**Homework 5**

**ECE 309 Fall 2019**

**Due: October 2, 2019**

**Free extension until October 4, 2019**

*This is a shared google document. This means (1) it may change to clarify content, and (2) other people can view your comments on this file. If you have questions, you are encouraged to comment directly on this document, but* ***do not add your answers here****. Make a copy into your private Google Drive and then edit the document.*

*DO NOT ADD ANSWERS TO THE SHARED DOC! THAT’S CONSIDERED CHEATING!*

# 1. Public Inheritance

[20 points] You’re implementing a new software application to help manage a grocery store’s inventory. There are lots of different kinds of items in a grocery store. The following table shows 5 kinds of instantiated objects in a grocery store that must be supported and the interfaces that are desired. Design a class hierarchy using public inheritance that allows these objects to be created and share as much code between them as possible. Provide the necessary data (strings, integers, double) as well in protected or private data members that are initialized using constructors. Also, an object should never be given a member function, even through inheritance, unless explicitly allowed to have it in the table below.

You should not name the class as “Avocado” or “Cereal”. These are merely examples of the kinds of objects. Instead, you should think about it more like “PreparedFood” or “Produce” in which these kinds of objects would fit.

Also, if you do not use public inheritance, your answer will earn 0 points.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Avocado | Cereal | Jam | Rotisserie Chicken | Banana |
| string getName(): return name of the item as a string, in this case it would return “Avocado”. | getName(): return name of the item as a string, in this case it would return “Cereal” | getName(): return name of the item as a string | getName(): return name of the item as a string | getName(): return name of the item as a string |
| string getBarcode(); | string getBarcode(); | string getBarcode(); | string getBarcode(); | string getBarcode(); |
| double costPerItem(); | double costPerItem(); | double costPerItem(); | double costPerItem(); | double  costPerPound(); |
| int getLocId() (return an ID for where its located) | int getLocId();  (return an ID for where its located) | int getLocId();  (return an ID for where its located) | string getDepartment() | int getLocId();  (return an ID for where its located) |
| string getDepartment(); Return which part of the store manages this item. | string getDepartment() | string getDepartment() | string getDatePrepared() | string getDepartment() |
| string getSupplier();  Return the name of the company that supplies this produce. | int getExpirationYear(); | int getExpirationYear() | int getLocId();  (return an ID for where its located) |  |
|  | string getBrand();  Return the name of the brand that makes this item. | string getBrand() |  | string getSupplier() |

# 

# 2. ZyLabs

[80 points] Submit the following ZyLabs before the deadline.

1. [15 points] ZyLabs 11.30, Implement a Queue using the has-a relationship without private inheritance.
2. [15 points] ZyLabs 13.8, Implement a Queue using private inheritance.
3. [50 points] ZyLabs 8.22 IntArray with operator overloading. Here’s some explanation of the missing functions you need to implement:

|  |  |
| --- | --- |
| IntArray& operator+=(const IntArray &rhs); | Accumulate rhs into this object and return a reference to this object. Keep size of this object fixed even if rhs is bigger. If rhs’s array is larger, just ignore those elements. |
| IntArray operator+(const IntArray &rhs) const | Add two IntArray objects together in a new IntArray object, and return the new object.  The new object has an array the same size as that in this object. If the rhs is smaller, just copy over the extra ones unmodified. |
| IntArray operator-(const IntArray &rhs) const; | Subtract two IntArray objects into a new IntArray object, and return the new object. For example, if the lhs was {1, 2, 3} and the rhs was {1, 2, 3}, the new array would be {0,0,0}, because we pair-wise subtracted the elements.  The new object has an array the same size as that in the lhs object. If the rhs is smaller, just copy over the extra ones unmodified. |
| IntArray operator\*(int x) const; | Multiply array by integer x. If the array was {1, 2, 3} and x=3, then the return value is a new array with {3, 6, 9}.  The new object has an array the same size as that in the lhs object. |