Министерство образования и науки Российской федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт ИТММ

Отчет

по учебной практике

**Вычисление арифметических выражений**

Выполнил:

 студент гр. 381703-1

Громов Н.В.

Проверил:

ассистент кафедры МОСТ ИИТММ

 Волокитин В.Д.

г. Нижний Новгород

2018г

Оглавление

[Введение 3](#_Toc530768225)

[Постановка учебно-практической задачи 4](#_Toc530768226)

[Руководство пользователя 5](#_Toc530768227)

[Руководство программиста 6](#_Toc530768228)

[Описание структур данных 9](#_Toc530768229)

[Заключение 12](#_Toc530768230)

[Список литературы 13](#_Toc530768231)

[Приложения 14](#_Toc530768232)

# Введение

Чаще всего для выполнения арифметических операций в программировании нужна реализация обратной польской записи и распределения входных данных на лексемы.

Обратная польская запись — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Также именуется как обратная польская запись, обратная бесскобочная запись, постфиксная нотация, бесскобочная символика Лукасевича, польская инверсная запись, ПОЛИЗ.

Обратная польская нотация была разработана австралийским философом и специалистом в области теории вычислительных машин Чарльзом Хэмблином в середине 1950-х на основе польской нотации, которая была предложена в 1920 году польским математиком Яном Лукасевичем.

# Постановка учебно-практической задачи

**Цель работы**

Разработать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение в качестве операндов может содержать вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки номера символов строки, в которых были найдены ошибки.

**Последовательность выполнения работы**

1. Разработка шаблонного класса Stack
2. Проверка корректности выражения:

* правильность расстановки скобок.
* правильность написания операндов или операций.
* недопустимые символы.

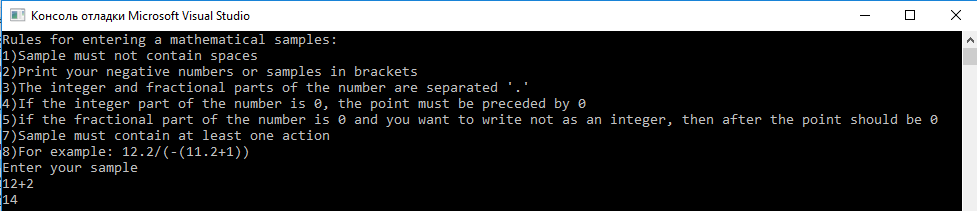
1. Разбиение исходного арифметического выражения на лексемы (т.е. выделить операнды, операции) и перевод выражения в обратную польскую запись
2. Вычисление выражения
3. Создание консольного интерфейса пользователя

**Исходные данные:**

C - Строка, содержащая арифметическое выражение

**Требуемый результат:**

Answer – ответ

**Пример:** 

# Руководство пользователя

1)Для начала работы с программой необходимо открыть файл arithmetic.exe.После запуска приложения на экране появится окно. Появятся инструкции по написанию выражения и просьба ввода. Пользователю необходимо ввести с клавиатуры арифметическое выражение, руководствуясь инструкциями ввода, и нажать Enter (см. рис. 1).

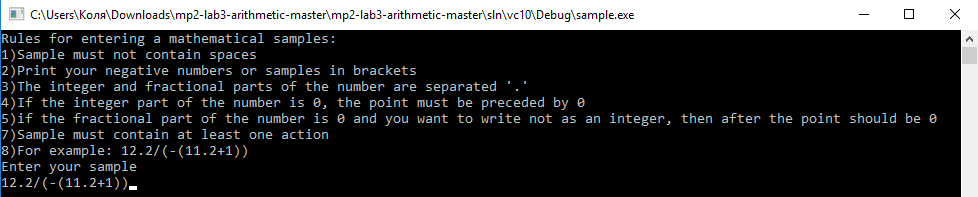
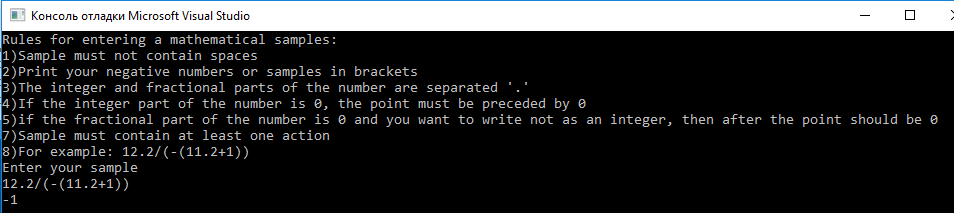


рис.1.Ввод строки

2)При правильном вводе программа выведет ответ (см. рис. 2).

