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

Измайлов И.Р.

Нижний Новгород

2021г.

**Содержание**

[Постановка задачи 3](#_Toc97587518)

[Алгоритм выполнения 4](#_Toc97587519)

[Руководство пользователя 5](#_Toc97587520)

[Описание программной реализации 8](#_Toc97587521)

[Приложение 10](#_Toc97587522)

# Постановка задачи

Передо мной была поставлена задача – повторить весь материал по языку программирования С++, который мы успели пройти на момент окончания 3 семестра. На основе этих знаний написать программу, которая бы реализовывала вычисление арифметических выражений.

Арифметическое выражение – выражение, в котором операндами являются объекты, над которыми выполняются арифметические операции.

Например (1+10)\*(100-45+78)/45 – инфиксная запись

Существует так называемая обратная польская форма записи, не содержащая скобок. Операторы следуют после собственных операндов. Для данного примера постфиксная запись будет иметь следующий вид:

1 10 + 100 45 – 78 + \* 45 /

Допустимые операции: +, -, /, \*. Допускается наличие унарного минуса.

И для вычисления арифметических выражений нам необходимо преобразовать инфиксную запись в постфиксную.

Предполагается, что вычисления, преобразования будут производиться с использованием структуры данных Стек.

Стек – структура данных, представляющий из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. Говорят, что стек устроен по принципу FILO(first in - last out, «первым пришел – последним вышел»).

# Алгоритм выполнения

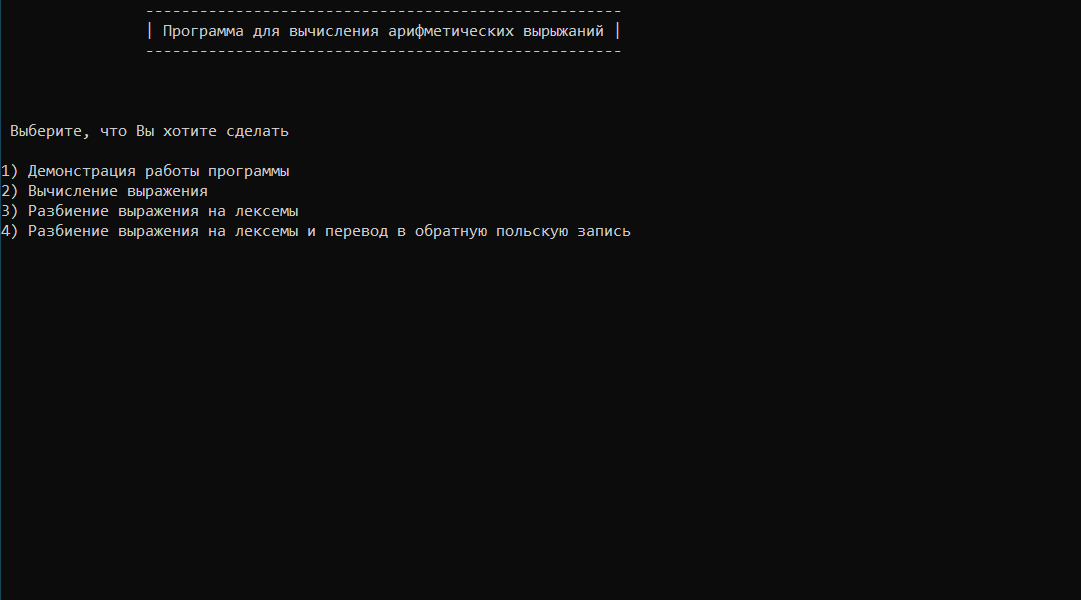
1. Разбиение исходного арифметического выражения на лексемы (т.е. выделить операнды, операции, переменные)
2. Проверка корректности выражения:

* правильность расстановки скобок (с использованием стека (TStack) для хранения индексов скобок).
* пропущены ли операнды или знаки операций
* недопустимые символы

1. Перевод выражения в постфиксную (польскую) запись
2. Вычисление выражения по постфиксной записи

# Руководство пользователя

Пользователь на стартовой странице выбирает, что необходимо сделать.

