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

# Вычисление арифметических выражений

**Выполнила:** студентка группы 831703-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шибаева И. В.

**Проверил**:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Волокитин В.Д.

# Содержание

[**Вычисление арифметических выражений**](#_djabqb4mhso9) **1**

[**Содержание**](#_f7iuj18fq39k) **2**

[**Введение**](#_30j0zll) **3**

[**Постановка учебно-практической задачи**](#_1fob9te) **3**

[**2. Руководство пользователя**](#_3znysh7) **3**

[**3. Руководство программиста**](#_2et92p0) **4**

[3.1 Описание алгоритмов арифметических действий и работы перегрузок](#_tyjcwt) 4

[3.2 Описание структур данных](#_3dy6vkm) 5

[3.3 Общее описание структур программного комплекса](#_i28vm9hdh54g) 7

[**Заключение**](#_macektxxkt3z) **8**

[**Список литературы**](#_wbcjq37wsnyu) **9**

# 

# **Введение**

Арифметическое выражение — это последовательность констант, переменных, функций, соединенных знаками арифметических операций.

Арифметические выражения можно представить как в префиксной и инфиксной, так и в постфиксной нотации.

Польская нотация, также известна как префиксная нотация, это форма записи логических, арифметических и алгебраических выражений. Характерная черта такой записи — оператор располагается слева от операндов.

# Постановка учебно-практической задачи

*Формулировка задачи:*

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

*Исходные данные:*

String s, содержащая арифметическое выражение.

*Требуемый результат:*

Число, полученное в результате подсчитанного выражения в форме обратной польской нотации.

*Контрольный пример:*

Введем выражение 1+(-2\*3 - 4). Программа должна посчитать его и вывести на экран -9.

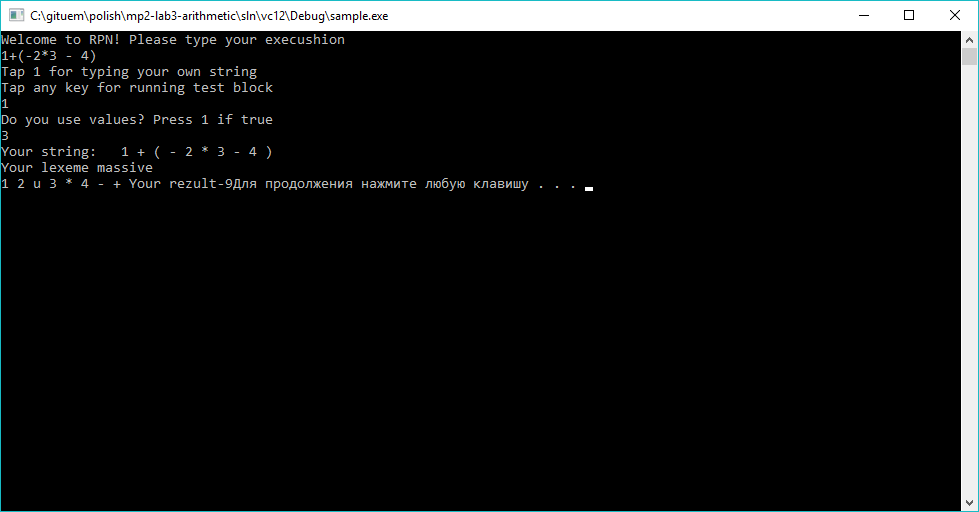


Рис. 1. Результат контрольного примера

# 2. Руководство пользователя

Для запуска программы необходимо открыть файл sample.exe. На экране консоли появится запрос строки или просьба запустить тестовый блок с готовыми примерами. Затем, если пользователь выбрал подсчет строки, то он должен указать, хочет ли он ввести в своё выражение переменную, и если да, то каково её значение.

