Fundamentals of C++11 and C++14 (code CPPF-11)

Course Contents

Part I Background

Design Goals

- Stability and compatibility with C++98
- Increase type safety
- Increase performance
- Make C++ easy to learn

Global Overview

- Core language usability enhancements
- Core language functionality enhancements
- C++ Standard Library changes
- Multithreaded memory model
- Removed and deprecated features

II Fundamentals and Essential Language Features

New Language Features

- 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 operator
- Rule-of-Three and Rule-of-Five

Memory Management and Smart Pointers

- Design rationale
- Class `shared_ptr`
- Destruction policies
- Class `weak_ptr`
- Class `unique_ptr`

Using Smart Pointers

- Smart pointers versus raw pointers
- Classes with embedded pointers
- Reengineering legacy code and software design patterns
- Move semantics with shared pointers

Bits and Pieces: Usability Enhancements

- Type alias (alias template)
- Automatic type deduction and auto specifier
- Range-based for loops
- nullptr
- New fundamental data types

III Data Types and Data Containers

Basic Types

- Unrestricted unions
- `std::bitset`
- `std::ratio`
- Timepoint and clock

Basic Containers

- std::forward_listForward list
- Fixed-sized array `std::array<>`
- `std::pair<>` and `std::tuple<>`
- Exception classes

Applications of Basic Containers

- Tuples as function return types and input arguments
- Fixed-sized matrices based on `std::array<>`
- Using tuples to hold configuration data; tuple nesting

IV C++ Classes and Class Modelling

New Class-related Functionality

- explicit specifier
- Deleted and defaulted member functions (delete, default)
- Generalised constant expressions (constexpr)
- override and final
- noexcept
- Uniform initialization

Class Templates

- Template declaration
- Implementing the member functions
- Using the template class
- Default template arguments
- std::decltype and std::declval
- Examples

Function Templates

- What is a function template?
- Defining the template
- Using the template
- Argument deduction
- Overloading function templates
- Advanced Template Programming
- Partial specialization
- Nested template classes (for example, 1:N aggregations)
- Traits and policy classes
- Template template parameters

V The C++ Function Panorama

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

Lambda Functions

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

A Taxonomy of Functions in C++

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

Applications

- Using `std::function` as universal function type
- `std::function` and its target methods
- Using lambda functions to configure applications
- `std::function` as an alternative to virtual functions
- Regex
- Random number generator

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