UNIVERSIDADE FEDERAL DE SERGIPE
CAMPUS PROFESSOR ALBERTO CARVALHO
DEPARTAMENTO DE SISTEMAS DE INFORMAÇÃO

Discentes: Eduardo Tavares, Guilherme Andrade, João Rios, Nadianne Galvão
Disciplina: Linguagens Formais e Tradutores
Docente: Andre Luis Meneses Silva

Linguagem de programação: **Rust**

Introdução

Rust é uma linguagem de programação desenvolvida para construção de softwares confiáveis e com bom desempenho. O foco da Rust é agilidade, concorrência e confiabilidade.

Ela surgiu através de um projeto pessoal desenvolvido pelo Graydon Hoare, empregado da empresa Mozilla, que passou também a apoiar o desenvolvimento do projeto. Sua primeira versão estável foi lançada em 2015 e durante 6 anos consecutivos, de acordo com pesquisas realizadas pelo Stack Overflow a linguagem Rust foi considerada a mais amada.

Identificador

Para criação de um identificador seguimos algumas regras, como:

- Nomes de variáveis começam apenas com letras e dígitos, números não são permitidos.
- O nome da variável é uma combinação de letras, dígitos e caracteres sublinhados (_).
- Palavras-chave e identificadores em Rust não são usados como um nome de variável.
- Os nomes das variáveis diferenciam maiúsculas de minúsculas (case sensitive), o que significa que os nomes em maiúsculas e minúsculas são diferentes.

Identificador Bruto

O identificador bruto é aquele que utiliza a palavra-chave que em outro momento não seria permitida, mas neste momento, se a palavra reservada vier acompanhada de r# você poderá utiliza-lá como identificador, segue um exemplo:

```rust
fn r#match(needle: &str, haystack: &str) -> bool {
    haystack.contains(needle)
}

fn main() {
    assert!(r#match("foo", "foobar"));
}
```

A palavra match não pode ser usada como identificador, mas usando o identificador bruto, ela pode dar nome a uma função.

Comentários

Os comentários são criados para auxiliar melhor compreensão do código, ou seja, são notas não compiladas, mas que servem para leitura.

Para realizar um comentário que não vai para documentação colocamos duas barras seguidas do comentário da linha, da seguinte forma:

```
// Comentario de linha
```

Comentários que abrangem um bloco são feitos assim:

```
/*
Este
é um
comentario de bloco
*/
```

Comentários que serão incluídos na documentação são iniciados com três caracteres:

```
//! Comentario para explicar o modulo
/// Comentario para explicar o modulo
```

Palavras reservadas

Essas palavras não podem ser usadas como identificadores como dito anteriormente, pois foram reservadas para uma função específica, a seguir veremos quais foram as palavras utilizadas.

> ***as*** - executa a conversão primitiva, elimina a ambiguidade da característica específica que contém um item ou renomeia itens em instruções de use
>
> ***async*** - retorna um Future em vez de bloquear o thread atual
>
> ***await*** - suspende a execução até que o resultado de um Future esteja pronto
>
> ***break*** - sai de um loop imediatamente
>
> ***const*** - define itens constantes ou ponteiros brutos constantes
>
> ***continue*** - continua para a próxima iteração do loop
>
> ***crate*** - em um caminho de módulo, refere-se à raiz da caixa
>
> ***dyn*** - despacho dinâmico para um objeto trait

else - fallback para if e if let control flow constructs

enum - define uma enumeração

extern - liga uma função ou variável externa

false - literal falso booleano

fn - define uma função ou o tipo de ponteiro de função

for - faz um loop sobre itens de um iterador, implementa uma característica ou especifica um tempo de vida de classificação mais alta

if - desvio baseado no resultado de uma expressão condicional

impl - implementa a funcionalidade inerente ou característica

in - parte da sintaxe do loop for

let - vincula uma variável

loop - loop incondicionalmente

match - corresponde um valor aos padrões

mod - define um módulo

move r - faça um fechamento se apropriar de todas as suas capturas

mut - denota mutabilidade em referências, ponteiros brutos ou associações de padrões

