Computing 2 - Labs

Lab 7: Inheritance in the Grocery Store

A grocery store is planning to introduce a better product management system. To this end, all products (class `Product`) are going to be labeled with a bar code (`bar_code_`) and a name (`name_`) . Since the grocery store only sells foodstuffs, there are either packaged food items (`class PrepackedFood`) or loose food items (`class UnpackedFood`). Prepacked food is priced per item (`item_price_`). Loose food such as vegetables is sold by weight (`weight_in_kg_`) based on a price per kilogram (`price_per_kg_`). As a first task, an entry system for each food item is to be built. This task can be solved using an inheritance hierarchy as shown in the UML diagram below.

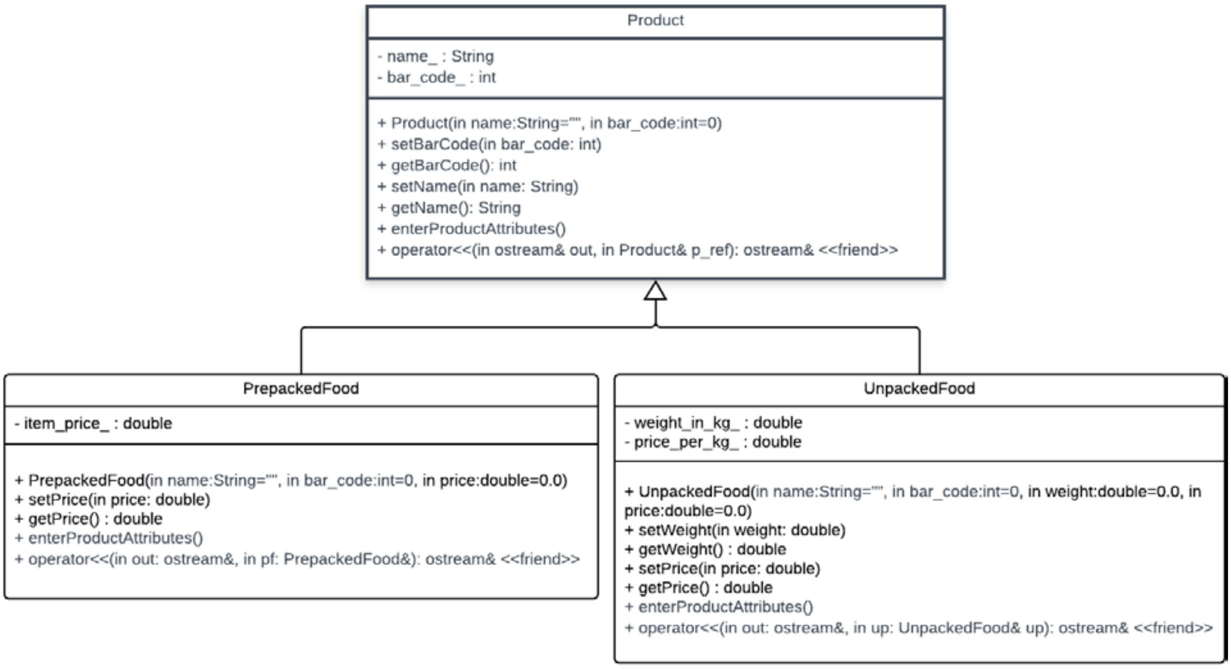

Product
- name_ : String
- bar_code_ : int
+ Product(in name:String="", in bar_code:int=0)
+ setBarCode(in bar_code: int)
+ getBarCode(): int
+ setName(in name: String)
+ getName(): String
+ enterProductAttributes()
+ operator<<(in ostream& out, in Product& p_ref): ostream& <<friend>>
PrepackedFood
- item_price_ : double
+ PrepackedFood(in name:String="", in bar_code:int=0, in price:double=0.0)
+ setPrice(in price: double)
+ getPrice() : double
+ enterProductAttributes()
+ operator<<(in out: ostream&, in pf: PrepackedFood&): ostream& <<friend>>
UnpackedFood
- weight_in_kg_ : double
- price_per_kg_ : double
+ UnpackedFood(in name:String="", in bar_code:int=0, in weight:double=0.0, in price:double=0.0)
+ setWeight(in weight: double)
+ getWeight() : double
+ setPrice(in price: double)
+ getPrice() : double
+ enterProductAttributes()
+ operator<<(in out: ostream&, in up: UnpackedFood& up): ostream& <<friend>>

Figure 1 UML diagram for a grocery's store product entry system.

There is a common base class `Product` and two derived classes `PrepackedFood` and UnpackedFood. The two derived classes inherit publicly the base class's member variables and member functions. Then they add their own attributes and behaviors. Note that constructors (as well as destructors) are not inherited. If you need something else than the default constructor and destructor the compiler provides, you have to write them yourself for each class.

