Rust for C++ Developers

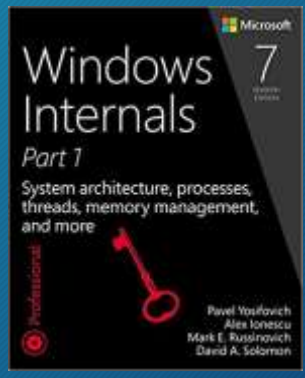
Pavel Yosifovich

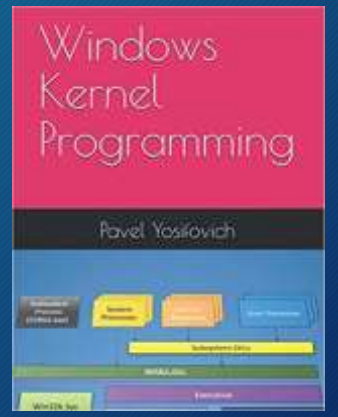
@zodiacon

zodiacon@live.com

About Me

- Developer, Trainer, Author and Speaker
- Book author
  - “Windows Kernel Programming” (2019)
  - “Windows Internals 7th edition, Part 1” (co-author, 2017)
  - “WPF 4.5 Cookbook” (2012)
- Pluralsight author
- Author of several open-source tools (http://github.com/zodiacon)
- Blogs: http://blogs.microsoft.co.il/pavely, http://scorpiosoftware.net





Agenda

- What is Rust?
- Rust vs. C++
- Code
- Summary
- Q & A

What is Rust?

- Performance
  - Rust is blazingly fast and memory-efficient
- Reliability
  - Rust’s rich type system and ownership model guarantee memory-safety and thread-safety
- Productivity
  - Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling
- What’s not to like?

Types

- Primitive types
- Structs / classes
- Enumerations
- Unions
- (Interfaces)
- Inheritance
- Polymorphism
- Attributes
- Templates

- Primitive types (+associated methods)
- Structs
- Enumerations
- Enumerations!
- Traits
- Trait Inheritance
- Polymorphism
- Traits!
- Generics

The Ownership Model

- Single owner or shared ownership
- Developer managed
- Assignment and copy construction mean “copy”
  - Unless R-value provided or `std::move` used explicitly (and there is a move ctor/assignment)

- Single owner
- Explicit
- Compiler enforced
  - a.k.a. “Borrow Checker”
- Assignment means “move”
  - Unless type implements `Copy` trait
- Borrowing

Copy vs. Move

```cpp
vector<int> v1{ 1, 2, 3 };
auto v2 = v1;
auto v3 = v1;

cout << v1.size() << " " << v2.size()
    << " " << v3.size() << endl;
```

```
3 3 3
```

```rust
let v1 = vec![1, 2, 3];
```

```
Running "cargo build":
Compiling hello v0.1.0
error[E0382]: use of moved value: `v1`
--> src\main.rs:18:14
|
```

```rust
let v1 = vec![1, 2, 3];
let v2 = v1.clone();
let v3 = v1.clone();

println!("{} {} {}",
    v1.len(), v2.len(), v3.len());
```