рис. 2. Вывод ответа



# Руководство программиста

Решение поставленной задачи осуществляется с помощью следующих функций и классов.

**Шаблонный класс стек(template <typename T> class Stack):**

**Конструктор стека(Stack(int nk)).**

Создает стек длинной nk.

**Деструктор стека(~Stack()).**

Освобождает память, выделявшуюся под массив значений стека.

**Функция, выдающая количество элементов в стеке(int Get\_pos()).**

**Функция увеличения стека при его полноте(void Resize\_IF()).**

Увеличивает длину массива в стеке в 1,5 раза.

**Функция проверки стека на пустоту(bool IsEmpty()).**

**Функция добавления элемента в стек(void Set(T k)).**

Добавляет в стек элемент, если стек заполнен, то вызывает функцию увеличения стека.

**Функция взятия элемента из стека(T Get()).**

Если стек не пустой, забирает верхний элемент стека.

**Функция просмотра верхнего элемента стека(T Top()).**

Если стек не пустой, возвращает верхний элемент стека без удаления.

**Класс лексем(Oper\_or\_numb):**

**Конструктор по умолчанию(Oper\_or\_numb()).**

Ставит в поле определения типа лексемы в классе «операция».

**Конструктор от операции(Oper\_or\_numb(char c).**

Ставит в поле определения типа лексемы в классе «операция».

Определяет ее преимущество перед другими операциями при вычислениях.

Записывает в поле операции саму операцию.

**Конструктор от числа(Oper\_or\_numb(double a)).**

Ставит в поле определения типа лексемы в классе «число».

Записывает значения числа в соответствующее поле.

**Деструктор(~Oper\_or\_numb()).**

Ставит в поле преимущества операции и в значение ‘0’ .

**Функция занесения операции в класс(void Set(char c)).**

Аналогична конструктору от операции.

**Функция занесения числа в класс(void Set(double a)).**

Аналогична конструктору от числа.

**Функция возврата типа лексемы(type Get\_type()).**

**Функция возврата значения лексемы, если оно число(double Get\_value()).**

**Функция возврата приоритета операции(int Get\_lvl()).**

**Функция возврата операции(char Get\_op()).**

**Перегрузка оператора ‘=’(Oper\_or\_numb& operator=(Oper\_or\_numb& a)).**

Переписывает значения полей.

**Класс вычислений(Lexeme):**

**Конструктор(Lexeme(char\* c1)).**

Определяет тип каждого символа и осуществляет все проверки на правильность написания.

**Функция возврата типа символа для заданного номера(type Get\_kind(int i)).**

**Функция возврата символа для заданного номера(char Get\_c(int i)).**

**Функция подсчета ответа(double Answer(Oper\_or\_numb)).**

В цикле записывает значения массива лексем в стек в правильном порядке.

Числа записываются сразу в стек. Операция достает из стека два числа, проделывает соответствующую операцию, а результат снова кладет в стек. Результат функции – оставшийся элемент в стеке.

**Функция выполнения алгоритма обратной польской записи(Oper\_or\_numb\* RPN())**.

В цикле смотрит тип каждого символа.

Цифры (точки и скобки, относящиеся к унарному минусу и сам унарный минус тоже относятся к цифрам) просматриваются в дополнительном цикле. Каждая цифра увеличивает счетчик главного цикла на один.

При окончании цифр записывает в массив лексем соответствующее число из стека (поля класса) (при унарном минусе в массив лексем записывается “-1”) и выходит из цикла.

