Computer Science 3-4
Problem Set #13: So Classy!

This problem set is going to be a bit different than the rest. There are quite a lot of details in classes and it can be a little bit overwhelming to make something on your own right off the bat. So we're going to start off by just walking through an example set of code. I encourage you to read along and actually type up the whole set of example code so you can run it and tinker with it a bit. Several of the homework problems will be to add some features to our basic program, so you'll eventually have to code it up anyways so start early!

At its heart, a *class* is just a way to declare a new *type* of variable. In the past we've seen lots of different types (int, string, char, etc.) and each has a specific use. The benefit of creating our own type is that it lets us create a blueprint for an *object*. An object is a data structure that lets us hold properties (variables) and actions (functions) in a specific programmer defined type. Let's get started with some code and you'll see what we mean.

Buffalo Herder: Adventure on the High Plains

We're going to make a simple little Tamagotchi-style game about raising buffalo (me being from Montana and all). We want to eventually end up with a whole herd of buffalo, but we also want to keep our sanity if we decide we need to have 50 of the critters running around our virtual ranch. Our first order of business is to make the "Buffalo" object, which we'll do by making its own class. Here's what the first version of our class looks like:

```
//Buffalo Herder - First Pass
//Demonstrates creating a new type, member variables and functions

#include <iostream>

using namespace std;

class Buffalo   // class definition -- defines a new type, Buffalo
{
public:
    int hungerLevel;       // data member
    void SayHowdy();       // member function prototype
};                         // this is the end of the class definition...
                           // note the ";"

void Buffalo::SayHowdy()   // member function definition
{
    cout << "Howdy. I'm a Buffalo. My hunger level is " << hungerLevel << ".\n";
}
// end of class related functions
```

```
int main()
{
    Buffalo Bill;
    Buffalo Bob;

    Bill.hungerLevel = 9;
    cout << "Bill's hunger level is " << Bill.hungerLevel << ".\n";

    Bob.hungerLevel = 3;
    cout << "Bob's hunger level is " << Bob.hungerLevel << ".\n\n";
    Bill.SayHowdy();
    Bob.SayHowdy();

    return 0;
}
```

Notice a couple of things right off the bat. First off our class is defined before we get to the main program. Second, notice the class holds two basic things…a variable (we're starting off with one but there can be as many as you want) and a function. This is really the crux of the class concept. These variables/functions belong very specifically to this class, and moreover it creates this sort of template that says…if I make a buffalo object it will have these characteristics. A good way to think about this is that when you make a class, you are really making a blueprint that describes two things: what are the properties of this object (i.e. variables that matter) and what are the actions the object can take (i.e. the member functions).

So for our buffalo, we want to keep track of how hungry he is and we want him to be able to tell us about his hunger level…hence the hungerLevel variable and SayHowdy member function. Notice that we don't actually put any code for our member in the class definition itself, we actually define the function below "void Buffalo::SayHowdy()". This is just like our normal function definition with one little twist: we need to specify that this function is associated with a specific class; hence the "Buffalo::" prefix. This helps us avoid confusion if we have multiple classes. Suppose we had a Buffalo and an Wolf class…we could easily have functions in each class that say "Eat", but they might do slightly different things because they are different animals/classes (Bison eat a ton more since they are vegetarians…so their function would get called more frequently!).

If you typed in all of the above code, you probably noticed that the output of our main is just showing us the hungerLevel in two different ways. First we use the class to actually define a specific variable we care about "Buffalo Bill". We use the "." operator to access things that are inside our created object (provided that they are in the public area of our class definition, but more on that later).

That being said what did we really accomplish? We made it so we could easily create two different bison and assign them some different hunger levels. This is a great start and we'll expand on this in the future.

Now some problems:

1. We will now write a program with a simple *Ship* object. The Ship class should have the following data members (i.e. variables): name (a string), fuelLevel (an integer). It should also have one function called reportStatus (a void that prints out the name and fuelLevel). The constructer for Ship should allow you to specify the name and fuelLevel your ship starts with…or default to values of your choosing. Your main function should have several instances of the Ship object that demonstrate multiple ships with different name/fuel combinations.
2. Modify the class from problem 1 to include two new functions: Move (a void that takes an integer distance as a parameter…note it takes one fuel to move one distance) and FillMeUp (a void that takes an integer fuelAmount and adds to the fuel of the ship). Make sure your Move function checks for the amount of fuel available…if there isn't enough fuel, then the ship shouldn't move! Also add in a variable to the Ship class that tracks the total distance the ship has moved. Your main function should have several instances of the Ship object that demonstrate the new features.

Now that we've cut our teeth on our Ship class, I want you to add some new functionality to the Buffalo class that finishes off our little Buffalo Herder game.

3. **Life on the High Plains** - We need to add in some functionality for our Buffalo to eat, sleep, roam and possibly die out on the range. Your task is to add variables to keep track of the Buffalo's huger level, level of exhaustion and happiness (a qualitative measure of the Buffalo's hunger + exhaustion). You will need to add some functions that let the Buffalo eat (lower hunger level, raise exhaustion), sleep (lower exhaustion, raise hunger), roam (raise hunger slightly, raise exhaustion slightly, displays entertaining message), Update(a function that is called by the other functions to see if the Buffalo is alive) and Grunt (reports happiness level by certain types of grunts). We will talk about this in class some, but you have some leeway in how this plays out based on your own imagination.