

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Дальневосточный федеральный университет»**

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ (ШКОЛА)**

**Департамент программной инженерии и искусственного интеллекта**

РАЗРАБОТКА ТРАНСЛЯТОРА, ПЕРЕВОДЯЩЕГО ПОДМНОЖЕСТВО ЯЗЫКА **RUST** В ЭКВИВАЛЕНТНОЕ ПОДМНОЖЕСТВО ЯЗЫКА **С++**

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

по дисциплине «Теория языков программирования и компиляторы»

по образовательной программе подготовки бакалавров по направлению 09.03.04 Программная инженерия

Выполнили студенты

гр. Б9120-09.03.04прогин

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Орлов Г. А.

Руководитель:

ассистент департамента ПИиИИ

Симаков В.К.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

(подпись)

г. Владивосток

2023 г

**Оглавление**

[Введение 4](#_Toc139040483)

[1 Неформальная постановка задачи 5](#_Toc139040484)

[2 Синтаксис входного языка 5](#_Toc139040485)

[3 Контекстные условия языка Rust 5](#_Toc139040486)

[4 Контекстные условия языка C++ 6](#_Toc139040487)

[5 Таблица соответствия языков 7](#_Toc139040488)

[6 Проект лексического анализатора 7](#_Toc139040489)

[6.1 Модель данных 8](#_Toc139040490)

[6.2 Таблица ключевых слов 8](#_Toc139040491)

[6.3 Таблица зарезервированных имен 9](#_Toc139040492)

[6.4 Типы лексем 10](#_Toc139040493)

[6.5 Типы ошибок 10](#_Toc139040494)

[6.6 Конечный автомат лексического анализатора 10](#_Toc139040495)

[7 Проект семантического анализатора 12](#_Toc139040496)

[7.1 Модель данных 12](#_Toc139040497)

[7.2 Описание алгоритма X семантического анализа 12](#_Toc139040498)

[7.3 Пример дерева разбора 12](#_Toc139040499)

[7.4 Типы ошибок 13](#_Toc139040500)

[8 Проект семантического анализатора 14](#_Toc139040501)

[8.1 Модель данных (если есть) 14](#_Toc139040502)

[8.2 Методы проверки нарушений контекстных условия 14](#_Toc139040503)

[8.3 Типы ошибок 14](#_Toc139040504)

[9 Проект генератора кода 15](#_Toc139040505)

[9.1 Модель данных (если есть) 15](#_Toc139040506)

[9.2 Метод кодогенерации 15](#_Toc139040507)

[9.3 Примеры кодогенерации 15](#_Toc139040508)

[10 Тестирование 16](#_Toc139040509)

[Заключение 17](#_Toc139040510)

# Введение

В данном отчёте рассматривается создание транслятора с языка Rust на C++. Транслятор - это программное средство, которое преобразует исходный код с одного языка программирования (в данном случае Rust) на другой язык программирования (C++).

Язык программирования представляет собой формализованный набор символов и правил, используемых для написания программного кода. В данном проекте входным языком является Rust, мощный системный язык программирования, который обеспечивает безопасность памяти и высокую производительность. Выходным языком является C++, популярный и широко используемый язык программирования, который обладает мощными возможностями и оптимизацией.

Основная цель создания данного транслятора - обеспечить возможность переноса существующего кода из языка Rust в C++. Это может быть полезно в случае необходимости использования существующих библиотек, инструментов и ресурсов, доступных только на языке C++.

В следующих разделах отчёта будут рассмотрены подробности реализации транслятора, используемые технологии, возможности и ограничения данного подхода, а также результаты тестирования и дальнейшие возможности развития проекта.

**Цель курсового проекта:** разработать проект транслятора, который преобразует программу, содержащую подмножество языка Rust, в программу, содержащую подмножество языка C++ генерируя эквивалентный исходный код.

**Задачи:**

1. описать грамматику подмножества входного языка;
2. описать контекстные условия входного и выходного языков;
3. описать соответствие конструкций входного и выходного языков;
4. разработать проект лексического анализатора;
5. разработать проект синтаксического анализатора;
6. разработать проект семантического анализатора;
7. разработать проект генератора кода выходного языка;
8. реализовать транслятор входного языка в выходной;
9. выполнить тестирование разработанного транслятора.

