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

Василевский А.П.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc122791657)

[Руководство пользователя. 4](#_Toc122791658)

[Руководство программиста. 7](#_Toc122791659)

[1.Описание структуры программы 7](#_Toc122791660)

[Stack 7](#_Toc122791661)

[Arithmetic 7](#_Toc122791662)

[2.Описание алгоритмов 11](#_Toc122791663)

[2.1 Разбиение на лексемы 11](#_Toc122791664)

[2.2 Проверка порядка следования лексем 11](#_Toc122791665)

[2.3 Перевод в постфиксную форму 12](#_Toc122791666)

[2.4 Вычисление арифметического выражения в постфиксной форме 13](#_Toc122791667)

[Результаты экспериментов 14](#_Toc122791668)

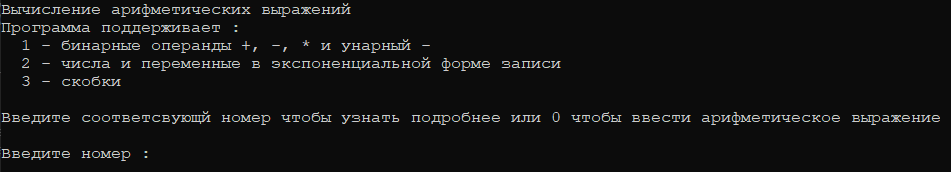
[Заключение 15](#_Toc122791669)

# Постановка задачи

Цель данной лабораторной работы - расчёт арифметических выражений с переменными и константами. Для данной цели будут реализованы алгоритм перевода в обратную польскую запись, структура данных стек, тесты для стека и класса арифметических выражений на базе Google Test.

# Руководство пользователя.

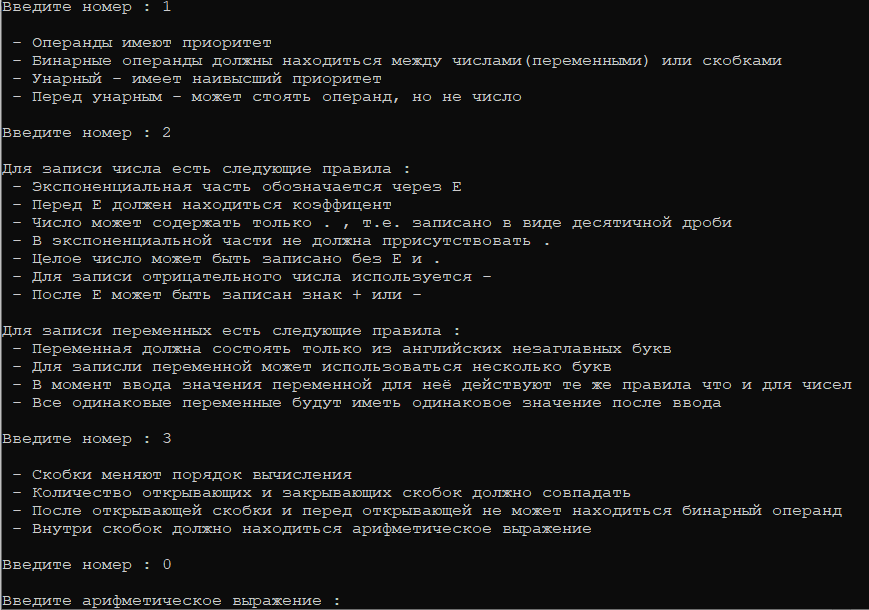
В начале работы программа выведет список поддерживаемых лексем и будет ожидать ввод пользователем числа 0 – для ввода арифметического выражения или для получения подробной информации о операциях – 1, числах и константах – 2, скобках – 3.



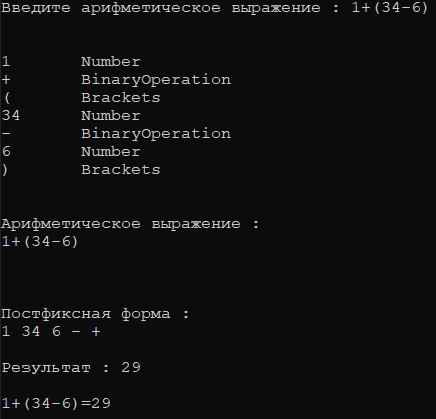
При некорректном вводе программа будет вновь запрашивать выбор пользователя.



