C++ Programming for Financial Engineering

Goals

The goal of this course is to introduce the C++ language to Masters students in financial engineering. This is a hands-on course and it consists of a number of level, each one dealing with a specific topic in C++. Each level is accompanied by small 'quizzies' to test if the general theory has been understood and by more extended programming exercises and applications in which you are expected to program in C++ and produce a working program. The compiler and environment used is Visual Studio because of its ease of use but other C++ compilers can be used.
This is the ideal preparation because it deals with the essential topics that are needed before progressing to more advanced applications in computational finance.

Prerequisites

No programming experience is required in order to follow this course. The course has been structured in such a way that you learn C++ from the ground up.
You can freely download the Visual Studio Express compiler from the Microsoft site.

The Course 'Process'

The course medium is high definition video. I use a combination of Powerpoint, Internet, Visual Studio in combination with audio and video. Each session discusses one logical topic (for example, C++ memory) and in general its duration is between 20 and 30 minutes. The approach is to combine theory and scaled examples to consolidate the learning process.
The examples will be 50% of a general nature (but related to engineering and math applications) and 50% to examples in computational finance.

Quizzes and Exercises

Short quizzes are given at the end of each level. They test what you have learned and they should be done before moving on to hands-on exercises in C++.
There are also four extended exercises for Computational Finance.

Course Content Originator and Trainer

Daniel J. Duffy is an internationally known trainer, writer and software designer. He is founder of Datasim Education and has been involved with C++ since 1988 as developer, designer and trainer. Clients are companies in all areas of software development over the world, including investment banks, hedge funds and MFE students at several universities. He has written ten books on software design, C++ and their application to computational finance.
Daniel Duffy has a PhD in numerical analysis from Trinity College, Dublin.

Part I Basic C/C++ Language and Syntax

In this block we introduce the fundamentals of programming in C and C++. Since we assume no prior programming knowledge on the part of the student we discuss all of the issues involved in learning a new programming language. In particular, we focus on the syntax of the C++ language and ensuring that you feel comfortable with the compiler, how to write and debug code and to prepare you for the object-oriented features in C++ that we discuss in later sections. Some of the syntax in Part I will be replaced by somewhat more programmer-friendly and modern syntax in later sections. We discuss it here for completeness because existing software systems use it and it is thus important to know.
Having successfully completed this part, you will have acquired a good knowledge of C/C++ syntax, the compiler and its environment and you will be prepared for the next part, namely object-oriented programming in C++.
The quizzes and exercises pertain to generic and numeric examples and they test your knowledge of fundamental data types and control structures.

The C++ Environment

- History of C++

- What Makes a C++ Program
- Steps for a C++ Program
- Compiling a C++ Program
- Structure of a C++ Source
- Example Program
- Placing Comments
- Lay-Out Source File
- The Visual Studio IDE explained
- C++ for computational finance: an overview

Data types

- Fundamental Data Types
- Type Magnitudes
- Type Specifiers and Qualifiers
- Constants

Variables, Operators and Expressions

- Variables
- Initialising Variables
- Concatenating Declarations
- Arithmetic Operators
- Special Cases
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement
- Conditional Operator
- The Comma Operator
- Type Conversion The sizeof Operator
- Bitwise Operators
- Operator Precedence

Decisions and Loops

- Blocks and Statements
- If Else
- Switch
- For
- While
- Do While
- Break and Continue
- Goto and Labels

Functions and Storage Classes

- Function Signatures
- Function Calls
- Functions and Return
- Function Arguments
- Recursive Functions

- Scope of variables
- Automatic Variables
- Global Variables
- External Variables
- Static Global Variables
- Functions and Static Variables
- Register Variables

The Preprocessor

- Introduction to Directives
- Include Files
- Macros
- Conditional Compilation
- Header Files and Directives

Pointers and Arrays

- Pointers
- Call by Reference
- Arrays
- Initialising Arrays
- Strings
- Using Strings
- Arrays and Pointers
- Pointer Arithmetic
- Multidimensional Arrays
- Function pointers and their applications