# 1 Неформальная постановка задачи

Требуется разработать транслятор с языка Rust на C++, который способен преобразовывать исходный код на языке Rust в эквивалентный код на языке C++.

Входной язык программирования, Rust, характеризуется следующим подмножеством функциональности:

Типы данных: целочисленные (integer) и другие стандартные типы данных (например, float).

Операторы: арифметические операторы (+, -, \*, /), логические операторы (==, !=, <, >) и прочие операторы (например, and).

Операторы циклов: while-цикл.

Операторы ветвления: if-else условные операторы.

Стандартные функции: например, функция print для вывода на экран.

Транслятор должен обеспечивать корректное преобразование синтаксиса и семантики кода на языке Rust в аналогичный код на языке C++. Результат выполнения программы транслятора должен быть эквивалентен оригинальной программе на языке Rust, с сохранением структуры и логики программы.

Также требуется обеспечить обработку возможных ошибок, которые могут возникнуть при трансляции кода, и предоставить информативные сообщения об ошибках для облегчения их исправления.

# 2 Синтаксис входного языка

Тут нужно написать про какие обозначения используются ниже, например для переноса строки или табуляции. Описываем грамматику входного языка (<PROGRAM> ::= <INSTRUCT> | <INSTRUCTIONS> и тд и тп.). Можно добавлять подразделы для описания некоторых конструкций, если вам так удобнее.

# 3 Контекстные условия языка Rust

Ключевые слова: Rust имеет определенный набор ключевых слов, таких как let, if, else, while, fn и т. д. Ключевые слова не могут быть использованы как идентификаторы.

Блочная структура: программы Rust имеют блочную структуру с помощью фигурных скобок {}. Блоки могут быть вложенными и используются для группировки логически связанных операций или инструкций.

Объявление переменных: переменные в Rust должны быть объявлены с использованием ключевого слова let, за которым следует имя переменной и тип (необязательно). Переменные в Rust имеют строгую типизацию.

Типы данных: в Rust доступны следующие типы данных: целочисленные типы (i32, u32 и т. д.), дробные типы (f32, f64), логический тип (bool), символьный тип (char) и другие композитные типы данных (например, массивы, кортежи, структуры и перечисления).

Операторы условия: в Rust используются операторы условия if-else для выполнения различных действий в зависимости от заданного условия.

Операторы цикла: в Rust используется оператор цикла while для повторения блока кода до тех пор, пока заданное условие остается истинным.

Функции: в Rust функции объявляются с использованием ключевого слова fn, за которым следует имя функции, параметры (необязательно) и тип возвращаемого значения (необязательно).

Модули: Rust поддерживает модули, которые используются для логической группировки кода и обеспечения его организации и повторного использования.

Обработка ошибок: Rust предоставляет механизмы обработки ошибок, такие как операторы Result и Option, для обнаружения и обработки возможных ошибок в программе.

Комментарии: в Rust можно добавлять комментарии в код с использованием символов // для однострочных комментариев. Комментарии игнорируются компилятором и служат для описания кода или временного отключения части кода.

# 4 Контекстные условия языка C++

Ключевые слова: C++ имеет определенный набор ключевых слов, таких как int, char, float, if, else, while, for, и т. д. Ключевые слова не могут быть использованы как идентификаторы.

Блочная структура: программы на C++ имеют блочную структуру с помощью фигурных скобок {}. Блоки могут быть вложенными и используются для группировки логически связанных операций или инструкций.

Объявление переменных: переменные в C++ должны быть объявлены с использованием ключевого слова, за которым следует имя переменной и тип (необязательно). Переменные в C++ имеют строгую типизацию.

Типы данных: в C++ доступны следующие типы данных: целочисленные типы (int, char, long и т. д.), дробные типы (float, double), логический тип (bool), символьный тип (char) и другие композитные типы данных (например, массивы, структуры и классы).

Операторы условия: в C++ используются операторы условия if-else для выполнения различных действий в зависимости от заданного условия.