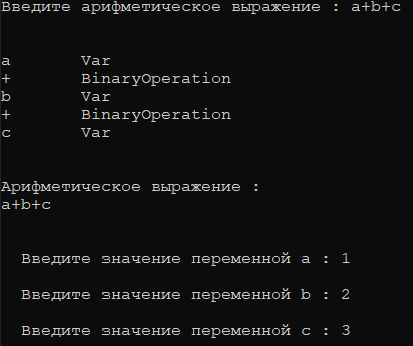
При корректно введённом выборе программа выведет информацию и будет ожидать дальнейшего выбора (0 для ввода арифметического выражения).



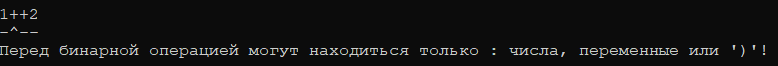
После ввода арифметического выражения программа разобьёт его на лексемы и вычислит.

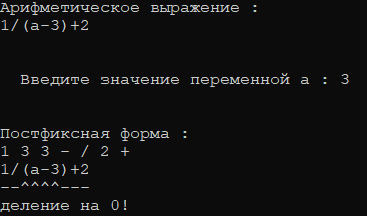


Если будут использоваться переменные после разбиения на лексемы программа запросит значения переменных.



Если в арифметическом выражении будет допущена ошибка или произойдёт деление на 0, то программа уведомит пользователя об этом и завершит работу.





# 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() – очищает стек (устанавливает индекс -1)

bool IsEmpty() – возвращает true если стек пуст и false в противном случае.

void Push(T elem) – добавляет элемент на вершину стека. Если стек полный, то перевыделяет память.

T Pop() – извлекает верхний элемент из стека.

int GetSize() - возвращает количества элементов в стеке

T Top() - возвращает верхний элемента (без удаления)

### Arithmetic

В классе Arithmetic содержаться следующие private поля:

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

BaseLexem\*\* lexems - используется для хранения арифметического выражения в инфиксной форме в виде указателей на лексемы.

BaseLexem\*\* postfix - используется для хранения арифметического выражения в постфиксной форме в виде указателей на лексемы.

int size – используется для хранения количества выделенной памяти в lexems и postfix

int last – используется для хранения индекса последнего элемента в lexems

int postfix\_size используется для хранения индекса последнего элемента в postfix

Также Arithmetic содержит класс BaseLexem и его наследников Operation для операций и Operand от которого наследуются Number для чисел, Var для переменных и.

Класс BaseLexem имеет private поля и public методы для получения их значения:

std::string name – используется для хранения типа.

std::string lexem – используется для хранения лексемы.

double double\_performance – используется для хранения численного представления.

int priority – используется для хранения приоритета.

int pos\_start – используется для хранения позиции начала лексемы.

int pos\_end – используется для хранения позиции конца лексемы.

std::string LexemType() – возвращает name.

std::string GetLexem() – возвращает lexem.

double Value() – возвращает double\_performance.

int GetSatrtPos() – возвращает pos\_start.

.int GetEndPos() – возвращает pos\_end.

int GetPriority() – возвращает priority.

Класс Operation имеет конструктор Operation(char data, int pos1, int pos2) который задаёт переменной name значение “BinaryOperation”, “UnaryOperation” или “Brackets” и устанавливает значение переменной priority от 0 до 3 в зависимости от того какой операцией является data, и задаёт pos\_start = pos1, pos\_end = pos2 и lexem = data.

Класс Operand имеет метод double ConvertNumber(std::string strlex) который преобразует строку в число и бросает исключение в случае некорректной записи.

Класс Number имеет 3 конструктора

- Number(std::string lexem, int pos1, int pos2) который задаёт переменной name значение “Number”, устанавливает значение переменной priority 4, и задаёт pos\_start = pos1, pos\_end = pos2, this -> lexem = lexem, double\_performance = ConvertNumber(lexem).

- Number(double value, int pos1, int pos2) который задаёт переменной name значение “Number”, устанавливает значение переменной priority 4, и задаёт pos\_start = pos1, pos\_end = pos2, double\_performance = value.

- Number() – конструктор по умолчанию (используется при создании стека), который задаёт переменной name значение “Number”, устанавливает значение переменной priority 4, и задаёт pos\_start = 0, pos\_end = 0, double\_performance = 0.

