Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

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

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Вычисление арифметических выражений»**

**Выполнил**:

студент группы 381903-2

Киселев Д. Д.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2019

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка 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 и сообщать пользователю вид ошибки и номера символов строки, в которых были найдены ошибки.

# Метод решения

Для решения задачи реализованы классы Stack, Lexem, содержащие методы, необходимые для разбиения исходного арифметического выражения на лексемы, проверки корректности выражения, перевода выражения в постфиксную запись и последующего вычисления результатов.

Класс Stack необходим для создания стеков, используемых в методах класса Lexem для перевода исходного выражения в обратную польскую запись и вычисления полученного выражения.

Класс Lexem содержит методы для вычисления выражения, переведенного в ОПЗ.

Функция priority определяет приоритет выполняемых операций.

Функция PolishRecord осуществляет перевод введенного выражения в ОПЗ.

Функция result принимает выражение в виде ОПЗ и считает результат выражения.

Функция errors проводит проверку выражения на ошибки.

Перевод в ОПЗ:

На вход подаётся массив лексем. Начинаем цикл по массиву и смотрим, если взятая лексема число, то кладем в новый массив лексем, если операция, то смотрим на стек. Если стек пустой, кладём операцию в стек. В стеке уже есть операция – сравниваем приоритеты операций. Если приоритет операции уже лежавшей в стеке больше приоритета поступающей, то достаем ее и кладем в новый массив. После выполнения цикла кладем все операции из стека в конец нового массива.

# Руководство пользователя

При запуске исполняемого файла пользователю предлагается ввести арифметическое выражение для последующего расчета.

Разрешено использование целочисленных и дробных значений, а также символов “+”, ”-”, ”\*”, ”/”, ”(”, ”)”.

После выполнения расчетов в консоли будет отображен результат вычислений. В случае возникновения ошибки будет указан вид ошибки.

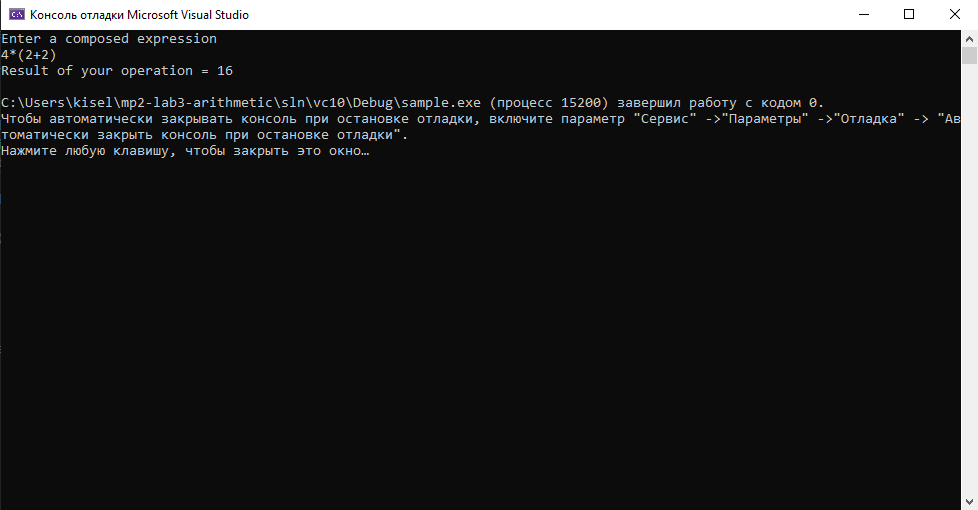


Рис.1 (Результат корректного выполнения программы)