Операторы цикла: в C++ используются операторы цикла while, do-while и for для повторения блока кода до тех пор, пока заданное условие остается истинным.

Функции: в C++ функции объявляются с использованием ключевого слова, за которым следует имя функции, параметры (необязательно) и тип возвращаемого значения (необязательно).

Классы и объекты: в C++ можно определять классы и создавать объекты на их основе. Классы могут содержать переменные-члены и функции-члены.

Обработка исключений: C++ предоставляет механизм обработки исключений с помощью ключевых слов try, catch и throw для обнаружения и обработки исключительных ситуаций в программе.

Комментарии: в C++ можно добавлять комментарии в код с использованием символов // для однострочных комментариев и /\* \*/ для многострочных комментариев. Комментарии игнорируются компилятором и служат для описания кода или временного отключения части кода.

# 5 Таблица соответствия языков

|  |  |  |
| --- | --- | --- |
| Языковая структура | Rust | C++ |
| Блочная структура | {} | {} |
| Объявление переменных | let | int, char, float, etc. |
| Типы данных | bool, i32, u32, f32, etc. | bool, int, float, char, etc. |
| Операторы условия | if, else | if, else |
| Операторы цикла | loop, while, for | while, do-while, for |
| Функции | fn | function, void, etc. |
| Классы и объекты | struct | class, object |

# 6 Проект лексического анализатора

Лексер (также известный как лексический анализатор) - это компонент в компиляторе или интерпретаторе, который преобразует последовательность символов в программе (такую как исходный код) в поток лексем. Лексемы представляют собой минимальные синтаксические единицы языка программирования, такие как ключевые слова, операторы, идентификаторы, константы и другие.

Лексический анализ - это процесс разбора исходного кода на отдельные лексемы, определение их типов и последовательности, а также их передачи синтаксическому анализатору для дальнейшей обработки. Таким образом, лексический анализ позволяет компилятору или интерпретатору понять структуру программы на основе ее лексических единиц.

В компиляторе или интерпретаторе лексер обычно работает следующим образом:

Сканирование: лексер последовательно читает исходный код и выделяет лексемы.

Классификация: лексемы классифицируются по типам данных, таким как операторы, идентификаторы, ключевые слова и т.д.

Генерация лексического потока: лексер создает поток лексем, который передается синтаксическому анализатору для дальнейшей обработки.

В результате лексического анализа и сгенерированного потока лексем компилятор или интерпретатор "понимает" структуру программы и может выполнять следующие шаги, такие как синтаксический анализ, анализ семантики и генерация исполняемого кода или выполнение программы.

## 6.1 Модель данных

Какие классы/структуры использовали, почему. Диаграммы классов можно вставить, если есть. Описать как представлены лексемы и токены в памяти

## 6.2 Таблица ключевых слов

Ключевые слова - это зарезервированные слова в языках программирования, которые имеют особое значение и не могут быть использованы в качестве идентификаторов (имен переменных, функций и т.д.).

Ключевые слова в языке Rust отличаются от остальных лексем (т.е. символов и последовательностей символов) тем, что они имеют заранее определенное значение и назначение в языке. Компилятор Rust распознает эти ключевые слова и использует их для определения структуры и семантики программы.

* as
* break
* const
* continue
* crate
* else
* enum
* extern
* false
* fn
* for
* if
* impl
* in
* let
* loop
* match
* mod
* move
* mut
* pub
* ref
* return
* self
* static
* struct
* super
* trait
* true
* type
* unsafe
* use
* where
* while

## 6.3 Таблица зарезервированных имен

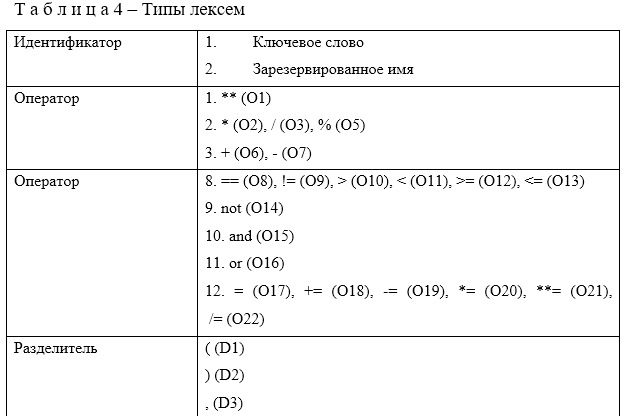
Что, зачем, в чем отличие от ключевых слов и других лексем. Ниже пример таблицы

