Distance Learning Advanced C++ - Programming Models, boost and Parallel Computing

Module 1: Quick Review of C++ Essentials

General Considerations

- The canonical class definition
- Why const is important
- Raw and smart pointers
- Robust C++ code: guidelines

Advanced Overloading

- Overloading index operators [] and ()
- The assignment operator and memory management
- Overloading the ostream operator <<
- Functors and function objects
- Comparing functors with function pointers

Simple Inheritance

- Inheritance and ISA Relationship
- Specialisation Scenarios
- Inheritance and Object Creation
- Using Base Class Constructors
- Accessibility of Base Members
- Overriding Functions

Polymorphism

- Pointers to the Base Class
- Function Visibility
- Polymorphism
- Defining an Interface
- Abstract Base Classes
- Virtual Destructors
- Operator Overloading and Inheritance

Module 2: Generic Programming and Policy-based Design

Programming with Templates I

- Multiple parameters
- Nested template class
- Inheritance and composition
- Compile-time and fixed-sized array classes

Programming with Templates II

- Default parameter values
- Template template parameters
- Some templated design patterns
- Template specialization; partial specialisation

Templated Software Components

- Traits classes
- Services and policy-based design
- 'Provides' and 'requires' interfaces
- Implementing policies in C++

Advanced GOF: Combining Components into larger Components

- Creating pattern languages
- Creating networks of inter-related patterns
- Using GOF patterns in larger architectures
- Finding the right patterns
- Contracts and where clauses

Generic Patterns and Generic Programming

- An introduction to generic programming
- Comparing OOP with GP
- Designing components in GP framework
- 'Provides' and 'requires' interfaces

The Design of Generic Components

- Traits and their applications
- Policy classes
- Combining policies and traits
- Test Case: a policy-based templated Command and Proxy patterns
- Examples

Module 3: Standard Template Library (STL)

Overview of Standard Template Library (STL)

- What is STL?
- STL Components
- Containers
- Main Container Types
- Algorithms
- Main Algorithm Categories
- Set-like Operations
- Iterators
- Function Objects
- Adaptors
- Allocators

- Strengths and Limitations of STL
- Student Prerequisite Knowledge

STL Containers

- Sequence Containers
- Vector
- Deque
- List

Sorted Associative Containers

- Multisets (Bags)
- Sets
- Set_like Operations on Sorted Structures
- Multimaps
- Maps

Iterators in STL

- What is an Iterator?
- Iterator Categories
- Iterator functions
- Iterator functions: Input Iterators
- Output Iterator Types
- Forward Iterators
- Bi-directional Iterators
- Random Access Iterators
- Qualifying Iterators: Mutable and Constant Iterators

Algorithms in STL

- Overview of STL Algorithms
- Algorithm Categories
- Algorithms with Function Parameters
- Non-mutating Sequence Algorithms
- Mutating Sequence Algorithms
- Sorting and searching

Module 4: Boost Containers, Data Structures and Higher-Order Programming

MultiArray

- Creating n-dimensional data structures
- Performance issues compared to STL
- Slicing and views
- Resize, reshape and storage
- Multi-index and sub-object searching

Range

- Modelling pairs of iterators
- Using ranges with generic algorithms and STL containers
- Raising the abstraction level
- Using metafunctions

Tuple

- Modelling n-tuples (pair is a 2-tuple)
- Using tuples as function arguments and return types
- Accessing the elements of a tuple
- Advantages and applications of tuples

Variant

- Creating discriminated unions with heterogeneous types
- Manipulating several distinct types in a uniform manner
- Type-safe visitation
- Avoiding type-switching for variant data

Any

- Value-based variant types
- Discriminated types
- Typesafe storage and retrieval Applications of Any

Multi-Index Containers

- Bidirectional maps
- Sets with several iteration orders
- Emulation of standard containers
- MRU lists
- Category: Function Objects and Higher-Order Programming

Bind

- Generalising and improving the STL Bind
- Uniform syntax for functors, (member) function pointers
- Functional composition and nested binders
- Bind as used in Boost.Function

Function

- Generalised callback mechanisms
- Storage and invocation of functors, (member) function pointers
- Useful in notification patterns (Observer, Signals and Slots)
- Example: separating GUIs from business logic

Signals and Slots

- Implementation of Observer (Publisher-Subscriber) pattern
- Event management with minimal inter-object dependencies
- Signals == Subject, Slots == Subscriber
- Application to Mediator and Observer patterns

Lambda

- Unnamed functions
- Useful for STL algorithms
- Avoiding creation of many small function objects
- Less code: write function at location where it is needed
- Lambda function in C++ 11

Module 5: Boost I/O and other Utilities

Filesystem

- Portable manipulation of paths, directories and files
- Defining functionality as in scripting languages
- Platform portability

Serialisation

- Saving arbitrary data to an archive (e.g. XML)
- Restoring data from an archive
- Versioning

Regex

- Regular expressions and pattern matching
- Processing large and inexact strings
- Emulating functionality as in Perl, awk, sed

Spirit

- Functional, recursive descent parser generator framework
- Command-line parsers
- Specifying grammar rules in C++ (EBNF syntax)
- Performance issues

Tokenizer

- Separate character sequences into tokens
- Finding data in delimited text streams
- User-defined delimiters

Time Series and Forecasting

- Linear and multiple regression
- Smoothing
- Exponential smoothing
- Multiplicative model

Boost Accumulator Library

- Overview and application areas
- The Statistical Accumulators Library
- Examples and test cases

Module 6: Boost Interprocess and Network Communication

Asynchronous Communication

- Network and low-level I/O
- Proactor design patterns
- Strands
- Custom memory allocation