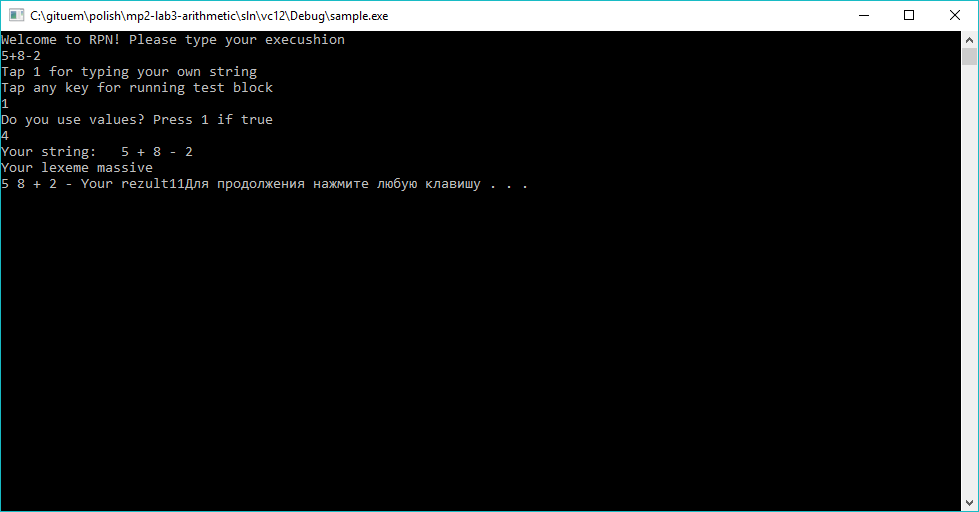


Рис. 2. Демонстрация запуска программы

# 3. Руководство программиста

### **3.1 Описание алгоритмов арифметических действий и работы перегрузок**

Для решения поставленной задачи были реализованы алгоритмы проверки строки, анализа и перевода в класс лексем и перевода в обратную польскую запись, подсчет выражения.

*1) Проверка корректности выражения*

bool Checking\_Block(string s, int valhere) - функция, не член класса, которая принимает на вход строку с выражением и флаг, отвечающий за наличие переменной в выражении.

Содержит три блока:

bool Skobki(string s) - проверка на правильность расстановки скобок. Если приходит открывающая скобка, она кладется в стек. Если на вход подается закрывающая скобка, открывающая удаляется из стека. В случае, если на пришедшую закрывающую скобку не найдется открывающей в стеке или по окончанию строки стек не пуст, выводится сообщение об ошибке.

bool CheckSequence(string s) - проверка порядка следования знаков выражении. Следит за тем, чтобы знак операции не шел после знака (за исключением случая с унарным минусом), скобки не закрывались без выражения внутри.

bool ValisOne(string s) - в случае, если пользователь собирается использовать переменную, проверяет строку на единство символов переменной в выражении.

*2) Анализ знаков в выражении*

vector<Lexem> Parsing (string a) - посимвольно проверяет строку и определяет каждый символ как знак или как число. Может считывать десятичные дроби. Создается массив лексем, туда записывается уже разобранная строка.

*3) Перевод в обратную польскую нотацию*

vector<Lexem> Polish(vector<Lexem> s) - принимает на вход массив лексем, числа кладутся в новый массив. Создается стек для операций. Если пришла операция - смотрим на знак и приоритет.

При появлении операции в исходном массиве:

а) если в стеке нет операций или верхним элементом стека является открывающая скобка, операции кладется в стек;

б) если новая операции имеет больший приоритет, чем верхняя операции в стеке, то новая операции кладется в стек;

с) если новая операция имеет меньший или равный приоритет, чем верхняя операции в стеке, то операции, находящиеся в стеке, до ближайшей открывающей скобки или до операции с приоритетом меньшим, чем

у новой операции, перекладываются в формируемую запись, а новая операции кладется в стек.

2. Открывающая скобка кладется в стек.

3. Закрывающая скобка выталкивает из стека в формируемую запись все операции до ближайшей открывающей

скобки, открывающая скобка удаляется из стека.

4.После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке,

перекладываются в формируемые выражение.

*3) Подсчет массива лексем*

double Calc(vector<Lexem> s) - принимает на вход массив лексем в обратной польской записи, возвращает число. Создается стек для чисел. Если элемент массива - число, он кладется в стек. Если операция - забирается элемент из стека, выполняется операция Pop и совершается действие, соответствующие пришедшему из массива знаку. Полученное число возвращается в стек.  
После завершения подсчета в качестве ответа возвращается число, оставшееся в стеке.