Операции записываются при помощи дополнительного стека. Первая операция кладется в доп. стек. При последующих сравниваются приоритеты поступившей и верхней в доп. стеке. Если больше или равно, то записывает поступившую в массив лексем, если меньше, то записывает ее в доп. стек.

Открывающая скобка просто записывается в доп. стек.

Закрывающая скобка достает все операции из доп. стека и записывает их в массив лексем.

Все оставшиеся в доп. стеке операции записываются в массив лексем.

# Описание структур данных

**Новые типы:**

**Enum type**(содержит разновидность типов символа).

**template <typename T>** **class** **Stack** (стек).

**Class Oper\_or\_numb** (класс лексем).

**Class Lexeme**(класс вычислений).

**Переменные:**

string c; - переменная, в которую пользователь записывает свое арифметической выражение.

char\* c1; - переменная, преобразующая выражение пользователя в необходимый тип данных.

Общее описание структуры програмного комплекса.

Решение поставленной задачи осуществляется с помощью многофайловой программы.

**Заголовочный файл stack.h**

В данном файле реализован следующий класс и методы:

template <typename T> class Stack

**Поля:**

int n; - размер массива q

T\*q; - массив, который хранит значения стека

int pos; - количество значений в стеке

**Методы:**

**Конструктор стека(Stack(int nk)).**

int nk – длина создаваемого стека

**Функция увеличения стека при его полноте(void Resize\_IF()).**

T\* tmp – массив, копирующий значения стека

**Функция добавления элемента в стек(void Set(T k)).**

T k – элемент, который добавляется в стек

**Заголовочный файл arithmetic.h**

В данном файле реализованы следующие классы и методы:

**Enum type**

**Поля:**

oper -операция

num-число

o\_b-открывающаяся скобка

c\_b-закрывающаяся скобка

**Class Oper\_or\_numb**

**Поля:**

int lvl - приоритет операций

char op – сама операция

type on – переменная, определяющая разновидность символа.

double val – значение числа

**Методы:**

**Конструктор от операции(Oper\_or\_numb(char c).**

char c – символ операции

**Конструктор от числа(Oper\_or\_numb(double a)).**

double a – само число

**Функция занесения операции в класс(void Set(char c)).**

char c – символ операции

**Функция занесения числа в класс(void Set(double a)).**

double a – само число

**Перегрузка оператора ‘=’(Oper\_or\_numb& operator=(Oper\_or\_numb& a)).**

Oper\_or\_num a – переменная, к которой приравнивается переменная слева ‘=’

**Class Lexeme**

**Поля:**

Stack <double> TStack – стек, в котором хранятся значения чисел из строки.

char\* c - строка, в которой записано выражение

type\* kind - массив типов символов из строки

int n – количество лексем в обратной польской записи

**Методы:**

**Конструктор(Lexeme(char\* c1)).**

char\* c1 – строка, в которой записано выражение.

int t – флаг для правильности написания скобок

double k – переменная для корректного написания цифр после ‘.’ в числе

double a – переменная, записывающая числа для стека TStack

**Функция выполнения алгоритма обратной польской записи(Oper\_or\_numb\* RPN()).**

Oper\_or\_numb\* o – массив лексем, в котором записана обратная польская запись

Oper\_or\_numb o1 – переменная для хранения лексем из строки

int t, k , a , b – флаги

Stack <double> S – стек, обратный к TStack

Stack <Oper\_or\_numb> S1 - стек, записывающий операции в o

int v – изначальное количество элементов в TStack

**Функция подсчета ответа(double Answer(Oper\_or\_numb)).**

double a,b - два числа, над которыми проводятся операции

Stack<double> S – стек, хранящий промежуточные вычисления и ответ

**Файл arithmetic.cpp**

Реализованы все прототипы из файла arithmetic.h.

**Файл main\_arithmetic.cpp**

Реализован диалог с пользователем и приведение его строки в массив символов.

string c – строка, введенная пользователем.

char\* c1 – массив символов, копия с.

# Заключение

В данной лабораторной работе был изучен ряд известных алгоритмов, и был создан программный комплекс, реализующий такие функциональные алгоритмы, как: алгоритм обработки лексем из строки, алгоритм преобразования лексем в обратную польскую запись, вычисление обратной польской записи, класс стек.

