Document Generated: 02/25/2026

Learning Style: Virtual Classroom

Technology:

Difficulty: Intermediate

Course Duration: 5 Days

Hands-on Rust Programming for C++ Programmers (TTRS2105)

About This Course:

Rust is a modern programming language designed for performance and safety, particularly in concurrent systems, offering a unique combination of efficiency and reliability. Learning Rust equips you with the ability to write high-performance, bug-

free code, greatly enhancing your programming toolkit.

Geared for C++ experienced developers, Hands-on Rust Programming for C++ Programmers is designed to help you transition from C++ to Rust with confidence. Led by an expert trainer, the course combines engaging lectures with hands-on labs, ensuring you grasp Rust's unique features and gain practical experience. You'll explore Rust's syntax, control flow, and module imports, and move on to advanced topics like memory management, concurrency, and pattern matching. By leveraging Cargo, you'll streamline project management and integrate popular crates like Serde for serialization, Tokio for asynchronous programming, and SQLx for database interactions, enabling you to build robust, scalable applications.

The workshop-style format dedicates half of the class time to hands-on labs, applying what you've learned in real-world scenarios. You'll gain expertise in writing concurrent programs using Rust’s powerful tools, such as threads, Mutex, RwLock, and async/await, ensuring your applications handle multiple tasks efficiently. You'll also master pattern matching, creating readable and maintainable code. By the end of the course, you'll be confident in writing comprehensive tests and documentation, making you a highly effective programmer ready to tackle complex Rust projects and advance your career

Course Objectives:

- Master Rust Syntax and Semantics: Get comfortable writing and understanding Rust code, including how to use control flow, functions, and module imports.
- Implement Memory Management: Learn to manage memory effectively in Rust by mastering concepts like ownership, borrowing, and lifetimes to ensure your code is both safe and efficient.
- Leverage Cargo and Crates: Become adept at using Cargo to manage Rust projects and dependencies, and explore how to integrate popular crates like Serde, Tokio, and SQLx to add powerful features to your applications.
- Utilize Rust's Concurrency Model: Discover how to write efficient and concurrent programs using Rust’s concurrency tools, such as threads, Mutex, RwLock, and async/await.
- Employ Pattern Matching and Generics: Understand and apply pattern matching with enums and functions, and use generics to create flexible and reusable code.
- Create and Test Rust Applications: Develop complete Rust applications, including writing tests with Rust’s testing framework and generating clear, comprehensive documentation using Rustdoc.

Audience:

- This course is designed for C++ developers looking to expand their programming skills with Rust. It is ideal for software engineers, system programmers, and developers interested in performance-critical and concurrent applications. Whether you are transitioning to a new role or enhancing your current one, this course provides the knowledge and skills to excel in Rust programming.

Prerequisites:

- Proficiency in C++ Programming: A strong understanding of C++ syntax, functions, and modules.
- Basic Knowledge of Programming Concepts: Familiarity with variables, expressions, control flow (loops and conditionals), and basic data structures.
- Experience with Version Control Systems: Basic experience using Git for version control, including committing, branching, and merging code.
- Understanding of Software Development Practices: Familiarity with project management tools and practices such as virtual environments, dependency management, and code testing.

Course Outline:

1. Getting Started with Rust

    - Rust’s Philosophy and Goals
    - History and motivation
    - Rust Community
    - The Rust Playground

2. Install Rust (Optional)

    - Script
    - macOS Homebrew
    - Platform Installers

3. Rust Editors

    - VSCode with Extensions
    - Rust Rover
    - Debug Rust in VSCode
    - GitHub Copilot

4. Hello World

- Create a new Project
- Main Function
- Print to the Console
- Comments

5. Cargo

- What is Cargo?
- Run Command
- Build Command
- Build Release Command
- Install Third-Party Crates

6. Scalar Types and Data

- Rust Types
- Constants
- Immutable Variables
- Mutable Variables

7. Code Logic

- If Statement
- Loop with Break
- While Loop

8. Functions

- Define a Function
- Call a Function
- Parameter Types
- Return Types
- Closure Functions

9. Modules

- Import Modules from Standard Library
- Import Modules from Third-Party Crates
- Define Custom Modules
- Import Custom Modules

10. Built-In Macros

- print! and println!
- format!
- assert!, assert_eq!, and assert_ne!
- vec!
- include_str! and include_bytes!