Borrowing


```
fn greet(s: String) {
}
```

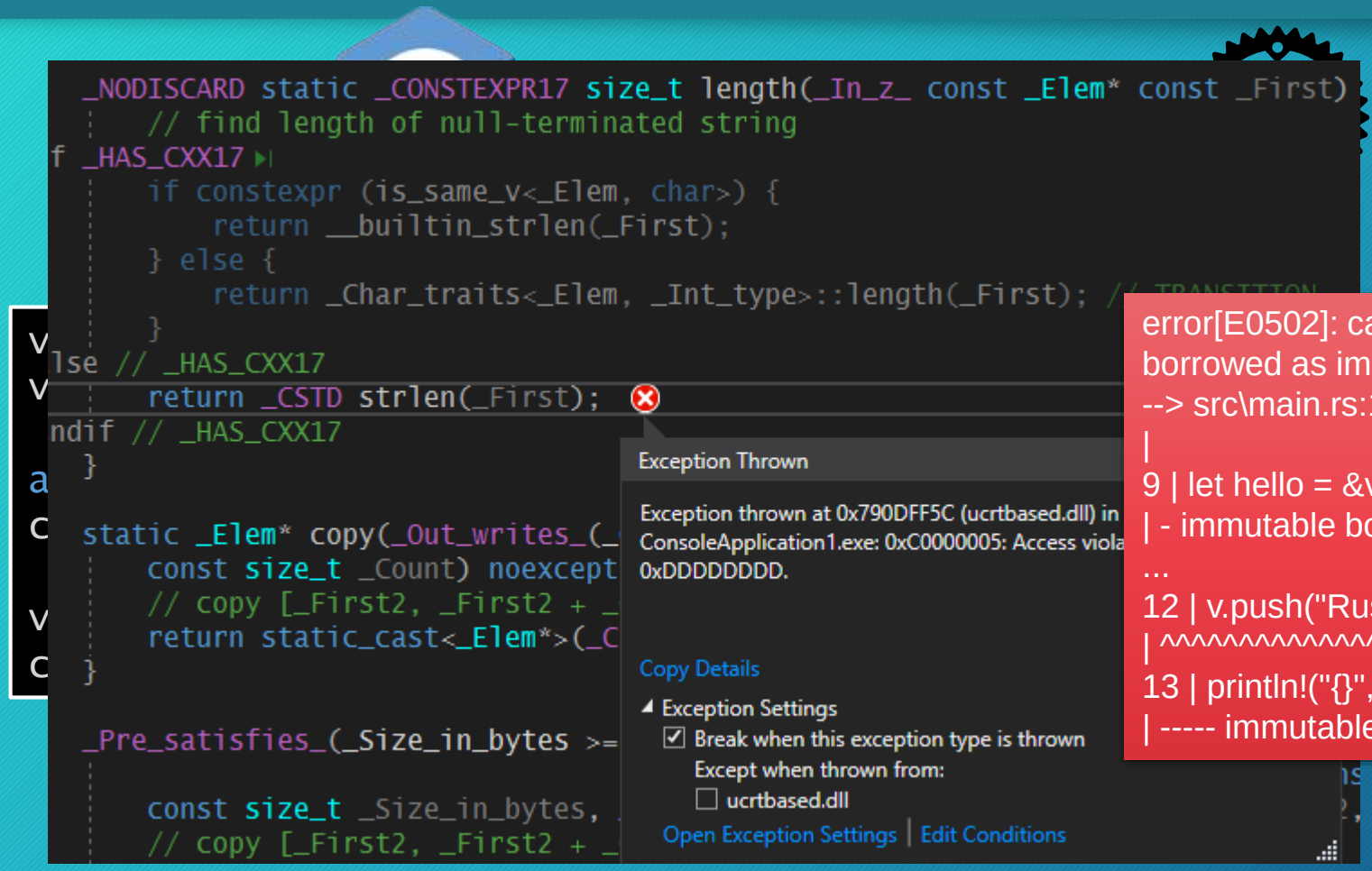
```
fr
}
```

```
error[E0382]: borrow of moved value: `name`
--> src\main.rs:41:31
|
39 | let name = String::from("Pavel");
| ---- move occurs because `name` has type
`std::string::String`, which does not implement the `Copy` trait
40 | greet(name);
| ---- value moved here
41 | println!("Hello again, {}!", name);
| ^^^^ value borrowed here after move
```

```
fn greet(s: &String) {
    println!("Hello, {}!", s);
}
```

```
fn main() {
    let name = String::from("Pavel");
    greet(&name);
    println!("Hello again, {}!", name);
}
```

Ownership & Borrowing




```
error[E0502]: cannot borrow `v` as mutable because it is also borrowed as immutable
--> src\main.rs:12:5
|
9 | let hello = &v[0];
| - immutable borrow occurs here
...
12 | v.push("Rust");
| ^^^^^^^^^^^^^^ mutable borrow occurs here
13 | println!("{}", hello);
| ----- immutable borrow later used here
```

Ownership and Borrowing

- `unique_ptr<T>`
- `shared_ptr<T>`
- References
- Default is non-const
  - Add `const` to declaration

- `Box<T>`
- `Rc<T>, Arc<T>`
- References (borrowing)
- Default is immutable
  - Add `mut` to declaration to mutate
- Multiple immutable references allowed
- Mutable reference means no other references can exist at that scope

Ownership

Demo

Enumerations (C++)

- A set of named values

```
enum class Season {
    Winter,
    Spring
    Summer,
    Fall
};
```

```
auto s = Season::Summer;
```

Enumerations (Rust)

- More than just named values (closer to C++ union)
- Can be generic
- Can have methods

```
enum Season {
    Winter,
    Spring,
    Summer,
    Fall
}
```

```
enum Result<T, E> {
    Ok(T),
    Err(E)
}
```

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

```
enum TurtleCommand {
    Forward(f32),
    Backwards(f32),
    Rotate(f32),
    RotateRight,
    RotateLeft,
    PenColor { r: u8, g: u8, b: u8 }
}
```

Pattern Matching and Enums

14

Demo

Traits

- Somewhat similar to C++ interfaces
  - Abstract class with pure virtual functions
- Some similar to attributes
- Basis for polymorphism
- Can inherit from other traits
- Some syntactic sugar in Rust is based on traits

Traits Examples


```
fn largest<T: PartialOrd + Copy>(list: &Vec<T>) -> T {
    let mut largest = list[0];

    for &item in list.iter() {
        if item > largest {
            largest = item;
        }
    }

    largest
}
```

```
let nums = vec![12, 33, 45, 3, 13, 40];
println!("largest: {}", largest(&nums));
```

Polymorphism with Traits

Demo

External Packages

- Large ecosystem
- The boost libraries
- Many other libraries out there
- No single repository
- (Microsoft has Nuget)

- Fast growing ecosystem
- Built-in package manager (Cargo)
- Each package is a “Crate”
- Central crates repository (crates.io)
- Easy to use and consistent

Packages

Demo

Summary

- Rust and C++ complete for (roughly) the same space
  - Both are native, statically typed, emphasize zero cost abstraction, stack over heap, etc.
- Rust has unique model for safety
  - “Borrow checker”
- C++ leaves safety to developers
  - Does provide types to help
- Rust supports pattern matching and functional style
  - Surprisingly rich in functionality and libraries
- Give rust a try!

Thank you!