**C++ Content Outline**

**Program Title: C++**

**Duration: 5 days**

**Overview:**

Advanced OO programming using C++ . The training course presents a thorough understanding of object oriented concepts and its application. The central concepts of C++ syntax and style are taught in the context of using object-oriented methods to achieve reusability, adaptability and reliability. Emphasis is placed on the features of extensive usage of Standard Template libraries.

**Objectives:**

* To learn the concepts of Object Oriented Programming (OOP)
* Understand the structure of a C++ program
* Explore classes, the building blocks of C++
* Work with constructors and destructors
* Understand function - and operator-overloading
* Learn about inheritance
* Discover the power of virtual functions
* Work with templates and exception handling
* Understand Runtime Type ID and the Casting Operators
* Utilize namespaces

**Suggested Audience**

Developers, Senior Developers who want to specialize in object-oriented C++ programming.

**Prerequisites:**

* Good knowledge and experience in C Programming constructs
* Good understanding of pointers and data structures implementation in C programs.

**Contents**

**DAY 1**

**AN OVERVIEW OF OBJECTORIENTED CONCEPTS**

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

**Relationships**

* Generalization
* Realization
* Association
* Aggregation
* Composition
* Dependency

**AN OVERVIEW OF C++**

* What is Object Oriented Programming?
* Two versions of C++
* C++ Console I/O
* C++ Comments
* Classes: A First Look
* Some Differences between C and C++
* Introducing function overloading
* C++ keyboards

**ARRAYS, POINTERS, REFERENCES**

* Arrays of objects
* Using Pointers to objects
* The this Pointer
* Using new And delete
* More about new And delete
* References
* Passing references to objects
* Returning references
* Independent references and restrictions

**FUNCTION OVERLOADING**

* Overloading constructor functions
* Creating and using a copy constructor
* The overload anachronism
* Using default arguments
* Overloading and ambiguity
* Finding the address of an overloaded function

**INTRODUCING CLASSES**

* Class Definition
* Class Objects
* Class Member Functions
* The Implicit this Pointer
* Pointer to Class Member
* Class Scope
* Nested Classes
* Classes as Namespace Members
* Local Classes

**DAY 2**

**OBJECT INITIALIZATION**

* Compiler Synthesized Constructor
* Deep copy v/s Shallow copy
* Overloaded constructor
* Copy constructor & Generic Copy constructor
* Explicit constructor
* Copy Constructor v/s Assignment operator

**INITIALIZATION LIST**

* Initialization List
* Order of Initialization
* Initialization v/s Assignment
* Default Arguments
* Calling base class constructor

**OBJECT CLEANUP**

* Destructor
* Compiler Synthesized Destructor
* Preventing destroying object instance

**DYNAMIC MEMORY MANAGEMENT**

**New operator**

* New vs malloc vs calloc
* Handling bad\_alloc exception
* Using new with nothrow
* Placement new
* Overloading new operator

**Delete operator**

* Delete vs Free
* Destroying objects on heap
* Destroying array of objects
* Smart pointer

**OBJECT ATTRIBUTES**

* Static members and objects on the heap
* Static members of a class
* Static data members
* Static member functions
* Const data members
* Const member functions

**FRIEND FUNCTIONS**

* Friend and classes
* Object communication
* Friend functions
* Friend classes
* Const objects and const member functions
* Object composition & destruction mechanism

**DAY 3**

**OPERATOR OVERLOADING**

* Operator Overloading
* Friends
* Operator =
* Operator []
* Operator ()
* Operator ->
* Operators ++ and –
* Operators new and delete
* User-Defined Conversions
* Selecting a Conversion
* Overload Resolution and Member Functions
* Overload Resolution and Operators

**INHERITANCE**

* Defining a Class Hierarchy
* Types of inheritance
* Identifying the Members of the Hierarchy
* Base Class Member Access
* Base and Derived Class Construction
* Base and Derived Class Virtual Functions
* Memberwise Initialization and Assignment
* Containment vs Inheritance
* Private vs protected inheritance
* Hybrid Inheritance / virtual base class
* Friend class v/s Derived class

**VIRTUAL FUNCTIONS**

* Virtual member function
* Pure virtual function
* Abstract class v/s Interface v/s Concrete class

**VIRTUAL FUNCTION ISSUES**

* Calling virtual function from constructor
* Calling virtual function from destructor
* Calling virtual function from non virtual member function
* Object Slicing

**VIRTUAL INTERNALS**

* Virtual functions in Single Inheritance
* Virtual functions in Multiple Inheritance
* Virtual Inheritance

**DAY 4**

**RUNTIME TYPE IDENTIFICATION AND CASTING**

* typeid function
* type\_info class object
* dynamic\_cast , static\_cast, const\_cast
* RTTI Internals

**NAMESPACES**

* The STD Namespace
* Custom Namespaces
* Namespaces and Their Limitations

**EXCEPTION HANDLING**

* Throwing exception
* try block
* catch block
* multiple catch blocks
* catch any block
* set\_terminate functions
* custom exception class

**EXCEPTION HANDLING ISSUES**

* Order of catch blocks
* Catching exception by value
* Throwing exception in constructor
* Throwing exception in destructor

**DAY 5**

**STANDARD TEMPLATE LIBRARY**

**GENERIC FUNCTION**

* Template Function
* Overloading vs Generic Function
* Function Template Internals
* Full Specialization
* Partial Specialization
* Template function overloading

**GENERIC CLASSES**

* Class templates
* Class Template Full specialization
* Class Template Partial specialization
* Code Blow and Class templates

**CONTAINERS**

* Types of containers
* Associative containers
* Sequential containers
* vector
* list
* map , multimap
* set ,multiset

**ITERATORS**

* Iterators and const\_iterators
* accessing iterator information in generic algorithms
* types of iterators
* input iterator
* output iterator
* forward iterator
* reverse iterator
* random iterator

**GENERIC ALGORITHMS**

* review: generic algorithms
* interaction between algorithms and iterators
* types of algorithms
* mutating algorithms
* non mutating algorithms