Изображение выглядит как стол

Автоматически созданное описание

## 6.4 Типы лексем

Аналогично (что, зачем). Ниже пример таблицы (кусочек) (название токенов в первом столбце вы даете сами, а справа - представители токена)

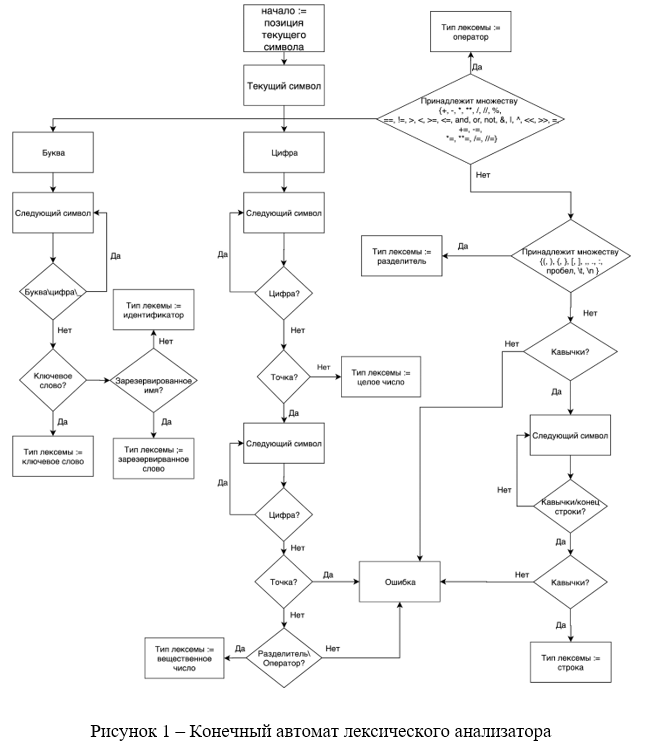
****

## 6.5 Типы ошибок

Все ошибки, которые может выдать лексер с примерами кода и сообщениями об ошибке

## 6.6 Конечный автомат лексического анализатора

Аналогично что зачем и диаграмму автомата, можно в виде блок-схемы как ниже, лучше в виде диаграммы конечного автомата – она будет короче. Еще нужно привести пример его работы на примере какой-нибудь (демонстрирующей возможности языка) программы на входном языке. Можно привести пример вывода своего лексера.

