Министерство науки и высшего образования

Федеральное государственное бюджетное образовательное учреждение высшего образования

Волгоградский государственный технический университет

Факультет электроники и вычислительной техники

Кафедра «Программное обеспечение автоматизированных систем»

КОНТРОЛЬНАЯ РАБОТА

по дисциплине «Теория формальных языков и методы трансляции»

на тему «Разработка компилятора для языка программирования Rust»

Выполнили:

ст.гр. ПрИн-466

Ломакин А. С., Чупинин А. П.

Проверил:

доцент кафедры «ПОАС»

Сычев О. А.

Волгоград, 2023

Оглавление

[1) Описание синтаксиса и семантики (смысла) компилируемого языка 3](#_Toc157091998)

[2) Пример программы на компилируемом языке, с использованием всех реализованных возможностей 5](#_Toc157091999)

[3) Таблица с распределением реализованной части 13](#_Toc157092000)

[4) Таблица лексем 16](#_Toc157092001)

[5) Грамматика языка 24](#_Toc157092002)

[6) Перечень таблиц, генерируемых на этапе семантического анализа 36](#_Toc157092003)

[7) Перечень ошибок, определяемых на этапе семантического анализа 41](#_Toc157092004)

[8) Перечень преобразований дерева и дополнительной информации для узлов дерева на этапе семантического анализа 42](#_Toc157092005)

[9) Выводы 42](#_Toc157092006)

1) Описание синтаксиса и семантики (смысла) компилируемого языка

Rust - современный системный язык программирования, ориентированный на безопасность, производительность и параллелизм. Синтаксис Rust чистый и выразительный, обеспечивая ясное определение переменных, функций и структур данных. Объявление переменных с использованием ключевых слов let и mut для неизменяемых и изменяемых переменных соответственно.

Функции в Rust определяются с использованием ключевого слова fn, а структуры данных - с помощью ключевого слова struct. Rust также предоставляет мощные механизмы управления потоком выполнения, такие как условные операторы, циклы, передачу по ссылке.

**Примеры синтаксиса:**

**Объявление переменных:**

let x = 5; // Неизменяемая переменная

let mut y = 10; // Изменяемая переменная

**Функции:**

fn add(a: i32, b: i32) -> i32 {

a + b

}

**Структуры данных:**

trait Printable {

fn print(&self);

}

// Пример структуры данных, реализующей трейт Printable

struct Dog {

name: String,

}

// Реализация трейта Printable для структуры Dog

impl Printable for Dog {

fn print(&self) {

println!("Dog named {}", self.name);

}

}

**Условия:**

if x > 0 {

println!("Positive");

} else if x < 0 {

println!("Negative");

} else {

println!("Zero");

}

**Циклы:**

for i in 0..5 {

println!("{}", i);

}

while condition {

// код

}

// Бесконечный цикл с использованием ключевого слова loop

loop {

println!("Current count: {}", counter);

// Увеличение счетчика

counter += 1;

// Проверка условия завершения цикла

if counter >= 5 {

break; // Выход из цикла

}

}

Одной из ключевых особенностей Rust является система владения памятью, которая позволяет избегать утечек памяти и ошибок с нулевыми указателями. Rust обеспечивает безопасность памяти на уровне компиляции, что позволяет программистам писать производительный и безопасный код.

Язык также активно поддерживает параллельное программирование с помощью концепции "владения", что предотвращает гонки данных и проблемы синхронизации.

Rust также обладает современными инструментами разработки, такими как система сборки Cargo и менеджер зависимостей. Экосистема Rust активно развивается, что делает его привлекательным выбором для широкого спектра задач от встраиваемого программирования до создания серверных приложений.

2) Пример программы на компилируемом языке, с использованием всех реализованных возможностей

struct MyStruct {  
 data: i32,  
}  
  
fn by\_value(mut obj: MyStruct) {  
 println\_int("Function by\_value: data before = {}", obj.data);  
 obj.data = 999;  
 println\_int("Function by\_value: data modified = {}", obj.data);  
}  
  
fn by\_reference(obj: &mut MyStruct) {  
 println\_int("Function by\_reference: data before = {}", obj.data);  
 obj.data = 888;  
 println\_int("Function by\_reference: data modified = {}", obj.data);  
}  
  
mod geometry {  
   
 pub struct Circle {  
   
 pub radius: f64,  
   
 side: f64,  
 }  
  
   
 pub mod triangle {  
   
 pub struct Triangle {  
   
 pub base: f64,  
   
 pub height: f64,  
 }  
  
 impl Triangle {  
   
 pub fn *new*(base: f64, height: f64) -> Triangle {  
 Triangle { base: base, height: height}  
 }  
  
   
 pub fn area(&self) -> f64 {  
 0.5 \* self.base \* self.height  
 }  
 }  
  
   
 fn private\_function() {  
 println("{}", "Это приватная функция внутри модуля triangle");  
 }  
  
   
 pub fn public\_function() {  
 println("{}", "Это публичная функция из модуля triangle");  
   
 private\_function();  
 }  
 }  
  
   
 mod inner\_module {  
   
 pub struct InnerStruct {  
 pub field: i32,  
 }  
  
   
 pub fn inner\_function() {  
 println("{}", "Это внутренняя функция из модуля inner\_module");  
 }  
 }  
  
   
 pub fn use\_inner\_module() {  
   
 let inner\_instance = inner\_module::InnerStruct { field: 42 };  
  
   
 println\_int("Поле InnerStruct: {}", inner\_instance.field);  
 inner\_module::inner\_function();  
 }  
  
 impl Circle {  
   
 pub fn *new*(radius: f64) -> Circle {  
 Circle { radius: radius, side: 0.0 }  
 }  
  
   
 pub fn area(&self) -> f64 {  
 3.14 \* self.radius \* self.radius  
 }  
 }  
  
   
 fn private\_function() {  
 println("{}", "Это приватная функция внутри модуля geometry");  
 }  
  
   
 pub fn public\_function() {  
 println("{}", "Это публичная функция из модуля geometry");  
   
 private\_function();  
 }  
}  
  
  
fn main() {  
 let mut ch : char = 'x';  
 let mut integ : i32 = 1000000000;  
 let mut f : f32 = 1.2;  
 let mut b : bool = false;  
 let mut strs : String = "Petya";  
 let x = 23\_\_\_;  
 println\_int("{}", x);  
  
 let decimal\_integer = 4\_2; // Десятичное целое число (42, \_ не несет смысловой нагрузки)  
 let hexadecimal\_integer = 0x2A; // Шестнадцатеричное целое число  
 let binary\_integer = 0b101010; // Двоичное целое число  
 let octal\_integer = 0o52; // Восьмеричное целое число  
  
 println\_int("{}", decimal\_integer);  
 println\_int("{}", hexadecimal\_integer);  
 println\_int("{}", binary\_integer);  
 println\_int("{}", octal\_integer);  
  
 {  
 let x = "abc";  
 let y = "aaa";  
  
 println\_bool("equal res {}", x == y);  
 println\_bool("not equal res {}", x != y);  
 println\_bool("greater res {}", x > y);  
 println\_bool("lesser res {}", x < y);  
 println\_bool("ge res {}", x >= y);  
 println\_bool("le res {}", x <= y);  
 }  
  
 let my\_object = MyStruct { data: 42 };  
  
 by\_value(my\_object);  
 let mut my\_object\_ref = MyStruct { data: 123 };  
  
 by\_reference(&mut my\_object\_ref);  
  
 println\_int("Main function: data = {}", my\_object\_ref.data);  
  
 let result = gcd\_recursive(24, 36);  
  
 let circle = geometry::Circle::new(5.0);  
 let triangle = geometry::triangle::Triangle::new(3.0, 4.0);  
  
 println\_float("Радиус круга: {}", circle.radius);  
 println\_float("Основание треугольника: {}", triangle.base);  
 println\_float("Высота треугольника: {}", triangle.height);  
   
 let circle\_area = circle.area();  
 let triangle\_area = triangle.area();  
   
 println\_float("Площадь круга: {}", circle\_area);  
 println\_float("Площадь треугольника: {}", triangle\_area);  
  
 geometry::public\_function();  
  
 geometry::triangle::public\_function();  
  
 geometry::use\_inner\_module();  
  
 {  
 let mut n = 0;  
 while n < 10  
 {  
 if n > 8 {  
 break;  
 }  
 println\_int("n = {}", n);  
 n = n + 1;  
 }  
 }  
  
 {  
 let mut n = 0;  
 while n < 10  
 {  
 n = n + 1;  
 if n == 9 {  
 continue;  
 }  
 println\_int("n = {}", n);  
 }  
 }  
  
 // Полиморфизм  
 {  
 let array: [Shape; 2] = [Circle{r:10.1}, Rectangle{x: 10.0, y:10.0} ];  
  
 for elem in array {  
 println\_float("Shape area {} = ", elem.area());  
 }  
 }  
  
 {  
 let x = 10;  
 let n = 0;  
 let a = (x != 10);  
 let k = (x != 10) && (x / n == 1);  
 let all = !((x != 10) && (x / n == 1));  
 println\_bool("{}", a);  
 println\_bool("{}", k);  
 println\_bool("{}", all);  
 let mut b = !( a && (x / n == 1));  
 println\_bool("{}", b);  
 if (!((x != 10) && (x / n == 1))){  
 println("result is {}", "true");  
 } else {  
 println("result is {}", "false");  
 }  
 }  
  
 //left operand and  
 {  
 let x = 10;  
 let n = 0;  
 let y = (x != 10) && (x / n == 1) ;  
 println\_bool("{}", y);  
 }  
  
 //left operand or  
 {  
 let x = 10;  
 let n = 0;  
 let y = (x == 10) || (x / n == 1) ;  
 println\_bool("{}", y);  
 }  
  
 //enum  
 {  
 // Создание переменных с типом данных Color  
 let COLOR1 = Color::*Blue*;  
 Color::getBright(COLOR1);  
 }  
  
