Rust

Madhavan Mukund, S P Suresh

Programming Language Concepts

Lecture 8, 01 February 2024

Introduction to Rust

- Everything: `https://www.rust-lang.org/`
- Installation: `https://www.rust-lang.org/tools/install`
- Documentation: `https://www.rust-lang.org/learn`

Creating programs

- Hello world, cargo

Collection of functions — start at main()

println! is a "macro"

cargo — package manager + builder

Variables, mutability, constants, shadowing

- Types
  - Static
    - Declarations (Java etc)
    - Explicit vs Implicit
      - + Static ⇒ Haskell
      - Type inference
    - Strong vs Weak
  - Dynamic
    - Python - by current value

Variables, mutability, constants, shadowing

- Weak typing

```
x = 1.5 * 3
if x {-- 3
```

Variables, mutability, constants, shadowing

- Rust
  - Strong (super strong)
  - Implicit } Type Inference — but "declared" via let
  - Static
  - Functions require explicit typing

Variables, mutability, constants, shadowing

- `let x = 5;` — x is an arbitrary but fixed value
- `let mut x = 5;` — x can be up
- Separate notion of a constant

```
const pi_approx : f64 = 3.1415927
```

explicit type annotation

Variables, mutability, constants, shadowing

- Shadowing

```
let x = 5;
let x = x+1; ← "new" x
```

Data types

- let mut x = 0.0;

  x = x + 1; ✗ Disallowed

  Can change types when shadowing

  let x = 0.0;

  let x = 5; ✓

Data types

- u8, i32, f64 etc

Can always explicitly type a variable

```
let x : f32 = 4.5;
```

Data types

- Booleans
  - true, false
  - Can't implicitly interpret a numeric value as a boolean expression
- Char
  - UTF-8

Functions, statements, expressions

- fn myfunction() {
  =
  }

Must provide type annotations for parameters and return values

Functions, statements, expressions

```
fn addtwo (x : i32, y:i32) -> i32 {
    return x+y;
}
fn addtwoV2 (x:i32, y :i32) -> i32 {
    x+y
}
```

What does this mean?

Functions, statements, expressions

- $x + y$ Expression

Value of a function is implicitly the last expression

Control flow

- if ↑ { } else { } — "Expression"

  valid Boolean expression

  loops — while _ { }

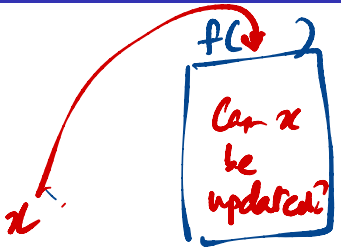
  loop { break } — Infinite

  iterator for

Ownership

- Mutability

  let mut ...



Passing references

Heap Python

```
l1 = [1,2,3]
l2 = l1
```

Ownership

- m1 = Manager() --

Manager contains a Data

m2 = m1