pub - denota visibilidade pública em campos struct, blocos impl ou módulos

ref - vincular por referência

return - retorno da função

Self - um alias de tipo para o tipo que estamos definindo e implementando

self - assunto do método ou módulo atual

static - variável global ou tempo de vida que dura toda a execução do programa

struct - define uma estrutura

super - módulo pai do módulo atual

trait - definir um traço

true - literal verdadeiro booleano

type - define um alias de tipo ou tipo associado

union —defina uma união ; é apenas uma palavra-chave quando usada em uma declaração de união

unsafe - denota código, funções, características ou implementações inseguras

use - traga símbolos para o escopo

where - denota cláusulas que restringem um tipo

while - loop condicionalmente baseado no resultado de uma expressão

Palavras reservadas para uso futuro

A seguir veremos palavras que ainda não são utilizadas, mas foram reservadas para possível uso no futuro.

abstract

become

box

do

final

macro

override

priv

try

typeof

unsized

virtual

yield

Operadores e Símbolos

Os operadores são símbolos próprios que estão determinados a uma operação específica.

!	ident!(...), ident! {...}, ident![...]	Expansão de macro
!	!expr	Complemento bit a bit ou lógico
!=	expr != expr	Comparação sem igualdade
%	expr % expr	Resto aritmético
%=	var %= expr	Resto aritmético e atribuição
*	expr * expr	Multiplicação aritmética
*=	var *= expr	Multiplicação aritmética e atribuição
*	*expr	Desreferência
*	*const type, *mut type	Ponteiro Bruto
&	&expr, &mut expr	Analisador de empréstimo
&	&type, &mut type, &'a type, &'a mut type	Analisador de empréstimo bit a bit

&	expr & expr	Operador lógico AND bit a bit
&	var &= expr	Operador lógico AND bit a bit e atribuição
&	expr && expr	AND em curto circuito
+	trait + trait, 'a + trait	Adição e atribuição aritmética
+	expr + expr	Adição aritmética
+=	var += expr	Adição e atribuição aritmética
-	- expr	Negação aritmética
-	expr - expr	Subtração aritmética
- =	expr + expr	Subtração aritmética e atribuição
->	var += expr	Função e tipo de retorno de fechamento
,	expr, expr	Separador de argumentos e elementos
.	expr.ident	Acesso de membro

..	.., expr.., ..expr, expr..expr	Literal de intervalo exclusivo à direita
..=	..=expr, expr..=expr	Literal de intervalo inclusivo à direita
..	..expr	Sintaxe de atualização literal de estrutura
..	variant(x, ..), struct_type { x, .. }	Encadernação padrão “E o resto”
...	expr...expr	(Descontinuado, use ..= em vez disso) Em um padrão: padrão de intervalo inclusivo
/	expr / expr	Divisão aritmética
/=	var /= expr	Divisão aritmética e atribuição
:	pat: type, ident: type	Restrições
:	ident: expr	Inicializador de campo de estrutura
:	'a: loop {...}	Rótulo de loop
;	expr;	Declaração e terminador de item

<<	expr << expr	Deslocamento à esquerda
<<=	var <<= expr	Deslocamento à esquerda e atribuição
<	expr < expr	Menor que
<=	expr <= expr	Menor que e igual a
=	var = expr, ident = type	Atribuição/equivalência
==	expr == expr	Comparação de igualdade
=>	pat => expr	Part of match arm syntax
>	expr > expr	Maior que
>=	expr >= expr	Maior que e igual a
>>	expr >> expr	Deslocamento para a direita
>>=	var >>= expr	Deslocamento para a direita

@	ident @ pat	Pattern binding

^	expr ^ expr	OU exclusivo bit a bit

^=	var ^= expr	OU exclusivo bit a bit e atribuição

|	pat | pat	Alternativas de padrão

|	expr | expr	OU bit a bit

|=	evar |= expr	OU bit a bit e atribuição

||	expr || expr	OU lógico em curto-circuito

?	expr?	Propagação de erros

- Analisador de empréstimo é uma característica da Rust, já que ela se propõe a identificar se o tempo de vida do objeto está sendo respeitado

Referência:

KLABNIK,S.;NICHOLS,C. **The Rust Programming Language**. Disponível em: https://doc.rust-lang.org/book/title-page.html Acesso em: 08 de fevereiro de 2023