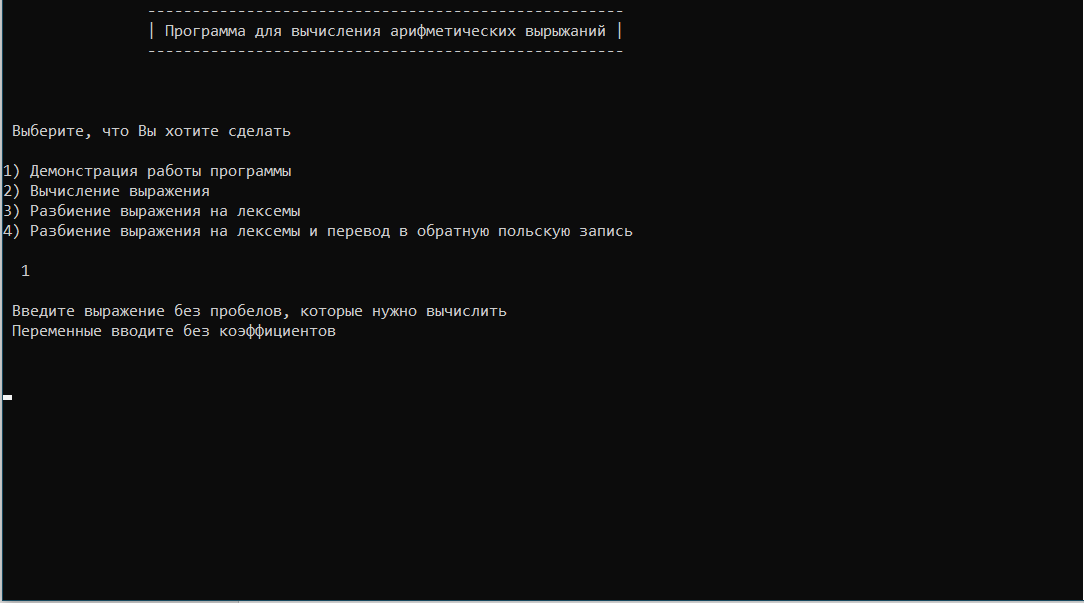


Программа может продемонстрировать по шагам порядок вычисления арифметического выражения. Для выбора этой функции пользователь вводит 1. Если необходимо, чтобы программа выводила только результат вычисления, то для этого вводится 2.

Для разбиения выражения на отдельные лексемы пользователю необходимо ввести 3.

Если нужно разбить выражения на отдельные лексемы и перевести ее в обратную польскую запись, то вводится 4.

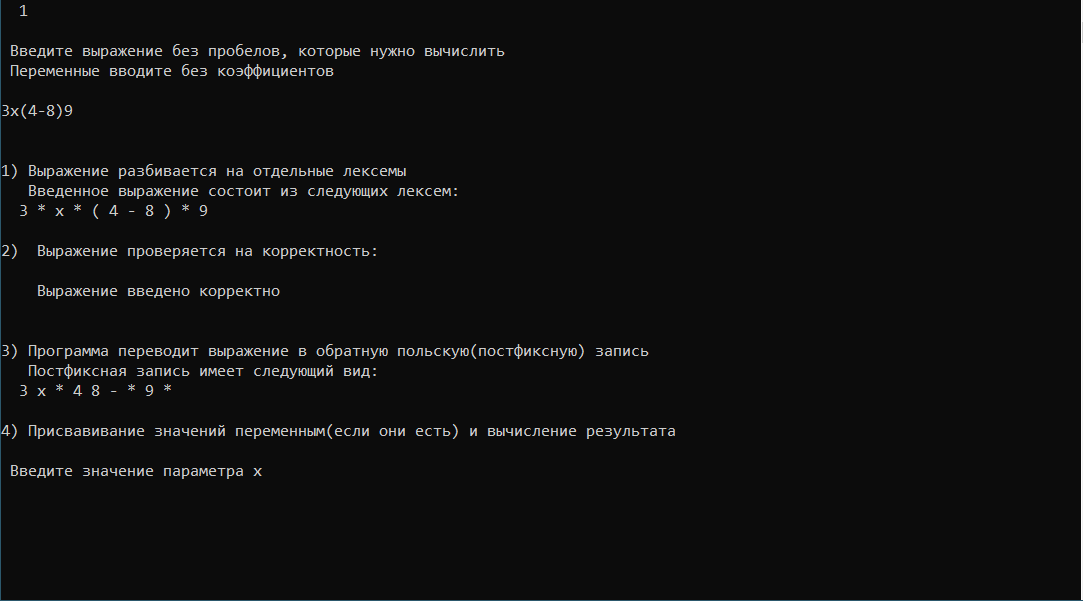
Для наглядности, продемонстрируем, каким образом программа вычисляет результат выражения.