Data Aggregates

- Introduction to Structures
- Using Structures
- Arrays of Structures
- Structures and Pointers
- Passing Structures
- Unions
- Typedef

Part II Object-Oriented Programming (OOP) in C++

In this part of the course we introduce the object-oriented programming model and how to implement it in C++. We focus on motivating on how to 'thinking objects' while at the same time avoiding going overboard by creating over-engineered classes and class hierarchies. Since C++ is a huge language it is often difficult for beginners to know where to begin and for this reason we concentrate on essential and high-priority topics which when learned will allow you to write non-trivial C++ code in a short period of time. The approach taken is incremental and this part of the course contains some of the most important C++ features that you will use when developing your applications in computational finance, for example public and private class members, use of the const keyword, operator overloading and efficient memory allocation.

Having successfully completed this part, you will have acquired a good knowledge of the C++ language that is needed to create efficient and robust classes. In the next part of the course we shall incrementally build on the methods here in order to create reusable and flexible class hierarchies.
The quizzes and exercises pertain to generic, numeric and financial examples and they test your knowledge of intermediate-level functionality relating to the creation of classes and objects in C++.

OOP Thinking and Modeling

- Classes and Objects; what are they?
- Discovering classes
- Encapsulation and implementation hiding
- Class relationships: aggregation and composition
- Class relationships: inheritance
- A little UML (Unified Modeling Language)
- Advantages and disadvantages of OOP
- OOP in combination with other programming paradigms

The Class Concept

- Abstract Data Types
- Classes and Objects
- C++ members
- Constructors and destructors
- The C++ header file
- Access specifiers
- The C++ source file
- Creating objects

Improving your Classes

- Function name overloading
- Call by reference vs. call by value
- The 'const' specifier
- The copy constructor

Basic Operator Overloading

- Introduction to operator overloading
- Binary operators
- Unary operators
- Assignment operators
- The 'this' pointer
- Returning references
- The Essential Header File

Memory Management, Fundamentals

- Stack, heap and static memory
- Who takes care of memory?
- The *new* and *delete* operators

- Dynamic arrays
- Classes with dynamic memory
- Smart pointers

Namespaces

- Why using namespaces
- Using classes in a namespace
- Placing classes in a namespace
- Nested namespaces
- Aliases
- Name lookups

Part III Inheritance and Polymorphism

In this part of the course we discuss a number of advanced features that allow us to justify the remark that C++ is an object-oriented language. In particular, we discuss how C++ realizes the Generalization/Specialization (ISA) relationship by its support for the inheritance mechanism. We consider both implementation and interface inheritance, how to implement them in C++ as well as the impact they have on the quality of your software.
Having successfully completed this part, you will have acquired a good knowledge of the inheritance mechanism in C++ in order create efficient and robust class hierarchies
The quizzes and exercises pertain to generic and financial examples and they test your knowledge of advanced-level functionality relating to the creation of classes, class hierarchies and applications in C++.

Object-Oriented Modeling

- Gen/Spec (ISA) relationship
- The Principle of Substitutability
- When to use ISA and when to use HASA
- Combining ISA and HASA
- Good and bad class hierarchies
- A little UML (Unified Modeling Language)

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

Exception Handling

- Error handling and exceptions
- Throwing and catching exceptions
- Exception hierarchy
- Polymorphic and explicit nets
- Order of handlers

Final Remarks and Guidelines

- Semantic correctness of class hierarchies
- Efficiency and maintainability issues
- Too much inheritance considered harmful
- What to use instead of multiple inheritance

Part IV Generic Programming in C++ and Standard Template Library (STL)

In this part we introduce the generic programming model and its realization by C++ template classes and template functions. The GP model is probably not as well known as the OOP model but it is pervasive in C++ and in a sense more important than OOP as can be seen in libraries such as STL (Standard Template Library) and Boost. We can combine the OOP and GP models and they can also be seen as competitors for solving a given problem.
It is possible to create template functions and classes based on the skills that you learned in Part III of the course. We do consider creating our own templates but the main focus is on using ready-made template data containers and algorithms.
Having successfully completed this part, you will have acquired a good knowledge of templates in C++ in order to create compile-time efficient and robust generic classes and functions.
The quizzes and exercises pertain to generic and financial examples and they test your knowledge of template classes and algorithms. In particular, eliminating and resolving compiler errors is an important skill you need to learn.

