CS 162 LAB #7 – Implementing Inheritance

In order to get credit for the lab, you need to be checked off by the end of lab. You can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the lab before the next lab and get checked off with your lab TAs during their office hours. For extenuating circumstances, contact your lab TAs and the instructor.

This lab is worth 10 points total. Here's the breakdown:

- 4 points: Fruit class, makefile, and application implemented
- 4 points: Apple and Watermelon classes implemented
- 2 points: Granny_Smith class implemented

In this lab, you'll start to work with inheritance in C++.

(4 pts) Step 1: Implement a generic Fruit class

We're going to work with fruits in this lab exercise. We'll create several classes to represent different fruit, some of them using inheritance. The first class we'll write is one to represent a generic fruit with a name, a color, and a unit price.

Create two new files, `fruit.h` and `fruit.cpp`, and in them, define a `Fruit` class. Here's the start of a class definition you should use:

```
class Fruit {
protected:
      const string name; //note the keyword 'const'
      string color;
      double unit_price;
public:
      ...
};
```

Your class should also have constructors, accessors, and mutators, as appropriate. In addition, your class should have a `price()` function for computing the fruit's price. For this generic `Fruit` class, the `price()` function can simply return 0, since we aren't actually defining the fruit itself.

In addition to your files `fruit.h` and `fruit.cpp`, create a new file `application.cpp`. In this file, write a simple `main()` function that instantiates some `Fruit` objects and prints out their information (name, color, and unit_price). In addition, write a `makefile` to specify compilation of your program. Make sure you compile your `Fruit` class into an object file first, separately from the compilation of your application, and then use that object file when you're compiling your application.

(4 pts) Step 2: Implement Apple and Watermelon classes

Create new files `apple.h`, `apple.cpp`, `watermelon.h`, and `watermelon.cpp`, and in them, implement an `Apple` class and a `Watermelon` class. Both of these classes should be derived from your `Fruit` class. The Apple class should have a `weight` and a `sweetness`, and the

`Watermelon` class should have a `seeds`. Here are the beginnings of definitions for these classes:

```
class Apple : public Fruit {
private:
      float weight;
      bool sweetness; //0 for sour, 1 for sweet
public:
      ...
};

class Watermelon : public Fruit {
private:
      bool seeds; //0 for seedless, 1 for seeded
public:
      ...
};
```

Both of these classes should have constructors, accessors, and mutators, as needed, and each one should override the `Fruit` class's `price()` function to compute prices that are appropriate for apples and watermelons:
Apple: price = weight * unit_price
Watermelon: if seeds is true, price = unit_price; if seeds is false, price = unit_price * 1.2

Add some code to your application to instantiate and print out some `Apple` and `Watermelon` objects, and add rules to your `makefile` to compile each of your new classes into separate object files, which you should then use when compiling your application.

(2 pts) Step 3: Implement a GrannySmith (green apple) class

Now, create new files `granny.h` and `granny.cpp`, and in them, implement a `Granny_Smith` class that derives from your `Apple` class. Your `Granny_Smith` class ***should not*** contain any new data members, nor may you change any members of the `Apple` class to `protected` or `public` access. Instead, you should figure out how to implement a public interface for your `Granny_Smith` class by appropriately using the `weight` and `sweetness` of your `Apple` class ***via its public interface*** (i.e. via the `Apple` class's constructors, accessors, and mutators). Specifically, the public interface to your `Granny_Smith` class should use the public interface of your `Apple` class while enforcing the constraint that a granny smith's sweetness has to be 0 (sour).

Hint: You probably want to redefine the `set_sweetness()` in the Granny_Smith class as if the user changes the sweetness, you need to modify the value of sweetness so that it remains 0 (sour).

Here's the start of a definition for your Granny_Smith class, with no new data members:

```
class Granny_Smith : public Apple {
public:
      void set_sweetness(bool);
```

```
        ...
};
```

Once your `Granny_Smith` class is written, add some lines to your application to instantiate and print out some `Granny_Smith` objects, and add a `makefile` rule to compile your class into an object file that's used in the compilation of your application.

Show your completed work and answers to the TAs for credit. You will not get points if you do not get checked off!

Submit your work to TEACH for our records **(Note: you will not get points if you don't get checked off with a TA!!!)**

1. Create a **zip archive** that contains all files you've created in this lab:
2. Transfer the tar file from the ENGR server to your local laptop.
3. Go to TEACH.
4. In the menu on the right side, go to **Class Tools → Submit Assignment**.
5. Select **CS162 Lab7** from the list of assignments and click "**SUBMIT NOW**"
6. Select your files and click the Submit button.