# Summer 2015.

# **Wedding Planner Application**

OOP244 Assignment

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

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

Create a class called Good. The class Good is responsible for encapsulating a general ReadWrite item.

Although the class Good is a ReadWrite (inherited from ReadWrite) it will not implement any of the pure virtual member functions, therefore it remains abstract.

The class Good 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 will be 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:**

Good is constructed by passing 5 values to the constructor:  
the UPC, the Name, the price, the Quantity needed and if the good is taxed or not.   
The constructor:

* Copies the UPC into the corresponding member variable up to MAX\_UPC\_LEN characters.
* Allocates enough memory to hold the name in the \_name pointer and then copies the name into the allocated memory pointed to by the member variable \_name.
* Sets quantity on hand to zero.
* Sets the rest of the member variables to the corresponding values received by the arguments.
* If value for good being taxed is not provided, it will set the \_taxed flag to the default value “true”

## **Dynamic memory allocation necessities**

Implement the copy constructor and the operator= so the item is copied from and assigned to another Good safely and without any memory leak. Also implement a virtual destructor to make sure the memory allocated by \_name 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 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() will be the same as price.

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

## **Member Operator overloads:**

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

This operator will compare the received constant character pointer to the UPC of the Good, if they are the same, it will return true or else, it will return false.

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

This operator will add the received integer value to the quantity on hand of the Good, returning the sum.

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

**Operator+=** : receives a double reference value as left operand and a constant Good reference as right operand and returns a double value;

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

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

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

After implementing the Good class, overload the operator<< and operator>> to work with ostream (cout) to print a Good to, and istream (cin) to read a Good from, the console. Use the display() and conInput()methods of ReadWrite class to implement these operator overloads.

Make sure the prototype of the 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.*