МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Динамический класс для представления многочленов от одной переменной**

**Выполнила:** студентка группы 381703-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гитлина З. М.

Подпись

**Проверил:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волокитин В. Д.

Подпись

Нижний Новгород  
2018

Оглавление

Введение3

1. Постановка учебно-практической задачи4

2. Руководство пользователя5

3. Руководство программиста6

3.1. Описание алгоритмов вычисления арифметического выражения6

3.2. Описание структур данных8

3.3. Описание структуры программного комплекса9

Заключение10

Список литературы11

Приложение12

**Введение**

Арифметическое выражение – это запись математической формулы с использованием констант, переменных, функций, знаков арифметических операций и круглых скобок. Формулы встречаются повсеместно в экономике, физике, химии, и, конечно же, в математике. Везде, где нужно символически представить какую-нибудь функцию или соотношение. Символическое представление, конечно, не единственное. И формула является самым компактным и универсальным способом.

Некоторые простые выражения мы можем посчитать самостоятельно и быстро. Но есть и такие, для вычисления которых потребуется очень много времени. Для этого и создаются программы, выполняющие вычисления арифметических выражений.

В отчете приводится постановка задачи вычисления арифметических выражений с использованием стека, описание алгоритмов вычислений, а также дается описание программы и правил ее использования, прилагается текст программы и результаты выполнения подсчетов.

**1. Постановка учебно-практической задачи**

*Формулировка задачи:*

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 и номера символов строки, в которых были найдены ошибки.

*Исходные данные:*

string s – строка, представляющая собой арифметическое выражение, введенное пользователем.

*Требуемый результат:*

double res – результат вычисления арифметического выражения.

*Контрольный пример:*

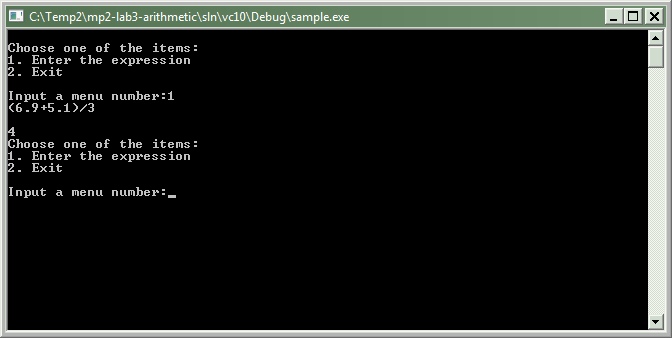
Результаты выполнения вычисления арифметического выражения (6.9+5.1)/3 приведен на Рис.1.

Рис. 1. Результат вычисления выражения

**2. Руководство пользователя**

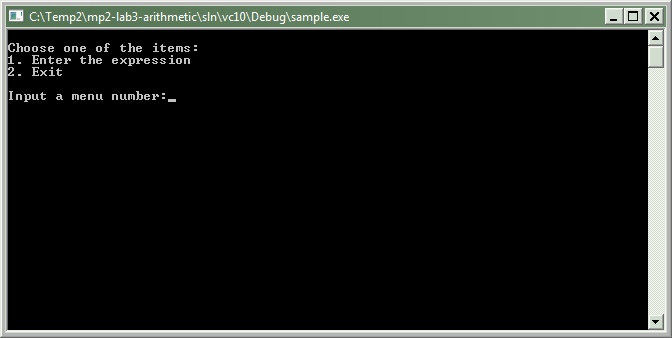
Для выполнения лабораторной работы по созданию динамического класса для представления многочленов от одной переменной необходимо запустить на выполнение файл sample.exe. На экране монитора после запуска программы открывается главное меню (Рис. 2).

Рис. 2. Главное меню программы

После выбора 1-ого пункта меню необходимо ввести арифметическое выражение, которое надо. В качестве результата программа выводит полученный результат или сообщение об ошибке.

Для выхода из программы необходимо выбрать 2 пункт меню.

**3. Руководство программиста**

**3.1. Описание алгоритмов вычисления арифметического выражения**

В программах комплекса реализованы алгоритмы проверки корректности арифметического выражения, его перевода в обратную польскую запись и вычисления. Приведем общее описание методов их выполнения.

*Проверка корректности арифметического выражения*

bool mistake(string s) – функция проверки корректности выражения, которая на вход принимает строку, а после проверки выводит сообщение о виде ошибки в выражении (если оно некорректно), на выходе она возвращает значение переменной true (если выражение введено верно) или false (если неверно).

