SHI | TRAINING & ADOPTION

Course Outline

Intermediate C++ Programming | Effective C++
Course TTCP2150: 4 days Instructor Led

About this course

Intermediate C++ Programming | Effective C++ is a lab-intensive, hands-on C++ training course geared for experienced C++ programmers who wish to take their development skills to the next level. Students will leave this course armed with the required skills to put advanced C++ programming skills right to work in a practical environment, using sound coding techniques and best practices.

- This comprehensive course consists of three modules. A preliminary module reviews topics, including inheritance, the ANSI C++ Standard Library, templates. I/O streams, and practical issues of C++ programming, such as reliability & testing. This material is covered as needed depending on the background of the students.

The second module covers more advanced topics. Advanced issues of inheritance and polymorphism are covered. Principles of effective class design, including, use of composition, templates and interface inheritance. The course covers exception handling and run-time type information (RTTI). Multiple inheritance is covered, including the complications that are introduced by this powerful feature. Advanced applications of C++ concepts are studied, including smart pointers and reference counting.

The third module introduces the Standard C++ Library. The main components of data structures, algorithms and iterators are covered. Illustrations are provided of a number of important containers, such as vectors, stacks, queues, lists and sets. Extensive programming examples and exercises are provided. A number of progressively developed case studies are used to illustrate object-oriented programming techniques and to give the student practical experience in putting together features of C++ learned in the course.

Throughout the course, new features of modern C++ (version 11 and beyond) are introduced as well as the impact on programming style. These new features include smart pointers, move constructors, functional programming and lambda expressions.

Audience profile

This is an intermediate level development course designed for developers with prior C++ programming experience. Students without prior C++ programming background should take the pre-requisite training.

Please see the Related Courses tab for specific Pre-Requisite courses, Related Courses that offer similar skills or topics, and next-step Learning Path recommendations.

Take Before: Incoming students should have practical skills equivalent to the topics in, or should have recently attended, one of these courses as a prerequisite:

- TTC2100: Introduction to C++ Programming
- Please contact us for recommended next steps tailored to your longer-term education, project, role or development objectives.

At course completion

After completing this course, students will be able to:

- Modern C++ 11, 14, 17
- Templates
- Memory Management
- Inheritance and Polymorphism
- Exception Handling
- Input/Output in C+
- Unit Testing in C++

SHI | TRAINING & ADOPTION

Course Outline

- Advanced Polymorphism and Inheritance
- Functional Programming
- Runtime Type Information
- Overview of Standard Library
- STL Containers
- STL Iterators
- Threads & Tasks

Course Outline

1. **Quick Review of C++**

- Implementing a basic O-O design
- Implementing Classes
- Visibility & friends
- File organization
- C++ types – structs, classes, interfaces, enums

2. **Modern C++**

- New keywords in C++ 11,14,17
- RAII - Modern memory management in C++ - overview
- Copy vs Move semantics
- Namespaces

3. **Templates**

- General Purpose Functions
- Function Templates
- Template Parameters
- Template Parameter Conversion
- Function Template Problem
- Generic Programming
- General Purpose Classes
- Class Templates
- Class Template Instantiation
- Non-Type Parameter
- C++ Containers overview
- Variadic Templates

4. **Memory Management**

- The handle/body (Bridge) pattern
- Using strings effectively
- Smart Pointers
- Move constructor in depth

Course Outline

- Other <memory> features

5. **Inheritance and Polymorphism**

- Inheritance Concept
- Inheritance in C++
- Protected Members
- Base Class Initializer List
- Composition
- Member Initialization List
- Order of Initialization
- Inheritance vs. Composition
- A Case for Polymorphism
- Dynamic Binding
- Pointer Conversion in Inheritance
- Polymorphism Using Dynamic Binding
- Virtual Function Specification
- Invoking Virtual Functions
- VTable
- Virtual Destructors
- Abstract Class Using Pure Virtual Function
- Interfaces

6. **Exception Handling**

- Exception Handling
- try and catch
- Exception Flow of Control
- Context and Stack Unwinding
- Handling Exceptions in best Context
- Benefits of Exception Handling
- Unhandled Exceptions
- Clean Up
- Multiple Catch Handlers

7. **Input/Output in C+**

- Input/Output in C++
- Built-in Stream Objects
- Output Operator <<
- Input Operator >>
- Character Input
- String Input
- Formatted I/O
- Streams Hierarchy (Simplified)
- File I/O
- File Opening
- Integer File Copy

Course Outline

- Character File Copy
- Overloading Stream Operators
- Implementing Overloaded Stream Operators

8. **Unit Testing in C++**

- Unit testing – Quick Overview
- Unit testing in C++
- Introduction <catch.hpp>

9. **Advanced Polymorphism and Inheritance**

- Good Class Design

- string Class
- Public Inheritance
- Public Inheritance Problems
- Inheritance and Semantics
- Private Inheritance
- Composition
- Composition vs. Private Inheritance
- Templates vs. Inheritance
- Protected Inheritance
- Implementation Encapsulation
- Interface Inheritance
- Multiple inheritance issues

10. **Functional Programming**

- Overview
- The IoC pattern
- Dependency Injection
- Functions as objects
- IoC via interface
- Functors
- IoC with Functors
- Implementing Functors
- Function Pointers
- IoC with Function Pointers
- Lambda Expressions
- Lambda Syntax
- IoC with Lambdas

11. **Runtime Type Information**

- Runtime Type and Polymorphism
- type_info Class
- typeid Operator

Course Outline

- Compiler Options
- Safe Pointer Conversions
- Dynamic Cast
- New C++ Style Casts
- Static Cast
- Reinterpret Cast
- Const Cast

12. **Overview of Standard Library**

- Perspective
- History and Evolution
- New Features
- The Standard Template Library
- Generic Programming
- Design Goals
- Header Files
- STL Components
- Containers
- Algorithms
- Iterators
- Threads & Tasks

13. **STL Containers**

- Vectors
- Vector.cpp
- Vector Operations
- Typedefs
- Deques
- deque as Stack
- deque Functionality
- Lists
- Generic Programming
- Tradeoff with Lists
- List Memory Allocation
- list Functionality
- Associate Containers
- Sets
- Sets with User Defined Objects
- Multisets (Bags)
- Maps
- Multimaps
- Functional Programming with Containers

14. **STL Iterators**

SHI | TRAINING & ADOPTION

Course Outline

- Pointers
- Template Version
- String Version
- A Generalization of Pointers
- STL Iterators
- Input Iterators
- Output Iterators
- Forward Iterators
- Bidirectional Iterators
- Random Access Iterators

15. **Threads & Tasks**

- Overview Threads
- Starting Threads
- Managing threads
- Overview of Tasks
- Tasks
- async
- Future & Promise