### **3.2 Описание структур данных**

Для работы программы были созданы классы TStack и Lexem

**Класс TStack:**

template <class TType>

class TStack

{

int top; - номер вершины

int size; - размер стека

TType \*mas; - массив для данных стека

public:

TStack();

TStack(int ssize);

void Push(const TType &val);

TType Pop();

bool Empty();

void NewLen();

~TStack() { delete[]mas; };

TType CheckLast();

void ClearStack();

int GetSize();

};

**Класс Lexem:**

class Lexem

{

char op; - переменная под знаки операций

double num; - переменная под число

bool notop; - операция или нет

bool binary; - бинарная ли операция

public:

Lexem() {}

Lexem(double number) { num = number; notop = true; binary = false; }

Lexem(char oper){num = 0; notop = false; op = oper; binary = true;}

bool Binar();

void IfUnar();

double GetNum();

char GetOper();

bool IsNum();

void IfVal();

~Lexem() {};

friend ostream& operator<<(ostream &out, vector<Lexem> &lex); };

### **3.3 Общее описание структур программного комплекса**

Приложение представлено следующими файлами:

1. В файле arithmetic.h объявлен класс Lexem и прототипы функций для счёта.
2. В файле arithmetic.cpp реализованы методы класса Lexeme, функции проверки, разбора выражения, перевода его в польскую нотацию и подсчет выражения.
3. В файле stack.h реализован шаблонный класс TStack.
4. В файле main\_arithmetic.cpp содержится интерфейс пользовательского приложения.
5. В файле test\_arithmetic.cpp написаны тесты проверки вычисления в ОПН, блока проверки и создания объектов класса Lexem.
6. В файле test\_stack.cpp реализованы тесты для стека.
7. Файл test\_main.cpp запускает все тесты сразу.

# 

# Заключение

Для данной работы был разработан программный комплекс, позволяющий вычислять арифметическое выражение. Представленный в отчете программный комплекс позволяет производить работу с выражениями, выводить результат перевода в ОПН и печатать результаты вычисления на экран. Также специально для данной программы был разработан класс стека.

# 

# Список литературы

1. Шилдт, Г. С++ для начинающих: самоучитель / Шилдт, Г. - Изд-во: Эком, 2013г.
2. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ, М.:МЦНМО, 1999.-960 с.б 263 ил.

**Приложение**

**Файл arithmetic.h**

#ifndef arif\_H

#define arif\_H

#include "stack.h"

#include <string>

#include <stdio.h>

#include <map>

#include <vector>

#include <iostream>

using namespace std;

class Lexem

{

char op;

double num;

bool notop;

bool binary;

public:

Lexem() {}

Lexem(double number) { num = number; notop = true; binary = false; }

Lexem(char oper){num = 0; notop = false; op = oper; binary = true;}

bool Binar();

void IfUnar();

double GetNum();

char GetOper();

bool IsNum();

void IfVal();

~Lexem() {};

friend ostream& operator<<(ostream &out, vector<Lexem> &lex);

};

vector<Lexem> PutVal(double a, vector<Lexem> &s);

vector<Lexem> Parsing(string a);

vector<Lexem> Polish(vector<Lexem> s);

double Calc(vector<Lexem> s);

// checking block

bool Checking\_Block(string s, int valhere);

bool Skobki(string s);

bool CheckSequence(string s);

bool ValisOne(string s);

void DoPriority();

#endif

**Файл arithmetic.cpp:**

#include "arithmetic.h"

map<char, int> priority;

vector<Lexem> Parsing(string a)

{

vector<Lexem> PrePolish;

int i = 0;

while (a[i]!='\0')

{

if (a[i] == ' ')

i++;

else if ((a[i] >= '0') && (a[i] <= '9'))

{

double dd = 0.0;

double ff = 0.0;

while ((a[i] >= '0') && (a[i] <= '9'))

{

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

i++;

}

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

{

++i;

int raz = i;

double rezerv = 1;

while ((a[i] >= '0') && (a[i] <= '9'))

{

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

i++;

}

for (int j = 0; j < i - raz; j++)

{

rezerv \*= 10;

}

ff /= rezerv;

}

PrePolish.push\_back(dd+ff);

}

else if ((a[i] == '+') || (a[i] == '-') || (a[i] == '/') || (a[i] == '(') || (a[i] == ')') || (a[i] == '\*'))

{

DoPriority();

PrePolish.push\_back(a[i]);