# Список литературы

# Приложения

## Приложение 1. Файл stack.h

#include <stdlib.h>

#ifndef STACK\_H

#define STACK\_H

template <typename T> class Stack

{

private:

int n;

T \*q;

int pos;

public:

Stack(int nk=10)

{

n = nk;

q = new T[n];

pos = 0;

}

~Stack()

{

pos = 0;

delete[] q;

}

int Get\_pos() { return pos; }

void Resize\_IF()

{

if (pos == n)

{

T\* tmp;

tmp = new T[1.5\*n];

for (int i = n - 1;i >= 0;i--)

tmp[i] = Get();

delete[] q;

q = tmp;

pos = n;

n = 1.5\*n;

}

}

bool IsEmpty()

{

if (!(pos))

return(true);

else

return(false);

}

void Set(T k)

{

Resize\_IF();

q[pos++] = k;

}

T Get()

{

if (!(IsEmpty()))

{

pos--;

return(q[pos]);

}

else

throw "Stack is empty";

}

T Top()

{

if (!(IsEmpty()))

{

return(q[pos - 1]);

}

else

throw "Stack is empty";

}

};

#endif STACK\_H

## Приложение 2. Файл arithmetic.h

#include <string>

#include "stack.h"

enum type{oper,num,o\_b,c\_b};

class Oper\_or\_numb

{

private:

int lvl;

char op;

type on;

double val;

public:

Oper\_or\_numb();

Oper\_or\_numb(char c);

Oper\_or\_numb(double a);

~Oper\_or\_numb() { lvl = 0; val = 0; };

void Set(double a)

{

on = num;

val = a;

}

void Set(char c) {

on = oper;

switch (c)

{

case '+':

{

lvl = 1;

op = '+';

break;

}

case '-':

{

lvl = 1;

op = '-';

break;

}

case '\*':

{

lvl = 2;

op = '\*';

break;

}

case '/':

{

lvl = 2;

op = '/';

break;

}

case '(':

{

op = '(';

break;

}

}

};

type Get\_type() { return on; }

double Get\_value() { return val; }

int Get\_lvl() { return lvl; };

char Get\_op() { return op; };

Oper\_or\_numb& operator=(Oper\_or\_numb& a);

};

class Lexeme

{

private:

Stack <double> TStack;

char\* c;

type\* kind;

int n;

public:

Lexeme( char\* c1);

type Get\_kind(int i) { return kind[i]; };

char Get\_c(int i) { return c[i]; };

double Answer(Oper\_or\_numb\* o);

Oper\_or\_numb\* RPN();

};

## Приложение 3. Файл arithmetic.cpp

#include "arithmetic.h"

#include "stack.h"

#include "iostream"

Oper\_or\_numb::Oper\_or\_numb()

{

on = oper;

}

Oper\_or\_numb::Oper\_or\_numb(char c)

{

on = oper;

switch (c)

{

case '+':

{

lvl = 1;

op = '+';

break;

}

case '-':

{

lvl = 1;

op = '-';

break;

}

case '\*':

{

lvl = 2;

op = '\*';

break;

}

case '/':

{

lvl = 2;

op = '/';

break;

}

case '(':

{

op = '(';

break;

}

}

};

Oper\_or\_numb::Oper\_or\_numb(double a)

{

on = num;

val = a;

}

Oper\_or\_numb& Oper\_or\_numb::operator=(Oper\_or\_numb& a)

{

lvl = a.lvl;

op = a.op;

on = a.on;

val = a.val;

return(\*this);

}

Lexeme::Lexeme( char\* c1)

{

kind = new type[strlen(c1)];

c = c1;

int t = 0;

double a;

double k = 1;

switch (c1[0])

{

case('1'):

case('2'):

case('3'):

case('4'):

case('5'):

case('6'):

case('7'):

case('8'):

case('9'):

case('0'):

{

a = c1[0]-'0';

kind[0] = num;

TStack.Set(a);

break;

}

case('('):

{

t++;

kind[0] = o\_b;

if (c1[1] == '-')

kind[0] = num;

break;

}

default:

