Introduction to Rust
What is Rust?

- Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.
  - www.rust-lang.org

- Out of Mozilla

- Used in Firefox today on Win/Mac/Linux (Android coming soon)

- The Servo HTML5 rendering engine (replacing Gecko) is their use-case
Why should I care?

- Fast like C with excellent C inter-op
- Segmentation faults are impossible*
- Null-pointer dereferences are impossible*
- Buffer overflows are impossible*
- First class build system / documentation generator / code formatting
- Rich, expressive type system
- But unlike C++, the types are sane (e.g. std::string)
The Type System

- Scalars (u8, i16, f64, etc)
- Arrays
- Structs
- Plain enums
- Tagged enums
- Char (32-bit Unicode Scalar Values)
- Strings (of Unicode characters stored as UTF-8 octets)
- Slices
- References

- Types (even scalars) can implement Traits – this is what Rust uses for inheritance
- The Drop Trait – Rust’s idea of destructors
- Types can use Generics
enum Message {
    Quit,
    ChangeColor(i32, i32, i32),
    Move { x: i32, y: i32 },
    Write(String)
}

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

enum Result<T, E> {
    Ok(T),
    Err(E)
}
trait Foo {
    fn method(&self) -> String;
}

impl Foo for u8 {
    fn method(&self) -> String {
        format!("u8: {}", *self)
    }
}

impl Foo for String {
    fn method(&self) -> String {
        format!("string: {}", *self)
    }
}

let msg = 35.method();
Memory safety

- Impossible to run off the end of an array
  - You take a ‘slice’ of an array – knows its own length

- Impossible to leak memory from the heap

- Impossible to return a reference to memory on the stack

- Impossible to dereference a null pointer
  - Recall Option<T> - allows you to express Some(T) or None

- Can break these rules by using raw pointers in ‘unsafe’ blocks
The Borrow Checker

- You can either have one mutable reference to an object, or multiple immutable references, but not both.

- Why? Guarantees no race-hazards.

- Checked at compile time.

- You can bypass this rule if you really need to, by using an immutable object to wrap a mutable object (subject to some constraints)
match – switch on steroids

```rust
fn get_value() -> Option<i32> { ... }

fn test() {
    let x = match get_value() {
        Some(i) if i > 5 => "Got an int > 5!",
        Some(..) => "Got an int!",
        None => {
            report_fail();
            "No such luck."
        },
    },
    println!("x = {}", x);
}
```
Modules

- No more header files!

- `struct Foo { ... };

- `pub struct Bar { ... };

- `use my_module::{Baz, Qux};

- `pub use my_module::Baz;

- (Also, proper macros, not just text substitution)
Debug output

```rust
use std::fmt;

struct Point { x: i32, y: i32, }

impl fmt::Debug for Point {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "Point {{ x: {}, y: {} }}", self.x, self.y)
    }
}

let origin = Point { x: 0, y: 0 };
println!("The origin is: {:+}", origin);
```
Debug output

```rust
use std::fmt;

#[derive(Debug)]
struct Point { x: i32, y: i32, }

let origin = Point { x: 0, y: 0 };
println!("The origin is: {:?}", origin);
```
Rich standard API

- Collections
  - Vector, HashMap, BTreeMap, BinaryHeap, etc
- Sockets/Networking
- Threads
- Processes
- Channels (message passing)
- I/O & path manipulation
- etc
Cargo – the build system

- cargo build
- cargo run
- cargo test
- cargo bench
- cargo install ...
[package]
name = "hello_world"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]

[dependencies]
time = "0.1.12"
regex = "0.1.41"
Crates.io – the package repository

- [https://crates.io](https://crates.io)

- First-come, first-served on package names
RustDoc – the documentation generator

- cargo doc

- Dog-fooding - as used by [https://doc.rust-lang.org/std/](https://doc.rust-lang.org/std/)
RustFmt – the code formatter

- cargo install rustfmt
- cargo fmt
- No more arguments about formatting…
What’s the bad news?

- Compiling for bare-metal requires unstable features (i.e. subject to change)

- Binary format is not yet stable
  - Cannot yet use an *rlib* built in compiler version A with compiler version B
  - So, crates are usually distributed as source

- The compiler’s a bit slow (but it’s getting faster)
Where can I learn more?

- Website - https://www.rust-lang.org
- API Docs - https://doc.rust-lang.org/std/
- Reddit – https://reddit.com/r/rust
- This Week In Rust

- Jonathan Pallant (github.com/thejpster), Doug Young, Dan Cannell, Andrew Featherstone…