Rust 101 - Lecture 1

Jordan Hall

2022

DoCSoc

Who am I?

- Jordan Hall
- Born and raised in South Wales
- 2nd year Computing student at Imperial College London

A bit of my CV

- ML engineer at The Tower
- Data visualisation engineer at ICT Blue Ltd
- Will be interning at Microsoft during the summer

My Social Media

Github: JordanLloydHall

LinkedIn: Jordan Hall

Software bugs

Run-time bugs are rampant:

- Data races
- Null-pointer bugs
- Segmentation faults
- Double frees
- Use after free
- Memory leaks

These bugs/errors were manageable back when our code only needed to run on one thread. We left the impossible task of parallelism to operating systems developers, database software developers, network developers, ect. Other developers only had to wait until the next processor generation came out to double the speed of their code.

Software bugs

We have a problem:

- The clock speed of modern CPUs are stagnating, but the number of cores per chip is growing rapidly.
- We don't really know how to write safe, secure, and reliable concurrent code.

Potential Solutions:

- We have a runtime packaged with our code (like the Java Runtime Environment).
- We can write static analysers that check our code for common mistakes.
- Or we can rethink the way we write code to be better suited to these problems.

Software bugs

The root of the problem is this:

- We don't keep track of what code is doing to our data.
- Several parts of our program can change and read the data at the same time.
- Keeping track of when we need to free memory is not trivial.

Rust promises to solve these problems at compile-time.

Software bugs

```c
int main()
{
    int **arr = malloc(sizeof(int*));
    int **tmp = &arr[0];
    arr[0] = malloc(sizeof(int));

    free(tmp);
    arr[0] = 0;

    printf("%d\n", arr[0][0]);
    return 0;
}
```

Compile-time bugs

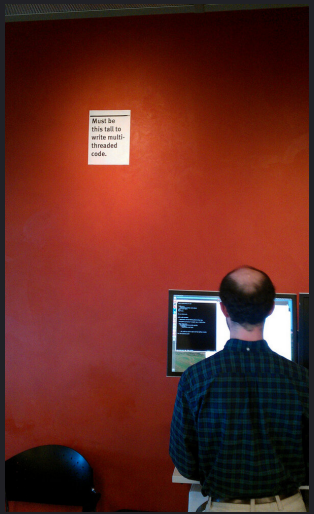
```c
int main()
{
    int **arr = malloc(sizeof(int*));
    int **tmp = &arr[0];                <- Unchecked Aliasing
    arr[0] = malloc(sizeof(int));       <- Mutation

    free(tmp);                          <- Free
    arr[0] = 0;                         <- Type incompatibility

    printf("%d\n", arr[0][0]);          <- Segmentation fault & use after free
    return 0;
}                                       <- Memory leak
```

What is Rust?

An actual photo from Mozilla offices:



What is Rust?

> My biggest compliment to Rust is that it's boring, and this is an amazing compliment.
>
> Chris Dickinson
> Engineer at npm, Inc

What is Rust?

Initially developed by Graydon Hoare, an employee at Mozilla in 2006.

Graydon noticed that making concurrent programs that are fast was practically impossible.

Mozilla had an feature request wherein a component of Firefox needed to be optimised for multi-core CPUs...

That issue was open for 7 years!

What is Rust?

Rust is a systems programming language with three major pillars:

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Where is Rust now?

I would love to see Rust take that place [of C/C++]. I have been following an OS written entirely in Rust, and it has great idioms.

Miguel de Icaza
Founder of GNOME, Mono, and Xamarin Projects

What is Rust?

In May 2015 Rust 1.0 was released into the public.

In 2020 Mozilla let go of 250 out of their 1000 employees - including the Rust team and the Servo team.

In 2021 the Rust Foundation was announced by its 5 founding companies: AWS, Huawei, Google, Microsoft and Mozilla.

Following this, Google added Rust support for the Android Open Source Project (making C/C++ no longer necessary).

Where is Rust now?

Rust is 7 years old now.

