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/ка группы 3821Б1ПМ2

Логинов С.С

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Оглавление**

[**Постановка задачи** 3](#_Toc122980560)

[**Руководство пользователя** 4](#_Toc122980561)

[**Руководство программиста.** 6](#_Toc122980562)

[**1.Описание структуры программы** 6](#_Toc122980563)

[**2. Описание алгоритмов.** 11](#_Toc122980564)

[**2.1 Разбиение на лексемы.** 11](#_Toc122980565)

[**2.2 Проверка на корректность следования лексем** 11](#_Toc122980566)

[**2.3 Перевод в постфиксную форму** 12](#_Toc122980567)

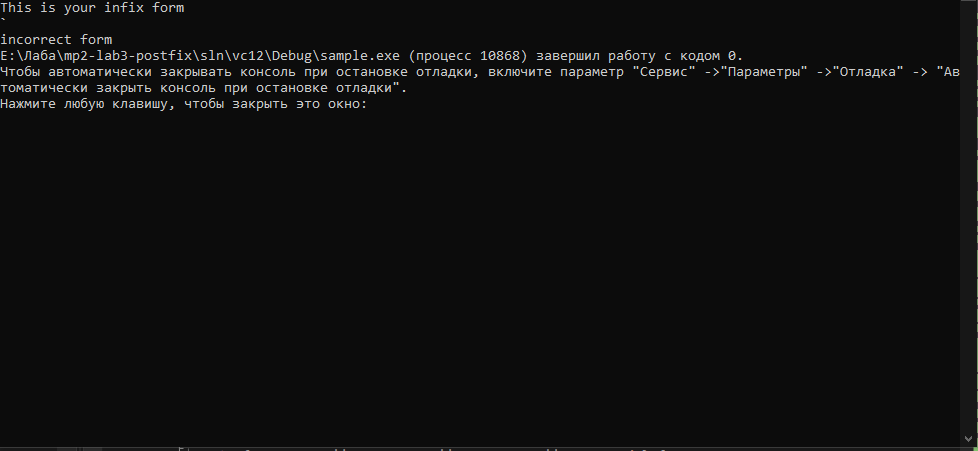
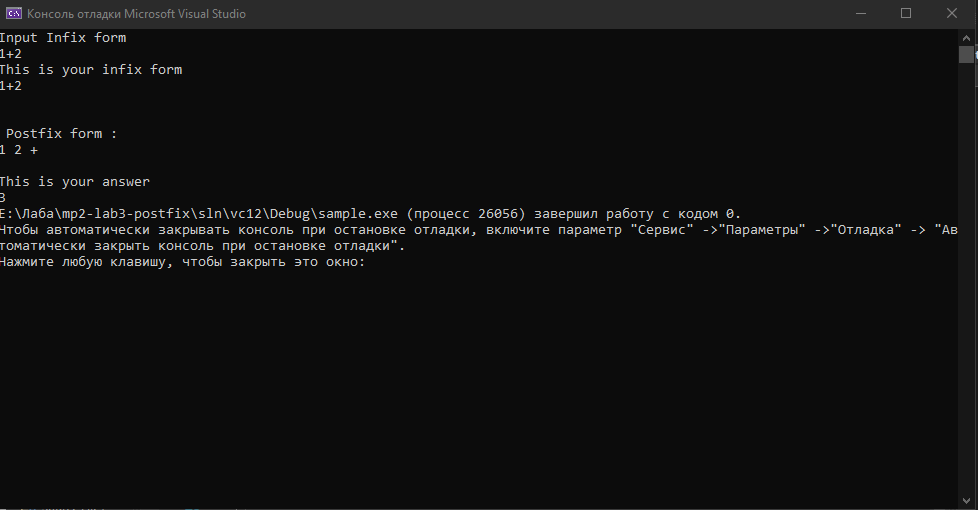
[**2.4 Вычисление арифметического выражения в постфиксной форме** 12](#_Toc122980568)

