IB9N7 C++ for Quantitative Finance
Worksheet 13

Small objects
Hints and solutions

11 February 2016
(Week 19)

Objectives for this lab session

By the end of this session, you should have played with some more objects

Exercises

Exercise 1: Colours

(a) Create an enumeration `Colour` of colours red, green and blue.

(b) Define a function

```
void printBlueLovers(const std::vector<std::tuple<std::string,int,Colour>>&
    people);
```

which takes a list of people's name, age and favourite colour, and prints the name of all those whose favourite colour is blue, in ascending order of age. (Do this by copying data to an appropriate second vector and sorting that.) The simplest solution is to use a tuple with the order chosen to match what we want. The whole program (including a test function) would look something like this.

```
#include<algorithm>
#include<string>
#include<iostream>
#include<tuple>
#include<vector>

using std::get;
using std::make_tuple;
using std::string;
using std::tuple;
using std::vector;

enum class Colour {RED, GREEN, BLUE};

void printBlueLovers(const std::vector<std::tuple<std::string,int,Colour>>&
    people){
    vector<tuple<int,string>> agesAndNames;
    for(auto& p : people){
        if (Colour::BLUE == get<2>(p)){
            agesAndNames.push_back(make_tuple(get<1>(p),get<0>(p)));
        }
    }
    std::sort(agesAndNames.begin(),agesAndNames.end());
    for(auto& p : agesAndNames){
```

```
        std::cout << get<1>(p) << "\n";
    }
}

int main(){
    vector<tuple<string,int,Colour>> people;
    people.push_back(make_tuple("Alpha",23,Colour::RED));
    people.push_back(make_tuple("Beta",23,Colour::GREEN));
    people.push_back(make_tuple("Gamma",23,Colour::BLUE));
    people.push_back(make_tuple("Delta",24,Colour::BLUE));
    people.push_back(make_tuple("Epsilon",23,Colour::GREEN));
    people.push_back(make_tuple("Zeta",22,Colour::GREEN));
    people.push_back(make_tuple("Eta",22,Colour::BLUE));
    people.push_back(make_tuple("Theta",22,Colour::BLUE));
    people.push_back(make_tuple("Iota",22,Colour::BLUE));
    people.push_back(make_tuple("Kappa",25,Colour::BLUE));
    printBlueLovers(people);
}
```

This illustrates how convenient tuples can be for some common tasks. It is slightly unsatisfactory that the string objects are copied into `agesAndNames` - we could perhaps be more efficient by keeping pointers to the original strings, but the code would be more complicated.

We could alternatively keep the structure of the data and define our own comparison function to use in the sorting. The implementation might look like this.

```
bool ageLessThan(const tuple<string,int,Colour>& a, const tuple<string,int,Colour
    >& b){
    return get<1>(a) < get<1>(b);
}

void printBlueLovers(const std::vector<std::tuple<std::string,int,Colour>>&
    people){
    std::vector<std::tuple<std::string,int,Colour>> people2;
    for(auto& p : people){
        if (Colour::BLUE == get<2>(p)){
            people2.push_back(p);
        }
    }
    std::sort(people2.begin(),people2.end(),ageLessThan);
    for(auto& p : people2){
        std::cout << get<0>(p) << "\n";
    }
}
```

Notice that our comparison function takes constant reference parameters. If we miss the `const`, the compiler error messages are very long! The order in which people with the same age will appear here is unspecified behaviour - the `sort` function makes no guarantees about the final order of elements neither of which is less than the other. An alternative, `stable_sort` works like `sort` but guarantees to leave such elements in the original order, which means it can be slower.

(c) Illustrate calling this function. This is shown in the main function above.

(d) Modify the function, so that if two people have the same age, their names are printed in alphabetical order. If we use the version with `tuple` above, then this already works and there is nothing to do here. If we used `ageLessThan`, then we can modify it to sort equal-aged people by name.

```
bool ageLessThan(const tuple<string,int,Colour>& a, const tuple<string,int,Colour
    >& b){
    return get<1>(a) < get<1>(b) ||
        ( get<1>(a)==get<1>(b) && get<0>(a) < get<0>(b) );
}
```

There is another way to do this, using the function `stable_sort` mentioned above, as follows.

```
bool ageLessThan(const tuple<string,int,Colour>& a, const tuple<string,int,Colour>& b){
    return get<1>(a) < get<1>(b);
}

void printBlueLovers(const std::vector<std::tuple<std::string,int,Colour>>& people){
    std::vector<std::tuple<std::string,int,Colour>> people2;
    for(auto& p : people){
        if (Colour::BLUE == get<2>(p)){
            people2.push_back(p);
        }
    }
    std::sort(people2.begin(),people2.end());
    std::stable_sort(people2.begin(),people2.end(),ageLessThan);
    for(auto& p : people2){
        std::cout << get<0>(p) << "\n";
    }
}
```

Exercise 2: array

What are some advantages and disadvantages of `array` over `vector`? `array` is only usable when the size is known at compile time. The fact the size and layout are known at compile time can make it easier for the compiler to do optimizations. It constructs all its members when it is constructed. It does not need to allocate a separate memory block to store its members - they are members; this is unlike `vector` whose elements live elsewhere. Members of an `array` will not get moved around in memory during the lifetime of the `array`.

If an object has no constructors, (so that default-initialization and value-initialization may be different) then it is possible to construct an `array` of uninitialised objects. Consider:

```
vector<int> v(10);  //this starts as 10 zeros
array<int,10> v{};  //this starts as 10 zeros
array<int,10> v;  //this starts as 10 uninitialised values, unachievable with vector
```

Exercise 3: Yield Curve Bootstrapping

(a) Download and extract 13_YieldCurve.zip, and open the project. This is a simplified Yield Curve Bootstrapper, a procedure for finding approximately realistic discount factors from the market. Have a look around and try to understand how it works. There may be some finance background which you have not seen. Ask for help.

(b) Consider how you would allow the use of the `Curve` object to be used in a class which implemented the `MarketData` interface from the EquityMC project.