                        while (!IsOperator(infix[i]))
                                        {
                                                polish+=infix[i++];
                                                if (i==len) break;
                                        }
                                        polish.push_back(' ');
                                }
                        }
                        else if (infix[i]=='(') OperationStack.push(infix[i]);
                        else if (infix[i]==')')
                        {
                                char s = OperationStack.pop();
                                while (s != '(')
                                {
                                        polish.push_back(s);
                                        polish.push_back(' ');
                                        s=OperationStack.pop();
                                }
                        }
                        else if (i!=0) // если не минус в начале
                        {
                                while ((OperationStack.getTop()>-1) &&
(GetOperationPrt(infix[i])<=GetOperationPrt(OperationStack.look())))
                                {
                                        polish.push_back(OperationStack.pop());
                                        polish.push_back(' ');
                                }
                                OperationStack.push(infix[i]);
                        }
        }
        //Когда прошли по всем символам, выкидываем из стека все оставшиеся там операторы
в строку
        while (OperationStack.getTop()>-1)
        {
                polish.push_back(OperationStack.pop());
                polish.push_back(' ');
        }
        return polish;
}

double Postfix::Result()
{
        double res(0);
        string polish = this->ConvertToPolish();
        int len = polish.length();
        Stack<double> ResultStack(len);
        for (int i(0);i<len;i++)
        {
                //Если цифра, то читаем все число и толкаем на вершину стека
                if (isdigit(polish[i])|| polish[i]=='.')
                {
                        string str;
                        double op;
                        while (!IsOperator(polish[i]))
                        {
                                str+=polish[i++];
                                if (i==len) break;
                        }
```

```
                istringstream(str)>>op;
                ResultStack.push(op);
                i--;
        }
        // Если встречен минус
        if (polish[i]=='-' && isdigit(polish[i+1]) && i<len-1)
        {
                i=i+1;
                string str;
                double op;
                while (!IsOperator(polish[i]))
                {
                        str+=polish[i++];
                        if (i==len) break;
                }
                istringstream(str)>>op;
                ResultStack.push(-op);
                i--;
        }
        else if (((IsOperator(polish[i]))==1) && (i!=0))
        {
                double op1=ResultStack.pop();
                double op2=ResultStack.pop();
                switch (polish[i])
                {
                case '+': res = op2 + op1; break;
                case '-': res = op2 - op1; break;
                case '*': res = op2 * op1; break;
                case '/':
                        if (op1!=0)
                        {
                                res = op2 / op1;
                                break;
                        }
                        else throw "Деление на ноль!";
                case '^': res = pow(op2,op1); break;
                }
                ResultStack.push(res);
        }
    }
    return ResultStack.look();
}
```

## sample.cpp

```
//////////////////////////////////////////////////////////////////////
// sample.cpp                                                       //
// Вычисление арифметических выражений                              //
// Автор - Краснов А.А., Нижний Новгород, 2016                      //
//////////////////////////////////////////////////////////////////////

#include "postfix.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    try
    {
        cout<<"\t\t***ЛР 3. Вычисление арифметических выражений***\n\n";
        cout<<"Правила ввода:"<< endl;
        cout<<"1) Допустимы любые вещественные числа"<<endl
            <<"2) Допустимые операции: +, -, /, *, ^" << endl
```

```
                    <<"3) В конце выражения допустим знак '='(не
обязательно)"<<endl<<endl;
                cout<< "Введите выражение:"<< endl;
                char s[256];
                cin.getline(s,256);
                string Expression(s);
                Postfix Convertor(Expression);
                string PolishExpression = Convertor.ConvertToPolish();
                double Rez = Convertor.Result();
                cout<<"Выражение в инфиксной записи: " << Expression<<endl;
                cout<<"Выражение в обратной польской записи: "<<PolishExpression<<endl;
                cout<<"Результат выражения: "<<Rez<<endl;
                return 0;
        }
        catch (const char* error)
        {
                cout<<error<<endl;
        }
}
```