Advanced C++ and Modern Design

Originator: Dr. Daniel J. Duffy, Datasim Education BV
Amsterdam

Objectives

The goal of this hands-on course is to introduce modern C++11 to C++20 and apply it to the design and implementation of applications in a number of domains. We describe the new syntax and libraries in detail and we show how to use these new features with the well-established Gamma and parallel designpatterns. We then apply this knowledge to designing applications in computational finance, mathematics, engineering and computer graphics. In this way we hope show the power and usefulness of C++ and as industry standard for software developers.

For whom is this Course and what is Prerequisite Knowledge?

We have developed this course for a range of software developers who use (or have used) C++ todevelop applications, specifically for those developers who are in any of the following cExegories:

- Have intermediate or advanced C++98 knowledge and who wish to learn C++.
- Have completed the Baruch/Quantnet *C++ Programming for Financial Engineering* course.
- Experienced C++ developers who wish to learn and apply design techniques in a multiparadigm programming environment.
- In general, this course is for C++ developers who wish to experience the full power of C++ forapplication development.

Course Structure and Process

This course uses a structured approach to learning C++ and using it in applications. The approach taken integrates advanced multiparadigm language features, software engineering and design principles to implement real-world applications in a number of technical domains. The modules are self-contained and logically related to previous modules. In this way we can progress from topic to topic in an orderly fashion. Each module consists of between seven and ten sections. Each section has an average length of one hour and it discusses a given topic in detail. In order to reinforce and consolidate the learning experience we have produced the following products:

- A set of quizzes to test how well you have learned the topic.
- Sample code to show how a particular technique works.
- Hands-on programming exercises that test how to apply the techniques.

In this way you can assess your progress before moving on to the next module.

Contents Overview

The main goal of this course is to apply modern C++ language features and to integrate them with design and system development methods to build software systems. In order to achieve this goal we have created six modules, each one dealing with one aspect of C++. The first three modules constitute a full treatment of C++ and is state-of-art. The focus in these modules is on learning what C++ offers in terms of new functionality and libraries. The last three modules integrate and apply C++ to Boost C++ libraries, patterns and system engineering.

The contents (audios, quizzes and exercises) are elaborated in an incremental fashion (both inter-module and intra-module) in order to make the learning process as seamless as possible.

1. C++11 and beyond essentials

In this module (7 sections) we introduce some of the most important and far-reaching functionality that C++11 offers. In particular, we discuss the multi-paradigm approach to modelling functions. We also introduce functionality that improves the robustness and efficiency of C++ code.

Lambda functions' essential functionality Comparing lambda functions with function 0bjects
Improving classes in C++11 (avoiding generation of special functions, move semantics, `explicit` constructors, `no except`) Variadic' fundamentals and polymorphic behaviorUniversal function wrappers and C++11 Bind
Function wrappers with Inheritance and CompositionTuples A-Z
Sealed classes
Other class improvements (alias template, explicit override)

C++17/20 Language Features I

Improving code quality (e.g. readability, reduce code bloat, reliability)
Auto deduction of correct variable by compiler (no human intervention)
Structured bindings to fixed arrays, tuples and member data
Class template argument deduction (CTAD) in initializers

C++17/20: Some Data Types

`std::variant`
Type-safe visitation on variants
Applications

2. Advanced C++ Features

In this module (8 sections) we introduce advanced syntax and functionality in C++11 and C++14. The main focus is on functionality that promotes the reliability and robustness of C++code, for example dealing with numeric overflow/underflow, round-off error and safe pointers; we also discuss advanced template programming such as variadic types, type traitsand platform-independent and platform-dependent error codes.