# **Постановка задачи**

Цель данной лабораторной работы - расчёт арифметических выражений. Арифметические выражение могут содержать переменные и константы. Для решение данной задачи будут реализованы:

* Алгоритм перевода в обратную польскую запись(RPN – Reverse Polish Notation).
* Стек, используемый как структура данных.
* Тесты для стека на основе Google Test
* Тесты для класса арифметических выражений на основе Google Test

# **Руководство пользователя**

1. В начале работы программа попросит пользователя ввести инфиксную форму числа(Input Infix form)
2. Если пользователь ввел неправильную инфиксную форму(форма, не являющаяся операцией,операндом,числом,константой,переменной),то пользователь увидит ошибку, гласящую “Некорректная форма”(Incorrect form)
3. Если пользователь ввел корректную инфиксную форму, то программа выведет сначала введенную пользователем инфиксную форму (This is your infix form). Затем программа выведет постфиксную форму (Postfix form). В конце, программа выведет результат вычислений (This is your answer)

# **Руководство программиста.**

## **1.Описание структуры программы**

Программа содержит два класса Stack и Arithmetic.

Stack

Шаблонный класс стек, при создании его экземпляра будет создан динамический массив T\* data на 2 элемента. Класс Stack также имеет private поля int index = -1 для хранения индекса текущего элемента и int real\_size = 2 хранения размера стека. Для работы со стеком реализованы следующие методы:

void Clear() – необходим для очистки stack(устанавливается индекс = -1)

bool IsEmpty() – Проверка на пустоту. Вернет true, несли стек пуст. Иначе вернет false.

void Push(T elem) – добавляет элемент на вершину стека. Перевыделяется память, если стек полный

T Pop() – извлекает верхний элемент из стека.

int GetSize() - возвращает количества элементов в стеке.

T Top() - просмотривает первый элемент в стеке.

**Класс Lex**

std::string name – Хранит в себе тип лексемы.

std::string lexem –Хранит в себе лексемы.

double double\_spot – Хранит в себе численное представление.

int pr – Хранит в себе приоритет.

std::string LexType() – возвращает наименование лексемы.

std::string GetLex() – возвращает саму лексему.

double value() – возвращает численное представление.

int GetPrior() – возвращает приоритет.

**Класс operations**

конструктор operations(char op) - задаёт переменной name значение “binary”, “unary” или “brackets”

Void SetPrior(char op) override - Задает приоритет в зависимости от операции.

**Класс operands**

double NumConv(std::string strlex) который преобразует строку в число и бросает исключение в случае некорректной записи.

void SetPrior(char op) - устанавливает значение переменной pr значение 2

**Класс Num**

- Num(std::string lex) - который задаёт переменной name значение “digit”, устанавливает значение переменной pr 2, задаёт lexem = lex, double\_spot = NumConv(lex).

- Num(double value) - который задаёт переменной name значение “digit”, устанавливает значение переменной pr 2, задаёт double\_spot = value.

- Num() - который задаёт переменной name значение “digit”, задаёт double\_spot =0. .

**Класс Var**

- Var(std::string lex); который задаёт переменной name значение “Var”, устанавливает значение переменной pr 2, и задаёт lexem = lex .

**Класс Arithmetic**

std::string infix – используется для хранения арифметического выражения в формате строки.

Lex\*\* lexem - Арифметическое выражение хранится в инфиксной форме в виде указателей на лексемы

Lex\*\* postfix - Арифметическое выражение хранится в постфиксной форме в виде указателей на лексемы

int size – Выделяемая память для lexem и postfix(хранение)

int lex\_size – Количество элементов в лексеме(хранение)

int postfix\_size - Хранит индекс последнего элемента в postfix

**Методы класса Arithmetic**

bool IsOperation(char v) – проверка, является ли символ операцией из перечисленных далее ('~' || '-' || '+' || '/' || '\*' || '^' || '(' || ')') (|| здесь является разделителем; операцией не является)

bool CheckOp() - проверка на корректность операций.