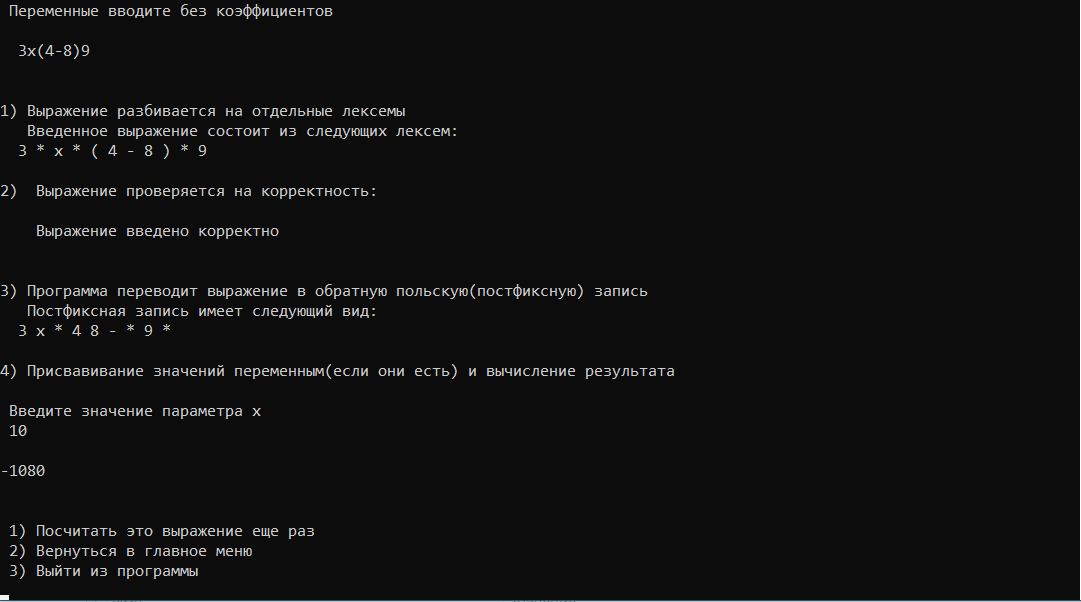
Первым шагом нам нужно ввести выражение:

Допускается вариант не ставить знак «умножение» перед скобкой и после нее. Разрешимый вариант:

3x(4-8)9 будет автоматически представлен в виде 3\*x\*(4-8)\*9



Так как мы ввели параметр x, программа просит присвоить её значение.

Присвоим “x” значение 10. После присвоения значения переменной, программа вычислит результат и выведет на экран.

После вычисления результата пользователь может посчитать это выражение еще раз, подставив новые значения переменным. Для этого необходимо выбрать цифру 1.

Можно вернуться в главное меню- цифра 2.

Для выхода из программы нужно ввести 3.

# Описание программной реализации

Для хранения лексем используется класс “lexeme”, которая может хранить значение(какое то число) или операцию(+,-,/,\*, унарный минус). Если isV = true, то это означает, что в объекте класса lexeme хранится какое-то значение. Если isV= false, то, соответственно – операнд.

**Подтверждение корректности**

Проверка корректности алгоритма осуществляется с помощью gtests.

# Приложение

vector<lexeme> arithmetic:: polish\_notation(vector<lexeme> array)

{

vector<lexeme> array2;

Stack<lexeme> st;

int l = 0;

for (int k = 0; k < array.size(); k++)

{

if (array[k].isValue() || array[k].isParameter())

{

array2.push\_back(array[k]);

l++;

}

else

{

if (st.isEmpty())

{

st.push(array[k]);

}

else

{

if (rang[array[k].getOp()] == 1)

{

st.push(array[k]);

}

else

{

while (!st.isEmpty() && rang[array[k].getOp()] <= rang[st.peek().getOp()])

{

if (rang[st.peek().getOp()] == 1)

{

st.pop();

break;

}

else

{

array2.push\_back(st.pop());

l++;

}

}

if (rang[array[k].getOp()] != 0)

{

st.push(array[k]);

}

}

}

}

}

while (!st.isEmpty())

{

array2.push\_back(st.pop());

l++;

}

return array2;

}

double arithmetic::calculate(vector<lexeme> arrt)

{

Stack<lexeme> s;

lexeme t;

double x1, x2;

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

{

if (arrt[i].isValue() || arrt[i].isParameter()) {

if (arrt[i].isValue())

{

s.push(arrt[i]);

}

if (arrt[i].isParameter())

{

cout << " Введите значение параметра " << arrt[i].getParameter() << "\n" << endl;

cin >> x1;

cout << "\n";

s.push(x1);

}

}

else

{

if (arrt[i].getOp() == "+")

{

x2 = s.pop().getValue();

x1 = s.pop().getValue();

t = x1 + x2;

}

if (arrt[i].getOp() == "-")

{

x2 = s.pop().getValue();

x1 = s.pop().getValue();

t = x1 - x2;

}

if (arrt[i].getOp() == "\*")

{

x2 = s.pop().getValue();

x1 = s.pop().getValue();

t = x1 \* x2;

}

if (arrt[i].getOp() == "/")

{

x2 = s.pop().getValue();

x1 = s.pop().getValue();

t = x1 / x2;

}

if (arrt[i].getOp() == "un-")

{

x2 = s.pop().getValue();

t = -1 \* x2;

}

s.push(t);

}

}

return s.pop().getValue();

}