

CMSC 330: Organization of Programming Languages

Safe, Low-level Programming with **Rust**

CMSC330 Spring 2019

Copyright © 2018-19 Michael Hicks, the University of Maryland. Some material based on <https://doc.rust-lang.org/book/second-edition/index.html>

What choice do programmers have today?

C/C++

- Low level
- More control
- Performance over safety
- Memory managed manually
- No periodic garbage collection
- ...

Java, OCaml, Go, Ruby...

- High level
- Secure
- Less control
- Restrict direct access to memory
- Run-time management of memory via periodic garbage collection
- No explicit malloc and free
- Unpredictable behavior due to GC
- ...

Rust: Type safety and low-level control

- Begun in 2006 by Graydon Hoare
- Sponsored as full-scale project and announced by Mozilla in 2010
 - Changed a lot since then; source of frustration
 - But now: **most loved programming language** in Stack Overflow annual surveys of **2016**, **2017**, and **2018**
- Takes ideas from **functional** and **OO** languages, and **recent research**
- Key properties: **Type safety** despite use of **concurrency** and **manual memory management**
 - And: **No data races**

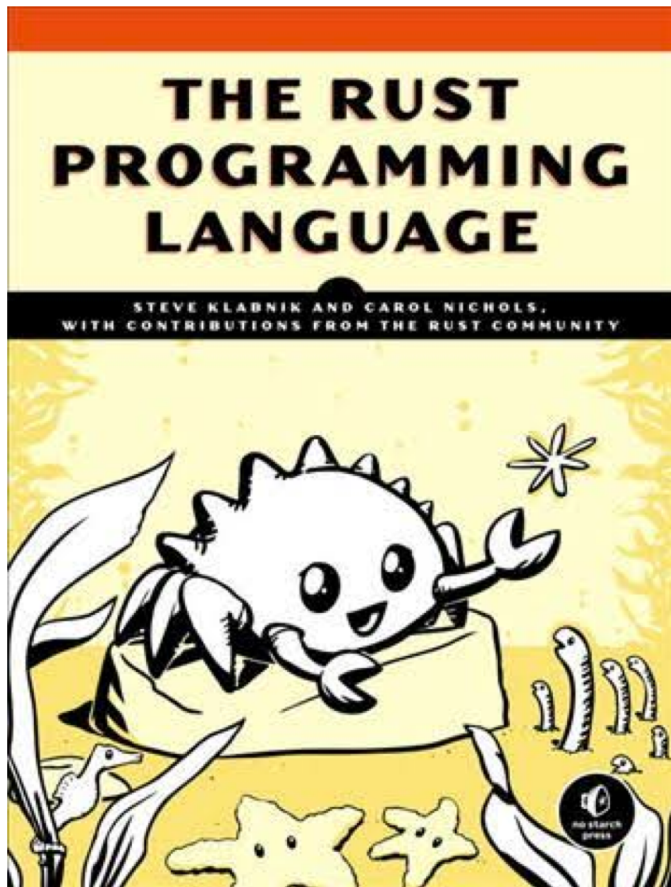
Features of Rust

- Lifetimes and Ownership
 - Key feature for ensuring safety
- Traits as core of object(-like) system
- Variable default is immutability
- Data types and pattern matching
- Type inference
 - No need to write types for local variables
- Generics (aka parametric polymorphism)
- First-class functions
- Efficient C bindings

Rust in the real world

- Firefox Quantum and Servo components
 - <https://servo.org>
- REmacs port of Emacs to Rust
 - <https://github.com/Wilfred/remacs>
- Amethyst game engine
 - <https://www.amethyst.rs/>
- Magic Pocket filesystem from Dropbox
 - <https://www.wired.com/2016/03/epic-story-dropboxs-exodus-amazon-cloud-empire/>
- OpenDNS malware detection components
- <https://www.rust-lang.org/en-US/friends.html>

Information on Rust



- **Rust book** free online
 - <https://doc.rust-lang.org/book/>
 - **We will follow it in these lectures**
- More references via Rust site
 - <https://www.rust-lang.org/en-US/documentation.html>
- Rust Playground (REPL)
 - <https://play.rust-lang.org/>

Installing Rust

- Instructions, and stable installers, here:

<https://www.rust-lang.org/en-US/install.html>

- On a Mac or Linux (VM), open a terminal and run

`curl https://sh.rustup.rs -sSf | sh`

- On Windows, download+run `rustup-init.exe`

<https://static.rust-lang.org/rustup/dist/i686-pc-windows-gnu/rustup-init.exe>

Rust compiler, build system

- Rust programs can be compiled using `rustc`
 - Source files end in suffix `.rs`
 - Compilation, by default, produces an executable
 - No `-c` option
- Preferred: Use the `cargo` package manager
 - Will invoke `rustc` as needed to build files
 - Will download and build dependencies
 - Based on a `.toml` file and `.lock` file
 - You won't have to mess with these for this class
 - Like `ocamlbuild` or `dune`

Using rustc

- Compiling and running a program

main.rs:

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

```
% rustc main.rs
```

```
% ./main
```

```
Hello, world!
```

```
%
```

Using cargo

- Make a project, build it, run it

```
% cargo new hello_cargo --bin
```

```
% cd hello_cargo
```

```
% ls
```

```
Cargo.toml    src/
```

```
% ls src
```

```
main.rs
```

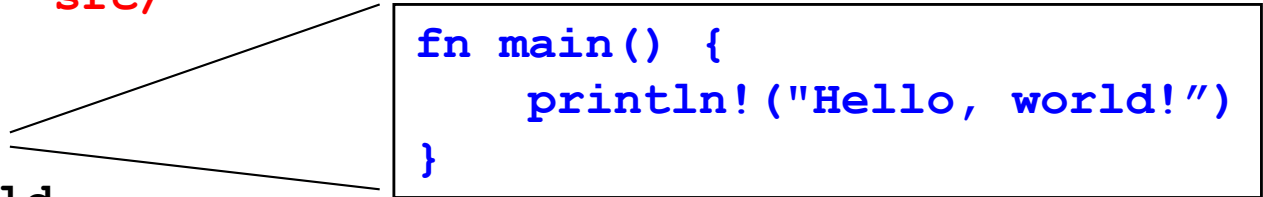
```
% cargo build
```

```
Compiling hello_cargo v0.1.0 (file:///...)
```

```
Finished dev [unoptimized + debuginfo] ...
```

```
% ./target/debug/hello_cargo
```

```
Hello, world!
```



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

More at <https://doc.rust-lang.org/stable/cargo/getting-started/first-steps.html>

Rust, interactively

- Rust has no top-level *a la* OCaml or Ruby
- There is an in-browser execution environment
 - See, for example, <https://doc.rust-lang.org/stable/rust-by-example/hello.html>

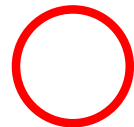
Hello World

This is the source code of the traditional Hello World program.

```
// This is the main function
fn main() {
    // The statements here will be executed when the compiled binary is called

    // Print text to the console
    println!("Hello World!");
}
```

Hello World!



Rust Documentation

- Your go-to to learn about Rust is the Rust documentation page
 - <https://doc.rust-lang.org/stable/>
- This contains links to
 - the Rust Book (on which most of our slides are based),
 - the reference manual, and
 - short manuals on the compiler, cargo, and more

Rust Basics

Functions

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

Hello, world!

Factorial in Rust (recursively)

```
fn fact(n:i32) -> i32
```

```
{  
    if n == 0 { 1 }  
    else {  
        let x = fact(n-1);  
        n * x  
    }  
}
```

```
fn main() {  
    let res = fact(6);  
    println!("fact(6) = {}",res);  
}
```

fact(6) = 720

If *Expressions* (not Statements)

```
fn main() {  
    let n = 5;  
    if n < 0 {  
        print!("{}", is negative", n);  
    } else if n > 0 {  
        print!("{}", is positive", n);  
    } else {  
        print!("{}", is zero", n);  
    }  
}
```

5 is positive

Let Statements

- By default, Rust variables are immutable
 - Usage checked by the compiler
- **mut** is used to declare a resource as mutable.

```
fn main() {  
    let a: i32 = 0;  
    a = a + 1;  
    println!("{}", a);  
}
```

```
fn main() {  
    let mut a: i32 = 0;  
    a = a + 1;  
    println!("{}", a);  
}
```