 // autofill matrix  
 {  
 let c = 23;  
 let mut matrix = [ [c + 12 \* 13 % 46; 3] ; 3];  
  
 for array in matrix {  
 for x in array {  
 print\_int("{} ", x);  
 }  
 println("{}", "");  
 }  
 }  
  
 //loop  
 {  
 let mut num = 1;  
 let result = loop  
 {  
 if num == 4 { break num \* 2;}  
 num = num + 1;  
 };  
 println\_int("result = {}", result); // result = 8  
 }  
  
 //arithmetic  
 {  
 let integer1 = 42;  
 let mut integer2: i32 = 500;  
 const *integer3*: i32 = 100;  
  
 let sum = integer1 + integer2;  
 let difference = integer2 - integer1;  
 let product = integer1 \* 5;  
 let quotient = (-*integer3*) / 2;  
  
 let x = - integer1; //унарный минус  
 let x1 = - 9;  
  
 println\_int("{}", sum);  
 println\_int("{}", difference);  
 println\_int("{}", product);  
 println\_int("{}", quotient);  
 println\_int("{}", x);  
 println\_int("{}", x1);  
  
 let mix = integer1 - integer2 + *integer3*;  
 println\_int("{}", mix);  
 }  
  
 //comparing  
 {  
 let integer1: i32 = 42;  
 let integer2: i32 = 500;  
 let integer3: i32 = 100;  
 let is\_equal = integer1 == integer2;  
 let is\_not\_equal = integer1 != integer2;  
 let greater\_than = integer2 > integer1;  
 let lt = integer2 < integer1;  
 let gte = integer1 >= 42;  
 let less\_than\_or\_equal = integer1 <= 42;  
  
 println\_bool("{}", is\_equal);  
 println\_bool("{}", is\_not\_equal);  
 println\_bool("{}", greater\_than);  
 println\_bool("{}", lt);  
 println\_bool("{}", gte);  
 println\_bool("{}", less\_than\_or\_equal);  
 }  
  
 //shield string  
 {  
 let mut strg: String = r###"Это ##"raw"## строка с тройными кавычками"###;  
 println("{}", strg);  
 let raw = r"Это \0 \n \t \xFF строка с тройными кавычками";  
 println("{}", raw);  
 let escaped\_string = "Это строка с символами:\n \\n (новая строка),\t \\t (табуляция)\0 \\0 (терминальный ноль)";  
 println("{}", escaped\_string);  
 }  
  
 //char decl  
 {  
 let char1: char = 'A';  
 let char2: char = 'B';  
 let escaped\_char = '\n';  
 let unicode\_char = '\u{007\_\_7\_\_\_\_\_\_\_\_\_\_}';  
  
 println\_char("{}", char1);  
 println\_char("{}", char2);  
 println\_char("{}", escaped\_char);println\_char("{}", unicode\_char);  
  
 let newline = '\n'; // Новая строка  
 let tab = '\t'; // Табуляция  
 let backslash = '\\'; // Обратный слеш  
 let single\_quote = '\''; // Одинарная кавычка  
 let double\_quote = '\"'; // Двойная кавычка  
 let hex\_escape = '\x41'; // \xHH - ASCII символы в шестнадцатеричной форме \x7F - предел)  
 let unicode\_symb = '\u{0077}'; //\u{XXXX} - Unicode символы \u{10FFFF} - предел  
   
 println\_char("{}", newline);  
 println\_char("{}", tab);  
 println\_char("{}", backslash);  
 println\_char("{}", single\_quote);  
 println\_char("{}", double\_quote);  
 println\_char("{}", hex\_escape);  
 println\_char("{}", unicode\_symb);  
 }  
}  
  
fn gcd\_recursive(a: i32, b: i32) -> i32 {  
 if b == 0 {  
 a  
 } else {  
 gcd\_recursive(b, a % b)  
 }  
}  
  
trait Shape {  
 // У любой фрормы можно посчитать площадь.  
 fn area(&self) -> f64;  
  
 fn fun(&self) {  
 println("{}", "It`s is a fun function");  
 }  
}  
  
  
trait HasAngles: Shape {  
 // У любой фигуры с углами можно посчитать количество углов.  
 fn angles\_count(&self) -> i32;  
}  
  
struct Rectangle {  
 x: f64,  
 y: f64,  
}  
  
// Прямоугольник является формой.  
impl Shape for Rectangle {  
 fn area(&self) -> f64 {  
 self.x \* self.y  
 }  
}  
  
// Прямоугольник является фигурой с углами.  
impl HasAngles for Rectangle {  
 fn angles\_count(&self) -> i32 {  
 4  
 }  
}  
  
struct Circle {  
 r: f64,  
}  
  
// Круг является формой  
impl Shape for Circle {  
 fn area(&self) -> f64 {  
 self.r \* self.r \* 3.14 }  
}  
  
// Объявление перечислимого типа данных  
enum Color {  
 *Red* = 1,  
 *Green* = 2,  
 *Blue* = 7,  
}  
  
impl Color{  
 fn *getBright*(c : Color){  
 if (c == Color::*Blue*) {  
 println("{}", "Ярко");  
 } else {  
 println("{}", "Неярко");  
 }  
 }  
}

3) Таблица с распределением реализованной части

|  |  |
| --- | --- |
| Язык: | Rust |
| Участники: | Ломакин А., Чупинин А., ПрИн-466 |

ВСЕГО:

С претензией на доп: 47 +1 резерв

С ожидаемыми штрафами: 41 +1 резерв

\_\_  
1-резерв кому будет надо за области видимости

**Ломакин**: 3 2 3 2 min +1 за автофилл в массивах и владение в ф-ях(10 (12))

+ 3 3 1 1 1 2(1 енам)| (11 (10)) = 23 – лучший, 20 - худший

**Чупинин**: 4 3 3 min +2 за if и циклы в виде expr (10 (12))

+ 4 2 2 (1 статики) | (8(7))

+ 1 4 (5) = 25 – лучший, 21 - худший

|  |  |  |  |
| --- | --- | --- | --- |
| Возможность[[1]](#footnote-1) | Вес[[2]](#footnote-2) | Особенности в языке[[3]](#footnote-3) | Кто реализует |
| Локальные переменные встроенных типов данных:   * целые числа (один тип), * символы * строки   Выражения с использованием локальных переменных, арифметических операций (4 вида), операций сравнения и присваивания | 4 min  (+1?) | Можно не задавать тип данных в объявлении, приведение типов через оператор as | Чупинин |
| Одномерные массивы (или контейнеры),  операция доступа к элементу массива | 3 min  (+1?) | Автофилл | Ломакин |
| Числовые (целые числа), символьные и строковые константы (литералы) с поддержкой всех видов констант и служебных последовательностей символов | 2 min |  | Ломакин |
| Управляющие структуры: развилки | 3 min  (+1?) | Сложность – возврат значения | Чупинин |
| Управляющие структуры: циклы | 3 min (+1?) | Сложность – возврат значения так как expr. | Чупинин |
| Функции (процедуры) | 3 min  (+1) | Понятия владения – при передаче в функцию | Ломакин |
| Классы/функции для работы с консолью (ввод/вывод базовых типов данных) | 2 min |  | Ломакин |
| Классы, свойства и методы, одиночное наследование с динамическим связыванием | 4 | Наследование через интерфейсы | Чупинин |
| Переменные, константы и операции (арифметические, сравнения, присваивания) для чисел с плавающей точкой | 3 |  | Ломакин |
| Многомерные массивы | 3 | Автофилл | Ломакин |
| Логические операции  (И, ИЛИ, НЕ) | 1 |  | Ломакин |
| Операции над строками, их участие в выражения | 1 |  | Ломакин |
| Контроль доступа к свойствам и методам класса (открытые/ защищенные свойства и методы) | 1 | Сложность во взаимодействии с модулями | Ломакин/Чупинин |
| Конструкторы, деструкторы, операция динамического создания объекта | 2 |  | Чупинин |
| Перегрузка функций по типу параметров | 1 | Невозможно в Rust |  |
| Перечислимый тип данных | 2 | Возможность писать к нему функции | Ломакин |
| Статические свойства и методы классов | 2 |  | Чупинин |
| For-each, range | 1 |  | Чупинин |
| Области видимости переменной | 1 |  | РЕЗЕРВ |
| Модули | 3 |  | Чупинин |

