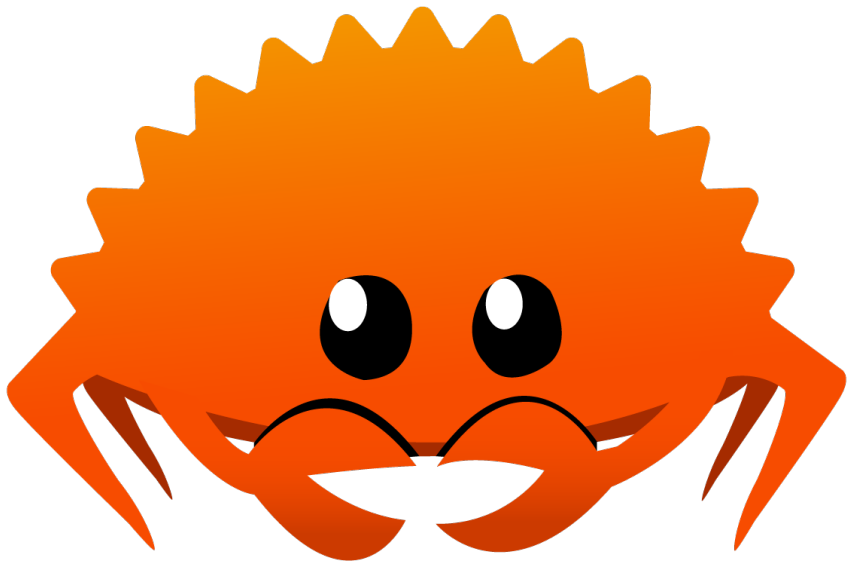
Lecture 2 — Rust Basics

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

November 15, 2020

We won’t tell you to just go learn Rust on your own...



Focus: important features: why & how they support the goal of performance.

Learning to Swim

Reading or watching about a programming language isn't super effective.

There's no substitute for writing code!

Suggestion: do practice exercises to become familiar with the language.

Goal-Setting

Some things aren’t here. We’re not covering the very basics of Rust.

The official docs are good and you will get used to the syntax as we use it.

Semicolon

Previously: C/C++/Java, where all statements end with semicolons.

In Rust that is not so: semicolons separate expressions.

The last expression in a function is its return value.

You can use `return` to get C-like behaviour, but you don't have to.

```
fn return_a_number() -> u32 {
  let x = 42;
  x+17
}

fn also_return() -> u32 {
  let x = 42;
  return x+17;
}
```

Change is Painful

Variables in Rust are, by default, immutable.

```
fn main() {
    let x = 42; // NB: Rust infers type "i32" for x.
    x = 17;     // compile-time error!
}
```

Immutability is good for performance.

The compiler can reason about the possibility of race conditions.

No writes? No race condition!

This has happened...

If you don't believe me, here's an example in C of where this could go wrong:

```c
if ( my_pointer != NULL ) {
    int size = my_pointer->length; // Segmentation fault occurs!
    /* ... */
}
```

What happened? We checked if `my_pointer` was null?

Diamonds are Forever

Immutability in Rust is forever (ish).

The compiler will not let you make changes to something via trickery.

Rust grudgingly permits such dark magicks, but you you have to brand your code with the `unsafe` keyword and are subject to undefined behaviour.

Change is Necessary

If you want for a variable's value to be changeable you certainly can, but you have to explicitly declare it as *mutable*

Add `mut` to the definition, like `let mut x = 42;`

Generally, minimize the number of times you use this.

Rust forces you to make mutability explicit & has the compiler check your work.

The Tao is Eternal

There are constants, which are different from global variables.

Constants are both immutable and immortal.

```
const SPEED_OF_LIGHT_M_S: u32 = 299_792_458;.
```

They don’t really exist at runtime and have no address.

Rust also has global variables, defined using `static`.

Shadowing

Shadowing is intended to address the problem of “What do I name this?”
An example from the docs:

```
let mut guess = String::new();

io::stdin().read_line(&mut guess)
    .expect("Failed to read line");

let guess: u32 = guess.trim().parse()
    .expect("Please type a number!");
```

Memory Management

In languages like C, memory management is manual: you allocate and deallocate memory using explicit calls.

In other languages like Java, it's partly manual—you explicitly allocate memory but deallocation takes place through garbage collection.

C++ supports memory management via RAII, and Rust does the same.

Rust does so at compile-time with guarantees, through ownership, which we'll discuss below.

Garbage Collection

You might be thinking: what's wrong with garbage collection?

The real answer is the magic word: performance!

Runtime and actual costs of collecting the garbage.