

Rust

- Baixo nível
- Segurança
- Confiabilidade



Brenno Lemos



Syndelis



O que a linguagem Rust oferece?


- Programação baixo nível multi-paradigma;
 - Imperativo, OO*, Funcional;
- Estaticamente tipada;
 - Equipada com um compilador inteligente e amigável;
- *Regras de Empréstimo* (veremos no futuro...)

- **Padrão único;**
- **Gerenciador de Pacotes;**
- **Impossibilita condições de corrida;**
- **Respeita a tipagem;**



O que é confiabilidade, na prática?

```
a = 1
a = true é igual a 1
a = 1
a ainda é igual a 1
```



```
#include <stdbool.h>
#include <stdio.h>

int main() {
    bool a = true;

    printf("a = %d\n", a);

    if (a == 1)
        printf("a = true é igual a 1\n");

    a = 10;
    printf("a = %d\n", a);

    if (a == 1)
        printf("a ainda é igual a 1\n");

    else
        printf("a é diferente de 1\n");
}
```

```

error[E0308]: mismatched types
  --> t.rs:6:10
   |
6 |     if a == 1 {
   |               ^ expected `bool`, found integer

error[E0308]: mismatched types
  --> t.rs:10:6
   |
2 |     let a = true;
   |           ---- expected due to this value
...
10 |     a = 10;
   |       ^^ expected `bool`, found integer

error[E0308]: mismatched types
  --> t.rs:13:10
   |
13 |     if a == 1 {
   |               ^ expected `bool`, found integer

```

error: aborting due to 3 previous errors

For more information about this error, try `rustc --explain E0308`.



```

fn main() {
    let a = true;

    println!("a = {}", a);

    if a == 1 {
        println!("a = true é igual a 1");
    }

    a = 10;
    println!("a = {}", a);

    if a == 1 {
        println!("a ainda é igual a 1");
    }

    else {
        println!("a é diferente de 1");
    }
}

```



Onde usar Rust?

- Web Backend;
 - **actix.rs**, **rocket.rs**, **diesel.rs**, **serde**;
- Web Frontend;
 - Compila para **WebAssembly**;
 - **yew.rs** possui sintaxe similar ao **React**;
- Sistemas Embarcados (nativo);
- Jogos;
 - **godot-rust**, **bevy-engine**;
- *Interoperabilidade com C*;

Prólogo: Tipos de Datos

RUST	C	RUST	C
u8/i8	[unsigned] char	u16/i16	[unsigned] short int
u32/i32	[unsigned] int	u64/i64	[unsigned] long int
[ui]128	__[u]int128_t	[ui]size	-
f32	float	f64	double
bool	bool	char	-



```
// Arrays e tuplas -----  
let array: [i32; 4] = [1, 2, 3, 4];  
let tuple: (i32, f32, &str) = (1, 2.5, "asd");  
  
let array_first = array[0];  
let tuple_first = tuple.0;
```

```
// Vetor dinâmico -----  
let mut v = Vec::new(); // Vetor vazio  
v.push(1); // Adiciona o número 1  
v.push(2);  
v.push(3);  
  
// Alternativa (mesma struct)  
let v = vec![1, 2, 3]; // Não precisa ser  
mutável!
```

```
// String -----  
let mut s = String::new();  
s.push_str("Olá mundo!");  
  
// Alternativa  
let s = String::from("Olá mundo!");  
// Não precisa ser mutável!
```



```
struct Cachorro {  
    nome: String,  
    idade: i32,  
    femea: bool  
}
```

```
// ...
```

```
let barney = Cachorro {  
    nome: String::from("Barney"),  
    idade: 5,  
    femea: false  
};
```



```
enum Mensagem {  
    Ligacao,  
    Texto(String),  
    Reacao {  
        emoji: char,  
        id_msg: i32  
    }  
}
```

```
impl Mensagem {  
    fn notificacao(&self) -> String {  
        match self {  
            Mensagem::Ligacao => "Te ligaram".into(),  
            Mensagem::Texto(texto) =>  
                format!("> {}", texto),  
            Mensagem::Reacao { emoji, id_msg: id } =>  
                format!("Reagiram a {} com {}", id,  
                    emoji)  
        }  
    }  
}
```