4) Таблица лексем

**Перечень выделяемых лексем.**

|  |  |  |
| --- | --- | --- |
| **Вид лексемы** | **Семантическое значение** | **Описание и способ задания** |
| **Служебные лексемы (символы и операторы)** | | |
| { и } | Открывающая и закрывающая фигурные скобки | Используются для объявления блока кода, создания структур и блоков инициализации. Пример: { let x = 5; } |
| [ и ] | Открывающая и закрывающая квадратные скобки | Используются для создания массивов, векторов и доступа к элементам по индексу. Пример: let arr = [1, 2, 3]; |
| ( и ) | Открывающая и закрывающая круглые скобки | Используются при вызове функций и методов, а также в выражениях и группировке. Пример: println!("Hello, World!"); |
| < и > | Угловые скобки | Используются при определении параметров типа (generics) и создании жизненных времен (lifetimes). Пример: fn foo<T>(x: T) { ... } |
| : | Двоеточие | Используется при объявлении переменных с указанием их типов, а также при аннотации типов в сопоставлении с образцом и в структурах. Пример: let x: i32 = 42; |
| ; | Завершающая точка с запятой | Используется для завершения выражений, операторов и объявлений. Пример: let x = 5; |
| , | Запятая | Используется для разделения элементов в списке, параметров функции, полей структуры и др. Пример: fn foo(x: i32, y: i32) { ... } |
| :: | Оператор разрешения области | Используется для доступа к элементам модуля, ассоциированным функциям и типам структур. Пример: std::cmp::min(1, 2) |
| = | Оператор присваивания | Используется для присваивания значения переменной. Пример: let x = 5; |
| => | Оператор указания типа в match | Используется для указания типа в выражении match. Пример: match result { Ok(x) => x, Err(e) => e } |
| @ | Сопоставление с образцом с "аспектом" | Используется в выражении match для создания переменной, содержащей совпавший с образцом фрагмент данных. Пример: match value { Some(x) => println!("Got {:?}", x), None => println!("Got nothing") } |
| .. | Две точки | Могут использоваться в различных контекстах, таких как диапазоны, импорты и т. д. Например, 1..5, use std::collections::HashMap::\*, .. (в значении "все остальные") |
| ..= | Включительный диапазон | Используется для обозначения включительного диапазона значений. Например, 1..=5 включает в себя 1, 2, 3, 4 и 5. |
| ? | Оператор обработки ошибок | Используется в выражении для обработки ошибок и возвращения результата Result или Option. Пример: value? |
| **Имена или идентификаторы** | | |
| Идентификаторы | Названия переменных, функций, структур и т. д. | Идентификаторы должны начинаться с буквы или подчеркивания, за которыми могут следовать буквы, цифры и подчеркивания. Например, my\_variable, value123. |
| Имена структур | Названия пользовательских структур данных | Имена структур обычно начинаются с заглавной буквы и следуют за структурой struct. Например, Person, Point. |
| Имена перечислений | Названия пользовательских перечислений | Имена перечислений обычно начинаются с заглавной буквы и следуют за перечислением enum. Например, Color, Direction. |
| Имена модулей | Названия модулей и подмодулей | Имена модулей обычно следуют синтаксису иерархии пакетов. Например, my\_module, submodule. |
| Имена типов | Названия пользовательских типов данных | Имена типов обычно начинаются с заглавной буквы и используются для создания новых типов данных. Например, MyStruct, Result. |
| Имена функций | Названия пользовательских функций | Имена функций обычно начинаются с маленькой буквы и следуют за ключевым словом fn. Например, add\_numbers, calculate. |
| Имена констант | Названия констант и статических переменных | Имена констант обычно записываются заглавными буквами и разделяются подчеркиваниями. Например, PI, MAX\_VALUE. |
| Имена методов | Названия методов в структурах и перечислениях | Имена методов обычно начинаются с маленькой буквы и используются в контексте объектно-ориентированного программирования. Например, calculate\_area(), to\_string(). |
| **Знаки операций** | | |
| + | Сложение | Используется для выполнения арифметической операции сложения. Например, x + y. |
| - | Вычитание | Используется для выполнения арифметической операции вычитания. Например, x - y. |
| \* | Умножение | Используется для выполнения арифметической операции умножения. Например, x \* y. |
| / | Деление | Используется для выполнения арифметической операции деления. Например, x / y. |
| % | Остаток от деления | Используется для получения остатка от деления двух чисел. Например, x % y. |
| == | Сравнение на равенство | Используется для проверки равенства двух значений. Например, x == y. |
| != | Сравнение на неравенство | Используется для проверки неравенства двух значений. Например, x != y. |
| < | Меньше чем | Используется для проверки, является ли значение слева от оператора меньшим чем значение справа. Например, x < y. |
| > | Больше чем | Используется для проверки, является ли значение слева от оператора большим чем значение справа. Например, x > y. |
| <= | Меньше чем или равно | Используется для проверки, является ли значение слева от оператора меньшим или равным значению справа. Например, x <= y. |
| >= | Больше чем или равно | Используется для проверки, является ли значение слева от оператора большим или равным значению справа. Например, x >= y. |
| = | Присваивание | Используется для присваивания значения переменной. Например, x = y. |
| += | Присваивание с добавлением | Используется для сокращенной записи операции сложения с присваиванием. Например, x += y эквивалентно x = x + y. |
| -= | Присваивание с вычитанием | Используется для сокращенной записи операции вычитания с присваиванием. Например, x -= y эквивалентно x = x - y. |
| \*= | Присваивание с умножением | Используется для сокращенной записи операции умножения с присваиванием. Например, x \*= y эквивалентно x = x \* y. |
| /= | Присваивание с делением | Используется для сокращенной записи операции деления с присваиванием. Например, x /= y эквивалентно x = x / y. |
| %= | Присваивание с остатком | Используется для сокращенной записи операции взятия остатка с присваиванием. Например, x %= y эквивалентно x = x % y. |
| ! | Логическое "не" | Используется для выполнения логической операции "не" (отрицания) над булевым значением. Например, !x. |
| && | Логическое "и" | Используется для выполнения логической операции "и" (логическое "и") над булевыми значениями. Например, x && y. |
| | | Вертикальная черта | Побитовое "или" и шаблон в сопоставлении с образцом.  let x = 0b1010; // Двоичное представление: 1010  let y = 0b1100; // Двоичное представление: 1100  let result = x | y; // Результат: 1110 (в двоичной системе)  match value { 0 | 1 => println!(“A”)} |
| & | Побитовое "и" и ссылка | Используется для побитовой операции "и" и создания ссылок на значения. Пример: let x = &y; |
| ^ | Побитовое "исключающее или" | Используется для побитовой операции "исключающее или". Пример: x ^ y |
| ! | Логическое "не" и оператор отрицания | Используется для логической операции "не" и в качестве оператора отрицания. Пример: !condition |
| && | Логическое "и" | Используется для логической операции "и". Пример: if x && y { ... } |
| **Константы** | | |
| () | Значение типа () | Тип данных () представляет собой кортеж без элементов и используется для представления "пустого" значения. |
| char | Знак Unicode | Константа типа char представляет собой один символ Unicode. Задается с использованием одинарных кавычек. |
| i8 | 8-битное целое число | Константа типа i8 представляет собой 8-битное знаковое целое число. Пример: let x: i8 = -42; |
| i16 | 16-битное целое число | Константа типа i16 представляет собой 16-битное знаковое целое число. Пример: let x: i16 = 1000; |
| i32 | 32-битное целое число | Константа типа i32 представляет собой 32-битное знаковое целое число. Пример: let x: i32 = -12345; |
| i64 | 64-битное целое число | Константа типа i64 представляет собой 64-битное знаковое целое число. Пример: let x: i64 = 9876543210; |
| u8 | 8-битное целое число | Константа типа u8 представляет собой 8-битное беззнаковое целое число. Пример: let x: u8 = 255; |
| u16 | 16-битное целое число | Константа типа u16 представляет собой 16-битное беззнаковое целое число. Пример: let x: u16 = 65535; |
| u32 | 32-битное целое число | Константа типа u32 представляет собой 32-битное беззнаковое целое число. Пример: let x: u32 = 4294967295; |
| u64 | 64-битное целое число | Константа типа u64 представляет собой 64-битное беззнаковое целое число. Пример: let x: u64 = 18446744073709551615; |
| f32 | 32-битное число с плавающей точкой | Константа типа f32 представляет собой 32-битное число с плавающей точкой (float). Пример: let x: f32 = 3.14; |
| f64 | 64-битное число с плавающей точкой | Константа типа f64 представляет собой 64-битное число с плавающей точкой (double). Пример: let x: f64 = 2.71828; |
| **Разделители** | | |
| ; | Знак точки с запятой | Разделитель для завершения выражений и инструкций. В конце каждой строки кода обычно ставится ;. |
| , | Запятая | Используется для разделения элементов в списке или параметрах функции. Например, let numbers = [1, 2, 3];. |
| : | Двоеточие | Используется при определении типов переменных и параметров функций, а также при создании меток (лейблов) в коде. |
| . | Точка | Используется для доступа к членам структур, перечислений и модулей. Например, my\_struct.field или my\_module::my\_function. |
| :: | Двойное двоеточие | Используется для доступа к элементам и методам модулей, структур и перечислений. Например, std::io::stdin(). |
| => | Стрелка | Используется в сопоставлении с образцом (pattern matching) для связывания значения с шаблоном. |
| = | Знак равенства | Используется для присваивания значения переменной. Например, let x = 42;. |
| -> | Стрелка справа налево | Используется при определении типов возвращаемых функцией значений. Например, fn add(x: i32, y: i32) -> i32. |
| @ | Знак собачки | Используется в сопоставлении с образцом для создания переменной, содержащей всю сопоставляемую структуру. |
| ; | Точка с запятой | Разделитель для завершения выражений и инструкций в блоках кода (например, в блоке for или if). |
| -> | Стрелка справа налево | Используется при определении типов возвращаемых функцией значений. Например, fn add(x: i32, y: i32) -> i32. |
| ; | Точка с запятой | Разделитель для завершения выражений и инструкций в блоках кода (например, в блоке for или if). |
| {} | Фигурные скобки | Используются для создания блоков кода и определения области видимости переменных. |
| **Комментарии** | | |
| Однострочный комментарий (//) | Игнорируется компилятором | Используется для добавления комментариев в код. Все текст после // до конца строки считается комментарием и игнорируется компилятором. Пример: // Это комментарий. |
| Многострочный комментарий (/\* \*/) | Игнорируется компилятором | Используется для создания многострочных комментариев. Все текст, заключенный между /\* и \*/, считается комментарием и игнорируется компилятором. Пример: /\* Многострочный комментарий \*/. |
| Документирующий комментарий (///) | Используется для документирования | Используется для создания документирующих комментариев, которые могут быть автоматически извлечены с помощью инструментов документации, таких как Rustdoc. Пример: /// Этот комментарий будет документирован. |
| Документирующий многострочный комментарий (/\*\* \*/) | Используется для документирования | Аналогичен документирующему комментарию ///, но может включать многострочные описания и дополнительные детали. Пример: /\*\* Этот комментарий будет документирован. \*/. |
| **Все ключевые слова** | | |
| as | Псевдоним, приведение типов | Используется для явного приведения типов или создания псевдонимов (алиасов) для типов данных. Пример: let x = 42 as f64;. |
| async | Асинхронный код | Объявляет асинхронную функцию или блок кода. Используется в асинхронном программировании. |
| await | Ожидание завершения асинхронной операции | Используется для ожидания завершения асинхронной операции в асинхронной функции. |
| break | Выход из цикла | Используется для прерывания выполнения цикла и выхода из него. |
| const | Константа | Объявляет константу с фиксированным значением, которое нельзя изменять после объявления. |
| continue | Продолжить следующую итерацию цикла | Используется для пропуска текущей итерации цикла и перехода к следующей. |
| crate | Корневой модуль | Используется для обращения к корневому модулю текущего крейта (проекта). |
| dyn | Динамический тип | Используется в контексте работы с трейтами для указания на динамический тип данных (позднее связывание). |
| else | Ветвь else условного оператора | Используется в конструкции if-else для выполнения кода, если условие в if не выполняется. |
| enum | Перечисление | Объявляет перечисление (enum) для создания пользовательских типов данных с фиксированным списком значений. |
| extern | Внешний код, FFI | Используется для объявления внешних функций и интерфейсов для взаимодействия с кодом на других языках. |
| false | Ложное значение булевого типа | Представляет ложное логическое значение false. |
| fn | Функция | Объявляет функцию. |
| for | Цикл for | Используется для создания цикла for, который итерируется по элементам коллекции или диапазону значений. |
| if | Условный оператор if | Используется для создания условного оператора if, который выполняет код в зависимости от условия. |
| impl | Реализация | Используется для реализации методов и трейтов для типов данных. |
| in | Проверка вхождения | Используется для проверки вхождения значения в коллекцию. |
| let | Объявление переменной | Используется для объявления переменной и присваивания ей значения. |
| loop | Бесконечный цикл | Создает бесконечный цикл, который выполняется до явного прерывания. |
| match | Сопоставление с образцом | Используется для сопоставления значения с шаблонами и выполнения соответствующего кода. |
| mod | Модуль | Объявляет модуль, который используется для организации кода в логические блоки. |
| move | Захват переменных | Используется для захвата переменных и передачи их внутрь замыканий. |
| mut | Изменяемость переменной | Объявляет переменную с возможностью изменения значения после объявления. |
| pub | Публичный доступ | Устанавливает публичный доступ к элементам модуля, структуры или функции. |
| ref | Ссылка на переменную | Используется для создания ссылок на переменные. |
| return | Возврат из функции | Используется для возврата значения из функции. |
| self | Текущий объект | Используется для обращения к текущему объекту или типу внутри методов. |
| Self | Текущий тип | Используется для обращения к текущему типу внутри методов. |
| static | Статическая переменная | Объявляет статическую переменную, которая существует на протяжении всего времени выполнения программы. |
| struct | Структура | Объявляет пользовательский тип данных структура (struct), состоящий из полей. |
| super | Родительский модуль | Используется для обращения к родительскому модулю из текущего модуля. |
| trait | Трейт | Объявляет трейт, который определяет интерфейс для типов данных. |
| true | Истинное значение булевого типа | Представляет истинное логическое значение true. |
| type | Псевдоним типа | Используется для создания псевдонимов типов данных. |
| unsafe | Небезопасный код | Обозначает блок кода, который может содержать небезопасные операции и обращения к указателям. |
| use | Импорт | Используется для импорта символов из других модулей или областей видимости. |
| where | Ограничение типов | Используется в контексте ограничений типов для указания допустимых типов данных. |
| while | Цикл while | Используется для создания цикла while, который выполняется, пока условие истинно. |
| box | Умный указатель на кучу (временно устарело) | Используется для создания умных указателей на кучу, но в текущей версии Rust лучше использовать Box. |
| override | Переопределение метода | Используется для переопределения метода в подклассе (структуре или enum). |

