Advanced C++ and C++ 11: The new Standard (Financial) (code CPPA)

Part I: C++ 11 as a 'better' C++

New Language Features I

- Keywords
- auto
- decltype
- noexcept
- constexpr
- nullptr

New Language Features II

- Uniform initialization and initializer lists
- Default template parameters
- Function declaration syntax
- New fundamental data types

Move Semantics

- What is move?
- Copying versus moving: performance
- Rvalue references
- Move constructor and move assignment

Using Move

- Arrays and Containers
- File streams
- User-defined copyable and movable classes
- Composition and inheritance
- Perfect forwarding

Exception Handling

- Exception class hierarchy
- Logic and run-time errors
- Exceptions thrown by the Standard Library
- Error codes compared to error conditions

Smart Pointers and Memory Management

- Design rationale
- Class shared_ptr
- Destruction policies
- Class weak_ptr
- Class unique_ptr
- Performance and reliability

Part II: Modelling Functions and STL

Background

- Traditional Approach
- Function pointers
- Function overloading and virtual functions
- The categories of polymorphic behaviour
- Using (and misusing) inheritance to realise subtype polymorphism

Fundamentals of Functional Programming (FP)

- Short history of FP
- Higher-order functions
- Recursion; passing a function to itself
- Strict and non-strict (delayed) evaluation
- Pure functions and lambda functions

Functional Programming

- Functions and Data
- Function composition
- Closures
- Currying and uncurrying
- Fold and continuations

Functional Programming in C++

- Overview
- C++ as a multi-paradigm programming language
- Universal function type
- (polymorphic) wrappers (std::function)
- Binders and predefined function objects (std::bind)
- Lambda functions versus binders
- A uniform function framework

Lambda Functions

- What is a lambda functions
- The closure of a lambda function: closure
- Using lambda functions with auto
- The mutable keyword

Using Lambda Functions

- Configuring applications
- With algorithms
- As sorting criteria
- As hash function
- Lambda functions versus function objects

A Taxonomy of Functions in C++

- Function pointers and free functions
- Object and static member functions
- Function objects
- Lambda functions

- Events and signals (Boost *signals2* library)

Part III: Data Structures and STL Review of STL

- Containers
- Sequence containers
- Associative containers
- Unordered containers
- Container adapters
- User-defined containers

Hashing

- 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

Unordered Containers

- Differences with (ordered) associative containers
- Abilities of unordered containers
- Complexity analysis
- Integration with STL and other Boost libraries
- The Bucket interface

Tuples

- 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

Fixed-sized Arrays `std:array<>`

- Why do we need `std:array<>` ?
- Operations and abilities
- Using arrays as C-Style arrays
- Combining arrays and tuples

Part IV: Other Libraries

Clocks and Timers

- Overview of Chrono library
- Duration and timepoint
- Clocks
- Date and time functions

Regular Expressions (Regex)

- Regex
- Match and Search Interface
- Subexpressions
- Regex iterators and token iterators
- Replacing regular expressions
- Flags and expressions

Random Numbers and Statistical Distributions

- What are random and pseudo-random numbers?
- Engines and distributions in C++
- Basic engines, engine adapters; adapters with predefined parameters
- Categories of distributions
- Examples and applications

Concurrency Fundamentals

- Threads in C++; properties
- Promises and return arguments
- Threads in detail
- Mutexes and locks

Advanced Concurrency

- Synchronisation and condition variables
- Futures and async()
- Launch policies
- Waiting and polling
- Example: Producer-Consumer pattern

Part V: C++ 11 Application Design

Advanced Templates

- Partial specialization
- Dynamic versus static polymorphism
- Generic programming
- Variadic templates
- Alias templates (template typedef)
- Generic lambda functions

Using C++ 11 in Applications: Epilogue

- Design patterns revisited and reengineered
- Multi-paradigm design in C++
- Software layering
- Software components
- Software assembly process

C++ 14 …

- Minor bug fixes and enhancements
- Generic lambdas and lambda captures expressions
- Function return type deduction for all kinds of function
- Aggregate member initialization
- New standard library features

Your Trainer

Daniel J. Duffy started the company Datasim in 1987 to promote C++ as a new object-oriented language for developing applications in the roles of developer, architect and requirements analyst to help clients design and analyse software systems for Computer Aided Design (CAD), process control and hardware-software systems, logistics, holography (optical technology) and computational finance. He used a combination of top-down functional decomposition and bottom-up object-oriented programming techniques to create stable and extendible applications (for a discussion, see Duffy 2004 where we have grouped applications into domain categories). Previous to Datasim he worked on engineering applications in oil and gas and semiconductor industries using a range of

numerical methods (for example, the finite element method (FEM)) on mainframe and mini-computers. Daniel Duffy has BA (Mod), MSc and PhD degrees in pure and applied mathematics and has been active in promoting partial differential equation (PDE) and finite difference methods (FDM) for applications in computational finance. He was responsible for the introduction of the Fractional Step (Soviet Splitting) method and the Alternating Direction Explicit (ADE) method in computational finance. He is also the originator of the exponential fitting method for time-dependent partial differential equations.

He is also the originator of two very popular C++ online courses (both C++98 and C++11/14) on www.quantnet.com in cooperation with Quantnet LLC and Baruch College (CUNY), NYC. He also trains developers and designers around the world. He can be contacted dduffy@datasim.nl for queries, information and course venues, in-company course and course dates