Lexem A = PrePolish.back();

PrePolish.pop\_back();

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

{

if (!(PrePolish.empty()))

{

Lexem B = PrePolish.back();

if (!B.IsNum() && (B.GetOper() != ')'))

{

A.IfUnar();

}

}

else {

A.IfUnar();

}

PrePolish.push\_back(A);

i++;

}

else if ((a[i] == '\*') || (a[i] == '/') || (a[i] == '+') || (a[i] == '(') || (a[i] == ')'))

{

PrePolish.push\_back(a[i]);

i++;

}

}

else if ((a[i] >= 'a') && (a[i] <= 'z'))

{

Lexem A(a[i]);

A.IfVal();

PrePolish.push\_back(A);

i++;

}

else i++;

}

return PrePolish;

}

vector<Lexem> PutVal(double a, vector<Lexem> &s)

{

Lexem A(a);

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

{

if (s[i].IsNum() && ((s[i].GetOper() >= 'a') && (s[i].GetOper() <= 'z')))

{

s[i]=A;

}

}

return s;

}

vector<Lexem> Polish(vector<Lexem> s)

{

vector<Lexem> Polish;

TStack<char> Op;

int k = 0;

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

{

if (s[i].IsNum())

{

Polish.push\_back(s[i].GetNum());

}

else

{

if (!s[i].Binar())

{

while (s[i].GetOper()=='-')

{

k++; ++i;

}

--i;

if (k % 2 == 1)

{

Lexem A('u');

A.IfUnar();

Op.Push(A.GetOper());

k = 0;

}

}

else if ((s[i].Binar())&&(Op.Empty()|| s[i].GetOper()=='(' || Op.CheckLast() == '(' || (priority[Op.CheckLast()] < priority[s[i].GetOper()])))

{

Op.Push(s[i].GetOper());

}

else if ((s[i].Binar()) && (priority[Op.CheckLast()] >= priority[s[i].GetOper()]))

{

if ((s[i].Binar()) && s[i].GetOper() == ')')

{

while (Op.CheckLast() != '(')

Polish.push\_back(Op.Pop());

Op.Pop();

}

else

{

while (!Op.Empty() && (priority[Op.CheckLast()] >= priority[s[i].GetOper()]))

Polish.push\_back(Op.Pop());

Op.Push(s[i].GetOper());

}

}

}

if (i == (s.size()-1))

{

while (!Op.Empty())

if (Op.CheckLast() == '(' || Op.CheckLast() == ')')

Op.Pop();

else Polish.push\_back(Op.Pop());

}

}

return Polish;

}

double Calc(vector<Lexem> s)

{

TStack<double> Num;

double Rez;

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

{

if (s[i].IsNum())

Num.Push(s[i].GetNum());

else if(!Num.Empty())

{

double frst=Num.Pop();

switch (s[i].GetOper())

{

case '+':

Rez = Num.Pop()+frst;

Num.Push(Rez);

break;

case '-':

Rez = Num.Pop() - frst;

Num.Push(Rez);

break;

case '\*':

Rez = Num.Pop()\*frst;

Num.Push(Rez);

break;

case '/':

if (frst != 0)

{

Rez = Num.Pop() / frst;

Num.Push(Rez);

}

else throw ("Division by null");

break;

case 'u':

double a = -frst;

Num.Push(a);

break;

}

}

}

Rez = Num.Pop();

return Rez;

}

bool Lexem::Binar()

{

return binary;

}

double Lexem::GetNum()

{

return num;

}

char Lexem::GetOper()

{

return op;

}

bool Lexem::IsNum()

{

return notop;

}

void Lexem::IfUnar()

{

binary = false;

}

void Lexem::IfVal()

{

notop = true;

}

bool Skobki(string s)

{

TStack<char> rez;

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

{

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

{

rez.Push(s[i]);

}

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

{

if(rez.Empty())

{

cout << "Wrong sintaxis" << endl;

return false;

}

else rez.Pop();

}

}

return rez.Empty();

}

bool CheckSequence(string s)

{

if ((s[0] == '\*') || (s[0] == '/') || (s[0] == '+'))

{

cout << "Wrong first element" << endl;

return false;

}

if ((s[s.size() - 1] == '-') || (s[s.size() - 1] == '+') || (s[s.size() - 1] == '\*') || (s[s.size() - 1] == '/'))