5) Грамматика языка

%{  
 #include <iostream>  
 #include "classesnodes.h"  
 void yyerror(char const\* s);  
 extern int yylex(void);  
 using namespace std;  
 ProgramNode\* global\_program;  
%}  
  
%union {  
 int int\_literal;  
 string\* string\_literal;  
 string\* identifier;  
 bool bool\_literal;  
 float float\_literal;  
 char char\_literal;  
 enum Visibility vis;  
  
 ProgramNode\* prg;  
 ExprNode\* expr;  
 ExprListNode \* expr\_list;  
 StmtNode\* stmt;  
 StmtListNode\* stmt\_list;  
 LetStmtNode\* let\_stmt;  
 ModuleStmtNode\* mod\_stmt;  
 ItemNode\* item;  
 ItemListNode\* item\_list;  
 StructFieldNode\* struct\_item;  
 StructFieldListNode\* struct\_items;  
 StructStructNode\* struct\_;  
 EnumStmtNode\* enum\_stmt;  
 EnumItemNode\* enum\_item;  
 EnumItemListNode\* enum\_items;  
 FuncStmtNode\* function\_stmt;  
 FuncParamNode\* function\_param;  
 FuncParamListNode\* function\_params;  
 ConstStmtNode\* const\_stmt;  
 TraitNode\* trait;  
 ImplStmtNode\* impl\_stmt;  
 TypeNode\* type;  
}  
  
%type <prg>Program  
%type <expr>ExprWithoutBlock  
%type <expr>ExprWithBlock  
%type <expr>BlockExpr  
%type <expr>IfExpr  
%type <expr>SimpleIfElseExpr  
%type <expr>SimpleIfExpr  
%type <expr>InfiniteLoopExpr  
%type <expr>LoopExpr  
%type <expr>PredicateLoopExpr  
%type <expr>IteratorLoopExpr  
%type <expr>PathCallExpr  
%type <expr>StructExprField  
%type <expr\_list>ExprList  
%type <expr\_list>ExprListEmpty  
%type <expr\_list>StructExprFieldList  
%type <stmt>Stmt  
%type <stmt>ExprStmt  
%type <stmt\_list>StmtList  
%type <let\_stmt>LetStmt  
%type <mod\_stmt>ModuleStmt  
%type <item>Item  
%type <item>SimpleItem  
%type <item\_list>ItemList  
%type <item\_list>ItemListEmpty  
%type <struct\_>StructStruct  
%type <struct\_>StructStmt  
%type <struct\_>TupleStruct  
%type <struct\_item>StructField  
%type <struct\_items>StructFieldList  
%type <struct\_items>StructFieldListEmpty  
%type <struct\_items>TupleFieldList  
%type <struct\_items>TupleFieldListEmpty  
%type <enum\_stmt>EnumStmt  
%type <enum\_item>EnumItem  
%type <enum\_items>EnumItemList  
%type <enum\_items>EnumItemListEmpty  
%type <function\_stmt>FuncStmt  
%type <function\_stmt>ImplFuncStmt  
%type <function\_stmt>DecFuncStmt  
%type <function\_param>FuncParam  
%type <function\_params>FuncParamList  
%type <function\_params>FuncParamListEmpty  
%type <const\_stmt>ConstStmt  
%type <item>AssociatedItem  
%type <item\_list>AssociatedItemList  
%type <item\_list>AssociatedItemListEmpty  
%type <impl\_stmt>ImplStmt  
%type <trait>TraitStmt  
%type <type>Type  
%type <vis>Visibility  
  
%type <int\_literal>INT\_LITERAL  
%type <string\_literal>STRING\_LITERAL  
%type <string\_literal>RAW\_STRING\_LITERAL  
%type <identifier>ID  
%type <bool\_literal>TRUE  
%type <bool\_literal>FALSE  
%type <float\_literal>FLOAT\_LITERAL  
%type <char\_literal>CHAR\_LITERAL  
  
  
%token BOOL CHAR FLOAT INT STRING  
%token TRUE FALSE CHAR\_LITERAL INT\_LITERAL FLOAT\_LITERAL STRING\_LITERAL RAW\_STRING\_LITERAL  
%token ID  
%token FOR LOOP WHILE IN CONTINUE  
%token IF ELSE  
%token LET MUT CONST  
%token FN ENUM STRUCT TRAIT  
%token IMPL SELF BIG\_SELF PUB SELF\_REF MUT\_SELF\_REF MUT\_REF MOD SUPER CRATE  
%token ';' RIGHT\_ARROW  
  
/\* BREAK и RETURN в документации почему-то присуствует в приоритетах операций. Стоит наверное с этим разобраться \*/  
  
%nonassoc BREAK RETURN  
%nonassoc '{' '}'  
%right ':'  
%right '='  
%nonassoc RANGE /\* .. \*/  
%left OR /\* || \*/  
%left AND /\* && \*/  
%left '<' '>' EQUAL NOT\_EQUAL LESS\_EQUAL GREATER\_EQUAL // == != <= >=  
%left '+' '-'  
%left '\*' '/' '%'  
%left AS  
%left '!' '&' MUT\_REF UMINUS USTAR /\* - \* \*/  
%nonassoc '?'  
%left '.' '[' DOUBLEDOTS  
%nonassoc '(' ')'  
  
%start Program  
  
%%  
  
  
/\* ---------------------- PROGRAM --------------------------- \*/  
  
Program: ItemListEmpty { $$ = global\_program = new ProgramNode($1); }  
/\* Необходимо уточнить, надо ли как-то обозначить, что Stmt все должны быть Item, иначе программа не заработает \*/  
  
/\* ----------------------------- STATEMENT ----------------------------- \*/  
  
  
StmtList: Stmt { $$ = new StmtListNode($1); }  
 | StmtList Stmt { $$ = StmtListNode::Append($1, $2); }  
 ;  
  
Stmt: ';' { $$ = new StmtNode(StmtNode::semicolon, NULL, NULL, NULL); }  
 | LetStmt { $$ = new StmtNode(StmtNode::let, NULL, NULL, $1); }  
 | ExprStmt { $$ = $1;}  
 | ConstStmt { $$ = StmtNode::ConstStmtToStmt($1)}  
 ;  
  
ItemListEmpty: /\*empty\*/ { $$ = 0; }  
 | ItemList { $$ = new ItemListNode($1); }  
 ;  
  
ItemList: Item { $$ = new ItemListNode($1); }  
 | ItemList Item { $$ = ItemListNode::Append($1, $2); }  
 ;  
  
Item: SimpleItem { $$ = $1;}  
 | Visibility SimpleItem { $$ = ItemNode::AddVisibility($1, $2);}  
 ;  
  