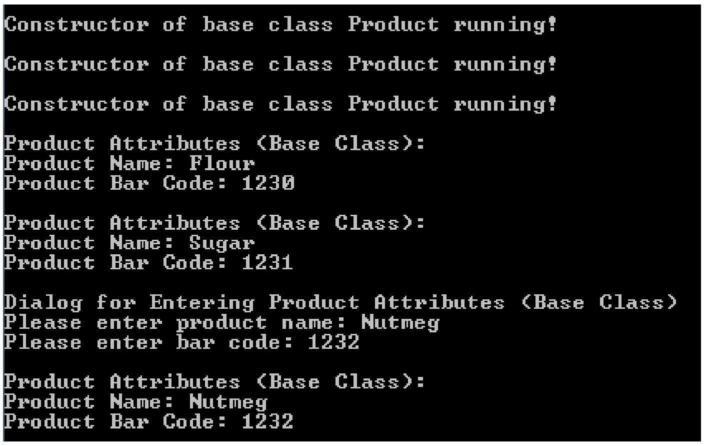
Beyond the typical get and set functions, there are two more methods that need to be explained in more detail:

1. `enterProductAttributes()`: this member function is to implement a dialog where the user is prompted to specify the data elements of the associated object one at a time.
2. Overloaded stream insertion operator `(<<):` this function prints the object's properties, i.e., the values of the data members. It can be implemented as a global (friend) function or as a normal global function using the get methods of each class.

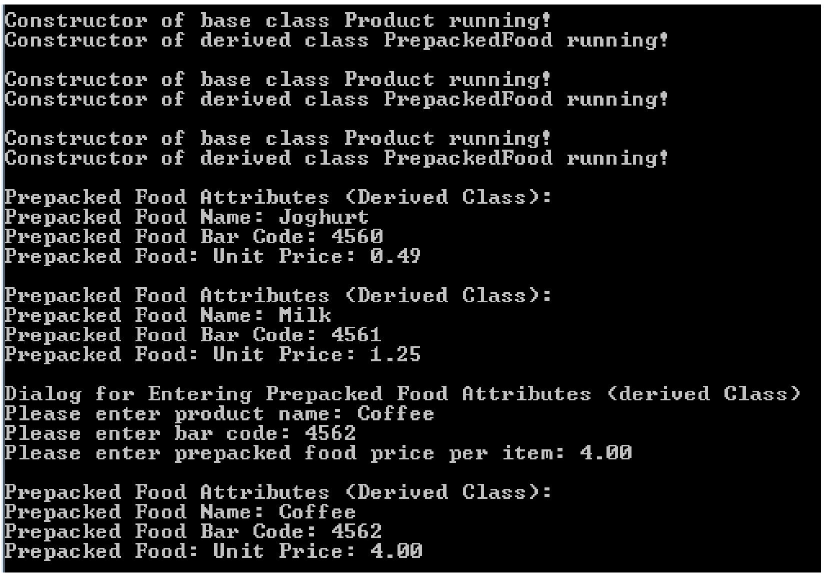
Tasks:

1. Based on the UML diagram, please implement the base class `Product` (i.e., write Product.h, Product.cpp).
2. Based on the code for the class `PrepackedFood` (PrepackedFood.h, PrepackedFood.cpp), write the derived class `UnpackedFood` (i.e. write UnpackedFood.h, UnpackedFood.cpp).
3. Then test your results using the driver program provided (main.cpp).

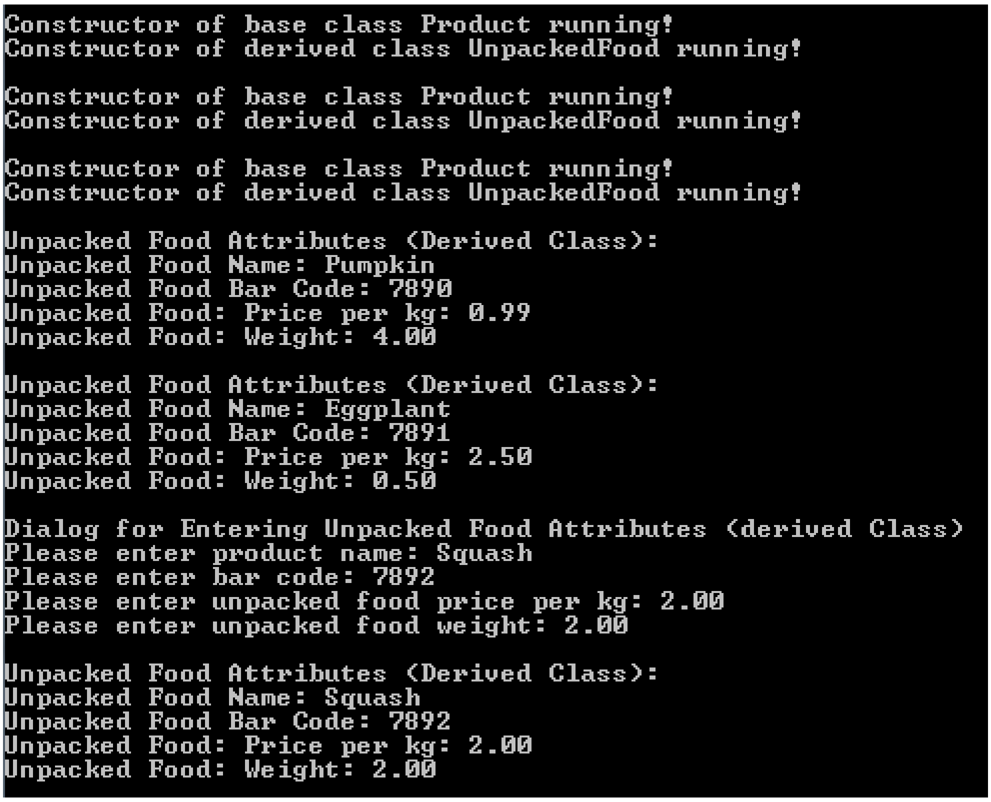
The result could look like this:



