МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Отчет по лабораторной работе**

**«Транслятор арифметических выражений»**

**Выполнил:**

студент группы 3823Б1ФИ1

Смышляев Александр Павлович

**Проверил:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панова Е.А.

Рецензент:

Нижний Новгород  
2024

# Содержание

[Содержание 2](#_Toc154184824)

[Введение 3](#_Toc154184825)

[1 Постановка задачи 4](#_Toc154184826)

[2 Руководство пользователя 6](#_Toc154184827)

[3 Руководство программиста 7](#_Toc154184828)

[3.1 Описание структуры программы 7](#_Toc154184829)

[3.2 Описание алгоритмов 8](#_Toc154184830)

[4. Реализация 14](#_Toc154184831)

[5. Эксперименты 15](#_Toc154184832)

[Заключение 20](#_Toc154184833)

[Список литературы 21](#_Toc154184834)

[Приложение 22](#_Toc154184835)

# Введение

На данный момент то или иное применение транслятора выражений можно встретить почти всегда, когда необходимо что-то посчитать с помощью программы, сайта или обработать некоторую строку, вводимую пользователем. Например, на основе подобных алгоритмов работают калькуляторы, интерпретаторы, различные математические приложения.

Рассмотрим транслятор арифметических выражений, казалось бы, ввиду широкого применения реализация должна быть тривиальной и состоять буквально из трех алгоритмов: лексического анализа, подразделяющего строку на отдельные компоненты, перевода в постфиксную форму, которая необходима для удобного и оптимизированного вычисления и, непосредственно, подсчет значения. Однако, в действительности, подходов к реализации достаточно много, и задача написания такого проекта дает большой простор для креатива и структурных решений. В частности, если программа рассчитана на работу с пользователем, должна быть некоторая система обработки ввода выражения и ошибок в нем, назовем ее синтаксическим анализом. И то, как его написать – вопрос, на который каждый программист ответит по-разному.

В данной работе будет представлена реализация транслятора арифметических выражений на языке C++ с пользовательским вводом, отслеживанием ошибок и поддержкой констант и переменных. К проекту, также, прилагаются тесты, проверяющие корректность работы программы и исключающие возможность возникновения ошибок и исключений.

В заключении представлен полноценный проект транслятора, дополненного синтаксическим анализом и тестами, готовый к использованию в последующих лабораторных работах.

# Постановка задачи

1. Изучить базовые методы для создания транслятора арифметических выражений.

* синтаксический анализ строки;
* лексический анализ строки;
* перевод массива компонентов в постфиксную форму;
* вычисление значения выражения по его постфиксной форме.

1. На основе полученных данных, разработать структуру программы и создать проект с использованием файлов Сmake.
2. Изучить принцип базовых методов (1.1), учитывая поддержку:

* целочисленных и вещественных чисел;
* буквенных переменных;
* констант PI и e;
* операций: +, -, \*, /;
* оператора присваивания;
* круглых скобок.

1. Разработать конечные автоматы для работы с пользователем и проведения парсиинга строки.
2. Реализовать несколько статических библиотек под следующие модули программы:

* Parser:

1. Class Parser (содержит основные функции для анализа строк);
2. Class Term (класс для описания компонентов при лексическом анализе);
3. Class Type (класс, содержащий перечисление типов компонентов).

* Interface:

1. Class Interface (класс, содержащий функции для работы с консолью).

* Converter:

1. Class Converter (класс, содержащий функцию для перевода набора компонентов в постфиксную форму).

* Stack:

1. Class Stack (класс, содержащий реализацию стэка для работы с постфиксной формой).

* Calc:

1. Class Calc (класс, содержащий функцию для вычисления итогового значения выражения).

* Connect:

1. Class Connect (класс, содержащий функции обеспечивающие связную работу всех модулей программы).
2. Протестировать и отладить работу приложения.
3. Написать тесты, проверяющие корректность работы программы (в том числе в крайних случаях) с помощью библиотеки GoogleTest, исключить недочеты.

