Advanced C++ Programming

C++, based on the C programming language, is an Object-Oriented Programming (OOP) language. Unlike C, C++ is built on the concept of "objects" instead of using data and actions on data as the basis for the program's logic. Using OOP, related data and routines are grouped into an object that then relates to other objects in the program. These objects can represent all of the parts and functions of a real entity or an abstract idea. C++ is a powerful language that is inherently useful for large-scale projects.

This course broadens the skills of a C++ programmer by presenting an in-depth treatment of templates, exceptions, memory management, advanced inheritance issues, disambiguation, adaptors, reference counting, runtime type identification, and the standard template library. Group discussions and lab exercises support the classroom lectures.

Course Objectives:

- Write programs using the C++ template facility
- Distinguish between the different forms of inheritance
- Identify the correct C++ feature to implement a particular design specification
- Implement multiple inheritance when necessary
- Write programs which utilize a robust set of data structure classes
- Understand programs which use function pointers in a wide variety of problems
- Use the exception handling capability of modern C++ compilers
- Use the algorithms, containers, and iterators from the new Standard Template Library
- Understand the complex set of rules which govern C++'s disambiguation algorithm
- Write programs that use the advanced I/O features from the iostreams library

Audience: Individuals interested in enhancing their knowledge of the C++ language.

Prerequisites: *C++ Programming*

Number of Days: 4 days

1. **Course Introduction**
   - Course Objectives
   - Overview
   - Suggested References
2. **What You Should Already Know-a Review**
   - Rationale for a new programming language
   - The language of Object Orientation
   - A typical C++ class
   - Issues regarding member functions vs. non-member functions
   - friend Or non friend
   - functions - returning references
   - Relationships
   - Initialization lists
   - Inheritance in C++
   - Access Levels
   - Simple C++ I/O
   - The uses of const
3. **Parameterized Types - Templates**
   - Templates
   - Overloading functions
   - Template functions
   - Specializing a template function
   - Disambiguation under specialization

- Template classes
- Instantiating a template class object
- Rules for template classes
- Non member function w/ a template argument
- Friends of template classes
- Templates with multiple type parameters
- Comments regarding templates

4. Relationships of All Kinds

- Uses of Member Initialization Lists
- Member initialization lists under composition
- Initialization lists under inheritance
- Initialization lists w/ Multiple Inheritance (MI)
- Initialization with MI and composition
- Efficiency
- operator= and composition
- Constructors and composition
- What is not inherited?
- operator=, construction, and inheritance
- Public inheritance
- Virtual functions
- A shape class hierarchy
- Polymorphism
- Pure virtual functions
- Abstract base classes
- Private inheritance
- Using relationships
- Associations

5. Multiple Inheritance

- Multiple inheritance
- Ambiguities
- virtual base classes
- The Dominance Rule
- Member initialization lists
- operator=

6. Data Structures

- Introduction
- A simple List
- Layering type safe classes upon List
- A template List class
- Iterators
- A template iterator
- Stack and Queue classes
- Templates and Inheritance

7. Function Pointers

- Why have function pointers?
- Passing functions as arguments
- Registering functions
- Function pointers in C++
- Callback functions
- A class with a callback object
- Registration of exceptions handlers

8. Exceptions

- What are exceptions?
- Traditional approaches to error handling
- try, catch, and throw
- A simple exception handler
- Multiple catch blocks
- The exception specification list
- Rethrowing an exception
- Cleanup
- Exception matching
- Inheritance and exceptions
- Resource allocation
- Constructors and exceptions
- Destructors and exceptions
- Catch by reference
- Standard exceptions

9. Standard Template Library

- The Standard Template Library
- Design goals
- STL Components
- Iterators
- Example: vector
- Example: list
- Example: set
- Example: map
- Example: find
- Example: merge
- Example: accumulate
- Function objects
- Adaptors

10. Disambiguation

- Conversion
- int Conversions
- float + double Conversions
- Arithmetic and pointer conversions

Inheritance based conversion
Overloaded functions
Exact match
Match with promotion
Match with standard conversion
User defined conversion
Constructors as conversion operators
Ambiguities
11. File I/O
Introduction
Manipulators
Writing your own manipulators
Overloading the I/O operators
Disk files
Reading and writing objects
Internal transmission of data
A spell checker
Handling Streams in the constructor and destructor
Treating a file as an array
12. Miscellaneous Topics
Namespaces
Use counts
Reference counts
RTTI
Casts
Having a limited number of objects
Smart pointers