SimpleItem: FuncStmt { $$ = ItemNode::DeclarationFunction(self, $1); }  
 | StructStmt { $$ = ItemNode::DeclarationStruct(self, $1); }  
 | EnumStmt { $$ = ItemNode::DeclarationEnum(self, $1); }  
 | ImplStmt { $$ = ItemNode::DeclarationImpl(self, $1); }  
 | TraitStmt { $$ = ItemNode::DeclarationTrait(self, $1); }  
 | ConstStmt { $$ = ItemNode::DeclarationConst(self, $1); }  
 | ModuleStmt { $$ = ItemNode::DeclarationModule(self, $1); }  
 ;  
  
/\* ---------- Function ------------ \*/  
  
FuncStmt: DecFuncStmt { $$ = $1; }  
 | ImplFuncStmt { $$ = $1; }  
 ;  
  
DecFuncStmt: FN ID '(' FuncParamListEmpty ')' ';' { $$ = new FuncStmtNode($2, 0, $4, 0); }  
 | FN ID '(' FuncParamListEmpty ')' RIGHT\_ARROW Type ';' { $$ = new FuncStmtNode($2, $7, $4, 0); }  
 | FN ID '(' FuncParamListEmpty ')' RIGHT\_ARROW IMPL Type ';' { $8->isImpl = true; $$ = new FuncStmtNode($2, $8, $4, 0); }  
 ;  
  
ImplFuncStmt: FN ID '(' FuncParamListEmpty ')' BlockExpr { $$ = new FuncStmtNode($2, 0, $4, $6); }  
 | FN ID '(' FuncParamListEmpty ')' RIGHT\_ARROW Type BlockExpr { $$ = new FuncStmtNode($2, $7, $4, $8); }  
 | FN ID '(' FuncParamListEmpty ')' RIGHT\_ARROW IMPL Type BlockExpr { $8->isImpl = true; $$ = new FuncStmtNode($2, $8, $4, $9); }  
 ;  
  
FuncParamListEmpty: /\* empty \*/ { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::static\_, 0); }  
 | FuncParamList { $$ = $1; }  
 ;  
  
FuncParamList: SELF { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::self, 0); }  
 | SELF\_REF { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::self\_ref, 0); }  
 | MUT\_SELF\_REF { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::mut\_self\_ref, 0); }  
 | FuncParam { $$ = new FuncParamListNode(FuncParamListNode::static\_, $1); }  
 | FuncParamList ',' FuncParam { $$ = FuncParamListNode::Append($1, $3); }  
 ;  
  
FuncParam: ID ':' Type {$$ = new FuncParamNode($1, $3, FuncParamNode::noMut); }  
 | ID ':' IMPL Type {$4->isImpl = true; $$ = new FuncParamNode($1, $4, FuncParamNode::noMut); }  
 | MUT ID ':' Type { $$ = new FuncParamNode($2, $4, FuncParamNode::mut); }  
 | MUT ID ':' IMPL Type {$5->isImpl = true; $$ = new FuncParamNode($2, $5, FuncParamNode::mut); }  
 | ID ':' MUT\_REF Type { $$ = new FuncParamNode($1, $4, FuncParamNode::mut\_ref); }  
 | ID ':' MUT\_REF IMPL Type { $5->isImpl = true; $$ = new FuncParamNode($1, $5, FuncParamNode::mut\_ref); }  
 | ID ':' '&' Type { $$ = new FuncParamNode($1, $4, FuncParamNode::link); }  
 | ID ':' '&' IMPL Type { $5->isImpl = true; $$ = new FuncParamNode($1, $5, FuncParamNode::link); }  
 ;  
  
/\* ========== Struct =========== \*/  
  
StructStmt: StructStruct { $$ = $1; }  
 | TupleStruct { $$ = $1; }  
 ;  
  
StructStruct : STRUCT ID '{' StructFieldListEmpty '}' { $$ = new StructStructNode($2, $4); }  
 | STRUCT ID ';' { $$ = new StructStructNode($2, 0); }  
 ;  
  
StructFieldListEmpty: /\* empty \*/ { $$ = 0; }  
 | StructFieldList { $$ = new StructFieldListNode($1); }  
 | StructFieldList ',' { $$ = new StructFieldListNode($1); }  
 ;  
  
StructFieldList: StructField { $$ = new StructFieldListNode($1); }  
 | StructFieldList ',' StructField { $$ = StructFieldListNode::Append($1, $3); }  
 ;  
  
StructField: ID ':' Type { $$ = new StructFieldNode($1, $3, self); }  
 | Visibility ID ':' Type { $$ = new StructFieldNode($2, $4, $1); }  
 ;  
  
/\*--- TupleStruct ----\*/  
  
TupleStruct: STRUCT ID '(' TupleFieldListEmpty ')' { $$ = new StructStructNode($2, $4); }  
 ;  
  
TupleFieldListEmpty: /\* empty \*/ { $$ = 0; }  
 | TupleFieldList { $$ = new StructFieldListNode($1); }  
 | TupleFieldList ',' { $$ = new StructFieldListNode($1); }  
 ;  
  
TupleFieldList: Type { $$ = new StructFieldListNode(new StructFieldNode(0, $1, self)); }  
 | Visibility Type { $$ = new StructFieldListNode(new StructFieldNode(0, $2, $1)); }  
 | TupleFieldList ',' Type { $$ = StructFieldListNode::Append($1, new StructFieldNode(0, $3, self)); }  
 | TupleFieldList ',' Visibility Type { $$ = StructFieldListNode::Append($1, new StructFieldNode(0, $4, $3)); }  
 ;  
  
/\* ============= ENUM ================ \*/  
  
EnumStmt: ENUM ID '{' EnumItemListEmpty '}' { $$ = new EnumStmtNode($2, $4); }  
 ;  
  
EnumItemListEmpty: /\* empty \*/ { $$ = 0; }  
 | ',' { $$ = 0; }  
 | EnumItemList { $$ = new EnumItemListNode($1); }  
 | EnumItemList ',' { $$ = new EnumItemListNode($1); }  
 ;  
  
EnumItemList: EnumItem { $$ = new EnumItemListNode($1); }  
 | EnumItemList ',' EnumItem { $$ = EnumItemListNode::Append($1, $3); }  
 ;  
  
EnumItem: ID { $$ = new EnumItemNode($1, self, 0, 0); }  
 | Visibility ID { $$ = new EnumItemNode($2, $1, 0, 0); }  
 | ID '=' ExprWithBlock { $$ = new EnumItemNode($1, self, NULL, $3); } /\* В таком случае ID должен быть всегда только целочисленным числом. Нельзя на парсере определить такое\*/  
 | ID '=' ExprWithoutBlock { $$ = new EnumItemNode($1, self, 0, $3); }  
 | Visibility ID '=' ExprWithBlock { $$ = new EnumItemNode($2, $1, 0, $4); }  
 | Visibility ID '=' ExprWithoutBlock { $$ = new EnumItemNode($2, $1, 0, $4); }  
 | Visibility ID '{' StructFieldListEmpty '}' { $$ = new EnumItemNode($2, $1, $4, 0); }  
 | ID '{' StructFieldListEmpty '}' { $$ = new EnumItemNode($1, self, $3, 0); }  
 ;  
  
/\* =========== IMPL ================ \*/  
  
ImplStmt: IMPL Type '{'AssociatedItemListEmpty '}' { $$ = new ImplStmtNode(ImplStmtNode::inherent, $2, 0, $4); }  
 | IMPL PathCallExpr FOR Type '{'AssociatedItemListEmpty '}' { $$ = new ImplStmtNode(ImplStmtNode::trait, $4, $2, $6); }  
 ;  
  
AssociatedItemListEmpty: /\* empty \*/ { $$ = NULL; }  
 | AssociatedItemList { $$ = $1; }  
 ;  
  
AssociatedItemList: AssociatedItem { $$ = $$ = new ItemListNode($1); }  
 | AssociatedItemList AssociatedItem { $$ = $$ = ItemListNode::Append($1, $2); }  
 ;  
  
/\* Необходима еще проверка для Impl то что FuncStmt является именно реализацией \*/  
AssociatedItem: FuncStmt { $$ = ItemNode::DeclarationFunction(self, $1); } // ImplFuncStmt  
 | ConstStmt { $$ = ItemNode::DeclarationConst(self, $1); }  
 | Visibility FuncStmt { $$ = ItemNode::DeclarationFunction($1, $2); }  
 | Visibility ConstStmt { $$ = ItemNode::DeclarationConst($1, $2); }  
 ;  
  
  
/\* ============ TRAIT ================ \*/  
  
TraitStmt: TRAIT ID '{' AssociatedItemListEmpty '}' { $$ = new TraitNode($2, $4); }  
 | TRAIT ID ':' ID '{' AssociatedItemListEmpty '}' { $$ = new TraitNode($2, $6, $4); }  
 ;  
  
/\* ============ CONST =============== \*/  
  
ConstStmt: CONST ID ':' Type '=' ExprWithBlock ';' { $$ = ConstStmtNode::ConstStmt($2, $4, $6); }  
 | CONST ID ':' Type '=' ExprWithoutBlock ';' { $$ = ConstStmtNode::ConstStmt($2, $4, $6); }  
 | CONST ID ':' Type ';' { $$ = ConstStmtNode::ConstStmt($2, $4, 0); }  
 ;  
  
/\* =========== Module ================= \*/  
  
ModuleStmt: MOD ID '{' ItemListEmpty '}' { $$ = new ModuleStmtNode($2, $4); }  
 ;  
  