Функция содержит в себе несколько циклов.

В первом цикле происходит последовательное сравнение каждого элемента строки с теми символами, которые допускаются выражении (они записаны в массив b). Если какой-то из элементов не совпадает ни с одним из предложенного списка, то выводится сообщение: "The wrong symbol".

Во втором цикле проводится проверка корректности выражения на скобки. Создается стек, в который при проходе по выражению помещается каждая открывающая скобка, которая извлекается из него, как только находится закрывающая скобка. Если в какой-то момент стек оказывается пустым, то выводится сообщение: "Wrong sequence of brackets".

В третьем и четвертом циклах проверяется корректность начала и конца выражения соответственно. Оно может начинаться только с открывающей скобки, минуса или числа, а заканчиваться может только на число или закрывающую скобку. В случае ошибки выводится сообщение "A wrong beginning of the expression" или "A wrong end of the expression".

В пятом цикле проверяется нет ли в выражении подряд идущих арифметических знаков (не считая минуса) или знака перед закрывающей скобкой.

В шестом цикле проверяется, что перед открывающей скобкой стоит также открывающая скобка или арифметический знак. А в седьмом – после закрывающей скобки стоит она же или арифметический знак.

*Перевод арифметического выражения в обратную польскую запись*

Lexeme\* Polish(string s, int & k) – функция на вход принимает строку и переменную k, а возвращает указатель на массив лексем в обратной польской записи.

В функции создается массив лексем, указатель на который она потом и возвращает. Далее в цикле последовательно рассматривается каждый элемент строки. Если это цифра до тех пор, пока за ней не будет арифметического знака, все элементы строки записываются в специально созданный массив, который затем с помощью функции atof() переводится в число типа double. Полученное число записывается в массив лексем: в поле Number записывается значение числа, а флаг становится равным двум. Если приходит арифметический знак, то если это первый знак в выражении, то он кладется в стек. Если это не первый знак, то если его приоритет выше чем у знака на вершине стека, то он кладется в стек, а если меньше или равен, то в цикле из стека извлекаются и записываются в массив лексем все знаки, приоритет которых выше или равен приоритета текущей операции в выражении, а затем в стек кладется сам знак этой операции. Если в выражение встречается открывающая скобка, то она кладется в стек, а когда – закрывающая, то из стека извлекаются все операции до открывающей скобки и она сама. Вконце из стека извлекаются все оставшиеся знаки операций. Также существует флаг, который считает количество подряд идущих знаков, и в зависимости от значения этого флага следующее число умножается на (-1) необходимое число раз.

*Вычисление арифметического выражения*

double result(Lexeme \*c, int k) – функция на вход принимает указатель на массив лексем в обратной польской записи и количество элементов в нем, а возвращает результат вычисления выражения

В функцию приходит массив лексем в обратной польской записи, по которому идет цикл. Если эта лексема представляет собой число, то она кладется в специально созданный стек. Если этой лексемой является знак операции, то из стека извлекаются два последних числа, над ними выполняется текущая операция, и ее результат кладется в стек. В конце выполнения цикла, в стеке, на его вершине, остается единственный элемент, значение которого равно результату вычисления арифметического выражения.

**3. 2. Описание структур данных**

В ходе выполнения данной лабораторной работы был создан класс стек (class Stack). Интерфейс класса приведен ниже:

class Stack

{

private:

ValType \* stack;// указатель на массив

int n;// размер стека

int end;// вершина стека

public:

Stack(int \_n = 100);// конструктор с параметрами, который создает стек указанного размера, также задан размер по умолчанию (100 элеменитов)

~Stack();// деструктор

ValType Pop();// извлечение элемента на вершине стека

void Push(double elem);// помещение элемента на вершину стека

bool IsEmpty();// проверка стека на пустоту

ValType Value();// значение элемента на вершине стека

int Size();// текущий размер стека

void Empty();// очистка стека

};

В данной лабораторной работе также использовался класс лексема (class Lexeme). Интерфейс класса приведен ниже:

class Lexeme

{

private:

double number;// число

char op;// операция

int flag;//флаг показывающий тип лексемы, если он равен 1, то это операция, а если 2, то число

public:

Lexeme() {};// конструктор без параметров

Lexeme(char, int);// конструктор инициализатор для операции

Lexeme(double, int); // конструктор инициализатор для числа

~Lexeme() {};// деструктор

void Set(char, int);// запись значения лексемы для операции

void Set(double, int); // запись значения лексемы для числа

void Print();// печать лексемы

int Flag() { return flag; };// значение флага

double Number() { return number; };// значение числа

char Op() { return op; };// тип операции

Lexeme& operator=(Lexeme&);// оператор присваивания

};