Класс Var имеет только конструктор

- Number(std::string lexem, int pos1, int pos2) который задаёт переменной name значение “Var”, устанавливает значение переменной priority 4, и задаёт pos\_start = pos1, pos\_end = pos2, this -> lexem = lexem.

В классе Arithmetic содержаться следующие private методы:

bool IsOperation(const char& ch) – проверяет является ли символ операцией или скобкой.

bool IsNumber(const char& ch) – проверяет является ли символ числовой константой.

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

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

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

void resize() – перевыделяет память для lexems.

std::string ShowError(int index, std::string text) – создаёт строку с указателем на позицию ошибки index и сообщением text.

std::string ShowError(int index1, int index2, std::string text) – создаёт строку с указателем на интервал ошибки начиная с index1 до index2 и сообщением text.

void FindIncorrectSymbols() – бросает исключение если в арифметическом выражении используются недопустимые символы.

void CheckLexemOrder() – проверяет порядок следования лексем.

void InputVarValue() – используется для ввода значений переменных.

void Postfix() – преводит арифметическое выражение в постфиксную форму.

void Parser() – разбивает арифметическое выражение на лексемы.

В классе Arithmetic содержаться следующие public методы:

Arithmetic(const std::string& arifmetic) – конструктор класса Arithmetic. Задаёт значение infix (при этом удаляет пробелы) и вызывает методы FindIncorrectSymbols(), Parser(), CheckLexemOrder(). Бросает исключение если нарушен порядок лексем или присутствует недопустимый символ.

double Calculate() – метод, возвращающий результат арифметического выражения. Вызывает методы InputVarValue(), Postfix() после чего производит необходимые вычисления. Может бросить ошибку, если значение переменной будет введено некорректно.

## 2.Описание алгоритмов

Для класса арифметического выражения используются 4 основных алгоритма:

### 2.1 Разбиение на лексемы

Перед разбиением все унарные – заменяются на ~

Создаётся пустая строка для хранения переменной или числа

Проходим циклом по символам инфиксной формы записи и проверяем является ли символ бинарной операцией (+, -, \*, /), унарной операцией (~) или скобками. Если условие истинно, то проверяем на пустоту строку. Если строка не пустая, то проверяем является она числом, символом или некорректно записанной (например, число сразу после которого идёт константа). В массив указателей на лексемы добавляем указатель на число или переменную (в случае некорректной записи будет брошено исключение) или бросаем исключение. После чего отчищаем строку. В независимости от пустоты строки добавляем в массив указателей на лексемы указатель на операнд.

После прохода по всем символам инфиксной строки проводим аналогичную проверку строки для чисел на пустоту и добавляем указатель на число при необходимости.

### 2.2 Проверка порядка следования лексем

После разбиения на лексемы проводится проверка на порядок следования лексем и соответствие количества открывающих и закрывающих скобок. Для этого существуют следующие правила:

- В начале арифметического выражения могут находится: унарные операции или '('.

- В конце арифметического выражения могут находится: числа, константы или ')'.

- Перед '(' не могут находиться: числа, переменные и ')'.

- Перед числом не могут находиться: переменные и ')'.

- Перед переменной не могут находиться: числа и ')'.

- Перед бинарной операцией могут находиться только: числа, переменные или ')'.

- Перед унарной операцией не могут находиться: числа, переменные или ')'.

- Перед ')' могут находиться только: числа, переменные и ')'.

- Количество открывающих и закрывающих скобок должно совпадать.

В случае невыполнения одного из требований будет брошено исключение с указанием на позицию в которой произошла ошибка.

### 2.3 Перевод в постфиксную форму

Для перевода арифметического выражения в постфиксную форму используется стек, хранящий операнды. Проходимся 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 – унарная (унарный -), то добавляем её в стек.

- Если все проверки оказались ложными (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.

Заведём стек для чисел и будем идти цикло 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 и добавляем полученный результат в стек.

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

# Результаты экспериментов

Для проведения экспериментов были реализованы тесты на базе Google Test. Классы Stack и Arithmetic успешно проходят все тесты.

# Заключение

В ходе лабораторной работы были реализованы структура данных Stack и класс арифметических выражений на языке программирования С++. Были описаны алгоритмы и проведена проверка корректности с использованием тестов.