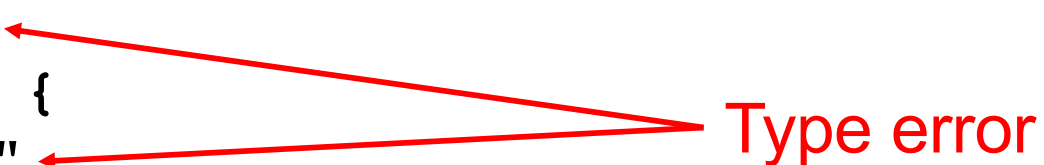
Compile error

Let Statements

```
fn main() {  
    let x = 5;  
  
    let x: i32 = 5; //type annotation  
  
    let mut x = 5; //mutable x: i32  
    x = 10;  
}
```

If Expressions

```
fn main() {  
    let n = 5;  
    let x = if n < 0 {  
        10  
    } else {  
        "a"  
    };  
  
    print!("{:?}", x);  
}
```



Type error

Let Statement Usage Examples

```
{  
  let x = 37;  
  let y = x + 5;  
  y  
} //42
```

```
{  
  let x = 37;  
  x = x + 5; //err  
  x  
}
```

```
{ //err:  
  let x:u32 = -1;  
  let y = x + 5;  
  y  
}
```

```
{  
  let x = 37;  
  let x = x + 5;  
  x  
} //42
```

```
{  
  let mut x = 37;  
  x = x + 5;  
  x  
} //42
```

```
{  
  let x:i16 = -1;  
  let y:i16 =  
x+5;  
  y  
} //4
```

Redefining a variable *shadows* it (like OCaml)

Assigning to a variable only allowed if **mut**

Type annotations must be consistent (may override defaults)

Quiz 1: What does this evaluate to?

```
{ let x = 6;  
  let y = "hi";  
  if x == 5 { y } else { 5 };  
  7  
}
```

- A. 6
- B. 7
- C. 5
- D. Error

Quiz 1: What does this evaluate to?

```
{ let x = 6;  
  let y = "hi";  
  if x == 5 { y } else { 5 };  
  7  
}
```

A. 6

B. 7

C. 5

D. Error – if and else have incompatible types

Quiz 2: What does this evaluate to?

```
{ let x = 6;  
  let y = 4;  
  let x = 8;  
  x == 10-y  
}
```

- A. 6
- B. true
- C. false
- D. error

Quiz 2: What does this evaluate to?

```
{ let x = 6;  
  let y = 4;  
  let x = 8;  
  x == 10-y  
}
```

- A. 6
- B. true
- C. false
- D. error

Using Mutation

- Mutation is useful when performing iteration
 - As in C and Java

```
fn fact(n: u32) -> u32 {  
  let mut x = n;  
  let mut a = 1;  
  loop {  
    if x <= 1 { break; }  
    a = a * x;  
    x = x - 1;  
  }  
  a  
}
```

infinite loop
(break out)

Other Looping Constructs

- While loops
 - `while e block`
- For loops
 - `for pat in e block`
 - More later – e.g., for iterating through collections

```
for x in 0..10 {  
    println! ("{}", x); // x: i32  
}
```

Other Looping Constructs

- These (and `loop`) are *expressions*
 - They return the final computed value
 - unit, if none
 - **break** may take an expression argument, which is the final result of the loop

```
let mut x = 5;
let y = loop {
    x += x - 3;
    println!("{}", x); // 7 11 19 35
    x % 5 == 0 { break x; }
};
print!("{}", y); //35
```

Quiz 3: What does this evaluate to?

```
let mut x = 1;  
for i in 1..6 {  
    let x = x + 1;  
}  
x
```

- A. 1
- B. 6
- C. 0
- D. error

Quiz 3: What does this evaluate to?

```
let mut x = 1;  
for i in 1..6 {  
    let x = x + 1;  
}  
x
```