{

cout << "Wrong last element" << endl;

return false;

}

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

{

if ((s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/' || s[i]=='(') && (s[i + 1] == '+' || s[i + 1] == '\*' || s[i + 1] == '/' || s[i+1]==')'))

{

cout << "operation after operation" << endl;

return false;

}

}

return true;

}

bool ValisOne(string s)

{

string qq;

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

{

if ((s[i] >= 'a') && (s[i] <= 'z'))

qq += s[i];

}

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

{

if (qq[1] != qq[i])

{

cout << "Wrong value name" << endl;

return false;

}

}

return true;

}

void DoPriority()

{

priority['u'] = 3;

priority['('] = 0;

priority[')'] = 0;

priority['\*'] = 2;

priority['/'] = 2;

priority['+'] = 1;

priority['-'] = 1;

}

ostream& operator<<(ostream &out, vector<Lexem> &lex)

{

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

{

if (lex[i].IsNum())

out << lex[i].GetNum()<< " ";

else

out << lex[i].GetOper()<<" ";

}

return out;

}

bool Checking\_Block(string s, int valhere)

{

if (!Skobki(s))

cout << "Error parentheses" << endl;

if (!CheckSequence(s))

cout << "Wrong syntax" << endl;

if (valhere == 1)

{

if (!ValisOne(s))

cout << "Please, use one value" << endl;

else return (Skobki(s) && CheckSequence(s) && ValisOne(s));

}

return (Skobki(s) && CheckSequence(s));

}

**Файл stack.h**

// объявление и реализация шаблонного стека

// стек поддерживает операции:

// - вставка элемента,

// - извлечение элемента,

// - просмотр верхнего элемента (без удаления)

// - проверка на пустоту,

// - получение количества элементов в стеке

// - очистка стека

// при вставке в полный стек должна перевыделяться память

#ifndef stack\_h

#define stack\_h

template <class TType>

class TStack

{

int top;

int size;

TType \*mas;

public:

TStack();

TStack(int ssize);

void Push(const TType &val);

TType Pop();

bool Empty();

void NewLen();

~TStack() { delete[]mas; };

TType CheckLast();

void ClearStack();

int GetSize();

};

template <class TType>

TStack <TType>::TStack()

{

size = 10;

top = -1;

mas = new TType[size];

}

template <class TType>

TStack <TType>::TStack(int ssize)

{

if (ssize <= 0)

throw ("Uncorr stack leight");

size = ssize;

top = -1;

mas = new TType[size];

}

template <class TType>

void TStack<TType>::Push(const TType &val)

{

if (size == top + 1)

NewLen();

mas[top + 1] = val;

top++;

}

template <class TType>

TType TStack<TType>::Pop()

{

if (Empty())

throw ("Stack is empty");

TType a = mas[top];

top--;

return a;

}

template <class TType>

void TStack<TType>::NewLen()

{

TType \*Copmas = new TType[size \* 2];

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

Copmas[i] = mas[i];

delete[]mas;

size \*= 2;

mas = Copmas;

}

template <class TType>

bool TStack<TType>::Empty()

{

if (top == -1)

return true;

else return false;

}

template <class TType>

TType TStack<TType>::CheckLast()

{

if (Empty())

throw ("Stack is empty");

return (mas[top]);

}

template <class TType>

void TStack<TType>::ClearStack()

{

top = -1;

}

template <class TType>

int TStack<TType>::GetSize()

{

return size;

}

#endif

**Файл main\_arithmetic.h**

// реализация пользовательского приложения

#include "arithmetic.h"

void main()

{

vector<Lexem> Primer, Pol;

double Rezz, value;

std::string s;

int react2, react1;

cout << "Welcome to RPN! Please type your execushion" << endl;

getline(cin, s);

cout << "Tap 1 for typing your own string\nTap any key for running test block " << endl;

cin >> react1;

if (react1 == 1)

{

cout << "Do you use values? Press 1 if true" << endl;

cin >> react2;

while (!Checking\_Block(s,react2))

{

cout << "Wrong syntaxis, check your string" << endl;

getline(cin, s);

}

Primer = Parsing(s);

if (react2 == 1)

{

cout << "Please, put your val" << endl;

if (value)

