IB9N7 C++ for Quantitative Finance
Worksheet 8

Introduction to Object Oriented Programming

30 November 2015
(Week 9)

Keep records of your solutions and show them to us when you have finished.
To help us judge the timing, please let us know how long it takes you to do the exercises.
If you are unable to finish the exercises during the lab session, please finish them before next time.

Objectives for this lab session

By the end of this session, you should have completed the following:

- Understand the program that demonstrates the `Date` class!
- Add/think about appropriate additional functionality for that class.
- Create a `Point` class that has appropriate member variables and functions with the appropriate access restrictions and validation.

Exercises

Exercise 1: Understanding

(a) Download, extract, compile and run the (complete version of the) `Date` example from the lecture `08_date.zip`. (It would be confusing for dev-cpp to try to open the files without extracting them.)

(b) Try to understand it. Focus on one member function at a time. Play with the debugger if you like.

(c) (i) What would happen if you tried to access the private members of the `Date` class from the `date_test_main` function?
e.g. `test.day_`

(ii) What would happen if you tried to access the non-static members of the `Date` class from a static context?
e.g. `Date::output_shortform()` from `date_test_main` or `output_shortform()` from `is_leap_year`.

(iii) What would happen if you tried to access the static members of the `Date` class from a non-static context?
e.g. `test.is_valid_date(1, 1, 2014)` from `date_test_main`

(d) (i) Write a free function that takes a `Date` as an argument and returns a `Date` corresponding to the following day.

(ii) Write a member function that does the same thing.
This should not modify the current Date.

(iii) Now write a member function that advances the Date of the current object instance by one day.

(iv) Think about what this kind of function (one returning the day following a given date) might look like if you could not use object-orientation at all.

(e) The `Date` class is currently very basic and there are lots of extra things that it could (and in some cases, should) be extended to do.

Think of some extra functionality that you might want the Date class to provide, and what additional members or non-members you might need to provide that functionality.

If you feel that you have the time and the necessary knowledge, try to provide some (but not necessarily all) of that functionality.

Exercise 2: Creating a new class

(This exercise may take some time but will help you to understand classes more. However, the class test will not involve you having to create your own classes — but the project might!)

(a) Design a class called `Point` to represent a point in a 2-dimensional Euclidean space. Do not start to implement any of the members until you have finished specifying all the entities to be part of the class.

You should think about:

- What data it needs.
- What accessors and mutators to provide.
- What validation needs to be performed.
- What methods it might be useful to have.
- Whether there is any need for any static methods.

Hints:

- A point can be specified in a number of co-ordinate systems. For example, the Cartesian co-ordinates $(x = 0, y = 1)$ and the polar co-ordinates $(r = 1, \theta = \pi/2)$ are two ways of representing the same point.
- C++ has a function `std::atan2(y, x)` for the principal value of the arc tangent of `y/x`, expressed in radians.
- Are there any co-ordinates in either representation that should be rejected (e.g. they represent invalid points)?
- Are representations in each form unique? If not, what are the implications for your class?
- One useful method might be to calculate the distance between the point and some other point.

(b) Implement and test your class!

(c) How hard would it be to extend the functionality to points in 3-dimensional (or *n*-dimensional) space? (You don't necessarily have to try to do it.)

Exercise 3: Linear interpolation example (extra exercise)

Another example of a class is provided in 09_Interp.zip. Note that the class has data members which are declared `const`, this means that they can only be set in the initialisation list in the constructor, and no member function can modify them.

(a) Look through the class and understand what it does.

(b) Data members of a class (like `x_` and `y_` here) can be references. Consider changing the line

```
const std::vector<double> x_, y_;
```

to

```
const std::vector<double> &x_, &y_;//to make two references, need two &s
```

or equivalently

```
const std::vector<double>& x_;
const std::vector<double>& y_;
```

. This would mean that the class holds a reference to the vectors which its constructor is given, rather than copies. What are the advantages and disadvantages of doing this?

(c) Implement the `get_many` function.

Remember to talk through your answers with the lab assistants!