Rust#

Rust — новый экспериментальный язык программирования, разрабатываемый Mozilla. Язык компилируемый и мультипарадигмальный, позиционируется как альтернатива C/C.

В Rust поддерживаются функицональное, параллельное, процедурное и объектно-ориентированное программирование, т.е. почти весь спектр реально используемых в прикладном программировании парадигм.

Синтаксис и особенности

Синтаксис языка строится в традиционном си-подобном стиле. Естественно, всем известные ошибки дизайна C/C++ учтены.

- В числовые константы можно вставлять подчеркивания. Удобная штука, сейчас эту возможность добавляют во многие новые языки. 0xffff_ffff_ffff_ffff_ffff_ffff

- Двоичные константы. Конечно, настоящий программист должен преобразовывать bin в hex в уме, но ведь так удобнее! 0b1111_1111_1001_0000

- Тела любых операторов (даже состоящие из единственного выражения) должны быть обязательно заключены в фигурные скобки. К примеру, в Си можно было написать if(x>0) foo();, в Rust нужно обязательно поставить фигурнные скобки вокруг foo()

- Зато аргументы операторов if, while и подобных не нужно заключать в кругные скобки

- во многих случаях блоки кода могут рассматриваться как выражения.

- синтаксис объявления функций — сначала ключевое слово fn, затем список аргументов, тип аргумента указывается после имени, затем, если функция возвращает значение — стрелочка >"и тип возвращаемого значения

- аналогичным образом объявляются переменные: ключевое слово let, имя переменной, после переменной можно через двоеточие уточнить тип, и затем — присвоить начальное значение. let count: int = 5;

- по умолчанию все переменные неизменяемые; для объявления изменяемых переменных используется ключевое слово mutable.

- имена базовых типов — самые компактные из всех, которые мне встречались: i8, i16, i32, i64, u8, u16, u32, u64,f32, f64

- поддерживается автоматический вывод типов

Типы данных.

Rust поддерживает структурную типизацию. Что такое структурная типизация? Например, у вас в каком-то файле объявлена структура (или, в терминологии Rust, «запись») type point = x: float, y: float;

Вы можете объявить кучу переменных и функций с типами аргументов «point». Затем, где-нибудь в другом месте, вы можете объявить какую-нибудь другую структуру, например

type MySuperPoint = x: float, y: float;

и переменные этого типа будут полностью совместимы с переменными типа point.

Структуры в Rust называются «записи» (record). Также имеются кортежи — это те же записи, но с безымянными полями. Элементы кортежа, в отличие от элементов записи, не могут быть изменяемыми.

Имеются вектора — в чем-то подобные обычным массивам, а в чем-то — типу std::vector из stl. При инициализации списком используются квадратные скобки, а не фигурные как в C/C++

Есть шаблоны. Их синтаксис вполне логичен, без нагромождений «template» из C++. Поддерживаются шаблоны функций и типов данных.

Язык поддерживает так называемые теги. Это не что иное, как union из Си, с дополнительным полем — кодом используемого варианта (то есть нечто общее между объединением и перечислением). Или, с точки зрения теории — алгебраический тип данных.

Сопоставление с образцом (pattern matching)

Для начала можно рассматривать паттерн матчинг как улучшенный switch. Используется ключевое слово alt, после которого следует анализируемое выражение, а затем в теле оператора — паттерны и действия в случае совпадения с паттернами.

В качестве «паттеронов» можно использовать не только константы (как в Си), но и более сложные выражения — переменные, кортежи, диапазоны, типы, символы-заполнители (placeholders, '_'). Можно прописывать дополнительные условия с помощью оператора when, следующего сразу за паттерном. Существует специальный вариант оператора для матчинга типов. Такое возможно, поскольку в языке присутствует универсальный вариантный тип any, объекты которого могут содержать

значения любого типа.

Указатели. Кроме обычных «сишных» указателей, в Rust поддерживаются специальные «умные» указатели со встроенным подсчетом ссылок — разделяемые (Shared boxes) и уникальные (Unique boxes). Они в чем-то подобны shared_ptr и unique_ptr из C++. Они имеют свой синтаксис: @ для разделяемых и для уникальных.

Замыкания, частичное применение, итераторы

В Rust полностью поддерживается концепция функций высшего порядка — то есть функций, которые могут принимать в качестве своих аргументов и возвращать другие функции.

1. Ключевое слово lambda используется для объявления вложенной функции или функционального типа данных. 2. Ключевое слово block используется для объявления функционального типа — аргумента функции, в качестве которого можно подставить нечто, похожее на блок обычного кода. 3. Частичное применение — это создание функции на основе другой функции с большим количеством аргументов путем указания значений некоторых аргументов этой другой функции. Для этого используется ключевое слово bind и символ-заполнитель "_"4. Чистые функции и предикаты

Чистые (pure) функции — это функции, не имеющие побочных эффектов (в том числе не вызывающие никаких других функций, кроме чистых). Такие функции выдяляются ключевым словом pure.

Предикаты — это чистые (pure) функции, возвращающие тип bool. Такие функции могут использоваться в системе typestate (см. дальше), то есть вызываться на этапе компиляции для различных статических

проверок.

Синтаксические макросы

Планируемая фича, но очень полезная. В Rust она пока на стадии начальной разработки.

Выражение, аналогичное сишному printf, но выполняющееся во время компиляции (соответственно, все ошибки аргументов выявляются на стадии компиляции). К сожалению, материалов по синтаксическим макросам крайне мало, да и сами они находятся в стадии разработки

Атрибуты

Концепция, похожая на атрибуты C# (и даже со схожим синтаксисом). Как и следовало ожидать, атрибуты добавляют метаинформацию к той сущности, которую они аннотируют.

Придуман еще один вариант синтаксиса атрибутов — та же строка, но с точкой с запятой в конце, аннотирует текущий контекст. То есть то, что соответствует ближайшим фигурным скобкам, охватывающим такой атрибут.