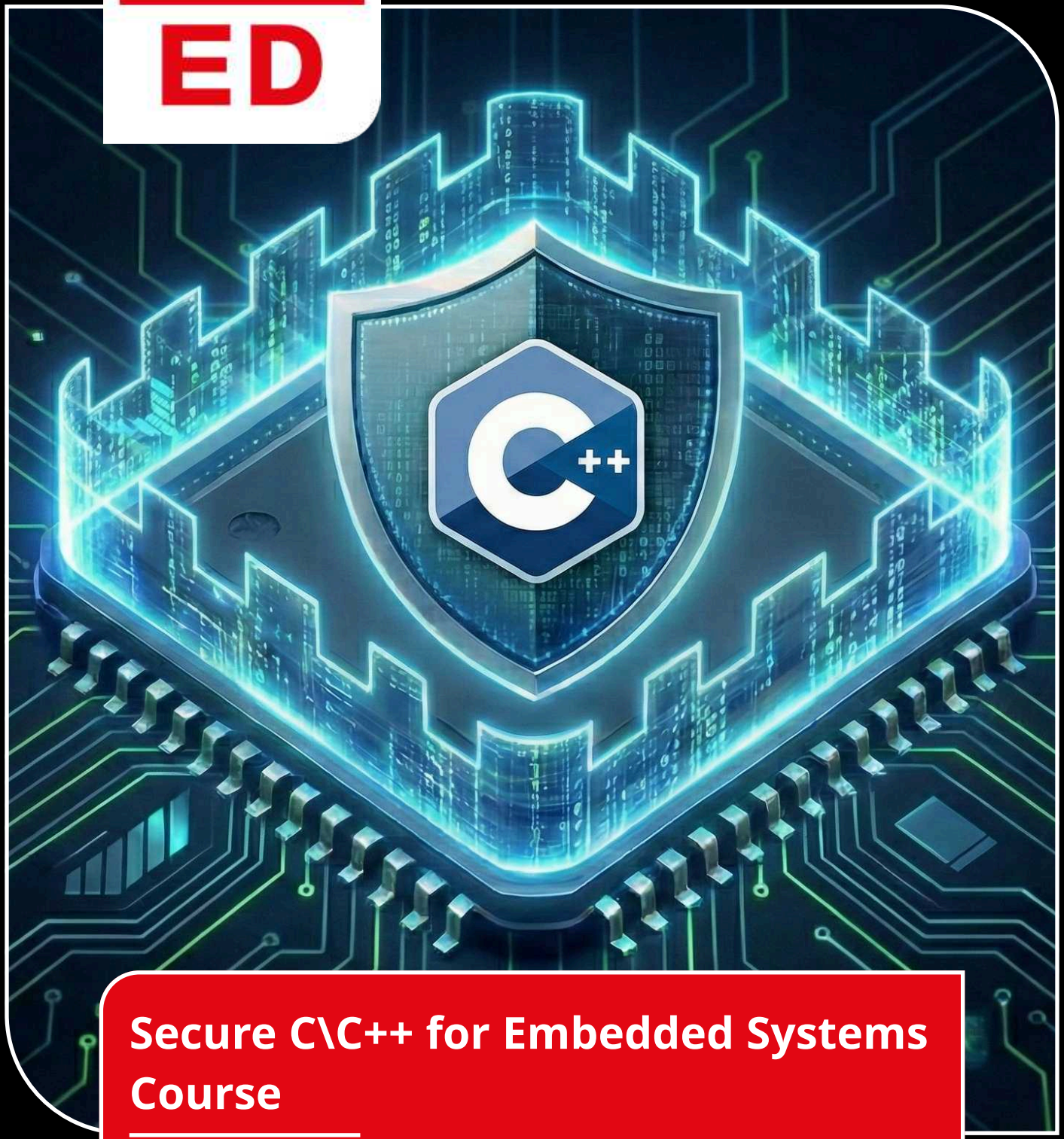
Real Time College

rt-ed.co.il info@rt-ed.co.il 077-7067057



Secure C\C++ for Embedded Systems Course

Duration: **40** Hour | **Hands-On Training:** 60%

TABLE OF CONTENTS

About RealTime Group 3

Course Overview 7

Who should attend, Prerequisite/Advantage 8

Course Curriculum 9

Miscellaneous 14

ABOUT REAL TIME GROUP

Experience the Industry. Master the Technology.

