

**Memory Safe Programming with Rust**

**Course Number:** RUST-110  
**Duration:** 2 days

**Overview**

Accelebrate's Memory Safe Programming with Rust training course gives attendees a thorough and practical understanding of memory safety, a concept that often poses a challenge for new Rust developers. While programmers with experience in other languages quickly grasp many Rust features, understanding and applying ownership and borrowing rules can be a steeper climb.

This advanced Rust course demystifies the Rust memory model and teaches attendees how to integrate it into their coding practice seamlessly. By the end of this Rust course, students have a deep understanding of memory safety in Rust and will be able to apply it to their projects immediately.

**Prerequisites**

This course assumes prior experience with Rust programming language. The course focuses on upskilling Rust developers to understand and apply Rust’s memory safety concepts.

**Materials**

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub through documentation and extensive code samples.

**Software Needed on Each Student PC**

* A free, personal GitHub account to access the courseware
* Permission to install Rust and Visual Studio Code on their computers
* Permission to install Rust Crates and Visual Studio Extensions

If students cannot configure a local environment, a cloud-based environment can be provided.

**Objectives**

* Understand Rust’s philosophy of memory safety
* Work with memory leaks and dangling pointers
* Review concepts such as the stack and heap
* Understand ownership and borrowing
* Work with references and mutability
* Use smart pointers
* Build self referential structures

**Outline**

* Introduction
  + How Memory is Managed on a Computer
  + How the Operating System Views Memory
  + How Memory is Allocated in a Process
  + How Programming Languages and Runtimes Manage Memory
  + Pitfalls with manual memory management in languages like C or C++
  + Pitfalls with garbage collection in languages like Python or C#
  + Memory Leaks and Dangling Pointers
  + Ensuring Memory Safety at Compile Time
  + Rust’s Approach to Safe Memory Management
  + Other Safe Memory Management Approaches
* 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
* Rust Memory Model
  + Ownership and Borrowing
  + References and Mutability
  + Stack Allocation vs Heap Allocation
  + Smart Pointers
  + Thread Safety through Atomics and Locks
  + Unsafe Rust
* Smart Pointers
  + What are Smart Pointers?
  + When to use Smart Pointers?
  + Unknown Size at Compile Time
  + Self-Referential Structures
  + Interior Mutability
* Smart Pointer Types
  + Box
  + Rc, Weak, and Arc
  + Cell and RefCell
  + RwLock and Mutex
* Conclusion