Part A

You are developing classes for the products books and software, which must be processed slightly differently. Potentially, in the future there will be many other product types added to this hierarchy, and you wish to provide a general class `Product` with `Book` and `Software` as sub-classes.

1. Write a class for a `Product`
   – Each `Product` has a data member which holds the net price, and a constructor which sets this price.
   – Each `Product` has a method `getGrossPrice()`, which calculates and returns the gross price (the gross price includes VAT at 21%)

2. Write 2 classes which inherit from the general `Product` class, of type `Software` and `Book`
   – The gross price for Software includes VAT at 21%,
   – Books are free of VAT, so the gross price is unchanged from the net price, and you will need to re-define the `getGrossPrice` method in this class

3. Write a program which does the following:

   a. Declare an vector of 8 pointers to `Product.` Use the vector from the STL.

   b. Declare a pointer to a `Book` and a pointer to `Software`

   c. Ask the user to enter details of the book, and of the software item, create the two items dynamically and store their addresses in your pointers.

   d. Check your method `getGrossPrice` works correctly with each type, and then add them as the first 2 pointers in the vector of product pointers.

   e. Now ask the user to enter details of the remaining 6 items to be stored in your vector. For each they must give the net price and say whether it is a book or software. Enter each item with
   `v[i] = new Software(price) or v[i] = new Book(price)`

   f. Cycle through the vector and print the gross price of each item using the `getGrossPrice` method. *Do your first two items return the same gross price as they did when you checked in an earlier step? Should they?*

   g. Using the sort algorithm from the STL. Sort the vector in ascending order of gross price, using an efficient sorting mechanism

   h. Repeat step f

Part B

You are given the following class definition

```
class Person
{
public:
Person(string); // initialise the name
virtual void printname();
protected:
string name;
};
```

1. Add to this another 2 classes, `Employee` and `Customer`, each of which are derived from `Person`.

2. Each of these classes also has a `printname()` method, which are implemented differently:

   – `printname` method of `Employee` class will print out the name and salary (which is an extra data member of the `Employee` class).

   – `printname` method of `Customer` class will print out the name and a message saying they want to make a complaint.

3. Define constructors for each of these classes which pass back a name variable to the `Person` class for initialisation. The constructor of the `Employee` class should, in addition, initialise the salary, which is an attribute of that class.

4. In your main program, declare a pointer to `Person` class – call it `personPtr`. This is the base class pointer.

   – Create and initialise a `Person` object and call its `printname` method via the pointer.
   – Create and initialise an `Employee` object, and call its `printname` method via the base class pointer.
   – Create and initialise a `Customer` object, and call its `printname` method via the base class pointer.

```
Sample Output
My name is Mark
My name is Tom and my salary is 34, 000
My name is Ed and I want to make a complaint
```

5. Now take out the virtual keyword in the `Person` class and re-run the program to see how the output differs.

6. Rewrite the `Person` class so that it is an abstract class, and see what difference this makes to your program.

7. Now add appropriate overloaded operators as member functions to the `Person` class which will enable you to compare and order people according to their name. (*Look at the compare method of the string class http://www.cplusplus.com/reference/string/string/*)

8. Add code to your main method to enable you to create 3 (or more) `Person` objects, store pointers to them in an vector of pointers to `Person`, and order them alphabetically.