Здравей, Rust

Въведение

7 октомври 2020

Административни неща

- Питайте въпроси в чата, ще правим паузи за отговори
- Записваме видео

Hello, world!

Защото винаги от там се почва:

```
fn main() {
    println!("Hello, world!");
}
```

The Rust Book

https://doc.rust-lang.org/stable/book/

Rust playpen

https://play.rust-lang.org/

Променливи

Променливи се декларират с `let`

```
let NAME = VALUE;
let NAME: TYPE = VALUE;
```

```
1  let x = 5;
2  let y: i32 = 3;
```

Променливи

Всяка променлива има тип, но можем да не пишем типа, ако е ясен от контекста

```
let x: i32 = 5;
let y = x;      // типа на `y` е `i32`, защото `x` е `i32`
```

```
let x = 5;      // типа на `x` е `i32`, защото `y` е `i32`
let y: i32 = x;
```

Променливи

shadowing

```
let x = 10;
let x = x + 10;
let x = x * 3;
```

Променливи

shadowing

```
let x1 = 10;
let x2 = x1 + 10;
let x3 = x2 * 3;
```

Променливи

mutability

Променливите са immutable по подразбиране

```
let x = 5;
x += 1;
```

Променливи

mutability

Променливите са immutable по подразбиране

```
1 let x = 5;
2 x += 1;
```

```
error[E0384]: cannot assign twice to immutable variable `x`
 --> src/bin/main_7db00a1a92dd466b5d0d2e943c3247d86695e529.rs:6:1
  |
5 | let x = 5;
  |     -
  |     |
  |     first assignment to `x`
  |     help: consider making this binding mutable: `mut x`
6 | x += 1;
  | ^^^^^^ cannot assign twice to immutable variable
```

Променливи

mutability

За да се направи mutable се използва ключовата дума `mut`

```
let mut x = 5;
x += 1;
```

Променливи

mutability

Това е различно от shadowing!

```
let x = 5;
let x = x + 1;
```

Основни типове

Целочислени типове (в различни бройни системи)

- Hex: `0xDEADBEEF`
- Octal: `0o77`
- Binary: `0b1010011010`

Основни типове

Булеви стойности

- `bool`
- стойности `true` и `false`

```
let x: bool = true;
```

Основни типове

Масиви

- `[T; n]`

```
let arr: [i32; 3] = [1, 2, 3];

let nested: [[i32; 3]; 2] = [
    [1, 2, 3],
    [4, 5, 6],
];
```

Основни типове

Кортежи (tuples)

- `(A, B, C, ...)`

```
let tuple: (i32, u32, bool) = (1, 2, false);
let unit: () = ();

println!("{}", tuple.0);
println!("{}", tuple.1);
println!("{}", tuple.2);
```

```
1
2
false
```

Основни типове

Сравнение със C

Length	Rust		C/C++	
	Signed	Unsigned	Signed	Unsigned
8-bit	i8	u8	char	unsigned char
16-bit	i16	u16	short	unsigned short
32-bit	i32	u32	int	unsigned int
64-bit	i64	u64	long long	unsigned long long
word	isize	usize	long	unsigned long / size_t

Length	Rust	C/C++
32-bit	f32	float
64-bit	f64	double

Length	Rust	C++
8-bit	bool	bool
-	()	void

Специфики

Няма автоматично конвертиране между различни числови типове

```
1  let x: i32 = 1;
2  let y: u64 = x;
```

```
error[E0308]: mismatched types
 --> src/bin/main_8b638dfa209c175f12a85c93aaa5a902798b2434.rs:5:14
  |
5 | let y: u64 = x;
  |        ---   ^ expected `u64`, found `i32`
  |        |
  |        expected due to this
  |
help: you can convert an `i32` to a `u64` and panic if the converted value doesn't f
  |
5 | let y: u64 = x.try_into().unwrap();
  |              ^^^^^^^^^^^^^^^^^^^^
```

Специфики

Аритметични операции не могат да се прилагат върху различни типове

```
1  let x = 4_u32 - 1_u8;
```

```
error[E0308]: mismatched types
 --> src/bin/main_8e5225b507b03e7c9f6c018e0cefb8bd8bc78341.rs:4:17
  |
4 | let x = 4_u32 - 1_u8;
  |                 ^^^^ expected `u32`, found `u8`
```

```
error[E0277]: cannot subtract `u8` from `u32`
 --> src/bin/main_8e5225b507b03e7c9f6c018e0cefb8bd8bc78341.rs:4:15
  |
4 | let x = 4_u32 - 1_u8;
  |               ^ no implementation for `u32 - u8`
  |
  = help: the trait `Sub<u8>` is not implemented for `u32`
```

Специфики

Аритметични операции не могат да се прилагат върху различни типове

```
1 | let y = 1.2_f64 / 0.8_f32;
```

```
error[E0308]: mismatched types
 --> src/bin/main_f1878bdeb672a355ef5d23dbdfc01d0241278399.rs:4:19
  |
4 | let y = 1.2_f64 / 0.8_f32;
  |                   ^^^^^^^ expected `f64`, found `f32`
```

```
error[E0277]: cannot divide `f64` by `f32`
 --> src/bin/main_f1878bdeb672a355ef5d23dbdfc01d0241278399.rs:4:17
  |
4 | let y = 1.2_f64 / 0.8_f32;
  |                 ^ no implementation for `f64 / f32`
  |
  = help: the trait `Div<f32>` is not implemented for `f64`
```