Primer = PutVal(value, Primer);

}

cout << "Your string: " << Primer << endl;

Pol = Polish(Primer);

cout <<"Your lexeme massive\n"<< Pol ;

Rezz = Calc(Pol);

cout << "Your rezult"<< Rezz;

}

else

{

string test1 = "3))+", test2 = "2+3+4+5+", test3 = "a+b-(2--4)", test4 = "2+(3-(5+7)-1)-2", test5 = "-2+d+(d-5)";

cout << "test1 " << test1 << endl;

cout << "test2 " << test2 << endl;

cout << "test3 " << test3 << endl;

cout << "test4 " << test4 << endl;

cout << "test5 " << test5 << endl;

if (!Checking\_Block(test1, 0))

cout << "test 1 end"<<endl;

if (!Checking\_Block(test2, 0))

cout << "test 2 end" << endl;

if (!Checking\_Block(test3, 1))

cout << "test 3 end" << endl;

if (Checking\_Block(test4, 0))

{

Primer = Parsing(test4);

Pol = Polish(Primer);

cout << "test 4 answer:" << Pol << endl;

Rezz = Calc(Pol);

cout << "test 4 answer:" << Rezz << endl;

}

if (Checking\_Block(test5, 1))

{

Primer = Parsing(test5);

Primer = PutVal(4,Primer);

cout << "test value 4" << endl;

Pol = Polish(Primer);

Rezz = Calc(Pol);

cout << "test 4 answer: " << Rezz << endl;

}

}

system("pause");

}

**Файл test\_stack.cpp**

// тесты для стека

#include "stack.h"

#include <gtest.h>

TEST(TStack, can\_generate\_stack\_without\_leight)

{

ASSERT\_NO\_THROW(TStack <int> stk);

}

TEST(TStack, can\_generate\_stack\_with\_leight)

{

ASSERT\_NO\_THROW(TStack <int> stk(20));

}

TEST(TStack, can\_create\_stack\_with\_wrong\_leight)

{

ASSERT\_ANY\_THROW(TStack <int> stk(-20));

}

TEST(TStack, new\_stack\_is\_empty)

{

TStack <int> stk(20);

EXPECT\_EQ(stk.Empty(), true);

}

TEST(TStack, can\_push\_to\_empty\_st)

{

TStack <int> stk;

ASSERT\_NO\_THROW(stk.Push(3));

}

TEST(TStack, can\_push\_and\_check)

{

TStack <int> stk;

stk.Push(45);

EXPECT\_EQ(stk.CheckLast(), 45);

}

TEST(TStack, can\_push\_and\_pop)

{

TStack <int> stk;

stk.Push(45);

stk.Pop();

EXPECT\_EQ(stk.Empty(), true);

}

TEST(TStack, overflow\_of\_massive)

{

TStack <int> stk(2);

stk.Push(1);

stk.Push(2);

stk.Push(3);

ASSERT\_NO\_THROW(stk.Push(4));

}

TEST(TStack, can\_pop\_empty\_stack)

{

TStack <int> stk;

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

}

TEST(TStack, check\_top\_elements)

{

TStack <int> stk(4);

stk.Push(1);

stk.Push(2);

stk.Push(3);

stk.Pop();

EXPECT\_NE(stk.CheckLast(), 3);

}

TEST(TStack, check\_dif\_size\_0f\_stk)

{

TStack <int> stk(2);

stk.Push(1);

stk.Push(2);

stk.Push(3);

stk.Push(4);

EXPECT\_NE(stk.GetSize(), 2);

}

TEST(TStack, check\_size\_of\_clear\_stk)

{

TStack <int> stk(20);

EXPECT\_EQ(stk.GetSize(), 20);

}

TEST(TStack, can\_check\_last\_in\_empty)

{

TStack <int> stk;

ASSERT\_ANY\_THROW(stk.CheckLast());

}

TEST(TStack, clear\_stk)

{

TStack <int> stk;

stk.Push(1);

stk.Push(1);

stk.Push(1);

stk.Push(1);

stk.Push(1);

stk.Push(1);

stk.Push(1);

stk.ClearStack();

EXPECT\_EQ(stk.Empty(), true);

}

**Файл test\_arithmetic.cpp**

// тесты для вычисления арифметических выражений

#include <gtest.h>

#include "arithmetic.h"