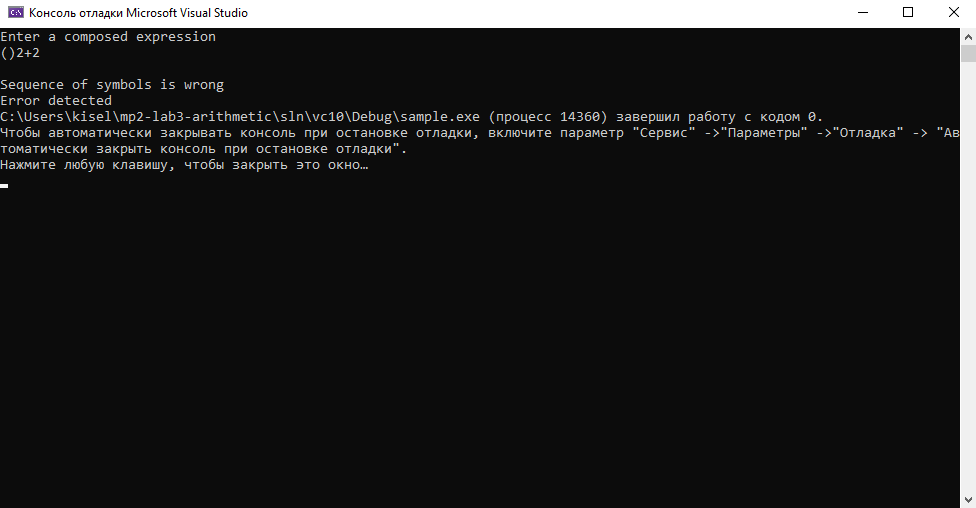


Рис.2 (Результат выполнения программы с синтаксической ошибкой на входе)

# Описание программной реализации

Программа представлена 4 проектами, объединенными в одно решение

1. Arithmetic

* arithmetic.h   
  Объявление класса Lexem и его методов, а так же функций для вычисления и проверки на ошибки исходного арифметического выражения.
* stack.h

Объявление шаблонного класса Stack, реализация его методов.

* arithmetic.cpp

Реализация методов и функций, объявленных в файле arithmetic.h.

1. Gtest

Библиотека Google unit тестов.

1. Sample

Реализация пользовательского интерфейса программы.

1. Tests

* Test\_arithmetic.cpp

Unit тесты класса Lexem.

* Test\_main.cpp

Запуск Unit тестов.

* Test\_stack.cpp

Unit тесты класса Stack.

Стек поддерживает операции:

* Вставка элемента
* Извлечение элемента
* Просмотр верхнего элемента (без удаления)
* Проверка на пустоту
* Получение количества элементов в стеке
* Очистка стека

class Stack:

* Stack(int n1 = 10)
* ~Stack()
* T pop()
* void push(T val)
* bool IsEmpty()
* void Clean()
* T Front()
* int Size()

class Lexem:

* int priority(char operation);
* Lexem\* PolishRecord(string, int&);
* double result(Lexem\*, int);
* bool errors(string);
* Lexem() {};
* ~Lexem() {};
* Lexem(char, int);
* Lexem(double, int);
* void set(char, int);
* void set(double, int);
* int sign();
* double number();
* char operation();

Ошибки, которые могут возникнуть при проверке выражения:

* The symbol is wrong
* Brackets are wrong
* The expression starts with a wrong symbol
* The expression ends with a wrong symbol
* Sequence of symbols is wrong

# Подтверждение корректности

Для подтверждения корректности выполнения программы реализованы Google тесты, проверяющие функции на наличие ошибок

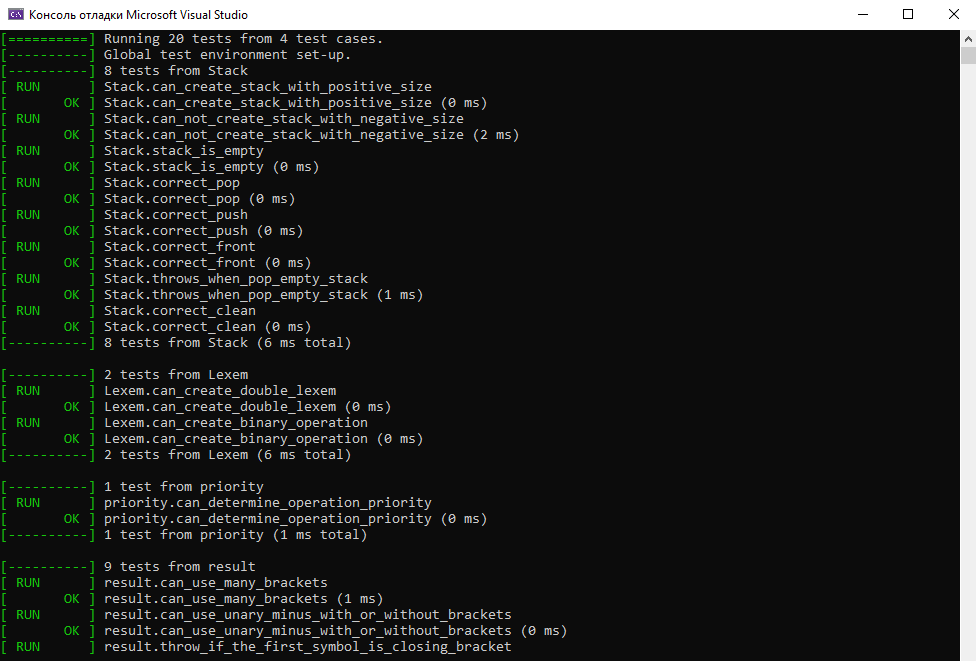


Рис.3 (Подтверждение корректности при помощи тестов)

# Заключение

В ходе лабораторной работы была реализована программа, выполняющая вычисление арифметических выражений с вещественными числами. Программа так же выполняет проверку входных данных на корректность.