Специфики

За конвертиране между типове се използва ключовата дума `as`

```
let one = true as u8;
let two_hundred = -56_i8 as u8;
let three = 3.14 as u32;

println!("one: {}\ntwo_hundred: {}\nthree: {}", one, two_hundred, three);
```

```
one: 1
two_hundred: 200
three: 3
```

Специфики

В режим debug, аритметични операции хвърлят грешка при препълване (integer overflow / underflow)

```
1  let x = 255_u8;
2  let y = x + 1;                    // 💥
3  println!("{}", y);
```

```
error: this arithmetic operation will overflow
 --> src/bin/main_c02999f2504c2ca084dca6d96adc70ddb7bf1bae.rs:4:9
  |
4 | let y = x + 1;                    // 💥
  |         ^^^^^ attempt to compute `u8::MAX + 1_u8`, which would overflow
  |
  = note: `#[deny(arithmetic_overflow)]` on by default
```

Специфики

Няма оператори ++ и --

```
1  x += 1;
2  x -= 1;
```

Коментари

Едноредов коментар

```
// So we’re doing something complicated here, long enough that we need
// multiple lines of comments to do it! Whew! Hopefully, this comment will
// explain what’s going on.
```

Rust поддържа и блокови коментари

```
/*
So we’re doing something complicated here, long enough that we need
multiple lines of comments to do it! Whew! Hopefully, this comment will
explain what’s going on.
*/
```

Control flow

if-клаузи

Синтаксис на if-клауза

```
if bool_expression {
    // ...
} else if another_bool_expression {
    // ...
} else {
    // ...
}

if 2 > 1 {
    println!("okay");
} else {
    println!("wait what");
}
```

Забележете, че няма скоби около условието и скобите за блок `{ }` са задължителни.

Control flow

Цикли

`for` цикъла работи с итератори, за които ще говорим в бъдеща лекция

```rust
for var in iterable {
    // ...
}

for n in [1, 2, 3] {
    println!("N: {}", n);
}
```

Отново няма скоби след `for` и скобите за блок `{ }` са задължителни.

Control flow

Цикли

Също така има и `while` и `loop` цикли.

```
while bool_expression {
    // ...
}
```

`loop` е същото като `while true`, но по-четимо.

```
loop {
    // ...
}
```

Statements & Expressions

Твърдение (statement)

Пример: можем да присвояваме стойността на израз на променлива с `let`, но не и стойността на твърдение (защото няма стойност)

```
1 | let x = (fn add(a: i32, b: i32) { a + b });
```

```
error: expected expression, found keyword `fn`
 --> src/bin/main_97ba87efcfac218791b39b5b3ee18a18037e022c.rs:4:10
  |
4 | let x = (fn add(a: i32, b: i32) { a + b });
  |          ^^ expected expression
```

Statements & Expressions

Много от конструкциите на езика са изрази.

Блоковете са израз - стойността им е стойността на последния израз в блока

```
fn main() {
    let x = {
        let a = 1;
        let b = 2;
        a + b
    };

    println!("x = {}", x);
}
```

```
x = 3
```

Statements & Expressions

if-else конструкцията е израз

```
let bigger = if a > b {
    a
} else {
    b
};
```

По тази причина няма тернарен оператор

```
let bigger = if a > b { a } else { b };
```

Statements & Expressions

`loop` е израз

```
let x = loop {
    break 5;
};
```

Функции

```
fn main() {
    println!("Hello, world!");
    another_function();
}

fn another_function() {
    println!("Another function.");
}
```

```
Hello, world!
Another function.
```

Функции

```
1  fn print_a(a: u32) {
2      println!("{}", a);
3  }
4
5  fn print_b(b: u32) -> () {
6      println!("{}", b);
7  }
```

- Не е нужно да пишем `-> ()` за функции който не връщат резулат

Функции

```
1  fn good_a(a: u32, a_is_bad: bool) -> u32 {
2      if a_is_bad {
3          return 0;
4      }
5
6      a
7  }
```

- Ако искаме да излезем от функцията преди последния ред, може да използваме `return`
- Използване на `return` на последния ред от тялото се счита за лоша практика

Анонимни функции (closures)

```rust
fn add1(a: u32, b: u32) -> u32 {
    a + b
}

fn main() {
    let add2 = |a, b| { a + b };

    println!("add1: {}\nadd2: {}", add1(1, 2), add2(1, 2));
}
```

```
add1: 3
add2: 3
```

- Ще говорим доста повече за тях по-нататък

println! macro

```
1  let x = 5;
2  let y = "десет";
3  println!("x = {:?} and y = {:?}", x, y);
```

```
x = 5 and y = "десет"
```

- Принтиране на конзолата
- `{:?}` placeholders

dbg! macro

```
let x = 5;
let y = "десет";

dbg!(x);
dbg!(y);
```

```
[src/bin/main_72fe3d84fb206495b3ff0c4f73c4dc2f69e38ea2.rs:5] x = 5
[src/bin/main_72fe3d84fb206495b3ff0c4f73c4dc2f69e38ea2.rs:6] y = "десет"
```

Въпроси