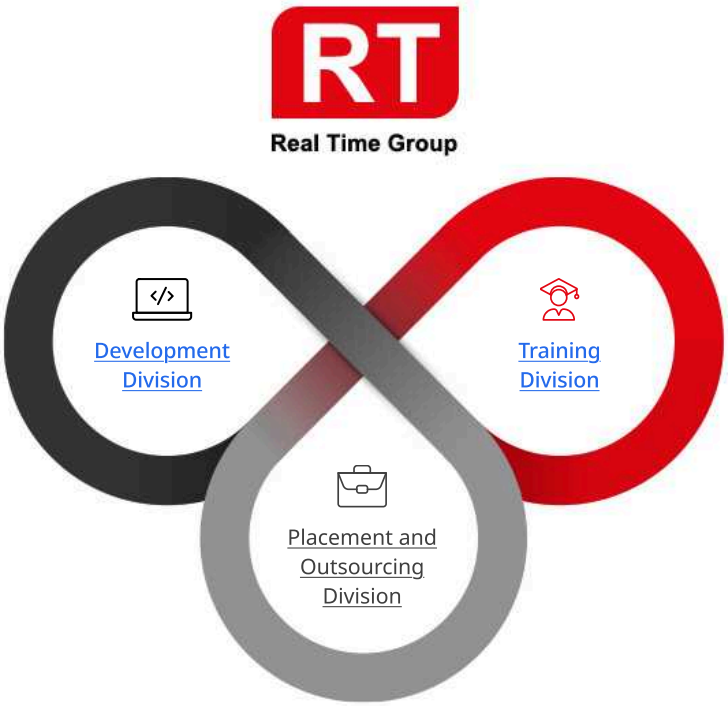
Real Time Group, established in 2007, is a technology company specializing in advanced software and hardware engineering solutions across RT Embedded Systems, Embedded Linux, FPGAs, Artificial Intelligence, Machine Learning, Computer Vision, DevSecOps, Cloud technologies, and Full Stack development.

The company combines deep hands-on development expertise with professional training programs designed to prepare engineers for the high-tech industry. Over the past two decades, Real Time teams have contributed to complex engineering projects for leading organizations including Apple, NVIDIA, Intel, the Israel Defense Forces (IDF), Rafael Advanced Defense Systems, Israel Aerospace Industries (IAI), Elbit Systems, and ELTA Systems.

This industry-driven experience forms the foundation of the Real Time training ecosystem.

OUR THREE-DIVISION ECOSYSTEM

WORKING TOGETHER TO LAUNCH YOUR CAREER:



RT-ED: TRAINING DIVISION

RT-ED provides advanced professional training programs focused on practical engineering skills. Courses are project-based and utilize industry-standard development environments, cloud platforms, and modern development tools.

Participants gain hands-on experience through structured labs, guided exercises, and industry-grade final projects.

DEVELOPMENT DIVISION

The Development Division delivers engineering and software development services across embedded systems, AI technologies, cybersecurity, and cloud infrastructures.

Through interaction with development teams and real-world workflows, students gain exposure to modern engineering practices and production-grade environments.

PLACEMENT AND OUTSOURCING DIVISION

Real Time Group provides structured placement and internship opportunities that support graduates in transitioning into the high-tech industry.

Students may participate in internships within active development teams, gaining real-world project experience under the guidance of experienced engineers and mentors.

COURSE OVERVIEW

This course focuses on secure programming techniques in C and C++ for embedded and real-time systems. It emphasizes writing safe, robust, deterministic, and maintainable code under hardware, timing, and memory constraints.

Students will learn how to prevent memory corruption, undefined behavior, race conditions, and security vulnerabilities, and how to apply industry standards and tooling used in safety-critical and mission-critical embedded products.

COURSE OBJECTIVES

By the end of this course, participants will be able to:

- Write secure, fault-tolerant C/C++ code for embedded systems
- Identify and prevent undefined behavior (UB) and memory-safety issues
- Design safe APIs and module interfaces
- Apply coding standards such as MISRA-C/C++, CERT C/C++
- Use static analysis, sanitizers, unit tests, and code reviews
- Handle errors, concurrency, and resource constraints safely
- Harden embedded software against faults and misuse

WHO SHOULD ATTEND\ PREREQUISITES

Who should attend:

- Embedded software engineers
- Real-time / firmware developers
- Safety-critical and defense-industry developers
- C/C++ developers working close to hardware

Prerequisite:

- Solid knowledge of C programming
- Basic understanding of embedded systems (bare-metal or RTOS)
- Familiarity with pointers, memory, and compilation
- C++ knowledge is an advantage but not mandatory

COURSE DURATION & FORMAT

- 40 academic hours (5 sessions × 8 Academic hours)
- 60% hands-on labs, 40% theory and case studies
- Development environment:
  - GCC
  - Make
  - GDB
  - Embedded target or host-based simulation with embedded constraints

COURSE CURRICULUM

Below is a breakdown of the core modules covered in this course, along with their descriptions and duration.

Select a module to view detailed content.