{

throw "Not able lexeme";

break;

}

}

switch (c1[1])

{

case('('):

{

t++;

kind[1] = o\_b;

if (c1[2] == '-')

kind[1] = num;

break;

}

case('.'):

{

if ((kind[0] != num) && (kind[2] != num))

throw "Point out of number";

else

{

k \*= 0.1;

kind[1] = num;

}

break;

}

case('1'):

case('2'):

case('3'):

case('4'):

case('5'):

case('6'):

case('7'):

case('8'):

case('9'):

case('0'):

{

kind[1] = num;

if (kind[0] == num)

{

TStack.Get();

a =10\*a+ c1[1] - '0';

}

else

{

a = c1[1] - '0';

}

TStack.Set(a);

break;

}

case('+'):

case('-'):

case('\*'):

case('/'):

{

if ((c1[1] == '-') && (c1[0] == '('))

switch (c1[2])

{

case('1'):

case('2'):

case('3'):

case('4'):

case('5'):

case('6'):

case('7'):

case('8'):

case('9'):

case('0'):

case('('):

{

kind[1] = num;

break;

}

default:

{

throw "Not able lexeme";

break;

}

}

else

kind[1] = oper;

break;

}

default:

{

throw "Not able lexeme";

break;

}

}

for (int i = 2;i < strlen(c1);i++)

{

if (t >= 0)

switch (c1[i])

{

case('1'):

case('2'):

case('3'):

case('4'):

case('5'):

case('6'):

case('7'):

case('8'):

case('9'):

case('0'):

{

if (c1[i - 1] == ')')

throw "Closed bracket before number";

if (kind[i - 1] == num)

{

if (k != 1)

{

TStack.Get();

if (a >= 0)

a += k \* (c1[i] - '0');

else

a -= k \* (c1[i] - '0');

k \*= 0.1;

}

else

if (c1[i - 1] == '-')

{

a = -1 \* (c1[i] - '0');

}

else

if (c1[i - 1] == '(')

break;

else

{

TStack.Get();

if(a>=0)

a = a \* 10 + (c1[i] - '0');

else

a = a \* 10 - (c1[i] - '0');

}

}

else

a = (c1[i] - '0');

TStack.Set(a);

kind[i] = num;

break;

}

case('+'):

case('-'):

case('\*'):

case('/'):

{

if ((kind[i - 1] != num) && (kind[i - 1] != c\_b))

throw "Open bracket or operator before operator";

k = 1;

if (i + 1 >= strlen(c1))

throw "Operator in the end";

if ((c1[i] == '-') && (c1[i - 1] == '('))

switch (c1[i + 1])

{

case('1'):

case('2'):

case('3'):

case('4'):

case('5'):

case('6'):

case('7'):

case('8'):

case('9'):

case('0'):

case('('):

{

kind[i] = num;

break;

}

default:

{

throw "Not able lexeme";

break;

}

}

else

kind[i] = oper;

break;

}

case('('):

{

if ((kind[i - 1] != o\_b) && (kind[i - 1] != oper)&&(c[i-1]!='-'))

throw "Number or closed bracket before open bracket";

t++;

kind[i] = o\_b;

if ((i + 1 < strlen(c1)) && (c1[i + 1] == '-'))

{

kind[i] = num;

}

break;

}

case(')'):

{

if ((kind[i - 1] != num) && (kind[i - 1] != c\_b))

throw "Operator or open bracket before closed bracket";

k = 1;

t--;

kind[i] = c\_b;

if ((kind[i - 1] == num) && (a < 0)&&(c[i-1]!=')'))

kind[i] = num;

break;

}

case('.'):

{

if ((kind[i - 1] != num) || (k!=1))

throw "Point out of number or some points in one number";

else

{

k \*= 0.1;

kind[i] = num;

}

break;

}

default:

{

throw "Not able lexeme";

break;

}

}

else

throw "Close bracket before open bracket";

}

if (t!=0)

throw "Not equal open and close brackets";

//std::cout << kind[1];

}

Oper\_or\_numb\* Lexeme::RPN()

{

Oper\_or\_numb\* o;

