FEABHAS

Real-Time Modern C++ (C++11/14/17)

Course category	C++ Training Courses
Training area	Programming Languages
Course code	C++11-502
Duration	5 days
Additional information	Available for on-site delivery only. Can be delivered remotely or Face-to-Face.

C++ is a remarkably powerful systems-programming language, combining multiple programming paradigms – Procedural, Object Oriented and Generic – with a small, highly-efficient run-time environment. This makes it a strong candidate for building complex high-performance embedded systems.

The C++11 standard marked a fundamental change to the C++ language, introducing new idioms and more effective ways to build systems. This new style of programming is referred to as 'Modern C++'.

This practical, hands-on course introduces the C++ language for use on hosted (Linux), multi-threaded embedded systems.

The focus is on developing effective, maintainable and efficient C++ programs. The course covers C++11, C++14 and C++17.

Overview:
A five-day course that provides a practical overview of Modern C++, focusing on developing object-oriented programs in a hosted, real-time environment. The course covers the current release of the C++ standard, known as C++17.

Course Objectives:

- To provide a solid understanding of the essentials of the C++ programming language.
- To give you practical experience of writing Modern C++ for Linux-based embedded applications.
- To give you the confidence to apply these new concepts to your next project.

Delegates will learn:

- Modern C++ syntax and semantics and idioms
- The Application Binary Interface (ABI) and memory model of C++
- Idioms and patterns for building effective C++ programs
- Real-time and concurrency design issues

Pre-requisites:

- A strong working knowledge of C
- Embedded development skills are useful, but not essential

Who should attend:

This course is aimed at C programmers working in a Linux environment who are moving to C++ for their embedded development.

Duration:

Five days.

Course Materials:

- Delegate manual

Course Workshop:

Attendees perform hands-on programming during course practicals. Approximately 50% of the course is given over to practical work. Students will be working in a Linux environment.

Part 1 - Core language

The C++ type model

- Objects and types
- Scalar types
- Pointers
- References
- Null terminated byte strings
- Type aliases

Initialisation

- Object initialisation
- Brace initialisation syntax
- Automatic type deduction
- Type conversion and casting

Constants

- const qualifer
- constexpr objects
- constexpr functions and classes
- enum classes

Control Structures

- Flow control statements
- if/switch initialisation
- iterator model
- range-for statement

Arrays

- C arrays
- std::array
- Value initialisation
- Iterators
- Range-for statement

Function parameters

- Function declaration and definition
- Procedure Activation Record
- Pass-by-value, pointer and reference
- Function overloading
- Default parameters
- Function inlining

Scope and lifetime

- Object scope
- Identifier resolution
- Internal Linkage
- Static and automatic object lifetimes
- Dynamic objects

Structuring code

- The build process
- Compilation Dependencies
- Include guards
- #pragma once

- Linkage to non-C++ code

Namespaces

- Organising large code bases
- Namespace resolution issues
- Nested namespaces and namespace aliases

Part 2 – Object Oriented Design

Principles of Object Oriented Design

- Modularisation
- Coupling, Cohesion, Encapsulation and Abstraction
- Object-based design concepts
- Client-server architecture

Classes

- User defined data types
- Attributes and operations
- Member functions and the ‘this’ pointer

Constructing objects

- Member variable initialisation
- Constructors and the Member Initalisation List
- Re-enabling the default constructor
- Delegating constructors
- Destructors
- Copy constructor and assignment
- Explicit constructors

Objects and functions

- In, Out and In-Out parameters
- const member functions
- Disabling copying and assignment

Object as return values

- Returning by value

- Named Return Value Optimisation (NRVO)
- Return Value Optimisation (RVO)
- Copy elision
- Factory functions

Compound types

- Structured bindings
- std::pair and std::tuple
- std::optional and std::variant

Class and static

- Static member variables
- Inline initialisation of static member variables
- Static member functions

Building connected objects

- One-to-one association
- One-to-many associations
- Bi-directional associations
- Friend functions
- Binding and lifetime management
- Forward references

Composite Objects

- Composition
- Aggregation
- Composite object initialisation
- Optional composite objects

Building substitutable objects

- Inheritance, base and derived classes
- Overriding methods
- Base class initialisation
- private and protected interfaces
- The Liskov Substitution Principle
- Static and dynamic binding
- Virtual destructors

Abstract Base Classes

- The Single Responsibility Principle
- Pure virtual functions
- Extending class interfaces
- dynamic_cast

Realising Interfaces

- The Dependency Inversion Principle
- Interfaces as design constructs
- Pure virtual classes
- Cross-casting

Part 3 – The Standard Library

Dynamic objects

- Dynamic object allocation
- std::unique_ptr
- std::shared_ptr
- std::weak_ptr

Sequence containers

- std::vector
- std::bitset
- std::list and std::forward_list
- Container classes and std::initializer_list

Associative containers

- std::set
- std::pair and std::map
- std::unordered map

Algorithms

- The Standard Library model
- std::fill and std::sort
- std::find, std::count and std::accumulate
- The Remove-Erase idiom
- std::transform and std::bind

- std::bind placeholders

Lambdas

- Lambda functor syntax
- Lambdas as a block-scoped function
- Capture context
- Capture initialisers