Introduction to Rust Programming for Developers

Course Summary

Description

Rust was developed by the Rust team at Mozilla – the FireFox company – who describe it as “a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.” System programming languages, like C, have to be able to exert fine grained control over system resources, run fast with small memory foot prints, but they are also notorious for allowing programmers to write unsafe code that results in run time errors.

Rust is designed to produce code that runs as fast as C, but Rust’s syntax and semantics make it impos-sible to write code that would result in any form of run time memory error. As a second generation systems language Rust is able to work with modern CPU architectures to provide advanced features like low level, safe concurrency, and parallelism. Rust has exceptional support for legacy C modules and libraries.

Rust supports both functional and imperative programming, but it is also full featured enough for application development – it is used by DropBox, Coursera, and FireFox – and has been voted by StackOverflow as programmers’ most loved language of 2017. The popularity of Rust has exploded since its introduction and many organizations are seriously considering Rust as a replacement for C and C++.

Because of the unique design of Rust, even though the syntax and semantics are similar to C++, novel program language constructs – such as owning, borrowing and loaning memory – often make it difficult from developers to transition to Rust from other languages. This course is designed to make that transition seamless and painless by exploring the Rust language, showing students how to leverage their existing programming skills into coding in Rust, but also emphasizes where they have to rethink how they write code in order to work with Rust effectively.

As students work through the features of Rust in an instructor led, hands on manner, the underlying con-cepts behind Rust and what is happening “under the hood” are explored so that students wind up not just knowing how to write Rust code, but what Rust code to write and why it should be written that way. At the end of the course, will be able to write Rust code that is consistent with Rust best coding and design prac-tices.

The class is designed to be about 50% hands on labs and exercises, about 25% theory and 25% instructor led hands on learning where students code along with the instructor.

Topics

- Why Rust? Design goals of Rust – fast, efficient and safe systems programming.
- The Cargo build system and LLVM back-ends.
- Zero-cost abstractions and the compilation cycle.
- Data types, variables, pointers, slices and references.
- Ownership, mutability, references and borrowing.
- Rust flow of control constructs.
- Rust function basics.
- Defining and using structs.
- Enums and pattern matching.
- Collections – vectors, strings and hash maps.

- Recoverable errors and unrecoverable panics.
- Generic data types and shared behaviour with Traits.
- Validating references with Lifetimes.
- Writing, organizing and running tests in Rust.
- Functional programming in Rust.
- Smart pointers and Boxes.
- Concurrency in Rust – threads, messages and shared resources.
- Pattern matching in Rust.
- Unsafe Rust.
- Using Cargo and crates.
Introduction to Rust Programming for Developers

**Course Summary (cont’d)**

**Audience**

This course is designed for programmers who want to get started programming in Rust.

**Prerequisites**

Before taking this course, an intermediate level of skill in and a solid knowledge of a high level programming language like Java, Python or C is essential. Students who do not have this prerequisite may struggle with the content and pace of the course.

**Duration**

Three days