It is already used everywhere:

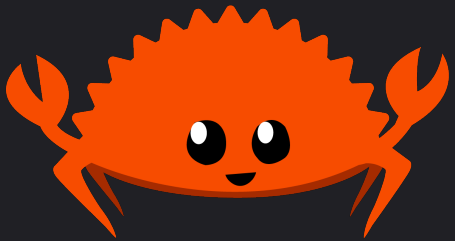
Microsoft, Google, Amazon, Facebook, Discord, Figma, Dropbox, NPM, etc.

In Stack Overflow's 2021 annual survey, Rust was rated the #1 most loved language - for the 6th year in a row.

Crates.io has 75,600 crates, with 13 billion downloads.

Has become the Linux Kernel's first ever secondary language, and even Linus happy about it!

Rust 101!



Roadmap


Rust 101
Concurrency
Mutex/Arc
Send/Sync Traits
Rayon
Types
Enums
Structs
Compound Types
Primitives
Functions
Pure Functions
Closures
Higher-Order Functions
Build Tools
Clippy
Automated Tests
Cargo
Formatter
Smart Pointers
Rc
Box
Traits
Generic Types
Traits
Lifetimes
Control Flow
If/While/For/Loop Statements
If/While Let
Control Flow as Expressions
Question Mark Operator
The Borrow Checker
Borrowing
Ownership
References

Basic Layout

```
$ cargo new hello_world
```

In src/main.rs:

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

```
$ cargo run
```

```
Hello World!
```

Types

Primitive Types

Unsigned Integers:

```
u8, u16, u32, u64, u128, usize
```

```
let x: u8 = 42;
```

Signed Integers:

```
i8, i16, i32, i64, i128, isize
```

```
let y = -42; // f32
```

Primitive Types (continued)

Floating Point Numbers:

```
f32, f64
```

```
let f1 = 3.141592654; // f64
```

```
let f2: f32 = 1.61803398875;
```

More types:

```
bool, str, char
```

```
let b = true;
let c = 'q'; // Also supports emojis! (but my Latex editor can't)
```

Printing Variables

```rust
fn main() {
    let answer = 42;
    println!("Your answer is: {}", answer);
}
```

```
Your answer is 42
```

Basic Arithmetic

```
fn main() {
    let x = 10 - 5;
    let y = x * 2;
    let z = y == 5;
    let w = x + z; // Is there something wrong here?
}
```

Questions:

1. What types are x, y, z?
2. What's wrong with w?

Variable Redefinition

```
fn main() {
    let x = 5;
    let x = true;
    println!("{}", x);
}
```

Questions:

1. What does the Rust compiler think about this?
2. What will be printed if this program is run?
3. What does this say about the scopes of variables?

Mutability

```
fn main() {
    let x = 96;
    x = 42;
    println!("{}", x);
}
```

Questions:

1. What does the Rust compiler think about this?
2. What will be printed if this program is run?
3. What does this say about variables in Rust?

Mutability (continued)

```
fn main() {
    let mut x = 96;
    x = 42;
    println!("{}", x);
}
```

1. What does the Rust compiler think about this?
2. What will be printed if this program is run?
3. What does this say about variables in Rust?

Constant and Static Variables

```
const I_AM_A_CONSTANT: u32 = 42;
static TRUTH: bool = I_AM_A_CONSTANT == 42;

fn main() {
    println!("Is I_AM_A_CONSTANT_42? {}", TRUTH)
}
```

Questions:

1. Do the types need to be specified?
2. What is the difference between static and const?

Unsafe Code

```
const I_AM_A_CONSTANT: u32 = 42;
static mut TRUTH: bool = I_AM_A_CONSTANT == 42;
fn main() {
    unsafe {
        TRUTH = false;
        println!(
            "I_AM_A_CONSTANT == 42 is {}, but TRUTH tells me it's {}",
            I_AM_A_CONSTANT == 42,
            TRUTH
        );
    }
}
```

Questions:

1. What does unsafe mean?
2. Why do we need unsafe for mutable static variables?

Type Casting

```
fn main() {
    let x: u32 = 10;
    let y = 8u8;
    let z: usize = x + y;
}
```

Questions:

1. What does the Rust compiler think of this?
2. What does this say about type inference in Rust?

Type Casting (continued)

```
fn main() {
    let x: u32 = 10;
    let y = 8u8;
    let z: usize = (x + (y as u32)) as usize;
    let w: usize = x as usize + y as usize;
}
```

Questions:

1. What does the Rust compiler think of this?
2. What does this say about type inference in Rust?

Compound Types

```
let arr = [1, 2, 3, 4, 5];
let arr = [1u8; 15];
let x = arr[14];

let tup = (1, true, 'h');
let i = tup.0;
let b = tup.1;
let c = tup.2;
```

Questions:

1. What do the types of arrays and tuples look like?
2. What does this say about type inference in Rust?

Compound Types (continued)

```
let arr: [i32; 5] = [1, 2, 3, 4, 5];
let arr: [u8; 15] = [1u8; 15];
let x: u8 = arr[14];

let tup: (i32, bool, char) = (1, true, 'h');
let i: i32 = tup.0;
let b: bool = tup.1;
let c: char = tup.2;
```

Questions:

1. What do the types of arrays and tuples look like?
2. What does this say about type inference in Rust?

Structs

```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}

fn main() {
    let tony = User {
        active: true,
        username: String::from("Tony"),
        email: String::from("OohBaby42@haskell.org"),
        sign_in_count: 12345
    };
    println!("{} has logged in {} times!", tony.username, tony.sign_in_count);
}
```

Structs (continued)

```
struct RGBColor(u8, u8, u8);

fn main() {
    let color = RGBColor(183, 65, 14);
    println!(
        "The color you have chosen is: R: {}, G: {}, B: {}",
        color.0, color.1, color.2
    );
}
```

Enums

```rust
enum Color {
    RGB(u8, u8, u8),
    Named(String),
    HexRGB(u32)
}
fn main() {
    let color = Color::HexRGB(12009742);
    match color {
        Color::RGB(r, g, b) => println!("R:{}, G:{}, B:{}", r, g, b),
        Color::Named(s) => println!("{}", s),
        Color::HexRGB(v) => println!("{:#06x}", v)
    }
}
```

```rust
let color = Color::HexRGB(12009742); // prints "0xb7410e";
let color = Color::Named("Rust"); // prints "Rust"
let color = Color::RGB(183, 65, 14); // prints "R:183, G:65, B:14"
```

Control Flow

If Statement

```
fn main() {
    let x = 42;
    if x % 2 == 0 {
        println!("The answer is an even number")
    } else {
        println!("The answer is an even number")
    }
}
```

Questions:

1. Why is there no need for semi-colons at the print statement's arms?
2. What types are allowed in the condition of a print statement?

If Statement (continued)

```
fn main() {
    let x = 42;
    let y = if x % 2 == 0 {
        x / 2
    } else {
        3 * x + 1
    };
}
```

Questions:

1. What happens if the arms of the expression are different types?
2. What is the type of y?
3. What happens if we don't add a second branch to the if statement?

While Statement

```
fn main() {
    let mut x = 42;
    let mut biggest_num = x;
    while x != 1 {
        x = if x % 2 == 0 { x / 2 } else { 3 * x + 1 };
        biggest_num = biggest_num.max(x);
    }
    assert_eq!(64, biggest_num);
}
```

Questions:

1. Prove or disprove that for all $x \in \mathbb{N}$, this program halts.

Loop Statement

```
fn main() {
    loop {
        println!("Are we done yet?")
    }
}
```

Loop Statement (continued)

```
fn main() {
    let n = 5;
    let mut counter = 1;
    let mut tmp = 1;
    let fact = loop {
        if counter == n {
            break tmp
        }
        counter += 1;
        tmp *= counter;
    };
    assert_eq!(120, fact);
}
```