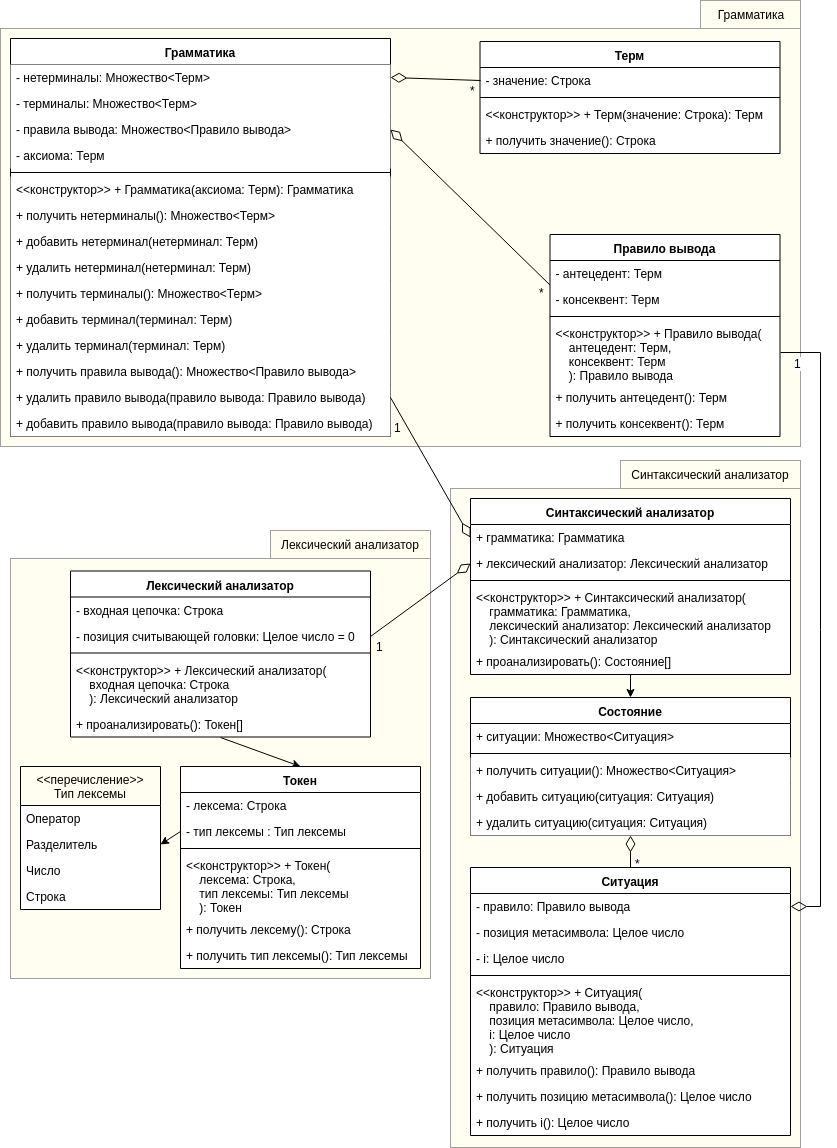


# 7 Проект семантического анализатора

Семантический анализатор (или семантический анализ) - это процесс анализа смысловой структуры текста с целью понимания его семантического значения и выявления связей между различными элементами.

## 7.1 Модель данных

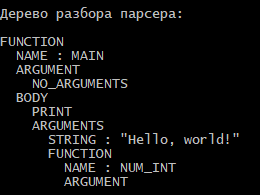
Средствами чего реализовывали, какой язык для этого выбрали (для написания синтаксического анализатора), привести диаграммы моделей данных. Показать как моделируется дерево разбора. Ниже пример диаграммы.



## 7.2 Описание алгоритма X семантического анализа

Тут понятно. Просто распишите как в лекциях

## 7.3 Пример дерева разбора



## 7.4 Типы ошибок

Аналогично разделу 6

# 8 Проект семантического анализатора

Аналогично 6

## 8.1 Модель данных (если есть)

Аналогично 6

## 8.2 Методы проверки нарушений контекстных условия

Как проверяются ошибки типов и тд (контекстных условий)

## 8.3 Типы ошибок

Аналогично 6

# 9 Проект генератора кода

Генератор кода в трансляторе является одной из важных компонентов, отвечающих за преобразование входного языка программирования в целевой код, который может быть выполнен на целевой платформе или архитектуре. Он работает в конце процесса трансляции, после анализа и синтеза исходного кода.

Генератор кода получает на вход абстрактное синтаксическое дерево (AST) или промежуточное представление программы (IR) и использует различные оптимизации и техники для преобразования кода в целевой язык или инструкции, понятные целевому исполнительному окружению.

Задачи генератора кода включают выбор правильных инструкций, управление памятью и регистрами, оптимизацию вычислений и преобразование алгоритмов в эффективный и оптимальный код. В зависимости от целевой платформы, генератор кода может выполнять дополнительные задачи, такие как генерация машинного кода или байт-кода.

