

Migrating to C++20

Course category	C++ Training Courses
Training area	Programming Languages
Course code	C++20-302
Duration	3 days
Additional information	Available for on-site delivery only

Overview:

This advanced 3-day course is designed to transition experienced C++ programmers to the latest features of C++17 and C++20. The focus is to teach good programming practice using Modern C++ and to put the latest features of the language into context.

Course Objectives:

- To provide an appreciation of the new features of C++17 and C++20
- To give you practical experience of working C++20 syntax and libraries
- To give you the confidence to apply these new concepts to your next project

Delegates will learn:

- New C++17/20 syntax, semantics and library features
- Class concepts, requirements and polymorphic allocators
- Ranges, views and coroutines for working with sequences
- Modules and interfaces for structuring source code

Pre-requisites:

- A good working knowledge of C++11/14 and the standard library
- An understanding of machine architectures is helpful.

Who should attend:

This course is aimed at C++ programmers who are using earlier standards of C++, and experienced C++ programmers who want to extend and expand their C++ skills.

Duration:

Three days.

Course Materials:

- Delegate manual

Course Workshop:

Attendees perform hands-on exercises during course practicals. Approximately 40% of the course is given over to practical work. The tools used are indicative of current modern working practices in the embedded arena.

Day 1**Introduction****Language Changes**

- constexpr virtual functions
- constexpr
- if constexpr ()
- flow control initialiser clauses
- using statement with enum and enumerated value t
- designated initializers for struct
- compiler diagnostics

Data type updates

- 2's complement integer type
- non arithmetic std::byte type
- restricted use of volatile objects
- string literals, string types, Unicode support
- using std::to_string
- the std::string_view class
- user defined literals
- string formatting and std::format
- std::map improvements
- invoking constructors with std::construct_at

Compound types

- C++17 structured bindings
- std::pair

- `std::optional`
- `std::tuple`
- `std::variant`

Template updates

- class template argument deduction
- abbreviated function templates
- template deduction guides
- template lambdas

Comparing objects

- comparing objects of the same/different types
- equality semantics
- equality testing with `operator==`
- default `operator==`
- ordering semantics: strongly ordered, and weakly ordered
- comparison (starship) operator `< = >` and default operator `< = >`

Day 2

Requirements

- implicit class requirements for templates
- defining template requirements with `requires`
- type traits
- function requirement modifiers
- ad hoc constraints
- non template constraints

Concepts

- concepts and requirements
- using concepts in templates
- standard concepts
- concepts and constraints
- `requires` expressions
- constrained auto types
- concepts and perfect forwarding

Ranges

- ranges concepts
- range-for structured bindings
- defining ranges with `std::span`
- algorithms and ranges
- range concept types
- projections
- writing a classic iterator
- using an end sentinel iterator

Views

- views concepts
- view pipelines
- writing views
- view iterator
- view adapter

Day 3

Polymorphic allocators

- problems with container allocators
- polymorphic allocator model
- polymorphic memory resources (PMR)
- writing a polymorphic allocator
- standard memory resources
- using `std::monotonic_buffer_resource`
- understanding `std::unsynchronized_pool_resource`

Coroutines

- coroutine concepts
- `co_yield` and `co_return` statements
- the promise and future model
- writing a coroutine
- coroutine support for range-for and algorithms

Modules

- module concepts
- mainstream compiler support for modules
- module, import and export statements
- Global Module Fragment
- single file modules

- module linkage
- multiple compilation units
- modules and namespaces
- modules and header files

Concurrency

- RAII/RDID threads using `std::jthread`
- atomic wait and notify
- binary and counting semaphores
- multi-thread synchronisation with `std::latch`
- multi-thread synchronisation with `std::barrier`

Feabhas Ltd - PO Box 4259, Marlborough, SN8 9FJ, UK · info@feabhas.com · www.feabhas.com