An Introduction to Generic Programming (GP)

- An introduction to generic programming
- Abstract data types and algorithms
- Generic polymorphism
- Composition and inheritance in GP
- Combining OOP and GP
- GP and modular programming

Template Classes in C++

- An introduction to C++ templates
- Parameterization
- Template classes and function templates
- Creating templates (source & header files)
- Type template parameter & value template parameter

- Operations on the generic argument
- Explicit template instantiation

An Introduction to STL

- The 6 STL component Categories
- Sequential data containers
- Iterators
- Algorithms
- Function objects
- Using STL in your applications

STL Containers and Algorithms

- An introduction to Complexity Analysis
- Complex numbers in STL
- Vectors and lists
- Maps and sets
- Modifying and nonmodifying algorithms
- Numeric algorithms

Combining OOP and GP

- GP with inheritance and composition
- Template specialization
- Integration with design patterns
- Policy-based design and software contracts
- Dynamic and parametric polymorphism
- Efficiency and flexibility

Part V An Introduction to Boost C++ Libraries

For completeness and a wish to discuss ready-made C++ functionality in computational finance applications we have decided to introduce a number of relevant libraries from the Boost suite of libraries. In particular, we focus on those libraries that we directly use in applications in Part VI of the course. The use of these libraries is similar to how libraries in Matlab and Fortran are employed in applications.

We focus on libraries that help us solve linear matrix systems, compute random numbers, special mathematical functions and statistical distributions.
Having successfully completed this part, you will have acquired a basic knowledge of how to use the Boost library.
There are no exercises in this Part V because they are incorporated into the exercises of Part VI.

An Introduction to Boost C++ Libraries

- What is Boost? Overview
- Matrix and Vectors: uBLAS library
- Continuous and Discrete Statistical Distributions
- Random number generation
- Special function in Math Toolkit

- Solution of nonlinear equations

Some test code and examples can be found on www.datasimfinancial.com

Part VI Creating Applications in Computational Finance

Having completed the first five parts of the course we are now ready to start developing some applications for computational finance. We take a number of well-known and well-documented examples from the financial literature. The software *process* in this part of the course is to take a financial model, set up the numerical model and then implement it in C++. We take an incremental approach "*get it working, then get it right, then get it optimized*" (in that order and if time permits.) You will be expected to make use of STL and Boost libraries to speed up productivity and to produce robust code.
Having successfully completed this part, you will have acquired a good knowledge of how to design and implement simple applications in computational finance as well as learning how to approach and solve software problems.

Applications and Test Cases

- Black Scholes equation: analytic solution
- A simple binomial method in C++
- A one-factor Monte Carlo simulator
- Using C++ numerical analysis libraries
- Software design essentials

Test Case 1 Monte Carlo Method

- Choose software framework
- Random number generators
- SDEs and Finite Difference approximations
- Performance and accuracy
- Some 2-factor problems

Test Case 2 Finite Difference Method

- One-factor plain options
- Euler, Crank-Nicolson methods
- The ADE (Alternating Direction Explicit) method
- Early exercise features
- FDM for the CIR pde
- Comparison of different methods

Test Case 3 Lattice Methods

- The one-factor binomial and trinomial methods
- Two-factor binomial method
- Early exercise
- Comparison with FD schemes

Test Case 4 Exact and quasi-Exact Methods

- Exact Black Scholes price and greeks

- Barone-Whaley
- Noncentral chiSquared distribution
- Bonds, swaps and swaptions