Advances in Programming Languages

Lecture 15: The Rust Programming Language

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 15 November 2016
Semester 1 Week 9

THE UNIVERSITY *of* EDINBURGH

Topic: Programming for Memory Safety

The final block of lectures cover features used in the **Rust** programming language.

- Introduction: Zero-Cost Abstractions (and their cost)
- Control of Memory: Ownership
- Concurrency and more

This section of the course is entirely new — Rust itself is not that old — and I apologise for any consequent lack of polish.

Previous Homework

1. Do this

Watch Kathleen Fisher, first HACMS project leader, explain to DARPA funders the research proposal background. **https://is.gd/hacms_video**

Read about how the project went and what's next in these two articles.

- **https://is.gd/hacms_quadcopter**
- **https://is.gd/hacms_helicopter**

You can get the quadcopter code at **http://www.smaccmpilot.org**.
It's in Haskell.

2. Read this

Ken Thompson
Reflections on Trusting Trust
Communications of the ACM, 27(8):761–763, 1984.
DOI: 10.1145/358198.358210

The Rust Programming Language

The **Rust** language is intended as a tool for *safe systems programming*. Three key objectives contribute to this.

- Zero-cost abstractions
- Memory safety
- Safe concurrency

Basic References

https://www.rust-lang.org
https://blog.rust-lang.org

The "systems programming" motivation resonates with that for imperative C/C++. The "safe" draws extensively on techniques developed for functional Haskell and OCaml. Sometimes these align more closely than you might expect, often through overlap between two aims:

- Precise control for the programmer;
- Precise information for the compiler.

Rust: When and How

Rust originated in 2006, took off with sponsorship from Mozilla in 2009, and reached its first stable release with Rust 1.0 in May 2015.

Mozilla use it for their experimental Servo concurrent HTML layout engine, and recently some parts of Firefox.

Dropbox rewrote their backend file system in Rust, to support their recent move off Amazon Web Services to their own storage infrastructure.

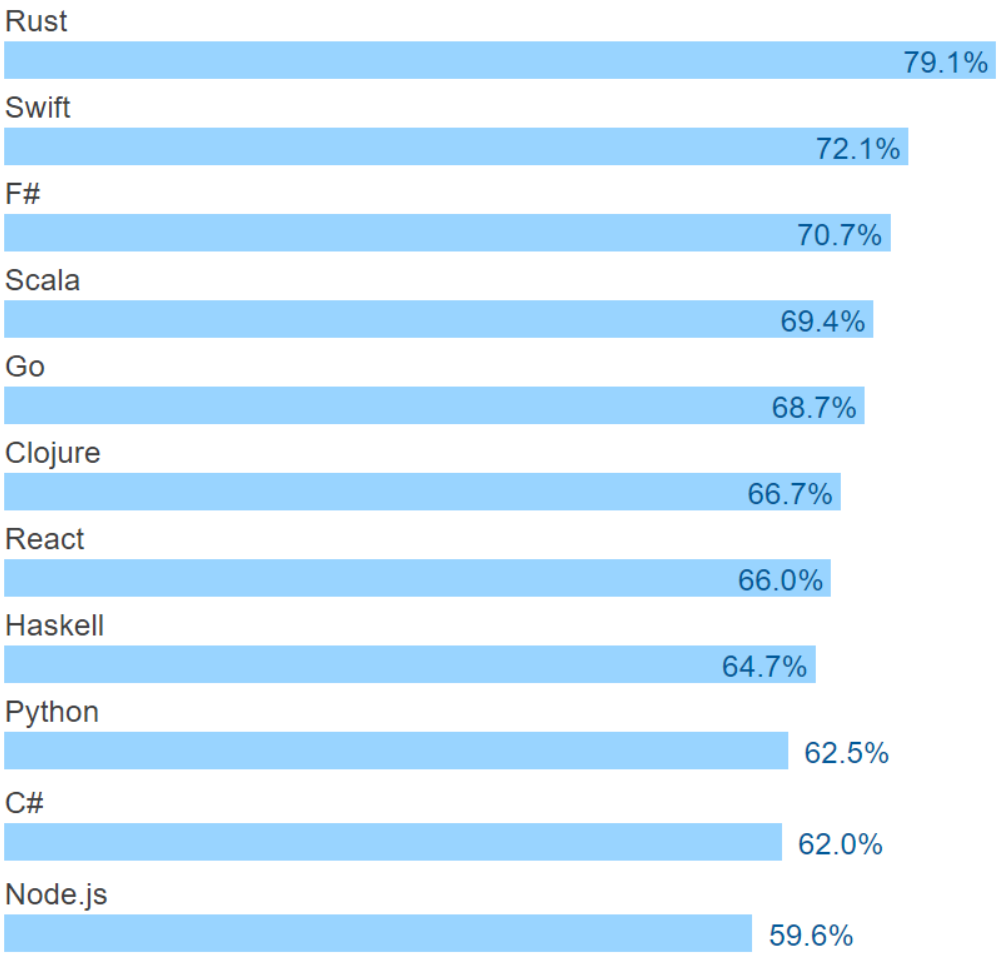
The "Friends of Rust" list identifies a number of organizations using Rust in production.

