Module IN3013/INM173 – Object Oriented Programming in C++

Solutions to Exercise Sheet 7

1. (a) Here are some simple classes (we sloppily make the field public to keep things simple):

```
class A { public: int x; };
class B : public A {};
class C : public A {};
class D : public B, public C {
public:
        void setB(int n) { B::x = n; }
        void setC(int n) { C::x = n; }
        int getB() const { return B::x; }
        int getC() const { return C::x; }
};
```

Now is we create an object of class `D`, we can check that it contains two copies of the `A` part:

```
D d;
d.setC(1);
d.setB(2);
cout << d.getC() << '\n';
```

The answer is `1`: setting the copy accessed through `B` does not affect the copy accessed through `C`.

(b) We now change the declarations of `B` and `C` to

```
class B : public virtual A {};
class C : public virtual A {};
```

Now if we repeat the above test, we get `2`, because the `B` and `C` parts share a single copy of `A`.

(c) The idea is to add constructors like

```
class B {
public:
        B() { cout << "initializing B\n"; }
```

(d) same as `draw` in the lecture.

(e)

2. Here is a simple employee class (omitting the methods):

```
class Employee {
protected:
        const string name;       // employee's name
        const string ni_number;  // employee's name
        double pay;              // pay earned so far this month

public:
        Employee(const string &n, const string &ni) :
                name(n), ni_number(ni), pay(0) {}
};
```

Now to define subclasses `HourlyEmployee` and `CommissionEmployee` of `Employee`. At this point, we have to decide whether it makes sense for a further class to derive from both of those, and if so whether the `Employee` part should be replicated. It does indeed make sense: we can have employees paid both hourly and by commission. Such people should have a single name and other attributes, so we want a single copy of the `Employee` part: we want repeated, or *virtual*, inheritance. So we define the derived classes:

```
class HourlyEmployee : public virtual Employee {
protected:
        double hourlyRate;     // hourly pay rate

public:
        HourlyEmployee(const string &n, const string &ni,
                       double rate) :
                hourlyRate(rate), Employee(n, ni) {}
};

class CommissionEmployee : public virtual Employee {
protected:
        double commission;     // hourly pay rate

public:
        CommissionEmployee(const string &n, const string &ni,
                           double percentage) :
                commission(percentage/100),
                Employee(n, ni) {}
};
```

Now class for employees paid in both ways will derive from both of these:

```
class BothEmployee : public HourlyEmployee,
        public CommissionEmployee {
public:
        BothEmployee(const string &name, const string &ni,
                     double rate,
                     double percentage) :
                Employee(name, ni),
                HourlyEmployee(name, ni, rate),
                CommissionEmployee(name, ni, percentage)
                {}
};
```

Note that in class selects the constructor of `Employee`. Even though `name` and `ni` are passed to the constructors of `HourlyEmployee` and `CommissionEmployee`, those constructors do not select an `Employee` constructor when they are invoked from `BothEmployee`. Thus their `name` and `ni` arguments are ignored.