# C++ fundamentals course with use cases from finance

## Introduction to the Course

The objective of the course is to teach students fundamentals of C++. The course does not assume any previous knowledge of C++, however, basic level of coding in any programming language is required. After completing of the course the students will be able to code simple applications in C++ understand the reasons for the errors and understand the concepts of C++ language. The course does not cover the design patterns of the advanced programming. The course’s focus is neither on generic algorithm nor on generic data structures. The course introduces the student to the Standard Library in C++ where the algorithms and data structures are implemented.

The instructor is Ivan Zhdankin - a quantitative researcher with experience in diverse areas of quantitative finance, including risk modelling, XVA, and electronic trading across asset classes, including commodity futures and G10 and emerging market currencies. Ivan was consulting various banks in quantitative modelling and has joined JP Morgan as a quantitative analyst. Ivan has a mathematical background from New Economic School and Moscow State University, where he studied under the celebrated Albert Shiryaev, one of the developers of modern probability theory.

## Course structure

* C++ Introduction (3h teaching and 1h tutorial)
* Introduction to OOP in C++ (3h teaching and 1h tutorial)
* Defining your own structures in C++ (3h teaching and 1h tutorial)
* Introduction to Standard Library and improving your code in C++ (3h teaching and 1h tutorial)

## Content

* **C++ introduction:**
  + Getting Started with C++
    - Why C++?
    - Not too hard for beginners
    - Standardization
    - The Standard Library
  + Tools
    - What do we need?
    - Building
    - Visual Studio
    - Application Structure
    - Creating Project
    - Errors
    - Stream I/O
  + Variable and Fundamental Types
    - Variables
    - Fundamental Types
    - Auto
    - Casting
  + Functions and Headers - Introduction
    - Functions
    - Overloads
    - Header Files
    - Errors
  + Flow of Control
    - Flow of Control
    - If, While, For
    - Switch, Immediate If
  + Tutorial
    - Stream I/O: User input is tenor and the corresponding value on the interest rate curve. The task is to output on the screen the tenor and its value.
    - Write a function that take 2 tenors on interest rate curve, the corresponding values on the curve, and another tenor to interpolate.
    - The function should return linear interpolation for the tenor.
    - Write function overloading in case we input just one tenor for interpolation (extrapolation)
    - Write the declaration of the function in a header file (.h) and implementation in a separate file (.cpp)
* **Introduction to OOP in C++:**
  + Writing Classes
    - Design
    - Coding Class
    - Some standard classes (collections): string, vector
    - Organising Files
    - Inline Functions
    - Encapsulation
    - Creating Instances – objects
  + User defined Types
    - Building
    - Constructors
    - Scope
    - Struct
    - Namespaces
    - Inheritance
    - Enumerations
    - The Preprocessor
  + Functions – Continue
    - Free Functions
    - Member Functions
  + Tutorial
    - For this tutorial organise different classes in different files with declarations in different header files
    - Define a base class “Curve” which contains a vector of tenors and a vector of curve values. The curve also has a name in a string variable.
    - Define some member functions of the curve class: setname and getname, add tenor with a value
    - Define a member function that takes a tenor as a parameter and return the largest tenor which is less than a given tenor
    - Define a member function that takes a tenor as a parameter and return the value corresponding to largest tenor which is less than a given tenor
    - Define a function that takes a tenor as a parameter and returns the smallest tenor which is larger than a given tenor
    - Define a member function that takes a tenor as a parameter and return the value corresponding to smallest tenor which is larger than a given tenor
    - Define a derived class Linear\_Curve
    - The derived class should have a member functions that take a tenor and return a linearly interpolated tenor or flat extrapolation
* **Defining your own structures in C++**
  + Operators
    - Arithmetic
    - Comparison
    - Bitwise
    - Operator Overloading
    - Writing Overload
  + Templates
    - Templates
    - Template Functions
    - Template Classes
    - Template Specialization
  + Indirections
    - Pointers and References
    - Const
    - Const with indirection
  + Memory
    - The Free Store
    - Manual Memory Management
    - Easy Memory Management
    - Smart Pointers
  + Tutorial
    - Create operator overloading that sums up two curves: the output is the curve where the tenors are computed as sum of the individual tenors.
    - Define template class “curve” that can have different types of the tenors (int, double). Create different instances of the class to validate your result.
    - Create template free function which take the class “curve” by reference and return the average value of the tenors.
* **Improving your code in C++ and Introduction to Standard Library**
  + Indirection and Inheritance
    - References and Inheritance
    - Virtual Function
    - Pointers and Inheritance
    - Slicing problem
    - Casting and Indirection
  + The Standard Library
    - What is the Standard Library?
    - Map
    - Collections
    - Sorting and Searching
  + Lambda
    - What is Lambda?
    - Lambda functions
    - Lambda returning value
    - Capturing
  + Exception handling
    - Errors
    - Exceptions
    - Try and Catch
  + Tutorial
    - Construct several classes: Bank Account, Business Account, Saving Account, Saving Account:
      * Bank Account is a base class that has variables such as id, first and last name of the person who has access, current balance. Bank account has such functions as withdraw some balance, calculate withdrawal fees.
      * Business Account and Saving Account are derived from the Bank account class. Implement different function that calculates withdrawal fees.
    - For Bank account implement a map that contain transactions as pairs of integer and amount
    - Implement function that returns the largest transaction in terms of the absolute value
    - Implement function that returns sum of transactions with positive amount using for\_each and lambda function
    - Implement a function that asks the user by how much the account should be debited. Use try and catch to capture invalid type of the input variable