Section Handout #6

Problem 1: The Philosophy of Objects and Classes

a) What is a C++ class? Describe the importance of the *wall of abstraction* and why it is useful to mark data as `public` or `private`. What problems could you encounter if there weren't the notion of `private` variables?

b) How does object-oriented programming help, from the viewpoints of both the person implementing a class and the person using it?

c) Imagine we're writing a small class to manage a Rectangle. It will encapsulate the location of the rectangle (x and y coordinates) and a width and height from which we can calculate things like the perimeter or area. We have four options as to how we will write the Rectangle class in order to afford the user a way to get its perimeter:

a. Make `width` and `height` public data members
b. Make `width` and `height` private, but allow access to them through `getWidth()` and `getHeight()` methods
c. Keep a public `perimeter` data member that the user can read directly
d. Add a `getPerimeter()` method

What are the pros and cons of each strategy? Which one is probably the best to choose?

Problem 2: A simple class

Design an interface for the Rectangle class. We will be keeping track of its size as well as its location, and should allow the user to find the rectangle's perimeter and area. You should also provide a way for the user to scale the rectangle by some amount and translate it in space.

Problem 3: Function Pointers and Templates

a) Write a generic function FindMax that searches a vector to find the largest element and returns it. The two arguments to the function are the vector and a comparison function. The comparison function should take two elements and return an integer that is negative if the first is less than the second, zero if the two are equal, and positive otherwise. The second parameter should be optional, if not given, the default comparison of built-in relational operators is used. Your function may assume that the vector has at least one element. FindMax should be written as a template and must work for vectors of any type.

b) Add the necessary code below to print the name of the safest car from the vector. The safest car is the one with the most airbags; among cars with the same number of airbags, the heavier car is safer. You should use the FindMax function from part a.

```
struct Car {
    string name;
    int weight;
    int numAirbags;
};
```

```
int main()
{
 Vector<Car> cars;
 // assume variables have been initialized here
 // you are to add code to get safest car from array using FindMax
```

Problem 4: Templatized Functions

Write a function **`Filter`** that removes all elements from a Queue that fits a user-supplied criteria. The function will take two parameters:

- A Queue, passed by reference, of the elements you want to filter.
- A function pointer to a bool function with one parameter; the type of the parameter is the same as the one stored in the Queue. The bool function returns true if the parameter fits a certain criteria. This type of function (one that returns a boolean) is sometimes called a predicate function.

For example, let's say that I'm cleaning out all of the recording in my DVR. The information regarding the individual recordings is kept in the following struct:

```
struct recordingT {
  string title;
  string genre;
  int rating;
};
```

If I want to remove all animated recordings with a rating of less than 8, I would pass a pointer to the following function:

```
bool IsBadAnimation(recordingT show)
{
  return (show.genre == "animation" && show.rating < 8);
}
```

If an element contained in the Queue causes the user-supplied function to return true, it should be removed from the Queue. In addition, function **`Filter`** should support Queues of any type.

Problem 5: More Templatized Functions

a) Write a templatized function **`RemoveDuplicates`** that finds all duplicate elements in a Vector and removes each appearance of the element beyond the first. For example, if it was passed the Vector [1, 2, 3, 4, 2, 3, 3] it would modify the Vector to be [1, 2, 3, 4]. It is passed the Vector by reference as well as a comparison function which two elements and returns an integer that is less than, equal to, or greater than zero depending on whether the first element is less than, equal to, or greater than the second element, respectively.

b) Suppose you have a Vector of ints, and you want only one occurrence of a number with a particular absolute value. That is, you only want one of 2 and -2 to be in the vector (although you don't care which it is). Write a callback function to accomplish this and show how you would call your function from part a to use it.