**3.3 .Описание структуры программного комплекса**

Программный комплекс представлен программой, состоящей из трех файлов:

1. В файле arithmetic.h представлен интерфейс класса Lexeme и прототипы функций;
2. В файле arithmetic.cpp реализованы методы класса Lexeme и функции поиска ошибок, перевода в обратную польскую запись и вычисления результата;
3. В файле stack.h представлен интерфейс класса Stack и реализованы его методы;
4. В файле main\_arithmetic.cpp находится основная программа.
5. Файл test\_stack.cpp содержит тесты для класса Stack.
6. Файл test\_arithmetic.cpp содержит тесты для класса Lexeme и функций, которые проверяют корректность выражения, переводят его в обратную польскую запись и вычисляют значение.
7. Файл test\_main.cpp запускает все google tests.

**Заключение**

В лабораторной работе разработан и создан программный комплекс, включающий в себя возможность вычисления значения арифметического выражения. Программный комплекс позволяет в режиме диалога вводить арифметическое выражение и печатать его. В работе использовалась структура данных стек, что показало ее удобство. Проведенные эксперименты доказали работоспособность данного программного комплекса.

**Список литературы**

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ. М.: МЦНМО, 1999.-960 с., 263 ил.
2. А. Шень Программирование: теоремы и задачи.- 6-е изд., дополненное.- М.: МЦНМО, 2017.-320с. : ил.

**Приложение**

**test\_stack.cpp**

#include "stack.h"

#include <gtest.h>

#include "stack.h"

#include <gtest.h>

#include "stack.h"

#include <gtest.h>

TEST(Stack, can\_create\_stack)

{

ASSERT\_NO\_THROW(Stack<int> st(10));

}

TEST(Stack, can\_push\_elem)

{

Stack<int> st(10);

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

}

TEST(Stack, can\_pop\_elem)

{

Stack<int> st(10);

st.Push(1);

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

}

TEST(Stack, can\_get\_size)

{

Stack<int> st(100);

for (int i = 0; i < 10; i++)

st.Push(i);

EXPECT\_EQ(10, st.Size());

}

TEST(Stack, can\_get\_top\_elem)

{

Stack<int> st(10);

st.Push(1);

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

}

TEST(Stack, can\_resize\_stack)

{

Stack<int> st(10);

for (int i = 0; i < 15; i++)

{

st.Push(i);

}

EXPECT\_EQ(15, st.Size());

}

TEST(Stack, throws\_when\_pop\_empty\_stack)

{

Stack<int> st(2);

st.Push(1);

st.Pop();

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

}

TEST(Stack, throws\_when\_get\_value\_from\_empty\_stack)

{

Stack<int> st(10);

int temp;

ASSERT\_ANY\_THROW(temp = st.Value());

}

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

{

ASSERT\_NO\_THROW(Stack<int> st(10));

}

TEST(Stack, throws\_when\_create\_stack\_with\_negative\_length)

{

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

}

TEST(Stack, can\_get\_stack\_current\_size)

{

Stack<int> st(10);

for (int i = 0; i < 5; i++)

st.Push(i);

EXPECT\_EQ(5, st.Size());

}

TEST(Stack, can\_check\_if\_stack\_is\_empty)

{

Stack<int> st(10);

EXPECT\_EQ(st.IsEmpty(), true);

}

TEST(Stack, can\_delete\_all\_elements)

{

Stack<int> st(10);

for (int i = 0; i < 5; i++)

st.Push(i);

st.Empty();

EXPECT\_EQ(st.IsEmpty(), true);

}

**test\_arithmetic.cpp**

#include <gtest.h>

#include "arithmetic.h"

#include <gtest.h>

#include <string>

using namespace std;

TEST(Lexeme, can\_create\_double)

{

Lexeme l(10.05, 2);

EXPECT\_DOUBLE\_EQ(l.Number(), 10.05);

}

TEST(Lexeme, can\_create\_binary\_operation)

{

Lexeme l('+', 1);

EXPECT\_EQ(l.Op(), '+');

}

TEST(Priority, can\_determine\_priority\_of\_operation) {

EXPECT\_EQ(3, prt('\*'));

EXPECT\_EQ(3, prt('/'));

EXPECT\_EQ(3, prt(':'));

