

Modern C++ Programming

Audience Modern C++ Programming Course

The course Modern C++ Programming is intended for developers who are familiar with [C++](#) but who wish to delve into the newest and most advanced techniques and features of the language.

Prerequisites Course Modern C++ Programming

To join this course knowledge of the basic concepts of [C++](#) and programming experience in C++ is required.

Realization Training Modern C++ Programming

The concepts are treated on the basis of presentations. Illustrative demo's are used to clarify the concepts. Theory and practice are interspersed and there is ample opportunity to practice.

Certification Modern C++

Participants receive an official certificate Modern C++ Programming after successful completion of the course.

Duration: 4 days

Price: € 2450

Open Schedule

C++98	C++11	C++14	C++17
<ul style="list-style-type: none"> Templates RTTI, including dynamic_cast Atomic types Thread-local storage Strongly typed enumerations 	<ul style="list-style-type: none"> Move semantics Atomic types Thread-local storage Strongly typed enumerations Relaxed memory ordering Relaxed memory ordering 	<ul style="list-style-type: none"> Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering 	<ul style="list-style-type: none"> Full support for C++11 Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering

Diagram illustrating the evolution of C++ memory models and synchronization primitives. It shows a progression from sequential to relaxed, shared, and future locks, with C++11, C++14, and C++17 highlighted as key milestones.

Modern C++

Diagram illustrating the evolution of C++ memory models and synchronization primitives. It shows a progression from sequential to relaxed, shared, and future locks, with C++11, C++14, and C++17 highlighted as key milestones.

C++98	C++11	C++14	C++17
<ul style="list-style-type: none"> Templates RTTI, including dynamic_cast Atomic types Thread-local storage Strongly typed enumerations 	<ul style="list-style-type: none"> Move semantics Atomic types Thread-local storage Strongly typed enumerations Relaxed memory ordering Relaxed memory ordering 	<ul style="list-style-type: none"> Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering 	<ul style="list-style-type: none"> Full support for C++11 Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering Relaxed memory ordering

Diagram illustrating the evolution of C++ memory models and synchronization primitives. It shows a progression from sequential to relaxed, shared, and future locks, with C++11, C++14, and C++17 highlighted as key milestones.

Content Course Modern C++ Programming

In the course Modern C++ the new and advanced aspects of the C++ language based on the standards C++11, C++14, C++17 and C++20 are extensively discussed.

C++11 Features

The course starts with an overview of the features that have been introduced in C++11 such as type inference, initializer lists, range based for loop, lambda functions and strongly typed enums.

Right References en Move Constructors

Next right references are discussed and the performance gains that can be booked by using move constructors in addition to copy constructors.

Inheritance Enhancements

The new possibilities with respect to inheritance with the keywords default, delete, override and final are also on the course program. The implementation of virtual functions and the need for virtual destructors are discussed as well.

Smart Pointers

Attention is paid to smart pointers like unique pointers, shared pointers and weak pointers.

Operator Overloading en Templates

Operator overloading and templates are also on the program and variadic templates and perfect forwarding are treated.

RAII Pattern

The modern C ++ Resource Acquisition is Initialization or RAII idiom is explained while discussing exception handling.

Multithreading

Attention is also paid to threads as well as the synchronization between threads which are now part of the C++ standard. This also includes asynchronous calls with promises and futures.

C++11, C++17 en C++20 Features

Next specific C++11, C++17 and C++20 features are discussed such as optional types, structured binding declarations and constructions from the world of functional programming such as fold expressions.

Advanced STL

The course concludes with an overview of advanced features of the Standard Template Library [STL](#).

Modules Course Modern C++ Programming

Module 1 : Modern C++ Features	Module 2 : Move Semantics	Module 3 : Inheritance
C++11 Features Type Inference Auto Keyword Deduction with decltype Uniform Initialization _INITIALIZER Lists Range Based for Loop Null Pointer Constant constexpr Keyword Static Asserts Lambda Functions Strongly Types Enums User Defined Literals Raw String Literals	Reference Initialization References and Pointers Rvalues and Rvalues in C++ Passing and Returning References Rvalue References Comparing Reference Types Rvalue Reference Usage Assignment Operator Copy Constructor Passing and Returning Objects Passing References to Objects Move Constructor Move Assignment Operator Golden Rule of 5	default and delete Keyword Delegating Constructors Inheritance Calling Base Class Constructors Multiple Inheritance Virtual Derivation Polymorphism Virtual Functions Abstract Classes Interfaces in C++ Destructors and Inheritance Virtual Destructors override Specifier final Specifier
Module 4 : Smart Pointers	Module 5 : Operator Overloading	Module 6 : Templates
unique_ptr Using unique_ptr Specialization for Arrays Replacement for std::auto_ptr std::make_unique shared_ptr Pointer Control Block shared_ptr Destruction Policy shared_ptr Interface Cyclic References weak_ptr	Syntax Operator Overloading Overloading Numeric Types Overloading Overview Overloading Restrictions When not to Overload Operators as Class Members Operators as Friend Functions Overloading Stream Operators Overloading ostream and istream Overloading Unary Operators Overloading Binary Operators	Template Functions Template Specialization Template Parameter List Inclusion Compilation Model Class Templates Template Member Functions Template Parameter Scope Templates and Statics Templates and Friends Alias Templates Perfect Forwarding
Module 7 : Exception Handling	Module 8 : Multiple Threads	Module 9 : Synchronization
Error Conditions and Exceptions Class Objects as Exceptions Parameter Catch Block Catching in Hierarchy Golden Rule Rethrowing Exceptions noexcept Specifier Preventing Resource Leaks RAII Idiom C++ Standard Exceptions User Defined Exceptions Exception Handling Costs	Multiple Threads Benefits and Drawbacks Thread Class Joining Threads Detaching Threads Thread ID Callables Passing Parameters Pass by Reference Pass by std::ref and std::move Member Function as Thread Thread Local Storage	Data Corruption Lock Guard Automatic Lock Management Mutex and RAII Recursive Locking Atomic Types Call Once Event Handling Condition Variables Wait and Notify Promises and Futures Asynchronous Tasks
Module 10 : C++14-17-20 Features	Module 11 : Standard Template Library	
Init-statement for if Selection Initialization Structured Binding Declarations const if Expressions Guaranteed Copy Elision Inline Variables Fold Expressions Optional Type Small String Allocations String View Generic lambdas Aggregate initialization	STL Core Components Containers, Algorithms and Iterators Vectors, Lists and Dequeues Adapters and Associative Containers Maps and Hash Maps Bitsets STL Iterators Reverse and Iostream iterators Function objects STL Algorithms Predicates and Comparators STL Allocators	