FEABHAS

Transitioning to Modern C++ (C++11/14/17)

Course category	C++ Training Courses
Training area	Programming Languages
Course code	AC++11-401
Duration	4 days

This four-day course introduces the new features of Modern C++ and how they relate to the previous incarnation, C++98.

The C++11 standard marks a fundamental change to the C++ language. Bjarne Stroustrup, originator of C++, refers to it as “feeling like a completely new language”. The course looks at some of the changes to the language and how they affect the way we write C++ code.

The course covers C++11, C++14 and C++17 and where relevant refers to C++20.

Course objectives:

- Provide a background into the C++ features that have changed
- Provide an overview of the new language features
- Understand how the new features change C++ programming style
- Give practical experience of the new features
- Give the confidence to apply these new concepts to your next project

Delegates will learn:

- The new extensions to the C++ language
- Some of the performance impacts of the new features
- The extensions to the Standard Template Library
- Some of the new Standard Libraries
- An introduction to the new C++ threading model

Pre-requisites:

This course is not intended to be a comprehensive C++ course and it is expected that students will already have a solid working knowledge of C++98, in particular.

- Object Oriented design

- RAII
- The Standard Template Library

Who should attend?
This course is aimed at experienced C++ developers who want to quickly understand the new facilities of C++11.

Duration:

- Four days

Course materials:

- Delegate manual

Course workshop:

At least 50% of the course is hands-on exercises. Students will be programming on a platform environment, either Windows or Linux, using an appropriate toolchain.

Simple types

- Automatic type deduction
- Constant-expressions
- Using aliases
- nullptr

Constructing objects

- Class definition and objects
- Cascading constructors
- Default constructors
- Brace initialisation syntax
- Initializer lists

Sequence containers

- std::array and std::vector
- Allocators
- Iterators
- Range-for

Associative containers

- std::tuple
- std::unordered_map

Specialisation

- Inheritance and substitution
- Overriding
- Dynamic polymorphism
- Pure virtual functions
- Interfaces
- Cross-casting

Resource Management

- Managing object lifetimes
- The Rule of Three
- The Copy-Swap idiom

Move Semantics

- rvalue references
- Resource pilfering
- Move constructors
- The Rule of Four (and a half)

Smart pointers

- unique_ptr
- shared_ptr
- weak_ptr

Template functions

- Generic functions
- Type deduction rules
- The template build mechanism

Template classes

- Generic classes

- Templates and polymorphism
- Policies

Perfect forwarding

- Meyers' Universal references
- Variadic templates

STL Algorithms

- The algorithm concept
- Adapters
- Binding

Function objects

- Lambdas
- Generic lambdas
- std::function

Threading

- Creating threads
- Joining and detaching threads
- Accessing the underlying OS

Atomic types

- std::atomics
- The C++ memory consistency model

Mutual Exclusion

- std::mutex
- scope-locked idiom
- Condition variables

Asynchronous tasks

- Deferred synchronous calls
- Promises and futures

- Packaged tasks
- std::async()

User-defined literals

- Rom-able classes
- operator ” ”