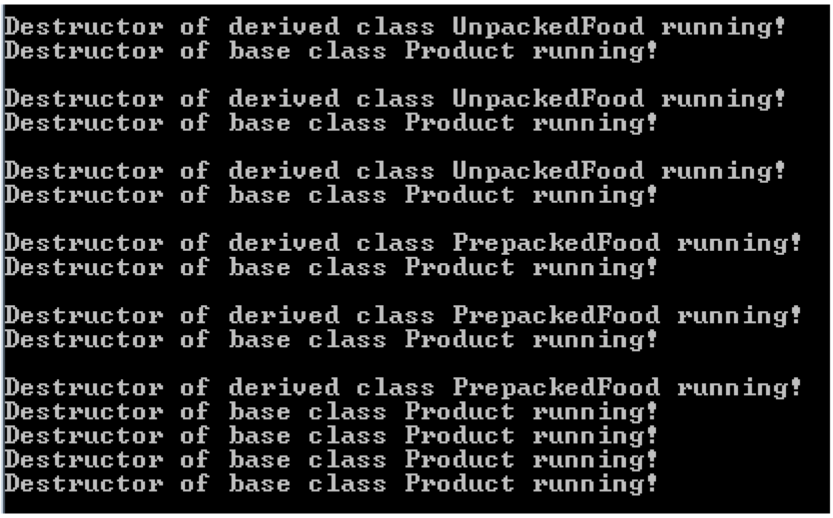
```
Constructor of base class Product running!

Constructor of base class Product running!

Constructor of base class Product running!

Product Attributes (Base Class):
Product Name: Flour
Product Bar Code: 1230

Product Attributes (Base Class):
Product Name: Sugar
Product Bar Code: 1231

Dialog for Entering Product Attributes (Base Class)
Please enter product name: Nutmeg
Please enter bar code: 1232

Product Attributes (Base Class):
Product Name: Nutmeg
Product Bar Code: 1232
```



```
Constructor of base class Product running!
Constructor of derived class PrepackedFood running!

Constructor of base class Product running!
Constructor of derived class PrepackedFood running!

Constructor of base class Product running!
Constructor of derived class PrepackedFood running!

Prepacked Food Attributes (Derived Class):
Prepacked Food Name: Joghurt
Prepacked Food Bar Code: 4560
Prepacked Food: Unit Price: 0.49

Prepacked Food Attributes (Derived Class):
Prepacked Food Name: Milk
Prepacked Food Bar Code: 4561
Prepacked Food: Unit Price: 1.25

Dialog for Entering Prepacked Food Attributes (derived Class)
Please enter product name: Coffee
Please enter bar code: 4562
Please enter prepacked food price per item: 4.00

Prepacked Food Attributes (Derived Class):
Prepacked Food Name: Coffee
Prepacked Food Bar Code: 4562
Prepacked Food: Unit Price: 4.00
```



```
Constructor of base class Product running!
Constructor of derived class UnpackedFood running!

Constructor of base class Product running!
Constructor of derived class UnpackedFood running!

Constructor of base class Product running!
Constructor of derived class UnpackedFood running!

Unpacked Food Attributes (Derived Class):
Unpacked Food Name: Pumpkin
Unpacked Food Bar Code: 7890
Unpacked Food: Price per kg: 0.99
Unpacked Food: Weight: 4.00

Unpacked Food Attributes (Derived Class):
Unpacked Food Name: Eggplant
Unpacked Food Bar Code: 7891
Unpacked Food: Price per kg: 2.50
Unpacked Food: Weight: 0.50

Dialog for Entering Unpacked Food Attributes (derived Class)
Please enter product name: Squash
Please enter bar code: 7892
Please enter unpacked food price per kg: 2.00
Please enter unpacked food weight: 2.00

Unpacked Food Attributes (Derived Class):
Unpacked Food Name: Squash
Unpacked Food Bar Code: 7892
Unpacked Food: Price per kg: 2.00
Unpacked Food: Weight: 2.00
```



```
Destructor of derived class UnpackedFood running!
Destructor of base class Product running!

Destructor of derived class UnpackedFood running!
Destructor of base class Product running!

Destructor of derived class UnpackedFood running!
Destructor of base class Product running!

Destructor of derived class PrepackedFood running!
Destructor of base class Product running!

Destructor of derived class PrepackedFood running!
Destructor of base class Product running!

Destructor of derived class PrepackedFood running!
Destructor of base class Product running!
Destructor of base class Product running!
Destructor of base class Product running!
Destructor of base class Product running!
```