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»**

**Выполнила**:

студентка группы 382003-2

Капустина Полина Дмитриевна

**Проверила**:

ассистентка каф. МОСТ,

Арисова Анастасия Николаевна

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя](#_Toc26962564) 6

[Описание программной реализации](#_Toc26962565) 7

[Подтверждение корректности](#_Toc26962566) 10

[Заключение](#_Toc26962568) 11

[Приложение 1](#_Toc26962569)2

# 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

Были реализованы шаблонный класс TStack, структура Lexem, класс PolishNotation.

TStack необходим для создания стеков, используемых в методах класса PolishNotation для перевода в обратную польскую запись и вычисление записанного в такой форме выражения.

Lexem применяется для хранения типа лексемы(операнд или операция), её приоритета, и того что в ней содержится (вещественное число, операция или переменная).

В PolishNotation реализованы методы переводящие в обратную польскую запись, вычисляющие значение выражения и проверяющие на ошибки:

Метод TranslationToPolishNotation переводит выражение в обратную польскую запись, внутри него вызывается метод CorrectionChecker, который проходясь с помощью цикла по введённой пользователем строке бросает исключение, если найден не корректный символ, либо если не корректно расставлены скобки. Допустимые символы: “+”, “-”, “\*”, “/”, “.”, “(”, “)” и цифры от 0 до 9. Если проверка не пройдена бросается исключение. После выражения разбивается на вектор лексем, отдельно хранятся операции и операнды. Приоритет присваивается на этом этапе по принципу: если лексема операнд или закрывающая скобка, то приоритет - 0, если лексема открывающая скобка - приоритет - 1, если лексема “+” или бинарный “-”, то приоритет - 2, если лексема “/” или “\*”, то приоритет - 3, если лексема унарный “-”, то приоритет - 4. После разбиения происходит проход по массиву лексем с помощью цикла, где значение каждой лексемы проверяется на некорректную расстановку точек (на случай если пользователь пропустил операцию между вещественными числами или случайно напечатал лишнюю точку). Если проверка не пройдена бросается исключение. Если проверка пройдена, создаем стек лексем, проходимся по массиву лексем с помощью цикла, переводим в постфиксную форму по следующему алгоритму:

1. Если встретили операнд записываем его в исходный вектор.
2. Если встретилась открывающая скобка, кладем ее в стек .
3. Если встретилась операция (кроме скобок) и стек пуст, кладем её в стек. Если стек не пуст, то извлекаем из него все операции, которые не ниже её по приоритету, записываем их в исходный вектор, а операцию кладем в стек.
4. Если встретилась закрывающая скобка, извлекаем из стека все операции до момента, когда встретится открывающая скобка, записывая их в исходный вектор, саму открывающую скобку извлекаем из стека никуда не записывая.
5. При достижении конца строки, если стек не пуст записываем из него все операции в исходный вектор.

Метод PolishNotationCalculate производит вычисление выражения в обратной польской записи.

Внутри создается стек действительных чисел, затем с помощью цикла проходим по вектору лексем:

1. Если лексема - операнд, кладем ее в стэк.
2. Если лексема операция - извлекаем из стэка необходимое для неё кличество операндов и кладём результат в стэк. Если в стэке не содержится необходимого количества операндов бросается исключение, так как пропущен операнд.
3. В конце цикла проверяем сколько элементов в стеке, если больше 1 элемента бросается исключение, так как пропущена операция. Если в стеке 1 элемент, то это и есть результат вычисления выражения.

# Руководство пользователя

Пользователь вводит выражение по следующим правилам:

a. Не допускается использование других символов, кроме «1234567890.+-\*/xyz»

b. Скобки должны быть расставлены по правилам математики

c. Нельзя ставить знак операции после знака операции, т.е. «1+/2», исключение составляет унарный минус

• Программа выдает результат в консоль. В случае нарушений правил ввода выдает причину ошибки.

# Описание программной реализации

В реализации используются 4 проекта:

1. **sample**

В нем находится файл main\_arithmetic.cpp, в нем находится функция main, в которой реализован пользовательский интерфейс программы.

1. **arithmetic**

Содержит файлы arithmetic.cpp, arithmetic.h, stack.h.

1. stack.h содержит интерфейс и реализацию методов класса TStack.

Поля:

size -- хранит размер стэка;

head -- хранит индекс вершины стэка;

stack -- динамический массив элементов, которые содержатся в стэке.

Имеет конструктор, создающий стек заднного размера и деструктор, очищающий память, выделенную под массив stack.