EXPECT\_EQ(2, prt('+'));

EXPECT\_EQ(2, prt('-'));

}

TEST(Polish\_notation, can\_create\_right\_sequence\_of\_calculations)

{

string s = "5+(4\*7-8)";

int k = 0;

Lexeme \*c = Polish(s, k);

int i = 0;

EXPECT\_EQ(c[i++].Number(), 5);

EXPECT\_EQ(c[i++].Number(), 4);

EXPECT\_EQ(c[i++].Number(), 7);

EXPECT\_EQ(c[i++].Op(), '\*');

EXPECT\_EQ(c[i++].Number(), 8);

EXPECT\_EQ(c[i++].Op(), '-');

EXPECT\_EQ(c[i++].Op(), '+');

}

TEST(Polish\_notation, can\_create\_right\_sequence\_of\_calculations\_with\_unary\_minus)

{

string s = "-5+(4\*-7-8)";

int k = 0;

Lexeme \*c = Polish(s, k);

int i = 0;

EXPECT\_EQ(c[i++].Number(), -5);

EXPECT\_EQ(c[i++].Number(), 4);

EXPECT\_EQ(c[i++].Number(), -7);

EXPECT\_EQ(c[i++].Op(), '\*');

EXPECT\_EQ(c[i++].Number(), 8);

EXPECT\_EQ(c[i++].Op(), '-');

EXPECT\_EQ(c[i++].Op(), '+');

}

TEST(Polish\_notation, can\_create\_right\_sequence\_of\_calculations\_with\_double\_numbers)

{

string s = "5.7+(4\*7-8.5)";

int k = 0;

Lexeme \*c = Polish(s, k);

int i = 0;

EXPECT\_DOUBLE\_EQ(c[i++].Number(), 5.7);

EXPECT\_DOUBLE\_EQ(c[i++].Number(), 4);

EXPECT\_DOUBLE\_EQ(c[i++].Number(), 7);

EXPECT\_DOUBLE\_EQ(c[i++].Op(), '\*');

EXPECT\_DOUBLE\_EQ(c[i++].Number(), 8.5);

EXPECT\_DOUBLE\_EQ(c[i++].Op(), '-');

EXPECT\_DOUBLE\_EQ(c[i++].Op(), '+');

}

TEST(Polish\_notation, can\_not\_count\_the\_wrong\_expression)

{

string s = "5.7+-(4\*7-8.5)";

int k = 0;

EXPECT\_EQ(mistake(s), false);

}

TEST(Polish\_notation, can\_not\_count\_the\_expression\_with\_wrong\_symbols)

{

string s = "5.7!+(4\*7-8.5)";

int k = 0;

EXPECT\_EQ(mistake(s), false);

}

TEST(Polish\_notation, can\_not\_count\_the\_expression\_with\_wrong\_symbols\_2)

{

string s = "5.7A+(4\*7-8.5)";

int k = 0;

EXPECT\_EQ(mistake(s), false);

}

TEST(Polish\_notation, can\_not\_count\_the\_expression\_with\_wrong\_symbols\_3)

{

string s = "\*5.7+(4\*7-8.5)";

int k = 0;

EXPECT\_EQ(mistake(s), false);

}

TEST(Polish\_notation, can\_not\_count\_the\_expression\_with\_wrong\_symbols\_4)

{

string s = "5.7+(4\*7-8.5)-";

int k = 0;

EXPECT\_EQ(mistake(s), false);

}

TEST(Result, can\_solve\_expression)

{

string s = "5+(4\*7-8)";

int k = 0;

Lexeme \*c = Polish(s, k);

EXPECT\_EQ(result(c, k), 25);

}

TEST(Result, can\_solve\_expression\_2)

{

string s = "-5+(4\*-7-8)";

int k = 0;

Lexeme \*c = Polish(s, k);

EXPECT\_EQ(result(c, k), -41);

}

TEST(Result, can\_solve\_expression\_3)

{

string s = "5.7+(4\*7-8.5)";

int k = 0;

Lexeme \*c = Polish(s, k);

EXPECT\_DOUBLE\_EQ(result(c, k), 25.2);

}

**test\_main.cpp**

#include <gtest.h>

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

**main\_arithmetic.cpp**

#include<string>

#include "arithmetic.h"

#include "stack.h"

using namespace std;

void main()

