|  |  |
| --- | --- |
| Лабораторная работа №2 | |
| Лексический анализатор для языка программирования Rust. | |
| Студенты | Ломакин А.С., Чупинин А.П. |
| Группа | ПрИн-466 |
| Преподаватель | Сычёв О.А. |
| Оценка |  |
| Дата, подпись |  |

**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). |

**Текст входного файла для flex.**