/\* ========= LetStmt ============ \*/  
LetStmt: LET ID '=' ExprWithBlock ';' { $$ = new LetStmtNode($2, 0, LetStmtNode::noMut, $4); }  
 | LET ID '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($2, 0, LetStmtNode::noMut, $4); }  
 | LET ID ':' Type '=' ExprWithBlock ';' { $$ = new LetStmtNode($2, $4, LetStmtNode::noMut, $6); }  
 | LET ID ':' Type '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($2, $4, LetStmtNode::noMut, $6); }  
 | LET MUT ID ';' { $$ = new LetStmtNode($3, 0, LetStmtNode::mut, 0); }  
 | LET MUT ID ':' Type ';' { $$ = new LetStmtNode($3, $5, LetStmtNode::mut, 0); }  
 | LET MUT ID '=' ExprWithBlock ';' { $$ = new LetStmtNode($3, 0, LetStmtNode::mut, $5); }  
 | LET MUT ID '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($3, 0, LetStmtNode::mut, $5); }  
 | LET MUT ID ':' Type '=' ExprWithBlock ';' { $$ = new LetStmtNode($3, $5, LetStmtNode::mut, $7); }  
 | LET MUT ID ':' Type '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($3, $5, LetStmtNode::mut, $7); }  
 ;  
  
/\* === Expression Statement === \*/  
ExprStmt: ExprWithoutBlock ';' {$$ = new StmtNode(StmtNode::exprstmt, $1, 0, 0);}  
 | ExprWithBlock {$$ = new StmtNode(StmtNode::exprstmt, $1, 0, 0);}  
 ;  
  
  
/\*----------------------- EXPRESSION ---------------------- \*/  
  
ExprListEmpty: /\*empty\*/ { $$ = 0; }  
 | ExprList ',' { $$ = new ExprListNode($1); }  
 | ExprList { $$ = new ExprListNode($1); }  
 ;  
  
ExprList: ExprWithBlock { $$ = new ExprListNode($1); }  
 | ExprWithoutBlock { $$ = new ExprListNode($1); }  
 | ExprList ',' ExprWithBlock { $$ = ExprListNode::Append($1, $3); }  
 | ExprList ',' ExprWithoutBlock { $$ = ExprListNode::Append($1, $3); }  
 ;  
  
ExprWithoutBlock: CHAR\_LITERAL { $$ = ExprNode::ExprFromCharLiteral(ExprNode::char\_lit, $1); }  
 | STRING\_LITERAL { $$ = ExprNode::ExprFromStringLiteral(ExprNode::string\_lit, $1); }  
 | RAW\_STRING\_LITERAL { $$ = ExprNode::ExprFromStringLiteral(ExprNode::raw\_string\_lit, $1); }  
 | INT\_LITERAL { $$ = ExprNode::ExprFromIntLiteral(ExprNode::int\_lit, $1); }  
 | FLOAT\_LITERAL { $$ = ExprNode::ExprFromFloatLiteral(ExprNode::float\_lit, $1); }  
 | TRUE { $$ = ExprNode::ExprFromBoolLiteral(ExprNode::bool\_lit, $1); }  
 | FALSE { $$ = ExprNode::ExprFromBoolLiteral(ExprNode::bool\_lit, $1); }  
 | ExprWithoutBlock '+' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }  
 | ExprWithoutBlock '+' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }  
 | ExprWithBlock '+' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }  
 | ExprWithBlock '+' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }  
 | ExprWithoutBlock '-' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }  
 | ExprWithoutBlock '-' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }  
 | ExprWithBlock '-' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }  
 | ExprWithBlock '-' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }  
 | ExprWithoutBlock '/' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }  
 | ExprWithoutBlock '/' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }  
 | ExprWithBlock '/' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }  
 | ExprWithBlock '/' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }  
 | ExprWithoutBlock '\*' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }  
 | ExprWithoutBlock '\*' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }  
 | ExprWithBlock '\*' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }  
 | ExprWithBlock '\*' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }  
 | ExprWithoutBlock '%' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }  
 | ExprWithoutBlock '%' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }  
 | ExprWithBlock '%' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }  
 | ExprWithBlock '%' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }  
 | ExprWithoutBlock AND ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }  
 | ExprWithoutBlock AND ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }  
 | ExprWithBlock AND ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }  
 | ExprWithBlock AND ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }  
 | ExprWithoutBlock OR ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }  
 | ExprWithoutBlock OR ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }  
 | ExprWithBlock OR ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }  
 | ExprWithBlock OR ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }  
 | ExprWithoutBlock '=' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }  
 | ExprWithoutBlock '=' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }  
 | ExprWithBlock '=' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }  
 | ExprWithBlock '=' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }  
 | ExprWithoutBlock EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }  
 | ExprWithoutBlock EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }  
 | ExprWithBlock EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }  
 | ExprWithBlock EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }  
 | ExprWithoutBlock NOT\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }  
 | ExprWithoutBlock NOT\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }  
 | ExprWithBlock NOT\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }  
 | ExprWithBlock NOT\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }  
 | ExprWithoutBlock '>' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }  
 | ExprWithoutBlock '>' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }  
 | ExprWithBlock '>' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }  
 | ExprWithBlock '>' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }  
 | ExprWithoutBlock '<' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }  
 | ExprWithoutBlock '<' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }  
 | ExprWithBlock '<' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }  
 | ExprWithBlock '<' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }  
 | ExprWithoutBlock GREATER\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }  
 | ExprWithoutBlock GREATER\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }  
 | ExprWithBlock GREATER\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }  
 | ExprWithBlock GREATER\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }  
 | ExprWithoutBlock LESS\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }  
 | ExprWithoutBlock LESS\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }  
 | ExprWithBlock LESS\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }  
 | ExprWithBlock LESS\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }  
 | '-' ExprWithoutBlock %prec UMINUS { $$ = ExprNode::OperatorExpr(ExprNode::uminus, $2, 0); }  
 | '-' ExprWithBlock %prec UMINUS { $$ = ExprNode::OperatorExpr(ExprNode::uminus, $2, 0); }  
 | '!' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::negotation, $2, 0); }  
 | '!' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::negotation, $2, 0); }  
 | ExprWithoutBlock '?' { $$ = ExprNode::OperatorExpr(ExprNode::question, $1, 0); }  
 | ExprWithBlock '?' { $$ = ExprNode::OperatorExpr(ExprNode::question, $1, 0); }  
 | '\*' ExprWithoutBlock %prec USTAR { $$ = ExprNode::OperatorExpr(ExprNode::ustar, $2, 0); }  
 | '\*' ExprWithBlock %prec USTAR { $$ = ExprNode::OperatorExpr(ExprNode::ustar, $2, 0); }  
 | '&' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::link, $2, 0); }  
 | '&' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::link, $2, 0); }  
 | MUT\_REF ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mut\_link, $2, 0); }  
 | MUT\_REF ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mut\_link, $2, 0); }  
 | '[' ExprListEmpty ']' { $$ = ExprNode::ArrExprFromList(ExprNode::array\_expr, $2); }  
 | '[' ExprWithoutBlock ';' ExprWithoutBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }  
 | '[' ExprWithoutBlock ';' ExprWithBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }  
 | '[' ExprWithBlock ';' ExprWithoutBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }  
 | '[' ExprWithBlock ';' ExprWithBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }  
 | ExprWithoutBlock '[' ExprWithoutBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }  
 | ExprWithoutBlock '[' ExprWithBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }  
 | ExprWithBlock '[' ExprWithoutBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }  
 | ExprWithBlock '[' ExprWithBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }  
 | ExprWithoutBlock '.' INT\_LITERAL { $$ = ExprNode::TupleExpr(ExprNode::tuple\_expr, $1, $3); }  
 | ExprWithBlock '.' INT\_LITERAL { $$ = ExprNode::TupleExpr(ExprNode::tuple\_expr, $1, $3); }  
 | CONTINUE { $$ = ExprNode::OperatorExpr(ExprNode::continue\_expr, 0, 0); }  
 | BREAK { $$ = ExprNode::OperatorExpr(ExprNode::break\_expr, 0, 0); }  
 | BREAK ExprWithoutBlock{ $$ = ExprNode::OperatorExpr(ExprNode::break\_with\_val\_expr, $2, 0); }  
 | BREAK ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::break\_with\_val\_expr, $2, 0); }  
 | RANGE { $$ = ExprNode::RangeExpr(ExprNode::range\_right, 0, 0); }  
 | RANGE ExprWithoutBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_right, $2, 0); }  
 | RANGE ExprWithBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_right, $2, 0); }  
 | ExprWithoutBlock RANGE { $$ = ExprNode::RangeExpr(ExprNode::range\_left, $1, 0); }  
 | ExprWithBlock RANGE { $$ = ExprNode::RangeExpr(ExprNode::range\_left, $1, 0); }  
 | ExprWithoutBlock RANGE ExprWithoutBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }  
 | ExprWithoutBlock RANGE ExprWithBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }  
 | ExprWithBlock RANGE ExprWithoutBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }  
 | ExprWithBlock RANGE ExprWithBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }  
 | RETURN { $$ = ExprNode::OperatorExpr(ExprNode::return\_expr, 0, 0); }  
 | RETURN ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::return\_expr, $2, 0); }  
 | RETURN ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::return\_expr, $2, 0); }  
 | ExprWithoutBlock '.' ID { $$ = ExprNode::CallAccessExpr(ExprNode::field\_access\_expr, $3, $1, 0); }  
 | ExprWithBlock '.' ID { $$ = ExprNode::CallAccessExpr(ExprNode::field\_access\_expr, $3, $1, 0); }  
 | ExprWithoutBlock '.' ID '(' ExprListEmpty ')' { $$ = ExprNode::CallAccessExpr(ExprNode::method\_expr, $3, $1, $5); }  
 | ExprWithBlock '.' ID '(' ExprListEmpty ')' { $$ = ExprNode::CallAccessExpr(ExprNode::method\_expr, $3, $1, $5); }  
 | PathCallExpr { $$ = $1 }  
 | PathCallExpr '(' ExprListEmpty ')' { $$ = ExprNode::StaticMethod(ExprNode::static\_method, $1, $3); }  
 | ExprWithoutBlock '{' StructExprFieldList '}' { $$ = ExprNode::FieldListAccess(ExprNode::struct\_creation, $1, $3); }  
 | '(' ExprWithBlock ')' { $$ = $2; }  
 |'(' ExprWithoutBlock ')' { $$ = $2; }  
 | ExprWithBlock AS Type { $$ = ExprNode::AsExpr($1, $3);} //*ToDo добавить в enum и создать новый элемент* | ExprWithoutBlock AS Type {$$ = ExprNode::AsExpr($1, $3);} //*ToDo добавить в enum и создать новый элемент Expr* ;  
  