Oper\_or\_numb o1;

o = new Oper\_or\_numb[strlen(c)];

int j = 0;

int s = 0;

int i;

int t=0;

int k = 0;

int a = 1;

int b = 0;

Stack <double> S;

Stack <Oper\_or\_numb> S1;

int v = TStack.Get\_pos();

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

{

S.Set(TStack.Get());

}

for (int i = 0;i < strlen(c);i++)

{

switch (Get\_kind(i))

{

case num:

{

if ((i > 0) && (c[i - 1] == '-') && (c[i - 2] == '('))

{

o[j].Set(-1.0);

k++;

t++;

j++;

s++;

o1.Set(c[i]);

S1.Set(o1);

}

while ((i < strlen(c) - 1) && (kind[i + 1] == num)&&(c[i+1]!='('))

{

i++;

}

if ((c[i] != '-')||(kind[i]==oper))

{

o[j].Set(S.Get());

j++;

s++;

}

break;

}

case oper:

{

o1.Set(c[i]);

if (S1.IsEmpty())

{

S1.Set(o1);

}

else

{

if (S1.Top().Get\_op() == '(')

{

S1.Set(o1);

}

else

if (S1.Top().Get\_lvl() >= o1.Get\_lvl())

{

o[j].Set(S1.Get().Get\_op());

S1.Set(o1);

s++;

j++;

}

else

{

S1.Set(o1);

s++;

}

}

break;

}

case o\_b:

{

t++;

if (S1.IsEmpty())

{

S1.Set(o1);

}

if ((i > 0) && (c[i - 1] == '-') && (c[i - 2] == '('))

{

o[j].Set(-1.0);

k ++;

t++;

j++;

s++;

}

o1.Set(c[i]);

S1.Set(o1);

break;

}

case c\_b:

{

t--;

if (((t == 0) || (t - k == 0)) && (k))

{

k--;

a++;

b++;

o[j+a-1].Set('\*');

j++;

s++;

}

else

{

if (b)

{

if (a)

{

o[j - a + 1].Set(S1.Get().Get\_op());

a--;

b--;

j++;

s++;

S1.Get();

}

}

else

{

while (S1.Top().Get\_op() != '(')

{

o[j].Set(S1.Get().Get\_op());

j++;

}

S1.Get();

}

}

break;

}

}

}

if (j>1)

for (;j <= s;j++)

{

o[j].Set(S1.Get().Get\_op());

}

n = j+1;

return(o);

}

double Lexeme::Answer(Oper\_or\_numb\* o)

{

double a, b;

Stack<double> S;

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

{

if (o[i].Get\_type() == num)

{

S.Set(o[i].Get\_value());

}

else

switch (o[i].Get\_op())

{

case '\*':

{

a = S.Get();

b = S.Get();

b \*= a;

S.Set(b);

break;

}

case '-':

{

a = S.Get();

b = S.Get();

b -= a;

S.Set(b);

break;

}

case '+':

{

a = S.Get();

b = S.Get();

b += a;

S.Set(b);

break;

}

case '/':

{

a = S.Get();

b = S.Get();

b /= a;

S.Set(b);

break;

}

}

}

return(S.Get());

}

## Приложение 4. Файл main\_arithmetic.cpp

#include "arithmetic.h"

#include <iostream>

using namespace std;

int main()

{

string c;

cout << "Rules for entering a mathematical samples: "<< endl;

cout << "1)Sample must not contain spaces " << endl;

cout << "2)Print your negative numbers or samples in brackets " << endl;

cout << "3)The integer and fractional parts of the number are separated '.' " << endl;

cout << "4)If the integer part of the number is 0, the point must be preceded by 0 " << endl;

cout << "5)if the fractional part of the number is 0 and you want to write not as an integer, then after the point should be 0 " << endl;

cout << "For example: 12.2/(-(11.2+1))"<< endl;

cout << "Enter your sample" << endl;

cin >> c;

char\* c1;

c1 = new char[c.length()+1];

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

{

c1[i] = c[i];

}

c1[c.length()] = '\0';

Lexeme a(c1);

cout << a.Answer(a.RPN()) << std::endl;

return 0;

}