MODULE NAME	DESCRIPTION
Secure Programming Fundamentals	1. What is secure programming in embedded systems 2. Typical embedded failure sources: hardware, software, timing, power 3. Relationship between secure Coding, Reliability, Safety, and Maintainability 4. Relevant standards: MISRA-C, MISRA-C++, CERT C/C++
Memory Model, Undefined Behaviour, And Architecture	1. Common memory failures and prevention: a. Buffer overflows b. Stack corruption c. Use-after-free 2. Alignment, padding, and data layout 3. Avoiding dynamic allocation (malloc/free) in real-time systems and using Static allocation strategies 4. Guard patterns and stack canaries 5. Alignment and endianness pitfalls 6. Integer promotions and overflows Lab: Overflows, alignment and static allocation.

Secure C/C++ for Embedded systems Course

MODULE NAME	DESCRIPTION
Secure C Programming Techniques	1. Useful GGC Flags (-Wall -Wextra -pedantic -Wcoversion) a. Code Organization (multiple files) b. c & .h files c. a & .so files 2. Integer overflow and underflow (-fwrapv) 3. Signed vs unsigned errors 4. Dangerous functions and techniques in C 5. Dangerous casts and implicit conversions 6. Why should we avoid macros 7. possible alternatives: a. inline functions b. C++ templates / C++ constexper-functions
Safe Pointer Usage	1. Safe use of pointers and arrays 2. NULL checking and secure access 3. Pointer arithmetic risks and safe patterns 4. memcpy strcpy strncpy memcmp memset 5. Secure C-Strings 6. Dangers of Malloc & Free 7. Const-correctness as a secure tool 8. strict aliasing 9. converting pointer arithmetic to index based code 10. Proper use of volatile and restrict 11. Secure initialization patterns Lab: Refactoring unsafe C code into secure code

Secure C/C++ for Embedded systems Course

MODULE NAME	COURSE DESCRIPTION
Secure C++ For Embedded Systems	1. Dangerous functions and techniques in C++ 2. Dangers of plain-casting in C++ 3. Dangers of "using namespace" 4. why polymorphism might not be viable for real-time 5. why heap-based collections might not be viable for real-time 6. static initialization order fiasco 7. const correctness 8. std::min/max bugs 9. Safe use of templates a. understanding rvalues and temporaries bound to references 10. constexpr functions 11. smart pointers std::unique_ptr std::shared_ptr 12. unnamed lock_guard bug 13. Rule of 3/5/6 14. Rule of Zero - reliance on RAII 15. Proper Initialization of fields 16. compiler deleted constructors and functions 17. compiler defaulted constructors and functions 18. destructor noexcept 19. issues with runtime assert() function
Error Handling And Fault Management	1. Error codes vs status objects 2. Watchdogs and fault recovery 3. Logging strategies in constrained systems 4. Error codes versus assertions 5. Debug vs release assertions 6. Watchdog-aware error handling Lab: Designing a robust error-handling API

Secure C/C++ for Embedded systems Course

MODULE NAME	DESCRIPTION
Concurrency And Real-Time Safety	1. Race conditions and data corruption 2. Interrupt context vs task context 3. Atomic operations and critical sections 4. Lock-free vs blocking designs 5. Secure RTOS usage patterns Lab: Fixing concurrency bugs in ISR/task interaction
Interrupts, Timing, And Concurrency	1. Safe ISR design 2. Reentrancy and thread safety 3. Atomic operations and lock-free patterns
Testing And Validation	1. Unit testing for embedded code 2. Host-based testing strategies 3. Mocking hardware 4. Boundary and stress testing 5. Introduction to fuzz testing (host-side) Lab: Writing unit tests for a firmware module

Secure C/C++ for Embedded systems Course

MODULE NAME	DESCRIPTION

Secure Coding Standards And Analysis Tools	1. MISRA-C & MISRA C++
	2. Compiler warnings as errors
	3. Static Analysis Tools
	  a. Coverity
	  b. Cppcheck
	4. Dynamic Analysis Tools
	  a. AddressSanitizer (ASan)
	  b. ThreadSanitizer (TSan)
	  c. UndefinedBehaviorSanitizer (UBSan)
	  d. Valgrind (Linux)
	5. Enabling Stack-Protectors in GCC
	6. Usage of -fPIC (Position Independent Code)
	7. Usage of -fPIE (Position Independent Executable)
	8. Usage of ASLR (Address Space Layout Randomization)
	9. Hardening Compiler Flags & Definitions
	
	Lab: static & Dynamic analysis

MISCELLANEOUS

• Course opening is contingent upon minimum enrollment.

• Registration fees are non-refundable except in cases where the college does not open the course.

• All content and implications of the syllabus are subject to college regulations.

• The college informs students that there may be changes in course content and scheduling.

• The college commits to notifying students of any changes.

• The college reserves the right to modify course content at its sole discretion.

Thank You!

Real Time College

rt-ed.co.il

info@rt-ed.co.il

077-7067057