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 382003-1

Филатов А. А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**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

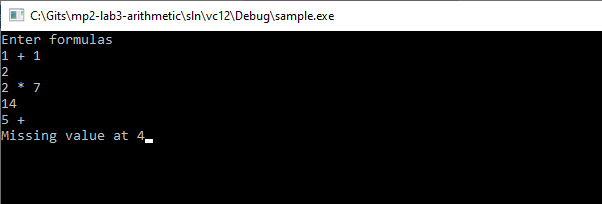
Разработать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение в качестве операндов может содержать переменные и вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Программа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

# Метод решения

В первую очередь программа считывает строку со всем выражением. Затем происходит разбиение на лексемы (операнды, операции или переменные). Каждая из лексем проверяется на недопустимые символы. После этого происходит преобразование из инфиксной записи в постфиксную (для удобности подсчёта). И уже по постфиксной записи происходит расчёт значения арифметического выражения.

# Руководство пользователя

Для пользователя был сделан простейший пользовательский интерфейс, который запускается с помощью файла sample.exe. В него можно вводить корректные арифметические выражения и получать значение, полученное после применения всех операций. В случае введения неверного выражения, программа постарается указать на тип и место ошибки.



# Описание программной реализации

Решение состоит из 4 проектов:

* Gtest - фреймворк Гугл тестов.
* Sample - реализация пользовательского интерфейса.
* Tests - юнит-тесты для программы.
* Arithmetic - основной проект, в котором находится весь рабочий код.

Arithmetic в свою очередь разделён на 3 файла:

1. Stack.h - полная реализация стандартного класса стек.
2. Arithmetic.h - header файл, в котором описаны классы Part и Formula.
3. Arithmetic.cpp - файл с реализацией методов классов Part и Formula.

Класс Part — это одна лексема. В случае если это число, то переменная type равняется v, а значение записывается в value. Если это скобка, то переменная type равняется b, а значение записывается в operation. Если же это операция, то переменная type принимает значение o, значение записывается в operation, а приоритет операции в pr.

Класс Formula состоит из классов Part. В нём важны два метода: конструктор и Solve. Конструктор получает на вход строку, разделяет её и преобразует в постфиксную запись. Метод Solve решает полученное постфиксное выражение и возвращает результат.

# Подтверждение корректности

Подтверждением корректности перевода из инфиксного в постфиксное выражение, является [алгоритм сортировочной станции](https://ru.wikipedia.org/wiki/Алгоритм_сортировочной_станции), предложенный Эдсгером Дейкстрой.   
Решение постфиксной записи основывается на стеке и самой идее обратной польской последовательности.

# Заключение

Были реализованы: стандартный класс стека, разделение строки на лексемы, перевод из инфиксной записи в постфиксную и решение арифметического выражения в обратной польской записи.

# Приложение

Formula::Formula(string& s)

{

stringstream ss;

ss << s;

string s1;

A = new Part[s.size() / 2 + 1];

size\_t l = 0;

TStack<Part> st(5);

char pr = 'n';

size\_t pos = 0;

bool minus = false;

stringstream sserr;

string serr;

while (ss >> s1)

{

try

{

Part p(s1);

}

catch (runtime\_error\* e)

{

sserr << "Wrong symbol from " << pos << " to " << pos + s.size();

getline(sserr, serr);

throw runtime\_error(serr);

}

Part p(s1);

char t = p.Type();

if (t == 'v')

{

if (pr == 'v')

{

sserr << "Missed operator between two values at " << pos - 1;

getline(sserr, serr);

throw runtime\_error(serr);

}

pr = 'v';

if (!minus)

{

A[l] = p;

}

else

{

p.Value() = -p.Value();

A[l] = p;

minus = false;

}

l++;

}

else if (t == 'o')

{

while (!((st.is\_empty()) || ((st.peek().Type() == 'b') && (st.peek().Operation() == "(")) || (p.Pr() > st.peek().Pr())))

{

A[l] = st.peek();

l++;

st.pop();

}

if ((p.Operation() == "-") && ((l == 0) || (pr == 'l')))

{

minus = true;

pos += 2;

continue;

}

if (l == 0)

{

sserr << "Missing value at " << pos;

getline(sserr, serr);

throw runtime\_error(serr);

}

st.push(p);

pr = 'o';

}

else if (t == 'b')

{

if (p.Operation() == "(")

{

if (pr == 'v')

{

sserr << "Missed operator before bracket at " << pos;

getline(sserr, serr);

throw runtime\_error(serr);

}

st.push(p);

pr = 'l';

}

else

{

if (pr = 'o')

{

sserr << "Missing value at " << pos - 1;

getline(sserr, serr);

throw runtime\_error(serr);

}

while (!((st.peek().Type() == 'b') && (st.peek().Operation() == "(")))

{

if (st.is\_empty())

{

sserr << "Missed left bracket. The right bucket was at " << pos;

getline(sserr, serr);

throw runtime\_error(serr);

}

A[l] = st.peek();

l++;

st.pop();

}

st.pop();

pr = 'r';

}

}

pos += 1 + s1.size();

}

if (pr == 'o')

{

sserr << "Missing value at " << pos;

getline(sserr, serr);

throw runtime\_error(serr);

}

while (!st.is\_empty())

{

if ((st.peek().Type() == 'b') && (st.peek().Operation() == "("))

{

sserr << "Missed right bracket";

getline(sserr, serr);

throw runtime\_error(serr);

}

A[l] = st.peek();

l++;

st.pop();

}

size = l;

}

double Formula::Solve()

{

TStack<double> st(size);

double t1, t2;

for (size\_t i = 0; i < size; i++)

{

if (A[i].Type() == 'v')

{

st.push(A[i].Value());

}

else

{

if (A[i].Operation() == "+")

{

t1 = st.peek();

st.pop();

t2 = st.peek();

st.pop();

st.push(t1 + t2);

}

else if (A[i].Operation() == "-")

{

if ((st.is\_empty()) || (A[i - 1].Type() == 'o'))

{

A[i + 1].Value() \*= -1;

}

t1 = st.peek();

st.pop();

t2 = st.peek();

st.pop();

st.push(t2 - t1);

}

else if (A[i].Operation() == "\*")

{

t1 = st.peek();

st.pop();

t2 = st.peek();

st.pop();

st.push(t1 \* t2);

}

else if (A[i].Operation() == "/")

{

t1 = st.peek();

st.pop();

t2 = st.peek();

st.pop();

st.push(t2 / t1);

}

}

}

return st.peek();

}