

# C++ 17

## Classes, Inheritance, Visibility, Templates, RTTI, Exceptions, Namespaces, Class Hierarchy Design

This course provides developers with an intensive introduction to programming in C++, together with an overview of its powerful standard library. C++ is a general purpose programming language, highly suited for objected oriented and generic application, component and systems development. It is used for advanced, complex, full functionality projects. It is fast – and often selected where high performance coupled with a rich object-oriented language is needed.

Above and beyond the new syntax aspects of C++, this course also examines how to profit from a range of new

object programming techniques that are evolving, such as genericity and refactoring. Also explored is the optimum architecture for modern C++ class hierarchies, the latest C++ standards (C++11, C++14, C++17) and interaction with C code.

A core competency of all software engineers is to be having completely mastery of the programming language they use. If you are currently a programmer moving to a new C++ project, then you will greatly benefit from attending this course.

<b>Contents of One-Day Training Course</b>	
<p><b>Target Audience</b> This training course is aimed at people who need to quickly get up to speed developing in C++.</p> <p><b>Prerequisites</b> Attendees should be experienced software developers with a good knowledge of any high-level programming language.</p>	<p><b>Tour of C++ Features</b> Primitives Flow control Enumerations Preprocessor Compilation</p> <p><b>Classes</b> Class members and this ptr Visibility – public, protected, private Constructors and destructors static</p> <p><b>Inheritance</b> Superclass and sub-classes Constructors and the inheritance hierarchy Impact of the virtual keyword</p> <p><b>Multiple Inheritance</b> Benefits and problems Which function is executed The diamond and vtbl issues</p> <p><b>Templates</b> Function templates Class templates Function specialization How to write code once and then use it with a variety of data types Debugging template code Designing with genericity in mind</p> <p><b>Lambda Expressions</b> What problem are these trying to solve Unnamed functions Expression syntax Writing code that accepts lambda expressions</p>
	<p><b>Exceptions</b> Benefits, costs and recommendations associated with C++ exception handling Families of exceptions Throw-try-catch syntax Declaring exceptions in header files</p> <p><b>Namespaces</b> Avoiding naming conflicts The std and other namespaces Creating &amp; using your own namespace</p> <p><b>RTTI</b> Run Time Type Information Exploring objects + their types at runtime</p> <p><b>Refactoring</b> Software grows and changes Sub-optimal solutions evolve Refactoring is a set of coding best practices which aim to fix &amp; improve</p> <p><b>C Code Interaction</b> How C++ code can call C code How C code can call C++ code extern C</p> <p><b>Design of C++ Class Libraries</b> Classes are seldom needed on their own How groups of classes can cooperate Designing hierarchies Optimally placing data and functionality</p> <p><b>Advanced Language Features</b> Temporaries/ Header file changes New keywords: mutable, typename, etc. Smart pointers: rationale &amp; how to use Potential problems to avoid C++20: What's coming soon</p>