{

int l;

bool res;

do {

cout << '\n' << "Choose one of the items:" << '\n';

char\*d[3] = { "1. Enter the expression ","2. Exit" };

for (int i = 0; i < 2; i++)

cout << d[i] << '\n';

cout << '\n' << "Input a menu number:";

cin >> l;

if (l == 2) break;

int k = 0;

string s;

getline(cin, s);

if (s == "")

{

getline(cin, s);

}

bool fl;

for (int i = s.length() - 1; i >= 0; --i)

{

if (s[i] == ' ')

s.erase(i, 1);

}

fl = mistake(s);

if (fl) {

Lexeme \*c = Polish(s, k);

double res;

try {

res = result(c, k);

cout << '\n' << res;

}

catch (int a) {

/\*if (a == 1)\*/

cout << '\n' << "DIV 0";

}

}

} while (l < 2);

}

**arithmetic.cpp**

#include "stack.h"

#include "arithmetic.h"

#include<string>

#include <cstdio>

#include <cstdlib>

#include <iostream>

using namespace std;

Lexeme::Lexeme(char \_op, int \_f)

{

op = \_op;

flag = \_f;

}

Lexeme::Lexeme(double \_number, int \_f)

{

number = \_number;

flag = \_f;

}

void Lexeme::Set(char \_op, int \_f)

{

op = \_op;

flag = \_f;

}

void Lexeme::Set(double \_number, int \_f)

{

number = \_number;

flag = \_f;

}

void Lexeme::Print()

{

if (flag == 1)

cout << '\n' << op;

else

cout << '\n' << number;

}

Lexeme& Lexeme::operator=(Lexeme & a) {

number = a.number;

op = a.op;

flag = a.flag;

return \*this;

}

int prt(char \_op) {

int prt;

switch (\_op) {

case'\*':

case':':

case'/': prt = 3; break;

case'+':

case'-': prt = 2; break;

case'=':prt = 1; break;

default:prt = -1;

}

return prt;

}

Lexeme\* Polish(string s, int & k) {

k = 0;

int l = 0;

int m = 0;

int n;

int q;

int g = 0;

Lexeme \*c;

c = new Lexeme[size(s)];

char \*d;

Stack<char> a(size(s));

for (int i = 0; i < size(s); i++)

if ((s[i] != '+') && (s[i] != '-') && (s[i] != '\*') && (s[i] != '/') && (s[i] != '(') && (s[i] != ')') && (s[i] != ':')) {

m = 0;

for (int j = i; (j < size(s)) && (s[j] != '+') && (s[j] != '-') && (s[j] != '\*') && (s[j] != '/') && (s[j] != '(') && (s[j] != ')'); j++)

m++;

d = new char[m + 1];

for (int j = i, n = 0; j < i + m; j++, n++)

d[n] = s[j];

d[m] = { '\0' };

double f = atof(d);

for(int i=1;i<g;i++)

f = -f;

g = 0;

c[k++].Set(f, 2);

i += m - 1;

}

else {

if (l == 0) {

if ((s[i] == '-') && (i == 0))

g = 2;

else

{

a.Push(s[i]);

l++;

g++;

}

}

else {

if (s[i] == ')') {

while (a.Value() != '(') {

c[k++].Set(a.Pop(), 1);

}

a.Pop();

}

else {

if ((!a.IsEmpty())&&((prt(a.Value()) < prt(s[i])) || (s[i] == '('))) {

if ((g == 0) || (s[i] == '('))

a.Push(s[i]);

if (s[i] != '(')

g++;

}

else {

if (g == 0) {

while ((!a.IsEmpty()) && (prt(a.Value()) >= prt(s[i])) && (a.Value() != '(')) {

c[k++].Set(a.Pop(), 1);

}

a.Push(s[i]);

}

else

g++;

}

}

}

}

while (!(a.IsEmpty()))

c[k++].Set(a.Pop(), 1);

return c;

}

double result(Lexeme \*c, int k) {

double res;

Stack<double> a(k);

for (int i = 0; i < k; i++) {

if (c[i].Flag() == 2) {

a.Push(c[i].Number());

}

else {

double m = a.Pop();

double l = a.Pop();

switch (c[i].Op())

{

case '+':

res = l + m;

break;

case '-':

res = l - m;

break;

case '\*':

res = l \* m;

break;

case ':':

case '/':

if (m != 0)

res = l / m;

else throw 1;

break;

}

a.Push(res);

}

}

if (k == 1)res = a.Pop();

return res;