# Руководство пользователя

1. Для начала работы программы запустите файл “AT\_exe.exe”.
2. Внимательно прочтите инструкцию использования, выведенную в консоли.
3. Для вычисления выражения введите его в предлагаемую строку. Ввод должен производиться без пробелов и вспомогательных символов.
4. Для подтверждения ввода нажмите “Enter”.
5. Если вы собираетесь использовать переменные, их необходимо инициализировать некоторым значением/выражением прежде, чем использовать в выражении.
6. Для того, чтобы создать переменную необходимо ввести ее имя, символ равно и выражение, которое вы хотите ей присвоить.
7. Если при вводе символ помечен красным и ввод невозможен, следовательно символ недопустим в данном случае, следует удалить его с помощью “Backspace” и продолжить ввод.
8. Если при нажатии “Enter” ничего не происходит или выводится красный вопросительный знак, следовательно выражение неполно, проверьте его на логичность и верное число скобок. Для продолжения работы сотрите вопросительный знак с помощью “Backspace”.
9. В программе поддерживаются следующие константы:
   1. PI = 3.14159265359;
   2. e = 2.71828182846.
10. Для работы с константами введите выражение включающее в себя название необходимого константного значения.
11. После вывода ответа нажмите “Enter” для продолжения работы или “Esc” для завершения программы.

# Руководство программиста

## Описание структуры программы

Программа разделена на несколько модулей (статических библиотек), каждый из которых отвечает за отдельную функциональность и организован как подпроект с использованием **CMake**. Основная структура представлена следующим образом:

1. **calc\_lib** (подпроект calc):

Содержит класс Calc, предоставляющий статический метод Calculate для вычисления выражений в постфиксной форме.

1. **converter\_lib** (подпроект converter):

Реализует преобразование выражений из инфиксной формы в постфиксную с помощью статического метода conv класса Converter.

1. **parser\_lib** (подпроект parser):

В данной библиотеке содержатся два класса: Term и Parser.

Первый используется, как вспомогательный класс для определения типа частей выражения, выделяемых при парсинге. Класс имеет два приватных поля, содержащих тип и значение лексемы, три конструктора, статичную функцию для перевода строки в вещественное число, две функции для доступа к полям и оператор сравнения, предназначенный для использования при тестировании класса. В нем, также, содержится перечисляемый класс Type, в котором объявлены все возможные типы вводимых лексем.

Класс Parser применяется для лексического и синтаксического анализа вводимой строки, в нем реализуются два конечных автомата для работы со строкой. Все методы класса являются статичными функциями: add\_ch и del\_ch необходимы для лучшей читаемости, в них вынесены повторы кода, term\_analis и synt\_analis – лексический и синтаксический анализы соответственно, synt\_analis\_fsm – отдельно вынесенная реализация конечного автомата синтаксического анализа необходимая для лучшей структуры кода.

1. **connect\_lib** (подпроект connect):

Данная библиотека содержит класс Connect, который обеспечивает связь между остальными классами. В нем содержатся два статичных метода: connection – позволяет вычислять произвольное количество выражений за один сеанс программы, handler – обрабатывает информацию о вводе пользователем выражения или приравнивании нового значения переменной, обеспечивает логику программы и запуск необходимых функций с верными параметрами.

1. **interface\_lib** (подпроект interface):

Обеспечивает взаимодействие программы с консолью. Класс Interface содержит методы для работы с текстом (например, print\_ch для вывода символов, delete\_ch для удаления последнего символа) и вспомогательные функции, такие как set\_color для установки цвета текста и clear\_console для очистки экрана. Используется библиотека windows.h, из-за чего проект несовместим с Linux.

1. **stack\_lib** (подпроект stack):

Содержит реализацию структуры данных стек, необходимой для выполнения операций в постфиксной записи.

## Описание алгоритмов

Разберем следующие основные алгоритмы, используемые в программе:

1. Синтаксический анализ.
2. Лексический анализ.
3. Перевод выражения в постфиксную форму.
4. Вычисление значения по постфиксной форме.

**Синтаксический анализ.**

Пользователю предлагается ввести выражение в консоль. Параллельно с этим запускается конечный автомат представленный на рисунке 1, который посимвольно принимает ввод пользователя, запрещая ввести синтаксически неверное выражение, и сохраняет в строку.

