C++ Programming

Duration:	5 days
Type:	beginner

Description

This course provides a comprehensive introduction to programming in C++. A strong emphasis is placed on best practises for modern C++ development, including good OO design, using containers rather than arrays or char pointers and taking advantage of emerging standard libraries such as Boost.

Prerequisites

Delegates must have existing programming experience, typically in C, Perl, Java or C#. Prior exposure to OO concepts and UML class and sequence diagrams is helpful but not essential.

List of Modules

Introduction to C++

- The history and evolution of C++
- Different generations of C++ developers
- Creating a basic program using streams
- Declaring and working with variables
- The pre-processor, compiler and linker
- Understanding pre-processor directives
- Placing declarations in header files
- Placing definitions in source files
- Compilation and linkage errors
- Using guards in header files

Namespaces

- How namespaces affect the scope of symbols
- Compiling against the std namespace
- Different forms of the using declaration
- Creating your own namespaces
- Working with nested namespaces
- Unnamed namespaces and static functions
- Defining an alias for a namespace

Types, Pointer and References

- Declaring and using arrays and structs
- Declaring and dereferencing pointers
- Working with pointers to arrays
- Working with pointers to structures
- Practical uses of pointers to pointers
- The const keyword and pointers
- Callbacks and function pointers
- Declaring and using references
- Mixing pointers and references

Storage Management

- Dynamically allocating memory
- The *delete* and *delete []* operators
- Handling out of memory conditions
- The placement new operator
- Casting with *static_cast*, *dynamic_cast*, *const_cast* and *reinterpret_cast*
- Using smart pointer classes

Functions

- Function prototypes and definitions
- Overloading a function name
- Supplying default parameter values
- Resolving overloaded functions
- Explaining inline functions
- The const type-qualifier

Object Oriented Development Part 1

- Why use object oriented programming?
- Creating a simple class declaration
- Passing objects by value and pointer
- Using forward references to classes
- Adding fields and methods to classes
- Working with static fields and methods
- Adding constructors to classes
- Copy and conversion constructors
- Providing an assignment operator
- Writing a destructor method

Object Oriented Development Part 2

- Declaring constructors as explicit
- Using initializer lists in constructors
- Declaring methods as being constant
- Using inheritance to create class hierarchies
- Calling base constructors in derived objects
- Protected fields verses protected helper methods
- Implementing polymorphism with virtual methods
- Creating abstract and pure virtual base classes

Object Oriented Development Part 3

- Slicing derived objects during pass by value
- Problems associated with hiding base methods
- Overloading operators using methods
- Overloading operators using free functions
- Common issues with operator overloading
- Reasons for overloading new and delete

Exception Handling

- The syntax for exception handling
- Understanding stack unwinding
- Safely recovering from exceptions
- Built in exception classes
- Adding exception specifications
- The syntax for function try blocks
- When *terminate()* is called
- When *unexpected()* is called

Templates

- Creating Function Templates
- Overloading Function Templates
- Creating Class Templates
- Functors as template parameters
- Function templates as template parameters
- How Templates are instantiated
- Linkage issues with Templates
- Fully specializing Templates
- Partially specializing Templates
- Using traits and policy classes
- Code generation using Templates

The Standard Template Library

- The evolution of the STL
- Why *std::string* is a Template
- Traversing containers with iterators
- Using the sequential containers
- Using the associative containers
- Algorithms for searching containers
- Algorithms for changing containers

The Boost Extensions to the STL

- Creating and running Regular Expressions via *Boost.Regex*
- Manipulating text with *Boost.String_algo* and *Boost.Tokenizer*
- Building lambda expressions using *Boost.Lamdba*
- Platform independent concurrency using *Boost.Thread*
- Test Driven Development using *Boost.Test*
- Review of the smart pointers in *Boost.Smart_ptr*
- Review of the Boost libraries for meta-programming