Networking

- TCP, UDP, ICMP
- Socket I/O streams
- SSL support
- Serial ports

Interprocess

- Shared memory
- Memory-mapped files
- Semaphores and mutexes
- File locking
- Message queues

UML Statecharts in Boost

- Hierarchical (composite, nested) states
- Orthogonal states
- Transitions and Guards
- Event delay

Module 7: Multi-threaded and Parallel Programming *Boost Thread*

Memory Systems

- Shared memory parallel computers (SMPs)
- Shared and cache memory
- Shared memory consistency models
- Distributed memory and shared distributed memory

Threads

- What is a thread?
- Thread attributes
- Thread execution lifecycle
- User threads and kernel threads

Data Access in Threads

- Fork-join (master/slave) model
- Shared and private data
- Thread synchronisation

Synchronisation in Detail

- Mutual exclusion (mutex) and condition variables
- Critical sections
- Memory synchronisation and fences
- Barriers

Troubleshooting

- Sequential consistency
- Removing data dependencies
- Race conditions
- Deadlock and livelock

Boost Threads

- Free thread functors
- Thread classes
- Non-member functions
- Status of Boost Thread

Synchronisation

- Mutex concepts

- Lock mechanism and lock types
- Condition variables
- Barriers
- Futures

Other Topics

- Thread local storage
- Emulations
- Conformance and Extension

Module 8: OpenMP, an Introduction

Overview

- Compiler directives
- Library routines
- Environment variables

My First OpenMP Program

- Writing the serial program
- Determining parallel code
- Adding OpenMP directives
- Debugging and performance measurement

Data Clauses in OpenMP

- Shared and private
- Lastprivate, firstprivate
- Default and nowait clause

OpenMP Synchronisation Constructs

- Barrier
- Ordered
- Critical and Atomic
- Locks, Master construct

Work Sharing in OpenMP

- Loop construct
- Sections and section
- Single construct
- Combined parallel work-sharing constructs
- Other Clauses
- Reduction clause
- Copyin clause
- Copyprivate clause
- Ordered clause

Configuration and Run-Time Information

- Setting environment variables' values
- Library functions for thread information
- Scheduling functions
- Lock functions
- Timing functions

Module 9: Applications: Design and Implementation

Structural Modelling, I

- Overview of Unified Modelling Language (UML)
- Class diagrams
- Component diagrams
- Sequence diagrams

Structural Modelling, II

- Semantic modelling
- Generalisation/Specialisation (Inheritance)
- Aggregation and Composition
- Association and association class

Whole-Part Pattern

- Modelling complex structured objects
- Whole-Part types
- Checklist: which type to use
- The steps in implementing Whole-Part object
- Applications for Whole-Part

Detailed Software Requirements for Components

- Throwaway, non-throwaway and production software
- What are the top software requirements?
- Functional and non-Functional requirements (FRs and NFRs)
- How FRs and NFRs affect component design

Combining Component and Object Technologies

- Comparing Component and Object Design
- The differences between OOD and COD
- Combining components and objects
- Assemblies and namespaces
- Developing components from objects
- Component loading and the object instantiation process

Using Components and Objects for GOF Patterns

- When to use interfaces and when to use abstract classes
- Using classes and objects in combination with components
- Stateless and Stateful GOF patterns
- Delegation and Composition

Designing C++ Applications

- Choice of programming models
- Complexity Analysis and data structures
- Which STL and boost libraries to use
- Design patterns

Performance of C++ I

- Classifying and discovering performance bottlenecks
- Virtual versus non-virtual functions
- Preventing unnecessary object creation
- Exceptional handling

Performance of C++ II

- Templates versus inheritance
- Using the appropriate data structures from STL

- Loop optimizing techniques
- Loop fission, fusion, unrolling and tiling

C++ 11 Update (contents similar to Wiki entry C++ 11)

- Core language usability enhancements
- Core language functionality improvements
- C++ standard library changes

Module 10: Linear and Nonlinear Data Structures

Overview

- Abstract data types and algorithms
- Taxonomy of data structures
- Mathematical tools for algorithm analysis
- Linear and nonlinear data types
- Design strategies

Review of Fundamental Data Structures

- Vectors, matrices and arrays
- Sets, stack and queues
- Linked lists

Complexity Analysis

- Computational and asymptotic complexity
- Big-O notations
- Other measures of complexity
- Potential problems
- NP-completeness

Recursion

- Basic concepts
- Function calls and recursive implementation
- Tail, nontail and nested recursion
- Backtracking

Binary Trees

- Mathematical properties
- Complete and full binary trees
- Computed the depth of a binary tree
- 2-trees

Introduction to Graph Theory

- Directed and Undirected Graphs
- Properties of Graphs and graph
- Paths and Connectivity
- Special Types of Graphs

Graph Structure and Algorithms

- Graph data structures and operations on graphs
- Minimum spanning tree (MST) problems
- Depth-first and Breadth-first searches in graphs
- Shortest path problems
- Connected components

Boost Functional Hash

- Hash function and hash table
- Categories of hash function
- Creating custom hash
- Applications

Boost Heap

- Heap ADT
- Variants (Fibonacci, skew, priority queue, etc.)
- Heap and computational efficiency
- Boost Heap versus STL heap

Boost Unordered

- Hashed associative containers
- Complexity analysis
- Applications
- Integration with STL and other Boost libraries
- Custom Types
- Controlling the number of buckets

Boost Bimap

- What is a bidirectional map?
- The three views of a bimap
- Integration with STL
- Implementing UML association class