TEST(Lexem, can\_lexeme)

{

double a = 4;

EXPECT\_NO\_THROW(Lexem S(a));

}

TEST(Lexem, can\_lexeme\_number)

{

double a = 2;

Lexem A(a);

EXPECT\_EQ(a, A.GetNum());

}

TEST(Lexem, can\_lexeme\_with\_more\_the\_one)

{

double a = 256;

Lexem A(a);

EXPECT\_EQ(a, A.GetNum());

}

TEST(Lexem, can\_lexeme\_Operation)

{

char a = '+';

Lexem A(a);

EXPECT\_EQ(a, A.GetOper());

}

TEST(Lexem, can\_parse\_one)

{

string a = "z";

vector<Lexem> A;

EXPECT\_NO\_THROW(A = Parsing(a));

}

TEST(Lexem, can\_parse\_more\_then\_one)

{

string a = "z+z+2+(4+5)";

vector<Lexem> A;

EXPECT\_NO\_THROW(A = Parsing(a));

}

TEST(Lexem, can\_lexeme\_value)

{

string a = "z";

vector<Lexem> A;

A = Parsing(a);

EXPECT\_EQ(A[0].IsNum(), true);

}

TEST(Lexem, unary\_minus)

{

string a = "-(12+2)";

vector<Lexem> A;

double a1;

A = Parsing(a);

A = Polish(A);

a1 = Calc(A);

EXPECT\_EQ(a1,-14);

}

TEST(Lexem, parsing\_acting1)

{

string a = "&^$#%0+2+4";

vector<Lexem> A;

A = Parsing(a);

EXPECT\_EQ(A[0].GetNum(), 0);

}

TEST(Lexem, parsing\_acting2)

{

string a = "4+2+4";

vector<Lexem> A;

A = Parsing(a);

EXPECT\_EQ(A[1].IsNum(), false);

}

//polish block

TEST(Lexem, do\_polish1)

{

string a = "4+2+4";

vector<Lexem> A;

double a1, b1=10;

A = Parsing(a);

A= Polish(A);

a1 = Calc(A);

EXPECT\_EQ(a1,b1);

}

TEST(Lexem, do\_polish2)

{

string a = "4\*2+4";

string b = "4\*(2+4)";

vector<Lexem> A,B;

double a1, b1;

A = Parsing(a);

A = Polish(A);

B = Parsing(b);

B = Polish(B);

a1 = Calc(A);

b1 = Calc(B);

EXPECT\_NE(a1, b1);

}

TEST(Lexem, do\_polish3)

{

string a = "f\*2+4+f";

vector<Lexem> A;

double a1, b1=10;

A = Parsing(a);

A = PutVal(2,A);

A = Polish(A);

a1 = Calc(A);

EXPECT\_EQ(a1, b1);

}

TEST(Lexem, division)

{

string a = "4+5/0";

vector<Lexem> A;

double a1;

A = Parsing(a);

A = PutVal(2, A);

A = Polish(A);

EXPECT\_ANY\_THROW(a1 = Calc(A));

}

TEST(Lexem, megaminus)

{

string a = "1--------1";

vector<Lexem> A;

double a1;

A = Parsing(a);

A = Polish(A);

a1 = Calc(A);

EXPECT\_EQ(a1, 2);

}

TEST(Lexem, dbl\_minus\_ex)

{

string a = "--(6+1)";

vector<Lexem> A;

double a1;

A = Parsing(a);

A = Polish(A);

a1 = Calc(A);

EXPECT\_EQ(a1, 7);

}

TEST(Lexem, ordinary\_nums\_with\_minus)

{

string a = "-5+7-2";

vector<Lexem> A;

double a1;

A = Parsing(a);

A = Polish(A);

a1 = Calc(A);

EXPECT\_EQ(a1, 0);

}

//checing block

TEST(string, can\_check\_syntax)

{

string s = "4+++4";

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

}

TEST(string, can\_check\_sk)

{

string s = "(((4+4))";

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

}

TEST(string, can\_check\_value)

{

string s = "a+b+c+d";

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

}

TEST(string, Check\_all)

{

string a = "4+2+()\*4";

EXPECT\_EQ(Checking\_Block(a,11), false);

}