  
PathCallExpr: ID { $$ = ExprNode::CallAccessExpr(ExprNode::id\_, $1, 0, 0); }  
 | SUPER { $$ = ExprNode::CallAccessExpr(ExprNode::super\_expr, new string("super"), 0, 0); }  
 | SELF { $$ = ExprNode::CallAccessExpr(ExprNode::self\_expr, new string("self"), 0, 0); }  
 | CRATE { $$ = ExprNode::CallAccessExpr(ExprNode::crate\_expr, new string("crate\_expr"), 0, 0); }  
 | PathCallExpr DOUBLEDOTS ID { $$ = ExprNode::PathCallExpr(ExprNode::path\_call\_expr, $3, $1); }  
 ;  
  
  
  
StructExprFieldList: StructExprField { $$ = new ExprListNode($1); }  
 | ',' StructExprField { $$ = new ExprListNode($2); }  
 | StructExprFieldList ',' StructExprField { $$ = ExprListNode::Append($1, $3); }  
 ;  
StructExprField: ID ':' ExprWithoutBlock { $$ = ExprNode::ExprFromStructField(ExprNode::struct\_field\_expr, $1, $3); }  
 | ID ':' ExprWithBlock { $$ = ExprNode::ExprFromStructField(ExprNode::struct\_field\_expr, $1, $3); }  
 ;  
  
ExprWithBlock: BlockExpr { $$ = $1; }  
 | LoopExpr { $$ = $1; }  
 | IfExpr { $$ = $1; }  
 ;  
  
  
BlockExpr: '{' StmtList '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, 0, $2); }  
 | '{' ExprWithoutBlock '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, $2, 0); }  
 | '{' StmtList ExprWithoutBlock '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, $3, $2); }  
 | '{' '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, 0, 0); }  
 ;  
  
LoopExpr: InfiniteLoopExpr { $$ = $1; }  
 | PredicateLoopExpr { $$ = $1; }  
 | IteratorLoopExpr { $$ = $1; }  
 ;  
  
InfiniteLoopExpr: LOOP BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_expr, 0, $2, 0); }  
 ;  
  
PredicateLoopExpr: WHILE ExprWithBlock BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_while, $2, $3, 0); }  
 | WHILE ExprWithoutBlock BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_while, $2, $3, 0); }  
 ;  
  
IteratorLoopExpr: FOR ID IN ExprWithBlock BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_for, $4, $5, $2); }  
 | FOR ID IN ExprWithoutBlock BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_for, $4, $5, $2); }  
 ;  
  
IfExpr: SimpleIfElseExpr { $$ = $1; }  
 | SimpleIfElseExpr ELSE BlockExpr { $$ = ExprNode::AddElseBlock($1, $3) }  
 ;  
  
  
SimpleIfElseExpr: SimpleIfExpr { $$ = ExprNode::IfExprList($1); }  
 | SimpleIfElseExpr ELSE SimpleIfExpr { $$ = ExprNode::AddIfBlock($1, $3); }  
 ;  
  
SimpleIfExpr: IF ExprWithoutBlock BlockExpr { $$ = ExprNode::IfExpr(ExprNode::if\_expr, $2, $3); }  
 | IF ExprWithBlock BlockExpr { $$ = ExprNode::IfExpr(ExprNode::if\_expr, $2, $3); }  
 ;  
  
  
/\*-------------------------TYPE -------------------------- \*/  
  
Type: BOOL { $$ = new TypeNode(TypeNode::bool\_); }  
 | CHAR { $$ = new TypeNode(TypeNode::char\_); }  
 | FLOAT { $$ = new TypeNode(TypeNode::float\_); }  
 | INT { $$ = new TypeNode(TypeNode::int\_); }  
 | STRING { $$ = new TypeNode(TypeNode::string\_); }  
 | PathCallExpr { $$ = new TypeNode(TypeNode::path\_call\_expr\_, $1); } //*ToDO поменять тип на pathCallExp в enum и хранить элемент в Expr* | '[' Type ';' ExprWithBlock ']' { $$ = new TypeNode(TypeNode::array\_, $2, $4); }  
 | '[' Type ';' ExprWithoutBlock ']' { $$ = new TypeNode(TypeNode::array\_, $2, $4); }  
 ;  
 /\* Не доделан. Можно добавить TupleType \*/  
  
/\*---------------------- VISIBILITY ------------------------- \*/  
  
Visibility: PUB { $$ = pub; }  
 | PUB '(' SUPER ')' { $$ = self; }  
 | PUB '(' SELF ')' { $$ = super; }  
 ;  
  
%%  
  
void yyerror(char const \*s)  
{  
 printf("%s\n",s);  
 cerr << "line " << LineNum::getLineNum() << ": " << s << "\n";  
}

6) Перечень таблиц, генерируемых на этапе семантического анализа

class ClassTableItem{  
public:  
 enum Type{  
 *struct\_*, *enum\_*, *trait\_*, *mod\_* // не факт что правильно  
 };  
  
 Type classType;  
 string name;  
 MethodTable methodTable;  
 FieldTable fieldTable;  
 string parentName; // имя родителя  
 bool isPub = false;  
 bool haveAbstract = false;  
  
 ConstTable constTable;  
  
 ClassTableItem();  
 ClassTableItem(FieldTable fieldTable, MethodTable methodTable, string parentName = "");  
 string toString();  
 bool isHaveParent();  
 bool isHaveAbstract();  
};  
  
class ClassTable {  
private:  
 static ClassTable\* \_instanse;  
 map<string, ClassTableItem> items;  
 ClassTable();  
public:  
  
 string toString();  
 static const string moduleClassName;  
 static const string globalClassName;  
 static const string RTLClassName;  
  
 static ClassTable\* Instance();  
 void addMethod(const string& className, const string& methodName, MethodTableItem methodTableItem);  
 void updateMethod(string className, string methodName, MethodTableItem methodTableItem);  
 void addField(string className, string fieldName, FieldTableItem fieldTableItem);  
 void updateField(string className, string fieldName, FieldTableItem fieldTableItem);  
 void updateClass(string className, ClassTableItem classTableItem);  
 void addClass(string className, ClassTableItem classTableItem);  
 void addParent(string childName, string parentName);  
 void addLocalParam(string className,string methodName, VarTableItem varTableItem);  
 void addFuncParam(string className,string methodName, VarTableItem varTableItem);  
 static int addLoopCounterVar(string className,string methodName);  
  
  
 bool isClassExist(const string& className);  
 bool isMethodExist(const string& className, const string& methodName);  
 bool isFieldExist(const string& className, const string& methodName);  
 bool isParamExist(const string& className, const string& methodName, const string& varName);  
 bool isLocalVarExist(const string& className, const string& methodName, const string& varName);  
 bool isLocalVarExist(const string& className, const string& methodName, const string& varName, const ExprNode\* blockExpr);  
 bool isParent(const string& child, const string& parentName);  
  
 ClassTableItem getClass(const string& className);  
 ClassTableItem getParentClass(const string &className);  
 FieldTableItem getField(const string& className, const string& fieldName);  
 MethodTableItem getMethod(const string& className, const string& methodName);  
 VarTableItem getParam(const string& className, const string& methodName, int paramNum);  
 VarTableItem getLocalVar(const string& className, const string& methodName, int localVarNum);  
 static string getDirectory(string className);  
 static void isCorrectChild(string childName, string parentName);  
 static void isCorrectTraitsImpl();  
 static bool isHaveParent(const string& child);  
 static bool isEqualDirectory(const string& first, const string &second);  
 static bool isSubDirectory(const string& subdir, const string &dir);  
 static bool isHaveAccess(const string& requesterClass, const string& requestedClass);  
 static bool isHaveAccessToMethtod(const string& requesterClass, const string& requestedClass, const string& requestedMethod);  
 static bool isHaveAccessToField(const string& requesterClass, const string& requestedClass, const string& requestedField);  
  
 static int addIntToConstTable(const string& className, int val);  
 static int addFloatToConstTable(const string& className, float val);  
 static int addStringToConstTable(const string& className, const string& str);  
 static int addClassToConstTable(const string& className, const string& addingClassName);  
 static int addMethodRefToConstTable(const string& className, const string& addingClassName, const string& method, const vector<DataType> &params,const DataType &returnType);  
 static int addFieldRefToConstTable(const string& className, const string& addingClassName, const string& field, const DataType& dataType);  
 static int addUTF8ToConstTable(const string& className, const string& utf8);  
  
 static int getStructFieldCount(const string& className);  
 static void isMainFunctionExist();  
 static void createConstTableCSV();  
  
 static bool isMethodExistDeep(const string& className, const string& methodName);  
 static bool isFieldExistDeep(const string& className, const string& fieldName);  
 static string getParentClassName(const string& className);  
 static FieldTableItem getFieldDeep(const string& className, const string& fieldName);  
 static MethodTableItem getMethodDeep(const string& className, const string& methodName);  
 static void isCorrectTraitImpl(const std::string &childName, const std::string &parentName);  
  
 static bool isEnum(const string& className);  
 static map<string, ClassTableItem> getItems();  
 static void makeMainForJavaFormat();  
 static void checkClassNames();  
 static void addAbstract(const string& className);  
};

class ConstTableItem {  
public:  
 enum ConstTableType : uint8\_t {  
 *CONSTANT\_CLASS* = 7,  
 *CONSTANT\_FIELD\_REF* = 9,  
 *CONSTANT\_METHOD\_REF* = 10,  
 *CONSTANT\_INTERFACE\_METHOD\_REF* = 11,  
 *CONSTANT\_STRING* = 8,  
 *CONSTANT\_INTEGER* = 3,  
 *CONSTANT\_FLOAT* = 4,  
 *CONSTANT\_LONG* = 5,  
 *CONSTANT\_DOUBLE* = 6,  
 *CONSTANT\_NAME\_AND\_TYPE* = 12,  
 *CONSTANT\_UTF8* = 1,  
 *CONSTANT\_METHOD\_HANDLE* = 15,  
 *CONSTANT\_METHOD\_TYPE* = 16,  
 *CONSTANT\_INVOKE\_DYNAMIC* = 18,  
 };  
  