Методы:

IsEmpty -- проверка стэка на пустоту, возвращает true если стек пуст;

IsFull -- проверка стека на полноту, возвращает true если стек полон;

Push - метод, позволяющий добавить элемент в стек, если стек полный перевыделяет память под массив stack;

Pop - метод, позволяющий извлечь элемент из стека, возвращает этот элемент;

GetHeadElement -- позволяет просмотреть верхний элемент стека, не извлекая его, возвращает этот элемент;

Get -- возвращает количество элементов в стеке;

Clear - очщает стек.

1. arithmetic.h содержит интерфейс класса PolishNotation, а также струкутуру Lexem.

Поля:

priority -- хранит приоритет той или иной лексемы;

type -- хранит тип лексемы(операнд или операция);

lexm -- хранит саму лексему.

1. arithmetic.cpp содержит реализацию методов класса PolishNotation.

Поля:

formula -- строка, хранит введенное пользователем выражение;

l -- вектор, хранит разбитое на лексемы выражение;

polish -- хранит выражение, переведенное в обратную польскую запись.

Имеет конструктор, присваивающий полю formula веденную пользователем строку.

Методы:

CorrectionChecker -- метод, проверяющий введенную пользователем строку на предмет недопустимых символов и скобок;

TranslationToPolishNotation -- сначала проверяет выражение введенное пользователем, хранящееся в поле formula на коректность, вызвав метод CorrectionChecker, затем если обнаружил символы, соответствующие переменным просит пользователя ввести их значение, после разбивает выражение на лексемы, проверяет лексемы на ошибки связанные с неправильной расстановкой точек, после переводит его в обратную польскую запись;

PrintPolishNotation -- выводит на экран выражение, переведенное в обратную польскую запись;

PolishNotationCalculate -- вычисляет выражение записанное в обратной польской записи, проверяет выражение на пропущенные операнды и операции, возвращает результат;

BinaryCalculate, метод который применяется если при вычислении выражения в обратной польской записи для бинарной операции хватает операндов, принимает на вход два вещественных числа и строку - операнды и операцию соответственно, производит действие соответствующее операции над операндами и возвращает результат.

1. **gtest**

**4. test**

Содержит файлы с реализацией Google-тестов: test\_stack.cpp, test\_arithmetic.cpp и test\_main.cpp.

1. test\_stack.cpp содержит тесты функций класса TStack.
2. test\_arithmetic.cpp содержит тесты, проверяющие корректность программы, переводящей арифметическое выражение в обратную польскую запись и вычисляющей результат этого выражения.
3. test\_main.cpp запускает все Google-тесты.

# Подтверждение корректности

Для подтверждения корректности были реализованы Google-тесты для классов TStack и PolishNotation.

# Заключение

В ходе работы реализована программа, переводящая введённое пользователем арифметическое выражение в обратную польскую запись, производящая вычисление этого выражения проверяющая введённое выражение на корректность.

# Приложение

class PolishNotation

{

private:

string formula;

vector <Lexem> polish;

vector <Lexem> l;

public:

PolishNotation(string s);

void CorrectionChecker(string s);

double BinaryCalculate(double a, double b, string s);

void TranslationToPolishNotation();

void PrintPolishNotation();

double PolishNotationCalculate();

};

double PolishNotation::PolishNotationCalculate()

{

double result;

TStack <double> st(polish.size());

unsigned int i = 0;

while (i < polish.size())

{

if (polish[i].type == "operand")

{

double k;

stringstream ss(polish[i].lexm);

ss >> k;

st.Push(k);

}

if ((polish[i].lexm == "-") && (polish[i].priority == 4))

{

if (st.IsEmpty())

{

throw "The problem is with the number of unary operators.";

}

double a = st.Pop();

double res = -a;

st.Push(res);

}

if ((polish[i].lexm == "+") || (polish[i].lexm == "\*") || (polish[i].lexm == "/"))

{

if (st.Get() < 2)

{

throw "The problem is with the number of unary operators.";

}

double b = st.Pop();

double a = st.Pop();

double res = BinaryCalculate(a, b, polish[i].lexm);

st.Push(res);

}

if ((polish[i].lexm == "-") && (polish[i].priority == 2))

{

if (st.Get() < 2)

{

throw "The problem is with the number of unary operators.";

}

double b = st.Pop();

double a = st.Pop();

double res = BinaryCalculate(a, b, polish[i].lexm);

st.Push(res);

}

i++;

}

if (st.Get() > 1)

{

throw "Missing operation.";

}

result = st.GetHeadElement();

return result;

}