IB9N7 C++ for Quantitative Finance
Worksheet 12

Dynamic polymorphism

03 February 2016
(Week 18)

Objectives for this lab session

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

- Work through, modify and extend some examples of virtual inheritance.

Exercises

Exercise 1: 'Simple' examples of virtual inheritance

Consider the following classes and `main` function.
Explain the output it produces.

```
#include <iostream>
#include <cstdlib>

class A
{
    public :
        void f () const {std::cout << "A::f" << std::endl;}
        void g () const {std::cout << "A::g" << std::endl;}
        virtual void h () const {std::cout << "A::h" << std::endl;}
};

class B : public A
{
    public :
        void h () const {std::cout << "B::h" << std::endl;}
    protected :
        void f () const {std::cout << "B::f" << std::endl;}
    private :
        virtual void g () const {std::cout << "B::g" << std::endl;}
};

class C : public B
{
    public :
        void f () const {g();}
        void g () const {h();}
};

void invoke_all(const A & a)
{
    a.f();
    a.g();
    a.h();
}

```

```cpp
int main ()
{
    ////////////////////////////////////////////////////////////////////////
    // Explain the output of each of the following member function calls
    ////////////////////////////////////////////////////////////////////////

    for (int i = 0; i != 78; ++i) std::cout << "-"; std::cout << std::endl;

    A a;
    invoke_all(a);

    for (int i = 0; i != 78; ++i) std::cout << "-"; std::cout << std::endl;

    B b;
    invoke_all(b);

    for (int i = 0; i != 78; ++i) std::cout << "-"; std::cout << std::endl;

    C c;
    invoke_all(c);

    for (int i = 0; i != 78; ++i) std::cout << "-"; std::cout << std::endl;

    return EXIT_SUCCESS;
}
```

Exercise 2: Numerical integration

(a) Run the example from the lecture (you may just use one cpp file for everything if you find it easier). You can download the file from the course page.

(b) Try to understand the numerical integrations that the code is performing.

(c) Try it without the keyword `virtual` in the base class `segment` function.

(d) Implement the midpoint quadrature rule in a new class.

Hint: for the grid interval $[a, b]$, the midpoint rule uses the approximation

$$\int_a^b f \approx (b-a) f\left(\frac{a+b}{2}\right).$$

(e) Modify the `NIntegrate` class so that it is abstract (by making the `segment` function pure virtual), and provide the trapezium rule functionality instead in a separate derived class.

(f) Currently the code only works for the provided function `f`, and so is not sufficiently general.

Define a pure abstract base class `Integrand` to replace the function `f` in the example. You may use any interface you wish (possibly but not necessarily overloading the parenthesis operator).

Exercise 3: Equity Monte Carlo

(a) Download and extract 12_EquityMC.zip, and open the project. This is a simple monte carlo pricer for european equity options. Have a look around and try to understand how it works. Ask for help. (There is some mathematical code here, in the EuropeanOption class the Black Scholes formula is implemented to price European options analytically, and in the Pricer class the Euler step scheme is implemented. I think you have learnt about this in other courses. It is just included to provide a realistic example. Don't focus on it here.)

(b) Implement a basket of options: create a new class `BasketOfOptions` which owns several `EuropeanOption`s which all expire at the same time, and which implements the Payoff interface. Its payoff should be the sum of the payoffs of the options it contains. Do not worry about an analytic price for it.

(c) Implement a basket Instrument: create a new class `Basket` which owns several `Payoff`'s which all expire at the same time, and which implements the Payoff interface. Its payoff should be the sum of the payoffs of the options it contains. Note that you will have to add a virtual destructor to `Payoff` to make this work.