11. Memory Management

- Problems with Manual Management
- Problems with Garbage Collection
- Ownership & Borrowing
- References
- Lifetimes

12. Strings and String Slices

    - What is a String and a String Slice?
    - String Slices
    - String Objects
    - Convert Between Slices and Strings
    - Print Strings with Interpolation

13. Tuples

    - What is a Tuple?
    - Heterogeneous Elements
    - Access Elements
    - Destructuring
    - Immutable

14. Enums

    - What is an Enum?
    - Define an Enum
    - Using Enums
    - Enum Variants
    - Enum Methods

15. Structs

    - What is a Struct?
    - Create Instance
    - Field Init Shorthand
    - Struct Methods
    - Constructor Pattern

16. Vectors

    - What is a Vector?
    - Create a Vector
    - Add and Remove Elements
    - Access Elements
    - Iterate over Elements

17. Collections and Iterators

    - Vectors, arrays, and slices
    - HashMaps and hash sets

- Iteration and iterators

18. Traits

    - What is a trait?
    - Defining a trait
    - Implementing a trait
    - Default implementations
    - Traits as parameters

19. Generics

    - What is a generic?
    - Defining a generic
    - Implementing a generic
    - Generic bounds
    - Multiple generic types

20. Pattern Matching

    - What is Pattern Matching?
    - Match Statement
    - If Let Statement
    - While Let Statement
    - Pattern Matching with Enums

21. Concurrent Programming

    - What is Concurrent Programming?
    - Using Multiple Threads
    - Mutex, RwLock, and Arc
    - Message Passing with Channels
    - Futures and Async/Await

22. Unsafe Rust

    - What is Unsafe Rust?
    - Raw Pointers
    - Dereferencing Raw Pointers
    - Calling Unsafe Functions
    - Creating Safe Abstractions

23. Macros and Metaprogramming

    - What is a Macro?
    - Define a Macro with macro_rules!
    - Using Pattern Matching
    - Define Expansion
    - Use the Custom Macro

24. Tests

- What is a Test?
- Test Functions
- Test Organization
- Test Attributes
- Test Coverage

25. Memory-Safe Programming

- How Memory is Managed on a Computer
- How the Operating System Views Memory
- How Memory is Allocated in a Process
- Pitfalls with manual memory management
- Rust’s Approach to Safe Memory Management

26. Memory Management

- Variables and their Data
- Variable Addresses and Data Addresses
- Mutability of Variables and their Data
- Variable and Data Ownership
- Rust’s Approach to Variables and their Data

27. Rust Memory Model

- Ownership and Borrowing
- References and Mutability
- Stack Allocation vs Heap Allocation
- Smart Pointers
- Thread Safety through Atomics and Locks

28. Smart Pointers

- What are Smart Pointers?
- When to use Smart Pointers?
- Unknown Size at Compile Time
- Self-Referential Structures
- Interior Mutability

29. Smart Pointer Types

- Box
- Rc, Weak, and Arc
- Cell and RefCell
- RwLock and Mutex

30. C++ and Rust Interoperability

- Challenges and Concerns

- Calling Rust from C++
- Calling C++ from Rust

31. Valgrind and Rust

32. Documentation with Rustdoc

- What is Rustdoc?
- Add Documentation to Rust Code
- Triple-Slash Comments and the #[doc] Attribute
- Generate Documentation
- Linking and Cross-Referencing Documentation

33. Wrapping It Up

- Recap of Key Rust Concepts
- Next Steps for Further Learning and Practice