# Summer 2015.

# **Wedding Planner Application**

OOP244 Assignment

**Milestone 3: the Good class V1.0**

**Due DAte Thu AUG 6th 23:59**

Create a class called Good. The Good class encapsulates a general ReadWrite item.

Although the Good class is a ReadWrite (derived from ReadWrite) it does not implement any of the pure virtual member functions and therefore remains abstract.

The Good class is implemented under the oop244 namespace. Code the Good class in the Good.cpp and Good.h files provided in FP2154MS3 repository on github:

<https://github.com/Seneca-OOP244/FP2154MS3>

You do not need the Date class for this milestone.

Good Class specs:

Private Member variables:

**\_upc:** Character array, MAX\_UPC\_LEN + 1 characters long

This character array holds the UPC (barcode) of the items as a string.

**\_name:** Character pointer  
 This character pointer points to a dynamic string that holds the name of the Good

**\_price**: Double  
 Holds the Price of the Good

**\_taxed:** Boolean  
 This variable is true if this item is taxed

**\_quantity:** Integer

Holds the on hand (current) quantity of the item.

**\_qtyNeeded:** Integer  
 Holds the quantity needed to purchase

# **Public member variables and constructors**

## **Constructor:**

A Good object is constructed from 5 values passed to the constructor:  
the UPC, the Name, the Price, the Quantity needed and the taxable status.   
The constructor:

* Copies the UPC into the corresponding member variable up to MAX\_UPC\_LEN characters.
* Allocates enough memory to hold the name, stores its address in the \_name pointer and copies the name received into the allocated memory starting at \_name.
* Sets \_quantity to zero.
* Sets the rest of the member variables to the corresponding values received by the arguments.
* If value for a Good being taxed is not provided, the constructor sets the \_taxed flag to the default value of “true”

## **Dynamic memory allocation necessities**

Implement the copy constructor and the operator= so that copying or assignment to the Good item occurs safely and without any memory leak. Also, implement a virtual destructor to make sure the dynamically allocated memory is freed when Good is destroyed.

**Accessors**

**Setters:**Create the following setter functions to set the corresponding member variables:  
- **upc**

- **price**

- **name**

- **taxed**

- **quantity**

- **qtyNeeded** (quantity Needed)

All of the above setters return void.

**Getters:**

Create the following getter functions to return the values or addresses of the member variables:

- **upc**, returns constant character pointer

- **price**, returns double

- **name**, returns constant character pointer

- **taxed**, returns boolean

- **quantity**, returns integer

- **qtyNeeded** (quantity Needed), returns integer

Also:

- **cost**, returns double

Cost returns the cost of the item after tax. If the Good is not taxed the return value of cost() is the same as the price.

All of the above getters are constant methods, which means they CANNOT modify the object.

## **Member Operator overloads:**

**Operator==** : receives a constant character pointer and returns a Boolean.

This operator compares the received address to the UPC of the current object and returns true if they are the same, false otherwise.

**Operator+=** : receives an integer and returns an integer.

This operator adds the received integer value to the quantity on hand and returns the sum.

## **Non-Member operator overload:**

**Operator+=** : receives a reference to a double as the left operand and an unmodifiable reference to a Good as the right operand and returns a double value;

This operator multiplies the cost of the Good by the quantity of the Good, adds that value to the left operand and returns the result.

Essentially this means that this operator adds the total cost of the item on hand to the left operand, which is a reference to a double, and then returns it.

# **Non-member IO operator overloads:**

After implementing the Good class, overload the operator<< and operator>> to insert a Good into an ostream (cout) object and istream (cin) to extract a Good from the console. Use the display() and conInput() methods of ReadWrite class to implement these operator overloads.

Make sure that the prototypes for these helper functions are in Good.h.

# **Submission:**

Please refer to your professor’s instructions for submission.

*Note: You should NOT have more than one return statement in a function. This rule (having one point of entry to and one point of exit out of a function) was established during the structured programming era decades ago and is not allowed in your code.*