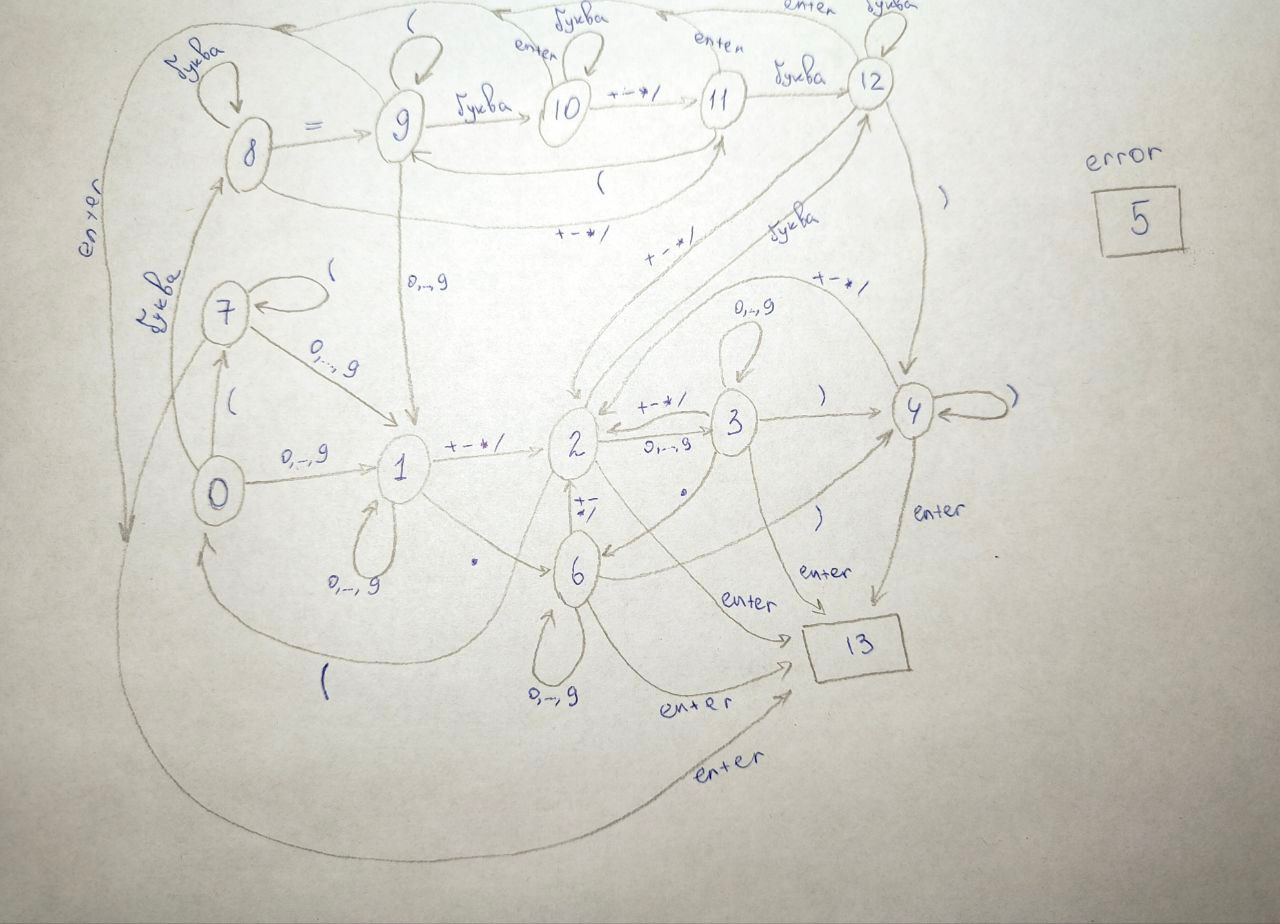


Рисунок 1

Состояния, пройденные при работе данного алгоритма, хранятся в стеке. Это позволяет возвращаться к ним при использовании пользователем Backspace.

