Advanced Modern C++ Programming

Course 4375 – 40 Hours

Overview

C++ is the standard language for implementing object-oriented designs where performance is a priority. Although long-term language stability is an important feature of C++, it has nevertheless continued to be developed. C++ 11 introduced a number of significant language and library features to improve safety and performance. These features allow us to more precisely express concepts from a design directly in code .
Although these changes could be considered incremental, in fact they allow and encourage a whole new approach to programming.

This course sets out from the beginning to embrace this new approach making full use of the facilities for writing well encapsulated robust code which is at the same time supremely efficient.

The course is written from a developers rather than an academics perspective, following the design of a simple library and introducing language and library features as they are encountered. In the process virtually all language features are explored and most of the standard library. In addition a variety of design patterns are examined and good practices are emphasised.

Although the course makes use of C++ 11/14 features throughout, most of the material is useful and relevant to pre C++ 11 users.

Delegates will gain a greater understanding of the capabilities and potential pitfalls of the C++ language and will be more able to use C++ language features to write robust, quality software.

This a comprehensive five-day course with a combination of lectures and practical sessions for each chapter to reinforce the topics covered throughout the course. The practicals avoid algorithmic difficulties so that delegates can concentrate on specific C++ features.

On Completion, Delegates will be able to

- Describe the modern approach to programming that C++ 11 facilitates.
- Describe and use advanced techniques for safe copying and forwarding
- Describe and use template techniques for efficient generic coding
- Understand portability issues and how to resolve them.
- Make appropriate use of operator overloading.
- Understand how to write robust constructors.
- Use C++ exceptions appropriately in libraries and in large programs.
- Understand and resolve issues with global variables.

- Understand the appropriate use of constexpr, volatile, unions and bitfields.
- Understand and use the library facilities for multi-threading
- Understand the issues of resource ownership and the use of RAII and smart pointers to make association explicit and safe.
- Describe advanced inheritance techniques, such as private inheritance, multiple inheritance, the template method pattern and down-casting.
- Take full advantage of the standard C++ library
- Make good use of the standard container classes and iterators
- Understand and use variety of OO call-back mechanisms

Prerequisites

This is not a course for beginners. It covers a great deal of material and those who do not meet the course prerequisites will find the course moves too quickly.

- You must have solid recent experience of writing C++.
- You must have a good appreciation of object-oriented principles.
- You must be comfortable with new and delete operators, class definitions and member functions, constructors and destructors, pointers, references, virtual functions, function overloading, specialization, inheritance and polymorphism.
- Ideally, you will have attended one of our C++ programming courses and have been using C++ solidly for at least six months.

Course Contents

Starting Well – (or replace with C++ refresher)

- Version Control
- Documenting Code
- Static Analysis
- Unit Testing
- 64-bit issues
- C/C library
- Precompiled Headers

Robust Design

- Single Responsibility Principle
- Reducing Complexity
- Encapsulation
- Keeping header files clean
- Conditional Compilation
- Coding Style
- C++ 11 game-changers

Const, Copying and Conversions

- Const consistency
- Logical vs physical const-ness and the mutable keyword
- The staticcast, dynamiccast, constcast and reinterpretcast keyword casts
- Converting constructors and the explicit keyword
- User defined conversion operators
- Copy construction and assignment
- Efficiency - Copy Elision and Return Value Optimisation

Move and Forward

- R-Value Reference
- Move rather than Copy
- Compiler Synthesised Member Functions
- Rule of Three / Five / Zero
- std::swap, std::move and std::forward
- Perfect forwarding

Strong Primitive Types

- Mission statement for a new design
- Standard libraries for Time and Date
- Creating strongly typed primitive types
- Class Definition Organisation

Operator Overloading

- Deriving operators from a sub-set
- Global functions for binary operators
- Templated operators
- Template Parameter Deduction
- Template Functions with non-argument type parameters
- Template Function Overloading and specialisation
- Template Instantiation and linkage
- Namespaces

Portable Integers and Robust Constructors

- Language specification for integers
- Discovering int size
- Specifying int size
- Template Classes
- Template Class Specialisation and Aliasing
- Single Point Of Maintenance for Constructors
- Argument Range Checking
- Custom Literals

Exception Handling

- Classifying and handling exceptions
- Catching exceptions
- Throwing exceptions
- The standard exception hierarchy
- Uncaught exceptions
- Resource acquisition and release
- Resource Acquisition is Initialisation idiom
- Exceptions and constructors
- Copy Before Release idiom
- Exceptions and destructors
- Commit or Rollback idiom
- STL exception guarantees

Bitfields and Unions

- Bitfields
- Anonymous Union
- Endedness and Bitfield Portability
- Creating a Portable Bitfield
- Template Parameter Checking
- Type Traits
- Static_Assert

Delegation Techniques

- Delegation principles
- Composition
- Adapter patterns
- Inheritance
- Multiple Inheritance
- Name Hiding
- Virtual inheritance
- Interface Classes
- Nested Classes

Statics and Globals

- Storage Class
- static class members
- Stateless Classes
- Static Local Variables
- Global and local access
- Problems with Global Variables
- Safe Global Variables
- The Singleton pattern
- Alternatives to Singleton

Volatile Variables

- Memory Mapped IO
- Volatile
- Placement New
- New and Delete operators
- Placing objects in memory
- Allocating without exceptions

Multithreading Techniques

- Multithreading concepts
- Creating multiple threads in C++
- Creating and managing locks
- Exception-Safe Lock Management
- Simultaneous Reads with a shared mutex
- Thread-Safe Assignment
- Atomic variables
- Condition variables
- Asynchronous functions and futures
- Exceptions in threads

Polymorphism

- Abstract Base Classes
- Benefits and Cost of Polymorphism
- Template Method Pattern
- Pure Virtual Functions
- Smart References and their uses
- Templated Outward Conversions
- The Curiously Recurring Template Pattern
- Downcasting

Safe Association

- Association for Independent Lifetimes
- Structured Lifetimes
- Unique_ptr
- Wrapping new
- 'Pointer' Function Parameters
- Shared_ptr
- Weak_ptr
- Safe Copying with Association

Design Patterns

- Composite
- Cheshire Cat
- Bridge
- Null Object
- Proxy
- Lazy Initialisation
- Dependency Inversion
- Factory
- Dependency Injection

Functional Abstraction

- Events and callbacks
- The Command pattern
- Functor Commands
- Lambda, an alternatives to functors
- Wrapping Traditional callbacks
- Member function pointers
- Function pointer adapters

Containers

- STL Container Classes
- Container Selection
- Container Functions
- Special List and Map/Set functions
- Iterators
- Algorithms
- Modifying Sort