Introduction to type traits and *Template Metaprogramming* (TMP) Advanced type traits (for example, creating type-independent code)
Advanced lambda programming (for example, *init captures* and functional programming in C++)
Smart pointers in Boost and C++11
IEEE 754 and C++ Floating Point Classification
Platform-dependent and platform-independent error codes and error conditions
STL Bitset, Boost dynamic Bitset and mini-applications
STL Function Objects and Lambdas
Veridic Template Programming

Multiparadigm Programming in C++

C++17/20 Language Features II:

Auto deduction rules for braced initialisation lists
The seven ways to initialise variables
`std::byte`
Three-way (spaceship) operator
Template parameter lists for lambda functions

C++17/20 Language Features III

Review of C++ parameter packs
Aggregation initialisation
Fold expressions for recursive data structures; applications
Concepts for the impatient

C++17/20 Language Features IV

`std::string_view`
String literals as template parameters
`std::optional`
`std::span`

3. C++ Libraries

In this module we discuss how C++ supports multi-threading, concurrencyand parallel programming. Prior to C++11 there was no direct support for multithreading inthe language and it was necessary to use either Boost Thread or some proprietary library. We discuss the details of thread programming as well as tasks, futures and promises.

Essentials of Boost and C++11 threading

Advanced C++ Concurrency
C++ Tasks and tasking applications

Extensions for Parallel STL

ISO/IEC TS 19570:2015 standard
Making STL algorithms work in parallel mode
Execution policy; kinds of parallelism.

4. STL, Data Structures, and Random Numbers

In this module we discuss introduce new data structures in C++11 and advanced treatment of STL algorithms. Furthermore, we discuss how to generate random numbers natively in C++11.

New data structures and data types
STL algorithms in a multi-paradigm programming environment
Random numbers and statistical distributions

5. Boost C++ Libraries

In this module (9 sections) we give a thorough overview of the Boost C++ libraries in order to acquaint the student with their potential and functionality. A number of the Boost libraries are also in C++11 but many have not been ported and it is for this reason that it is worth investigating what

Boost has to offer. Furthermore, the libraries are well-documentedin the main and the online documentation is useful. We do recommend the books “Introduction to the Boost C++ Libraries” (Volumes I and II) with full source code by Robert Demming and Daniel J. Duffy (contact info@datasim.nl for information).

Boost Libraries Overview String Algorithm Library Regular expressions and RegexHash and Unordered
BitmapHeap
Matrix Libraries (Boost bullas) Signals and Slots (Boost signals2)

6. System and Design Patterns

In this module (8 sections) we discuss the popular design and system patterns that are based on the object-orient model. We concentrate on approximately 20% of the patterns that account for 80% of the effectiveness in software development. We also show how to create *next-generation patterns* using the multiparadigm programming models that C++11 supports. The topics from Module 5 can be seen as the building blocks as input to Module 6.

Class and Component diagrams in UMLWhole-Part Pattern
Object-Oriented MetricsCreational patterns Structural patterns Behavioral Patterns

Next Generation Design PatternsExamples and applications

7. Modern Design

In this module (6 sections) we present a *defined process* (based on a combination of *Structured Analysis*, System patterns and C++11) to architecting, design and implementationof complex software systems. This approach allows us to create software systems as a sequence of working prototypes in C++.

Locating and bounding the Software System Decomposition
Presentation Abstraction Control model *Policy-Based Design* (PBD) in C++ Example and applications
Principles of Parallel Programming and Libraries
Option: Design using C++20 (Concepts, Modules, parallel).

C++ Concepts Motivation and Overview

Introduction to C++20 Concepts library
Syntactical and semantical elements
Concepts as protocols and modelling constructs
Provides-requires interfaces
“Rules of engagement” between client and server

Concepts: Nuts and Bolts

Keywords: concept and requires
Core language concepts
Concepts and type traits
Conjunction, disjunction and composite concepts

Concepts in Applications

- Concepts as central element in a new design philosophy
- System decomposition into components
- Designing components using concepts (“Concepts-based design (CBD)”)
- Moving from Policy-based design to CBD

Modules: Motivation

- What is a module? The “componentisation” of C++
- Logical division of C++ code
- Keywords `export` and `import`
- Migrating legacy code to modules