Bool IsOperand (const char& lexem) - проверка, является ли символ операндом

bool IsDigit(const char& ch) – проверяет является ли символ числом от 0 до 9, или точкой или экспонентой

bool IsNumber(std::string num) – проверяет является ли строка числовой константой.

bool IsVar(const char& ch) – проверяет является ли символ переменной.

bool IsVariable(std::string num) – проверяет является ли строка переменной.

Void CheckBrackets() - проверяет корректность расставленных скобок. Соответственно количество скобок.

Void IsBrackets(const char& ch) - проверяет, является ли символ скобкой.

Void CorrectOrder() - проверяет порядок лексем.

void resize() – перевыделяет память для lexem.

void IncorrectSymbols() – бросает исключение если в арифметическом выражении встречаются недопустимые символы.

void VarValue() – ввод значений переменных.

void Postfix() – превод арифметического выражения в постфиксную форму.

void Parser() – арифметическое выражение разбивается на лексемы.

Arithmetic(std::string arithmetic) - Конструктор арифметического выражения.Кидает исключение при пробеле. При повторении минусов заменяет первый минус на унарный. Если i-1 символ бинарная операция или ~ то i символ заменяется на ~. Запускает проверки IncorrectSymbols(), CheckBrackets(), Parser().

Double Calculate() - Проводит операции над постфиксом.

~Arithmetic() - Деструктор.

# **2. Описание алгоритмов.**

## **2.1 Разбиение на лексемы.**

* Создаётся пустая строка для хранения переменной или числа
* Проверяем наличие пробелов.
* Присваеваем размер size
* Выделяем память на lexem и создаем динамический массив указателей
* Все унарные минусы заменяются на ~
* Осуществляем проверки в цикле.
* Проверка, является ли символ бинарной операцией (+,-,\*,/)
* Проверка, является ли символ унарной операцией (~)
* Проверка, является ли символ скобкой ( )
* Создаем указатель на symbols.

## **2.2 Проверка на корректность следования лексем**

После разбиения на лексемы проверяется корректность следования. Она в себя включает:

* Перед открывающей скобкой не может стоять число, переменная или закрывающая скобра
* Перед унарной операцией не может стоять число, переменная или закрывающая скобра
* Перед числом либо переменной не может стоять закрывающая скобка
* Перед бинарной операцией не может стоять число,переменная и закрывающая скобка
* Перед закрывающей скобкой не может стоять число,переменная и закрывающая скобка
* Если лексема начинается с закрывающей скобки или бинарной операции, то бросаем ошибку
* Если предпоследний элемент это открывающая скобка, либо бинарная операция либо унарная операция, то бросаем ошибку

### **2.3 Перевод в постфиксную форму**

Используется стек, хранящий операнды.

Проходим по всем лексемам и выполняем:

* Если текущая лексема - скобка:
* Если открывающая скобка, то добавляем в стек.
* Если закрывающая скобка, то добавляем из стека операции в постфиксную форму до тех пор, пока не встретим открывающую скобку, которую изымаем из стека без добавления в постфиксную форму.
* Если текущая лексема – бинарная операция:
* Добавляем из стека операции в постфиксную форму до тех пор, пока не встретим операцию с меньшим приоритетом. Добавляем в стек текущую операцию.
* Если текущая операция – унарный -, то добавляем его в стек.
* Если текущая лексема число или переменная, то добавляем её в постфиксную форму.

### **2.4 Вычисление арифметического выражения в постфиксной форме**

Перед вычислением арифметического выражения необходимо ввести значения переменных и заменить их на числа.

Заводим стек для чисел и идем циклом по всем лексемам из постфиксной формы.

- Если текущая лексема - число, то добавляем её в стек.

- Если текущая лексема – унарная операция, то изымаем из стека число, применяем унарную операцию и добавляем результат в стек

- Если текущая лексема – бинарная операция, то изымаем из стека сначала правый операнд, затем левый. Применяем бинарную операцию к правому и левому операнду( они могут меняться местами, в зависимости от операции) и добавляем полученный результат в стек.

После прохода по циклу в стеке будет находиться одно число – результат арифметического выражения.