Valor Nulo

Segmentation Fault, difícil rastreabilidade

- Ponteiros sempre podem ser nulos, mas nem sempre isto é checado;
- “O erro de um bilhão de dólares” - Tony Hoare, criador do NULL;
- Em Rust, referências devem sempre ser válidas e apontar para um valor de memória existente;

```
typedef struct {  
    int x, y;  
} Rect;
```

```
Rect *pode_retornar_nulo();
```

```
Rect *rect = pode_retornar_nulo();  
rect->x = 10; // Segfault
```



```
struct Rect { x: i32, y: i32 };
```

```
fn pode_retornar_nulo() -> Option<Rect>;
```

```
let rect = pode_retornar_nulo();  
match rect {  
    Some(rect) => rect.x = 10,  
    None => panic!("O retângulo não existe!")  
}
```



Sistema de Empréstimo

Memory Leaks, Condições de Corrida, etc

- Variáveis devem ser explicitamente mutáveis;
- Duas variáveis não podem alterar o mesmo valor no mesmo escopo;
- Não podem haver variáveis lendo um valor no mesmo escopo no qual outra variável escreve;
- Valores não utilizados são desalocados imediatamente;



```
let x = 10;
```

```
x = 20; // Inválido
```

```
x += 10; // Inválido
```

```
error[E0384]: cannot assign twice to immutable  
variable `x`
```

```
--> m.rs:4:2
```

```
|  
2 |     let x = 10;  
|     -  
|     |  
|     first assignment to `x`  
|     help: consider making this binding  
mutable: `mut x`  
3 |  
4 |     x = 20; // Inválido  
|     ^^^^^^ cannot assign twice to immutable  
variable
```

```
// Podemos fazer que `x` seja mutável ----
```

```
let mut x = 10;
```

```
x = 20;
```

```
x += 10;
```

```
// OU podemos sombreadar a variável ----
```

```
let x = 10;
```

```
let x = 20;
```

```
let x = x + 10;
```

```
let mut x = 10; // Inteiro mutável
```

```
let mut y = &x; // Referência mutável para inteiro  
imutável
```

```
*y = 20; // ❌
```

```
let mut x = 10; // Inteiro mutável
```

```
let y = &mut x; // Referência imutável para inteiro  
mutável
```

```
*y = 20; // ✅
```

```
let mut v1 = vec![1, 2, 3, 4, 5]; // Novo vetor dinâmico
```

```
v1.push(6);
```

```
let mut v2 = v1; // Os dados de `v1` são movidos para `v2`
```

```
v2.push(7);
```

```
v1.push(8); // ❌ `v1` não é mais o dono do vetor!
```



```
error[E0382]: borrow of moved value: `v1`
--> t.rs:8:5
|
2 |     let mut v1 = vec![1, 2, 3, 4, 5]; // Novo vetor dinâmico
|         ----- move occurs because `v1` has type `Vec<i32>`, which does not
implement the `Copy` trait
...
5 |     let mut v2 = v1; // Os dados de `v1` são movidos* para `v2`
|         -- value moved here
...
8 |     v1.push(8); // Inválido! `v1` não é mais o dono do vetor!
|     ^^^^^^^^^^^ value borrowed here after move
```

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.

Traços e Implementações

Features de Orientação a Objetos

- Implementações são blocos de código que uma struct/enum possui;
- Traços são blocos de declaração que contém cabeçalhos de funções;
- Traços podem ser utilizados por funções monomórficas e polimórficas;
- Não existe herança de structs, mas traços podem implicar em outros traços;



```
trait Geometrico {fn area(&self) -> f64;}
struct Quadrado { lado: f64 }

impl Geometrico for Quadrado {
    fn area(&self) -> f64 {
        self.lado * self.lado
    }
}

fn dobro_area_mono<T: Geometrico>(g: &T) -> f64
{
    2.0 * g.area()
}

fn dobro_area_poli (g: &dyn Geometrico) -> f64 {
    2.0 * g.area()
}

fn main() {
    let q = Quadrado { lado: 2.0 };
    dobro_area_mono (&q) == dobro_area_poli (&q);
}
```

Zero-cost abstractions

soma_quadrados_pares:

mov eax, 1540

ret

```
int soma_quadrados_pares() {  
    int s = 0;  
    for (int i = 0; i <= 20; i++)  
        if ((i&1) == 0)  
            s += i * i;  
  
    return s;  
}
```



```
fn soma_quadrados_pares() -> i32 {  
    (0..=20)  
        .filter(|i| i&1 == 0)  
        .map(|i| i * i)  
        .sum()  
}
```





Como aprender Rust?

1. “The Book”: rust-lang.org/learn;
 - A bíblia do Rust;
2. Rustlings: [GitHub rust-lang/rustlings](https://github.com/rust-lang/rustlings);
 - Questões interativas de Rust na sua IDE favorita;
3. “Rust by Example”;
 - Pequenos exemplos de casos de uso comuns de várias *features* da linguagem;