}

bool mistake(string s) {

bool f = true;

int g = 0;

char b[19] = { '0','1','2','3','4','5','6','7','8','9','-','+','\*','/',':','.',',','(',')' };

for (int i = 0; i < size(s); i++) {

for (int j = 0; j < 19; j++)

if (s[i] == b[j]) {

g++;

}

if (g == 0) {

cout << '\n' << "The wrong symbol";

f = false;

}g = 0;

}

Stack<char> a(size(s));

int t = 0;

char br;

for (int i = 0; i < size(s); i++)

{

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

a.Push('(');

else

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

{

if (!(a.IsEmpty()))

{

br = a.Pop();

}

else

{

t++;

}

}

}

if (t != 0) {

f = false;

cout << '\n' << "Wrong sequence of brackets";

}

for (int j = 11; j < 17; j++)

if ((s[0] == b[j]) || (s[0] == ')')) {

f = false;

cout << '\n' << "A wrong begining of the expression";

}

for (int j = 10; j < 18; j++)

if (s[size(s) - 1] == b[j]) {

f = false;

cout << '\n' << "A wrong end of the expression";

}

for (int i = 0; i < size(s); i++) {

for (int j = 10; j < 19; j++)

if (s[i] == b[j])

if ((i + 1) < size(s)) {

for (int l = 11; l < 16; l++)

if (((s[i + 1] == b[l])||((s[i+1]=='-') &&(s[i + 2])=='(' ))&&(s[i]!=')')) {

f = false;

cout << '\n' << "This is not right sequence of symbols in the expression";

}

}

}

for (int i = 1; i < size(s); i++) {

if (s[i] == '(') {

for (int j = 10; j < 15; j++)

if ((s[i - 1] == b[j]) || (s[i] == '(')) {

g++;

}

if (g == 0) {

f = false;

cout << '\n' << "This is not right sequence of symbols in the expression";

}

}g = 0;

}

for (int i = 0; i < size(s) - 1; i++) {

if (s[i] == ')') {

for (int j = 10; j < 15; j++)

if ((s[i + 1] == b[j]) || (s[i + 1] == ')')) {

g++;

}

if (g == 0)

{

f = false;

cout << '\n' << "This is not right sequence of symbols in the expression";

}

}g = 0;

}

return f;

}

**stack.h**

#pragma once

#include <iostream>

template <class ValType>

class Stack

{

private:

ValType \* stack;

int n;

int end;

public:

Stack(int \_n = 100);

~Stack();

ValType Pop();

void Push(double elem);

bool IsEmpty();

ValType Value();

int Size();

void Empty();

};

template <class ValType>

Stack<ValType>::Stack(int \_n) {

if (\_n <= 0)

throw 2;

else {

stack = new ValType[\_n];

end = -1;

n = \_n;

}

}

template <class ValType>

Stack<ValType>::~Stack()

{

delete[]stack;

stack = NULL;

end = -1;

}

template <class ValType>

ValType Stack <ValType>::Pop()

{

if ((\*this).IsEmpty())

throw ("error");

else {

int temp = end;

end--;

return stack[temp];

}

}

template <class ValType>

int Stack <ValType>::Size()

{

return (end + 1);

}

template <class ValType>

void Stack<ValType>::Push(double elem)

{

if (end + 1 >= n) {

ValType \*temp;

temp = new ValType[2 \* n];

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

temp[i] = stack[i];

delete[]stack;

stack = temp;

n = 2 \* n;

}

end++;

stack[end] = elem;

}

template <class ValType>

void Stack<ValType>::Empty()

{

end = -1;

n = 0;

delete[]stack;

stack = NULL;

}

template <class ValType>

bool Stack<ValType>::IsEmpty()

{

if (end == -1)

return true;

else return false;

}

template <class ValType>

ValType Stack <ValType>::Value()

{

if (IsEmpty())

throw 1;

else

return stack[end];

}

**arithmetic.h**

#ifndef \_ARITHMETIC\_H\_

#define \_ARITHMETIC\_H\_

#include <string>

#include <vector>

#include <set>

using namespace std;

class Lexeme

{

private:

double number;

char op;

int flag;

public:

Lexeme() {};

Lexeme(char, int);

Lexeme(double, int);

void Set(char, int);

void Set(double, int);

void Print();

int Flag() { return flag; };

double Number() { return number; };

char Op() { return op; };

Lexeme& operator=(Lexeme&);

~Lexeme() {};

};

Lexeme\* Polish(string, int &);

int prt(char op);

double result(Lexeme\*, int);

bool mistake(string);

#endif \_ARITHMETIC\_H\_