Workshop Advanced C++11, C++14 and C++17 for Computational Finance

Course Contents

Day 1 New Essential Features

The objective of this part is to discuss new C++ feature that represent major improvements on C++03 and that we use as building blocks in applications.

Move Semantics

- What is move?
- Copying versus moving: performance
- Rvalue references
- Move constructor and move assignment operator
- Rule-of-Three and Rule-of-Five for classes

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

IEEE 754

- Overview of IEEE 754
- Numerics and IEEE 754
- Rounding rules and exception handling
- Normal, subnormal and infinite numbers; NaN
- Machine precision
- Rounding and cancellation errors

Numerics in C++

- `std::numeric_limits<>`
- Directed roundings
- Floating-point decomposition functions
- Error analysis
- Comparing floating-point numbers

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

Day 2 Advanced Language Features

In this part we introduce the functional programming model in C++, type traits (meta template programming), new abstract data types and how to us them to create robust code.

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 function?
- The closure of a lambda function
- 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)

Introduction to Type Traits
- Introduction to metaprogramming
- Defining behaviour based on type
- Type categories
- Using type traits in applications and libraries

Type Categories
- Primary (is a generic type of a given type?)
- Composite (is a type scalar, compound or object?)
- Properties (e.g. is a class abstract)
- Relationships (comparing types In some way)

Some Applications of Type Traits
- Robust numerics libraries
- Compile-time *Bridge* design pattern
- Type-independent code

Advanced Data Containers and Types
- Bitsets
- Hash function and hash table
- Unordered containers
- C++17 variant type

Random Number Generation and Statistical Distributions
- Random number engines (basic, adapters and adapters with predefined parameters)
- Seeding an engine and a collection of engines
- Univariate distributions in C++11
- Some applications
- A generic class for generating variates of an arbitrary distribution

The new C++ Memory Model
- Sequential consistency
- Ordering non-atomic operations
- Relaxed consistency models
- Total order

Day 3 Libraries and Modern Software Design in C++

We introduce a number of C++11 libraries that are important for computational finance, in particular libraries for random number generation, multithreading and multitasking. We also gibe a short critique of traditional software design patterns and how we replace them by modern multiparadigm-based patterns in C++. We also show how system decomposition works and how to design parallel code for finance.

Introduction to C++ Threads
- What is a thread?
- Creating a thread with various callable objects
- Thread function: pros and cons
- Waiting on a thread; detaching a thread
- Using lambda functions

Atomics
- Atomic types and atomic operations
- Atomic load, store, increment, decrement
- Atomic flags
- Smart pointers and thread-safe pointer interface

Quick Review of Gamma (GOF) Patterns
- Core techniques: subtype polymorphism and composition
- Summary of creational, structural and behavioural patterns
- 'Pattern pruning' and redundant patterns
- Most important patterns

Modern Software Design in C++11
- Distinguishing between object structure and object behaviour
- Multi-paradigm design patterns in C++11
- High-priority patterns

Walkthrough: Monte Carlo Model Case (C++11), I
- System decomposition and logical components
- Designing physical components
- Assembling and configuring the application
- Extending the functionality

Day 4 Interoperability, Applications and Performance

In this last part we set up a student project to analyse, design and implement two mini-applications using the techniques from the first three days. In particular, we focus on the requirements for PDE (one-factor and two-factor models) and Monte Carlo applications and we create the final software product in a series of prototypes, with each prototype delivering more functionality than its predecessor.

We provide the PDE and Monte algorithms that you design and implement in C++. The percentage theory/practice in this part is approximately 30%/70%.

Parallel Design in C++11

- Task/data decomposition and task dependency graph
- Implementing tasks with futures and promise
- Parallel design patterns
- Parallel applications in for computational finance

PDE Model Case, I

- Modelling one-factor and two-factor PDEs in C++
- PDE types
- Creational patterns for PDEs
- PDEs and aggregate structures

PDE Model Case, II

- Modelling the finite difference method (FDM) in C++
- Realising software requirements: customisability, efficiency and maintainability
- Designing FDM in multi-paradigm C++

Schemes Used

- Crank Nicolson
- Alternating Directions (ADI and ADE variants)
- Soviet Splitting
- Method of Lines (MOL)

Models

- One-factor American options
- Convertible bonds (Hull-White model)
- Two-factor baskets
- Hesston and SABR

Essential Design Patterns used in Software Solution

- Mediator (system decomposition)
- Bridge and Strategy (interoperability and portability)
- Event notification and observers
- Parallel patterns and speedup

Model Validation Requirements

- Computing sensitivities (AD, complex step method)
- Optimisation (Differential Evolution, simulated annealing, LevMar)
- Comparing accuracy with Monte Carlo and lattices

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