IB9N7 C++ for Quantitative Finance
Worksheet 10

Operator overloading
Hints and solutions

21 January 2016
(Week 16)

Objectives for this lab session

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

- Understand and experiment with operator overloading

Exercises

Exercise 1: Stream insertion and others

(a) Return to your `Point` class, and add any operators that you feel should be provided.

(Hint: some of these are provided in the lectures.)

Check that they work.

See `point_with_operators.zip` on my.wbs.

(b) Going back to the `TimeSeries` class, add an overload of the stream insertion operator as a member function (you can just invoke your `output` function, but note that the stream is not necessarily `std::cout` now).

You might change your `output` signature to have the following overloads:

```
void output() const;
void output(std::ostream & o) const;
```

...with the following implementations:

```
void TimeSeries::output() const
{
    ////////////////////////////////////////////////////////////////////
    /// Your implementation for question 1c
    output(std::cout);
    ////////////////////////////////////////////////////////////////////
}

void TimeSeries::output(std::ostream & o) const
{
    ////////////////////////////////////////////////////////////////////
    /// The rest of your implementation for question 1c
    for (std::vector<double>::size_type i = 0u; i != Values_.size(); ++i)
    {
        o << Values_.at(i) << std::endl;
    }
    ////////////////////////////////////////////////////////////////////
}
```

In TimeSeries.hpp, you should also declare the following outside (after) the class specifier:

```
std::ostream & operator<<(std::ostream &, const TimeSeries &);
```

Its implementation within TimeSeries.cpp might look like this:

```
std::ostream & operator<<(std::ostream & o, const TimeSeries & ts)
{
    ts.output(o);
    return o;
}
```

Similarly, write an overload of the stream insertion operator that takes a std::vector argument. Think about the best place to define this. This is not specific to the TimeSeries class so deserves its own translation unit that can be reused between different projects.

E.g.: vector_stream.hpp

```
#include<vector>
#include<iostream>
std::istream& operator>> (std::istream& s, std::vector<double>& v);

```

...and vector_stream.cpp

```
#include<vector_stream.hpp>
std::istream& operator>> (std::istream& s, std::vector<double>& v){
    double d;
    while(s>>d){
        v.push_back(d);
    }
    return s;
}

```

Exercise 2: Complex number class

Can you define a class for complex numbers? (Hint: complex numbers are analogous to points, but with additional possible operations. E.g., while it does not make sense to compare points with scalars, one can have equality between complex numbers and real numbers.)

N.B. the standard template library already defines a (template) std::Complex<T> class, but the purpose of this exercise is to try to write your own (simpler) Complex class to improve your understanding of operators.

Exercise 3: Dates

Extend your Date class so that you can:

(a) Use the stream insertion operator.

(b) Add a scalar to a date.

(c) Subtract a scalar from a date.

(d) Subtract two dates.

At least think about the method prototypes even if you struggle with the implementations.