This year's Stack Overflow developer survey awarded Rust:

Most Loved Language of 2016

II. Most Loved, Dreaded, and Wanted

Loved | Dreaded | Wanted



Variables that Don't

Bindings

```
let x = 10;
let y = true;
let (a,b,c) = (7,8,56);

x = x+1;                    // Error: bindings immutable by default

let mut z = a*b;
z = c+c+c;                  // OK: z declared as mutable
```

By default all "variables" are immutable. Everything here is statically typed: so x : i32 and y : bool, but all is inferred.

Functions and Expressions

Function declaration

```
fn abs (x:i32) -> i32 {
    if x > 0 { x } else { -x }
}
```

Function abs must declare the type of its argument, and it looks a little like C as we get to use braces (curly brackets) a lot. Everything else, though...

- Strict static typing
- Type inference
- Binding x is immutable
- The body if is an expression, not a statement
- So are its branches x and -x
- Function abs is itself a value, of type fn(i32) -> i32

Control flow

Assorted looping

```
loop {
  println!("Around we go");
}

while b {                    // At this point we probably do
  code;                      // want some mutable state and effectful
}                            // imperative programming

for c in 1..5 {
  println!( "Now c is {}", c );   // Will in fact print 1, 2, 3, 4
}
```

Control flow

Where Rust is not C

```
for (c = 1; c < 15; c++) {
    printf( "Now c is %d\n", c );
}
```

Rust does not provide the general for loop of C, intentionally.

All of loop, while, for are "zero-cost", in that they can readily be compiled in the simplest way possible and exactly as you would expect.

They also have the advantage of abstraction:

- Enables the programmer to do more;
- Enables the compiler to do more.

There is still a cost, though, in the constraint on the programmer.

Structured Values

Tuples

```
let v = (2, true, -3.0);
let w = v;
let a = w.1;
let b = w.2;
let (x,y,z) = w;
```

Structured Values

Structs

```
struct Point {
  x: f64,
  y: f64,
}

let p = Point { x: 1.0, y: -2.5 };
let (a,b) = p;

let mut q = Point { x: 0.0, y: 0.0 };
q.x = q.x + 3.4;
```

Structured Values

Tuple Structs

```
struct ThreePoint(i32,i32,i32);
struct Date(i32,i32,i32);

let xunit = ThreePoint(1,0,0); // These two values
let today = Date(2016,11,15); // have different types

struct Inches(f64);

let height = Inches(43.2);      // height: Inches

let Inches(h) = height;         // h: f64
```

Enumerations

Declaring enumerations

```
enum Draw {
  MoveTo { x: i32, y: i32 },
  PenUp,
  PenDown { r: i32, g: i32, b: i32},
  Quit,
}

let up = Draw::PenUp;

let start = Draw::PenDown{ r:255, g:255, b:255 };
```

Enumerations

Using enumerations

```
match b {
  true => println!("It's true!"),
  false => println!("It's not true!"),
}

match drawcommand {
  PenUp => println!("Raise pen"),
  PenDown { r, g, b } => println!("Red {} Green {} Blue {}",r,g,b),
  MoveTo { x, y } => println!("On the move"),
  Quite => println!("All done"),
}
```

Generics (Parametric Polymorphism)

```
enum Option<T> {
    Some(T),
    None,
}
```

```
let x: Option<i32> = Some(5);

let mut y: Option<f64> = None;

fn exchange<T,U>(a: T, b: U) -> (U,T) {
   (b,a)
}

let (p,q) = exchange (5,3.2); // Monomorphised at compile time
```

Not Homework

CertiKOS: Certified OS Kernels

Structured clean-slate kernels, with separated components, extensive specification, and verified compilation for concurrent execution on shared-memory multicore machines.

Yale press release 2016-11-14: https://is.gd/certikos_press

Project site: http://flint.cs.yale.edu/certikos

Java Futures – A Sneak Peak

Talk last week at Devoxx 2016 on language features coming to Java now and over the next few years. Lots of things, how they fit together, and the challenge of doing this in an established language.

Video: https://is.gd/voxx_peek

Guest Speaker !

The Hack programming language: Types for PHP

Andrew Kennedy
Facebook

3pm Friday 18 November 2016