For Loop

```
fn main() {
    for i in 1..=15 {
        let fizz = i % 3 == 0;
        let buzz = i % 5 == 0;
        print!("{:02} ", i);
        if fizz {
            print!("fizz");
        }
        if buzz {
            print!("buzz");
        }
        println!();
    }
}
```

For Loop (continued)

```
fn main() {
    let arr = [1, 2, 3, 4, 5];
    let mut sum = 0;
    for i in arr.into_iter() {
        sum += i;
    }
    assert_eq!(15, sum);
}
```

Questions:

1. What does into_iter() do?
2. Is there a better way again to do this?

Match Statement

```
fn main() {
    let age = 42;
    match age {
        0..=17 => println!("This person can't buy alcohol!"),
        120.. => println!("Are you sure you have the right age?"),
        _ => println!("This person can buy alcohol!")
    };
}
```

Questions:

1. What happens if not all of the cases are met?
2. What happens when you put floating point numbers in the match arms?

Match Statement (continued)

```
fn main() {
    let age = 42;
    let can_buy_alcohol = match age {
        0..=17 | 120.. => false,
        _ => true
    };
    assert!(can_buy_alcohol);
}
```

Questions:

1. What happens if not all of the cases are met?
2. What happens when you put floating point numbers in the match arms?

Functions

Primitive Functions

```
fn print_me(x: i32) {
    println!("{}", x);
}
fn main() {
    print_me(42);
}
```

Questions:

1. What are the types in this function?

Primitive Functions (continued)

```
fn multiply(x: i32, y: i32) -> i32 {
    x * y
}
fn main() {
    assert_eq!(12, multiply(3, 4))
}
```

Questions:

1. Where is the return statement?

Recursive Functions

```
fn fib(n: u32) -> u32 {
    match n {
        0 | 1 => n,
        _ => fib(n-1) + fib(n-2)
    }
}
fn main() {
    assert_eq!(55, fib(10));
}
```

Questions:

1. Do the types of each return need to match?

```
fn double(x: i32) -> i32 {
    x * 2
}
fn main() {
    let arr = [1, 2, 3, 4, 5];
    assert_eq!([2, 4, 6, 8, 10], arr.map(double));
}
```

Multiple comments about this implementation:

1. Defining a global function just to be used once as a higher order function is messy.
2. Function primitives can't capture scope.

But there's a better way!

Closures

```
fn main() {
    let double = |x| x * 2;
    assert_eq!(8, double(4));
}
```

Questions:

1. What are the types of double()?
2. What happens if we remove the use of double?

Closures (continued)

```
fn main() {
    let n = 2;
    let arr = [1, 2, 3, 4, 5];
    assert_eq!([2, 4, 6, 8, 10], arr.map(|x| x * n));
}
```

A Problem with Functions

```
fn sign_in_user(mut user: User) -> User {
    user.sign_in_count += 1;
    user.active = true;
    user
}
fn main() {
    let tony = User {
        active: false,
        username: String::from("Tony"),
        email: String::from("OohBaby42@haskell.org"),
        sign_in_count: 12345
    };
    let tony = sign_in_user(tony);
    assert_eq!(12346, tony.sign_in_count);
    assert!(tony.active);
}
```

A Problem with Functions (continued)

```
fn length_of_username(user: User) -> usize {
    user.username.len()
}
fn main() {
    let tony = User {
        active: false,
        username: String::from("Tony"),
        email: String::from("OohBaby42@haskell.org"),
        sign_in_count: 12345
    };
    let l = length_of_username(tony);
    assert_eq!(4, l);
    assert!(!tony.active)
}
```

A Problem with Functions (continued)

```
error[E0382]: use of moved value: `tony`
  --> src/main.rs:20:14
   |
12 |     let tony = User {
   |         ---- move occurs because `tony` has type `User`, which does not implement
   |                the `Copy` trait
...
18 |     let l = length_of_username(tony);
   |                                ---- value moved here
19 |     assert_eq!(4, l);
20 |     assert!(!tony.active)
   |             ^^^^^^^^^^ value used here after move

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