При вводе символа, который не ожидается в данном состоянии, осуществляется переход в состояние ошибки (5 состояние). Из него можно выйти, удалив последний символ с помощью клавиши Backspace.

Переход в состояние 13 происходит при нажатии клавиши Enter только в том случае, если введено неверное количество скобок или предыдущее состояние не подразумевает конец выражения.

Таким образом, после окончания работы данного конечного автомата получаем синтаксически верную строку, которую можно отправлять в лексический анализ без проверок на правильность.

**Лексический анализ**

В начале работы алгоритм получает синтаксически верную строку и разбивает ее на лексемы. Для его реализации используется аналогичный конечный автомат, главное отличие которого – отсутствие проверок на ошибки и состояний 5 и 13.

Проходя по строке, он создает объекты класса Term, передавая в конструктор выделенные подстроки, и добавляет их в массив.

На выходе имеем массив лексем, где у каждой лексемы есть значение в виде строки и ее тип.

**Перевод выражения в постфиксную форму**

В первую очередь создаются вспомогательные структуры данных: стек для временного хранения операторов и открывающих скобок и вектор для хранения результата в постфиксной форме. Задается таблица приоритетов операторов (например, \* и / имеют более высокий приоритет, чем + и -).

Далее начинается работа непосредственно алгоритма перевода в постфиксную форму и составления массива операндов. Осуществляется проход по массиву лексем и в зависимости от типа текущей лексемы выполняются следующие действия:

1. **Число:**

Добавляется в выходной вектор, так как числа в постфиксной форме располагаются непосредственно перед их обработкой.

Добавляется в массив операндов.

1. **Открывающая скобка:**

Помещается в стек для дальнейшего сопоставления с закрывающей скобкой.

1. **Закрывающая скобка:**

Элементы извлекаются из стека и записываются в итоговый вектор до тех пор, пока не встретится открывающая скобка. Открывающая скобка удаляется из стека, но не добавляется в результат.

1. **Оператор:**

Проверяется приоритет оператора относительно верхнего элемента стека:

* операторы с более низким или равным приоритетом извлекаются из стека и добавляются в результат;
* после чего текущий оператор помещается в стек.

1. **Переменная:**

Если переменная существует, её значение добавляется в выходной вектор и массив операндов как число. В противном случае алгоритм прерывается, указывая на ошибку.

**Завершение работы алгоритма:**

После обработки всех элементов входного выражения содержимое стека извлекается и добавляется в выходной вектор.

На выходе формируется последовательность элементов (чисел и операторов), записанных в постфиксной форме, готовая для выполнения операций.

**Вычисление значения по постфиксной форме**

В начале работы алгоритм инициализирует стек для хранения операндов, две переменные для проведения промежуточных вычислений и некоторый индекс для прохода по массиву операндов.

Далее начинается обработка постфиксного выражения. Алгоритм проходит по постфиксной форме, выполняя для каждого типа лексем (операнд или оператор) следующие действия:

**Число**:

Из списка операндов извлекается число (по индексу) и помещается в стек. Индекс увеличивается.

**Оператор**:

1. Извлекаются два верхних элемента из стека.
2. К этим значениям применяется оператор в соответствии с его типом:
   * +: Сложение;
   * -: Вычитание;
   * \*: Умножение;
   * /: Деление;
3. Результат операции помещается обратно в стек.

Таким образом, после обработки всех элементов постфиксного выражения в стеке остается единственный элемент, возвращаемый алгоритмом. Он и будет являться результатом вычислений.

# Реализация

В программе реализованы следующие библиотеки:

Статическая библиотека Calc\_lib (Приложение 1.1);

Статическая библиотека Converter\_lib (Приложение 1.2);

Статическая библиотека Parser\_lib (Приложение 1.3);

Статическая библиотека Connect\_lib (Приложение 1.4);

Статическая библиотека Interface\_lib (Приложение 1.5);

Статическая библиотека Stack\_lib (Приложение 1.6).

В качестве исполняемого подпроекта можно выбрать само приложение или тесты к нему.

При запуске приложения пользователь видит инструкцию по работе с ним, также, пользователю предлагается ввести интересующее его выражение. После вычислений приложение не закрывается автоматически, дает возможность продолжить работу.