Реализованы Google-тесты для проверки работы функций.

# Приложение

Lexem\* PolishRecord(string stroka, int& k)

{

k = 0;

int a = 0;

int b = 0;

int ñ = 0;

double e;

char\* tmp;

Lexem\* x;

x = new Lexem[size(stroka)];

Stack<char> stc(size(stroka));

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

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

b = 0;

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

b++;

tmp = new char[b + 1];

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

tmp[n] = stroka[j];

tmp[b] = { '\0' };

e = atof(tmp);

for (int i = 1; i < ñ; i++)

e = -e;

ñ = 0;

x[k++].set(e, 2);

i += b - 1;

}

else

{

if (a == 0)

{

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

{

ñ = 2;

}

else

{

stc.push(stroka[i]);

a++;

ñ++;

}

}

else

{

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

{

while (stc.Front() != '(')

{

x[k++].set(stc.pop(), 1);

}

stc.pop();

}

else

{

if ((stc.IsEmpty() == false) && ((priority(stc.Front()) < priority(stroka[i])) || (stroka[i] == '('))) {

if ((ñ == 0) || (stroka[i] == '('))

stc.push(stroka[i]);

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

ñ++;

}

else

{

if (ñ == 0) {

while ((stc.IsEmpty() == false) && (stc.Front() != '('))

{

x[k++].set(stc.pop(), 1);

}

stc.push(stroka[i]);

}

else

ñ++;

}

}

}

}

while (stc.IsEmpty() == false)

x[k++].set(stc.pop(), 1);

return x;

}

double result(Lexem\* x, int k)

{

Stack <double> stc(k);

double res;

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

{

if (x[i].sign() == 2)

{

stc.push(x[i].number());

}

else

{

double a = stc.pop();

double b = stc.pop();

switch (x[i].operation())

{

case '+':

res = b + a;

break;

case '-':

res = b - a;

break;

case '\*':

res = b \* a;

break;

case '/':

if (a != 0)

res = b / a;

else throw 1;

break;

}

stc.push(res);

}

}

if (k == 1)

res = stc.pop();

return res;

}

bool errors(string stroka)

{

bool a = true;

int b = 0;

Stack <char> stc(size(stroka));

int t = 0;

char br;

char c[17] = { '0','1','2','3','4','5','6','7','8','9','-','+','\*','/','.','(',')' };

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

{

for (int j = 0; j < 17; j++)

if (stroka[i] == c[j])

{

b++;

}

if (b == 0)

{

cout << endl << "The symbol is wrong";

a = false;

}

b = 0;

}

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

{

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

stc.push('(');

else

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

{

if (stc.IsEmpty() == false)

br = stc.pop();

else

t++;

}

}

if (t != 0)

{

a = false;

cout << endl << "Brackets are wrong";

}

for (int j = 11; j < 15; j++)

if ((stroka[0] == c[j]) || (stroka[0] == ')'))

{

a = false;

cout << endl << "The expression starts with a wrong symbol";

}

for (int j = 10; j < 16; j++)

if (stroka[size(stroka) - 1] == c[j])

{

a = false;

cout << endl << "The expression ends with a wrong symbol";

}

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

{

for (int j = 10; j < 17; j++)

if (stroka[i] == c[j])

if ((i + 1) < size(stroka))

{

for (int l = 11; l < 15; l++)

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

a = false;

cout << endl << "Sequence of symbols is wrong";

}

}

}

for (int i = 1; i < size(stroka); i++)

{

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

{

for (int j = 10; j < 14; j++)

if ((stroka[i - 1] == c[j]) || (stroka[i - 1] == '(')) {

b++;

}

if (b == 0)

{

a = false;

cout << endl << "Sequence of symbols is wrong";

}

}b = 0;

}

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

{

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

{

for (int j = 10; j < 14; j++)

if ((stroka[i + 1] == c[j]) || (stroka[i + 1] == ')'))

{

b++;

}

if (b == 0)

{

a = false;

cout << endl << "Sequence of symbols is wrong";

}

}

b = 0;

}

return a;

}