## 9.1 Модель данных (если есть)

## 9.2 Метод кодогенерации

## 9.3 Примеры кодогенерации

Изображение выглядит как текст, снимок экрана, Шрифт

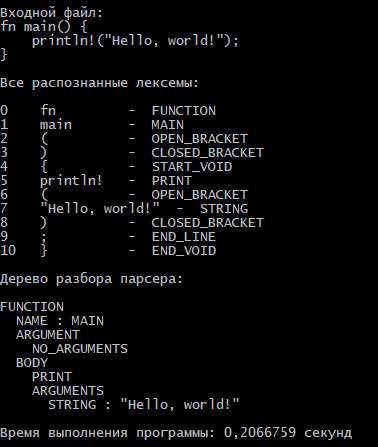
Автоматически созданное описание

# 10 Тестирование

Тестовый файл 1:

Изображение выглядит как текст, снимок экрана, Шрифт

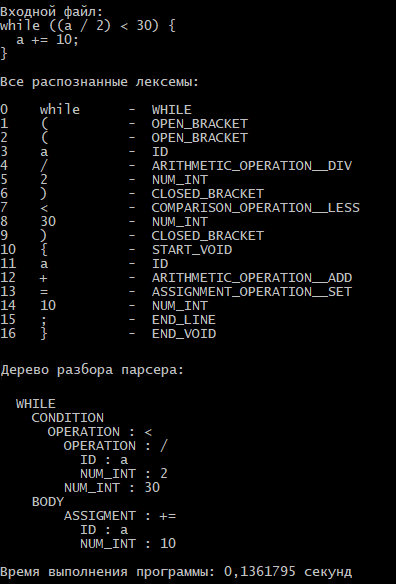
Автоматически созданное описание

Результат работы программы:

Тестовый файл 2:

Изображение выглядит как текст, снимок экрана, Шрифт

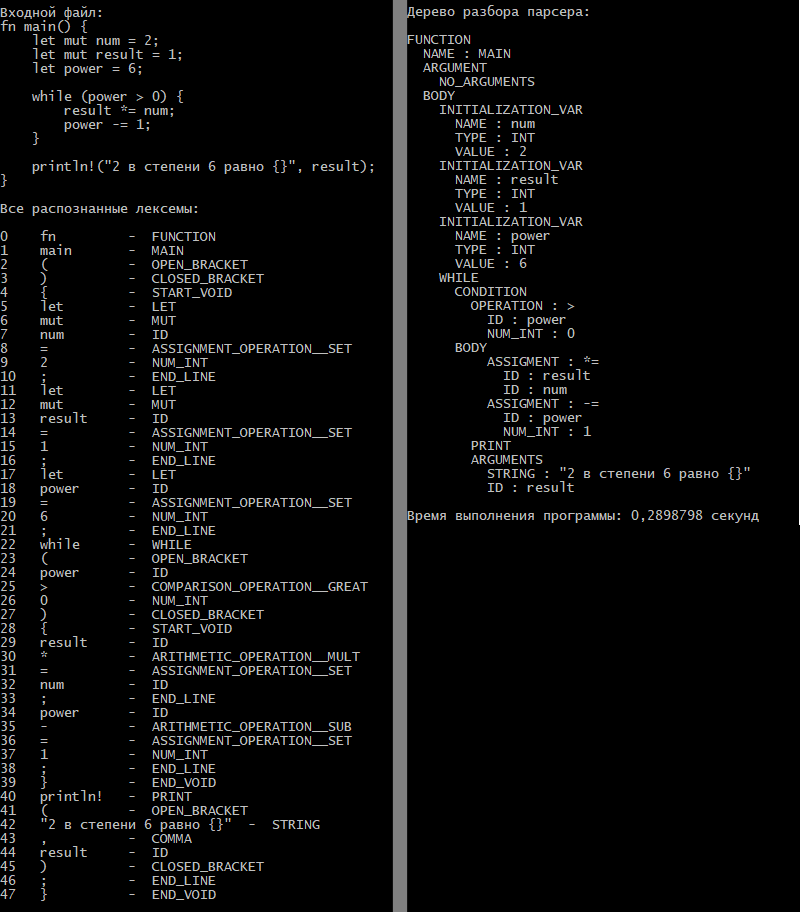
Автоматически созданное описание

Результат работы программы:

Тестовый файл 3:

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Автоматически созданное описание

Результат работы программы:

# Заключение

И сдали они все ТЯПИК, и всё у них было хорошо 😊