# Эксперименты

*Цель эксперимента*: Протестировать написанные библиотеки для подтверждения их работоспособности.

*Ход эксперимента:*

Для проверки функциональности программы были реализованы тесты с использованием библиотеки GoogleTest и фикстур (Приложение 2.1). Тестирование охватывало следующие статические библиотеки:

1. Calc\_lib.
2. Connect\_lib.
3. Convert\_lib.
4. Parser\_lib.

Данные тесты покрывают как общие, так и крайние случаи программы, что исключает возможность возникновения ошибок. Тесты выполнились без ошибок.

Также, вручную была протестирована правильность работы синтаксического анализа. Были выбраны крайние случаи на основе описанного выше конечного автомата и устранены возможности ошибок в них.

Результаты тестирования подтверждают корректную работу программы. Написанные библиотеки функционируют стабильно и соответствуют заданным требованиям.

# Заключение

В ходе выполнения лабораторной работы была разработана программа-транслятор арифметических выражений, реализованная на языке C++ с использованием модульной структуры. Архитектура проекта основывается на статических библиотеках, каждая из которых отвечает за отдельную задачу (парсинг, преобразование выражений, вычисления и т.д.). Использование CMake упростило управление зависимостями, сборку и тестирование программы.

Особое внимание было уделено обработке пользовательского ввода: разработанный синтаксический анализатор на основе конечного автомата гарантирует корректность строки на этапе ввода. Это обеспечивает защиту от ошибок, таких как несоответствие количества скобок или использование недопустимых символов. Подсвечивание ошибок и возможность исправления ввода делают программу удобной для пользователя.

Проведенные тесты с использованием GoogleTest подтвердили корректность работы всех основных модулей программы. Также вручную были протестированы крайние случаи, выявленные на этапе проектирования конечных автоматов. В результате программа продемонстрировала стабильность и соответствие поставленным требованиям.

В процессе выполнения работы были изучены и применены следующие методы и технологии:

* создание и тестирование статических библиотек;
* использование CMake для управления сборкой проекта;
* реализация конечных автоматов для синтаксического и лексического анализа;
* тестирование с использованием библиотеки GoogleTest.

Разработанный проект обладает гибкостью и модульностью, что позволяет легко расширять его функционал. Например, в будущем можно добавить поддержку новых операций, функций или использование в следующих работах.

Работа дала возможность углубить знания в области разработки алгоритмов, проектирования программного обеспечения и использования современных инструментов тестирования.

# Список литературы

1. Официальная документация GoogleTest [Электронный ресурс].   
   URL: <https://google.github.io/googletest/> (дата обращения: 28.11.2024).
2. Официальная документация CMake [Электронный ресурс].   
   URL: <https://cmake.org/documentation/> (дата обращения: 17.11.2024).

# Приложение

**Приложение 1.1. Статическая библиотека Calc\_lib**

Доступно по ссылке:

<https://github.com/Allor-maker/arithmetic_translator/tree/add_code/include/calc>

**Приложение 1.2. Статическая библиотека Converter\_lib**

Доступно по ссылке:

<https://github.com/Allor-maker/arithmetic_translator/tree/add_code/include/converter>

**Приложение 1.3. Статическая библиотека Parser\_lib**

Доступно по ссылке:

<https://github.com/Allor-maker/arithmetic_translator/tree/add_code/include/parser>

**Приложение 1.4. Статическая библиотека Connect\_lib**

Доступно по ссылке:

<https://github.com/Allor-maker/arithmetic_translator/tree/add_code/include/connect>

**Приложение 1.5. Статическая библиотека Interface\_lib**

Доступно по ссылке:

<https://github.com/Allor-maker/arithmetic_translator/tree/add_code/include/interface>

**Приложение 1.6. Статическая библиотека Stack\_lib**

Доступно по ссылке:

<https://github.com/Allor-maker/arithmetic_translator/tree/add_code/include/stack>

**Приложение 2.1. Тесты для библиотеки**

Доступно по ссылке:

<https://github.com/Allor-maker/arithmetic_translator/tree/add_code/test>