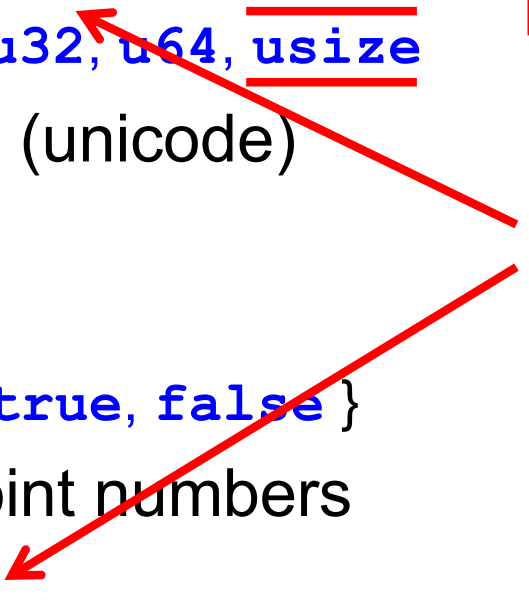
A. 1

B. 6

C. 0

D. error

Data: Scalar Types

- Integers
 - `i8`, `i16`, `i32`, `i64`, `isize`
 - `u8`, `u16`, `u32`, `u64`, `usize`
 - Characters (unicode)
 - `char`
 - Booleans
 - `bool` = { `true`, `false` }
 - Floating point numbers
 - `f32`, `f64`
 - Note: arithmetic operators (+, -, etc.) *overloaded*
- Machine word size
- Defaults (from inference)
- 

Compound Data: Tuples

- Tuples
 - n-tuple **type** (*t1*, ..., *tn*)
 - `unit ()` is just the 0-tuple
 - n-tuple **expression** (*e1*, ..., *en*)
 - Accessed by pattern matching or like a record field

```
let tuple = ("hello", 5, 'c');  
assert_eq!(tuple.0, "hello");  
let (x,y,z) = tuple;
```

Compound Data: Tuples

Distance between two points $s:(x_1,y_1)$ $e:(x_2,y_2)$

```
fn dist(s: (f64, f64), e: (f64, f64)) -> f64 {  
    let (sx, sy) = s;  
    let ex = e.0;  
    let ey = e.1;  
    let dx = ex - sx;  
    let dy = ey - sy;  
    (dx*dx + dy*dy).sqrt()  
}
```

Compound Data: Tuples

Can include patterns in parameters directly, too

```
fn dist2( (sx,sy) : (f64,f64) , (ex,ey) : (f64,f64) ) -> f64 {  
    let dx = ex - sx;  
    let dy = ey - sy;  
    (dx*dx + dy*dy).sqrt()  
}
```

We'll see Rust `structs` later. They generalize tuples.

Arrays

- Standard operations
 - Creating an array (can be mutable or not)
 - But must be of fixed length
 - Indexing an array
 - Assigning at an array index

```
let nums = [1,2,3];  
let strs = ["Monday", "Tuesday", "Wednesday"];  
let x = nums[0]; // 1  
let s = strs[1]; // "Tuesday"  
let mut xs = [1,2,3];  
xs[0] = 1; // OK, since xs mutable  
let i = 4;  
let y = nums[i]; //fails (panics) at run-time
```

Array Iteration

- Rust provides a way to **iterate over a collection**
 - Including arrays

```
let a = [10, 20, 30, 40, 50];  
for element in a.iter() {  
    println!("the value is: {}", element);  
}
```

- `a.iter()` produces an **iterator**, like a Java iterator
 - This is a **method call**, *a la* Java. More about these later
- The special `for` syntax issues the `.next()` call until no elements are left
 - No possibility of running out of bounds

Quiz 4: Will this function type check?

```
fn f(n: [u32]) -> u32 {  
    n[0]  
}
```

- A. Yes
- B. No

Quiz 4: Will this function type check?

```
fn f(n: [u32]) -> u32 {  
    n[0]  
}
```

A. Yes

**B. No – because
array length not
known**

Fun Fact

- The original Rust compiler was written in OCaml
 - Betrays the sentiments of the language's designers!
- Now the Rust compiler is written in ... Rust
 - How is this possible? Through a process called **bootstrapping**:
 - The first Rust compiler written in Rust is compiled by the Rust compiler written in OCaml
 - Now we can use the binary from the Rust compiler to compile itself
 - We discard the OCaml compiler and just keep updating the binary through self-compilation
 - So don't lose that binary! 😊