 ConstTableType constTableType;  
 string utf8;  
 float floatVal = -1;  
 int val1 = -1, val2 = -1;  
  
 ConstTableItem();  
  
 ConstTableItem(ConstTableType constTableType, string utf8);  
  
 ConstTableItem(ConstTableType constTableType, int val1);  
  
 ConstTableItem(ConstTableType constTableType, int val1, int val2);  
  
 ConstTableItem(ConstTableType constTableType, float floatVal);  
  
 string toString(int index);  
  
 vector<char> toBytes();  
};  
  
  
class ConstTable {  
public:  
 std::vector<ConstTableItem> items;  
  
 string toString();  
  
 string toCSV();  
  
 int Class(const string &className);  
  
 int Int(int val);  
  
 int Float(float val);  
  
 int String(const string &str);  
  
 int UTF8(const string &str);  
  
 int Val(ConstTableItem::ConstTableType constTableType, int val);  
  
 int Val(ConstTableItem::ConstTableType constTableType, int val1, int val2);  
  
 int Method(const string &method, const vector<DataType> &params, const DataType &returnType);  
  
 int Field(const string &field, const DataType &dataType);  
  
 int MethodRef(const string &className, const string &method, const vector<DataType> &params,  
 const DataType &returnType);  
  
 int FieldRef(const string &className, const string &field, const DataType &dataType);  
  
 int add(ConstTableItem constTableItem);  
  
 static string MethodParam(const vector<DataType> &params, const DataType &returnType);  
  
 ConstTable();  
  
 vector<char> toBytes();  
 static string formatClassName(const string& className);  
  
 inline static const string moduleClassName = "ModuleClass";  
 inline static const string globalClassName = "src";  
 inline static const string RTLClassName = "RTL";  
 inline static const string init = "<init>";  
 inline static const string clinit = "<clinit>";  
 inline static const string objectClassName = "java/lang/Object";  
 inline static const string stringClassName = "java/lang/Spring";  
};

class ExprNode;  
class FieldTableItem {  
public:  
 DataType dataType;  
 bool isConst = false;  
 bool isInit = false;  
 bool isPub = false;  
 ExprNode \*value; // хз надо ли  
  
 FieldTableItem();  
 FieldTableItem(DataType dataType, bool isConst);  
 FieldTableItem(DataType dataType, bool isConst, ExprNode \*value);  
 string toString();  
 bool isEquals(const FieldTableItem& fieldTableItem);  
};  
  
  
class FieldTable {  
public:  
 map<string, FieldTableItem> items; // ключ - имя, значение понятно  
 string toString();  
};

class MethodTableItem {  
public:  
 explicit MethodTableItem(const VarTable &paramTable);  
  
public:  
 DataType returnDataType;  
 VarTable paramTable;  
 VarTable localVarTable;  
 bool isStatic = true; // статический или не статический метод  
 bool isHasBody = false;  
 bool isPub = false;  
 bool isRefSelf = false;  
 bool isRTL = false;  
 ExprNode\* body;  
  
 MethodTableItem();  
 MethodTableItem(DataType returnDataType, VarTable paramTable,VarTable localVarTable, bool isStatic = true);  
 string toString();  
 bool isEqualsDeclaration(const MethodTableItem& other);  
 vector<DataType> getParamDataTypes();  
 string paramsToConstTableFormat();  
 static MethodTableItem initMethod();  
 static MethodTableItem clinitMethod();  
};  
  
  
class MethodTable{  
public:  
 map<string, MethodTableItem> items; // название класса  
 string toString();  
};

class ExprNode;  
class VarTableItem {  
public:  
 VarTableItem(DataType dataType, bool isMut, bool isRef, bool isInit);  
  
 VarTableItem(DataType dataType, bool isMut, bool isRef, bool isInit, ExprNode \*blockExpr);  
  
 string id = "";  
 //добавить номер  
 DataType dataType;  
 bool isMut = false;  
 bool isRef = false; // является передачей по ссылке или нет  
 bool isConst = false;  
 bool isInit = false;  
 ExprNode\* blockExpr; // блок, с которым связан переменная  
 ExprNode\* value; // для const;  
  
 VarTableItem();  
 VarTableItem(string name, DataType dataType, bool isMut,bool isRef,bool isInit, bool isConst);  
 VarTableItem(string name, DataType dataType, bool isMut,bool isRef,bool isInit, bool isConst, ExprNode\* blockExpr);  
 VarTableItem(string name, DataType dataType, bool isMut,bool isRef,bool isInit, bool isConst, ExprNode\* blockExpr, ExprNode\* value);  
 string toString();  
  
 bool isEquals(const VarTableItem &varTableItem);  
};  
  
class VarTable {  
public:  
 vector<VarTableItem> items;  
 string toString();  
 bool isExist(const string& varName);  
 bool isEquals(const VarTable & varTable);  
 bool isExist(const string& varName, const ExprNode\* blockExpr);  
 int getVarNumber(const string& varName, const ExprNode\* blockExpr);  
 VarTableItem getVar(int varNum);  
 static const string SELF\_PARAM;  
 size\_t size() const;  
};

7) Перечень ошибок, определяемых на этапе семантического анализа

1. Не определена fn main функция
2. Определено две и боее main функции
3. Несоответствие типов при инициализации
4. Несопоставимые типы при присвоениии
5. Несоответствие типов при передаче в функцию
6. Ожидаемый тип не совпадает с полученным типом 0308
7. Указана недопустимая область видимости (self/super)
8. Операнды имеют неприводимые типы
9. Недопустимое левое выражение в операции присваивания
10. Невозможно вычислить значение. (деление на ноль или целочисленное переполнение)
11. В enum два и более элемента имеют одно дискриминантное значение
12. Одинаковое имя полей у структуры
13. Возврат return из функции чей возвращаемый тип не ()
14. Бесконечный размер при определении рекурсивной структуры
15. Каждое поле при инициализации struct может быть проинициализировано лишь один раз
16. Не все поля при инициализации структуры были указаны.
17. Для диапазона range используется что-то другое кроме чисел и символов
18. Начало диапазона должно быть меньше или равно концу диапазона range
19. Ошибка вызова метода. Два и более метода имеют одинаковый прототип.
20. Попытка определить константу в trait
21. Конфликт реализаций (impl) одного trait для одной struct
22. Метод не имеет параметров (у метода всегда должен быть хотя бы &self)
23. Не найдены реализации в impl элементов, определённых в trait
24. Реализация в impl элемента с одним идентификатором из trait два и более раз
25. Попытка реализовать impl не для struct
26. Несоответствующие реализации в impl элементов, определённых в trait
27. Недопустимое кол-во аргументов при вызове функции
28. Несколько параметров функции имеет одно и то же имя
29. Значения для statics и constants должны быть известны к компиляции
30. Попытка изменить const или static
31. Вызов non-const функции при инициализации const
32. Обращение к необъявленному имени struct
33. Обращение к необъявленному имени module
34. Обращение к необъявленному в impl методу struct
35. Обращение к необъявленной свободной функции
36. Обращение к необъявленному локальной переменной
37. Обращение к необъявленному полю struct
38. Обращение к необъявленному имени trait
39. Использование не проинициализированной переменной
40. Использование в теле функции self когда он не передается в параметрах
41. Использование переменной после того как ее содержимое было перемещено (владение)
42. Неразрешенное имя (идентификатор совпадает с зарезервированными именами)
43. Struct или Module Определен более одного раза.
44. Использование ключевого слова pub там где оно не разрешено
45. EnumItem не является isize (i32)
46. Декларация функции без реализации находится вне trait
47. if возвращает разные типы значений
48. for while используют break со значением

8) Перечень преобразований дерева и дополнительной информации для узлов дерева на этапе семантического анализа

* Вычисление размера массива, если он задан с помощью инициализатора без указания размера при объявлении.
* Добавление тип данных к каждому узлу.
* Соединение struct с impl, для того чтобы мы могли получать из структуры доступ к методу
* Соединение struct и trait. Trait выступает родителем struct и представляет собой как будто абстрактный класс.
* Преобразование assign в array\_assign в случае, если слева располагается массив.
* Преобразование assign в point\_assign в случае, если слева располагается обращение к свойству.
* Преобразование строки println в сложение строк.

9) Выводы

В процессе выполнения задания мы изучили внутреннее устройство компилятора программ и его ключевые этапы. Особое внимание было уделено лексическому и синтаксическому анализаторам, которые преобразуют код программы в структурированное дерево.

Семантический анализ и генерация кода позволили углубить знания о работе со сложными структурами данных, включая обработку деревьев. Генерация кода дала представление о процессе создания байт-кода и его взаимодействии с исполнительной средой.

В итоге наше знакомство с компиляцией программ улучшило понимание внутреннего устройства языков программирования и расширило навыки анализа и обработки кода. Открыв для нас каждый этап, мы получили более четкое представление о том, как программы из набора букв, составляющих лексемы, превращаются в исполняемый код, который понимает компьютер.

1. Строки возможностей, не планируемых к реализации в рамках работы, следует удалить. В конце таблицы оставлены пустые строки для особых возможностей языка, не входящих в данный перечень. Их вес определяется при утверждении карточки преподавателем. [↑](#footnote-ref-1)
2. Вес возможностей языка, реализуемых студентом, является коэффициентом для определения его оценки за курсовую работу. Надписью «min» помечены возможности, реализация которых обязательна, их суммарный вес составляет 20 – по 10 на участника работы. [↑](#footnote-ref-2)
3. В этой графе вы кратко описываете особенности реализации данных возможностей в языке. Вес возможности может быть скорректирован, если ее реализация в данном языке значительно легче (тяжелее) чем в других языках. [↑](#footnote-ref-3)