

**Course Description:** This course broadens the skills of a C++ language programmer by examining sophisticated C++ concepts such as templates, exceptions, memory management, advanced inheritance issues, disambiguation of overloaded functions, private and protected inheritance, binary I/O and class libraries.

**Who Should Attend:** This course is for anybody who has programmed in C++ and wishes to enhance their knowledge of the language.

**Prerequisites:** Students should have completed an introductory C++ programming course or have equivalent knowledge.

**Benefits of Attendance:** Upon completion of this course, students will be able to:

- Decide between global functions, friend functions and member functions.
- Code their own memory management routines by overloading operators new and delete.
- Write classes and functions with parameterized types.
- Understand and handle exceptions in C++ programs.
- Disambiguate data and functions using multiple inheritance.
- Understand the difference between various kinds of inheritance.
- Use pointers to class member functions.
- Understand the C++ mechanism to resolve overloaded functions.

### Course Outline:

#### What You Should Already Know - A Review

Rationale for a new programming paradigm  
The language of Object Orientation  
A typical C++ class - a string class  
Issues regarding member functions  
Relationships  
Simple C++ I/O  
The uses of const

#### Parameterized Types - Templates

Templates  
Overloading functions  
Template functions  
Specializing a template function  
Disambiguation under specialization  
Template classes  
An array template class  
Instantiating a template class object  
Rules for templates  
Non member function w/ a template argument  
Friends of template classes  
Templates with multiple type parameters  
Comments regarding templates

#### Relationships Of All Kinds

Uses of Member Initialization Lists  
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  
Designing for inheritance  
Public inheritance  
Private inheritance  
Private inheritance vs composition  
Using relationships  
Associations

#### Multiple Inheritance

Multiple inheritance  
Ambiguities  
Removing ambiguities  
virtual base classes  
virtual base classes and the dominance rule  
Member initialization lists with MI  
operator= and MI  
Designing for inheritance

#### Data Structures

Introduction  
A simple List  
Implementation of the list functions

Layering type safe classes upon List  
A template List class  
Iterators  
A template iterator  
Stack and Queue classes  
A derived template array class

#### Function Pointers

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

#### 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

#### The C++ Standard Template Library

History and evolution  
New features  
The Standard Template Library  
STL Components  
Iterators  
Example: vector  
Example: list  
Example: set  
Example: map  
Example: find  
Example: merge  
Example: accumulate  
Function objects  
Adaptors

#### Disambiguation

Conversion  
int Conversions  
float  
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

#### File I/O

Introduction  
Error checking  
Overloading &&t;&t; and >>  
Formatted I/O  
Disk files  
Examples of seekg, tellg, and close  
Reading and writing objects to the disk  
Internal transmission of data  
A larger I/O example: Spell checker  
Treating a file as an array

#### Miscellaneous Topics

New features in the standard C++ language  
Namespaces  
Use counts  
Run Time Type Identification  
New casts  
Overloading operator new and delete  
How virtual functions are implemented  
Having a limited number of objects  
Smart pointers