Programming Assignment #1
CS 302 Programming Methodologies and Software Implementation

***** Make sure to read the Background Information first *****

Background:

When beginning this project, the first thing to keep in mind is that we are no longer working on CS163 programs! In CS163, we were concerned about creating Abstract Data Types. Instead, this term we will be focusing on how to create solutions where there are a series of classes, or abstractions, working together to solve a problem. An ADT may be part of that solution – but it certainly shouldn't be the primary focus. Instead, our goal should be to learn how to divide the problem into smaller abstractions (classes) that interact with each other to ultimately solve the problem. You will want to focus on developing classes that have specific "jobs", where one class is based on a more general class, whenever appropriate. We will learn to "push up the common" and "derive the differences". We will be developing applications with multiple classes, so you will want to focus on dividing the design into smaller components that have specific jobs working together to solve the problem.

Every assignment this term needs to have at least 5 classes. Divide up the work that each class needs to do. If each class performs a small specific job, then as a collective, a much larger problem can be solved. Your program should delegate responsibilities. Avoid the overuse of functions to "give" another class hidden data (**these are called "getters" and should be avoided**). If you give data to another class, it means that the other class will need to know all about the details of how to process that data, causing a duplication of effort. Instead, each class should be responsible for managing their own data. Instead of giving your data away – implement the functionality necessary to process that data based on what the client would have needed to do.

This term, we will begin by guiding you through programming with abstractions by focusing your attention first on the core components of the problem that manages the underlying data; this is done through progress submissions. Then we will move to the next phase where we begin to write classes that use that hierarchy, implementing the assigned data structures.

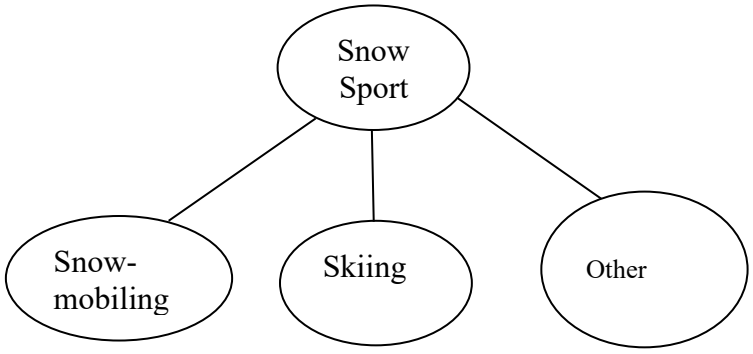
***This means you should pay close attention to the order of the progress submission requirements ***

Overview of the Problem:
How was your Winter break? We had relatives from Texas visit and had planned to go up to the mountain to enjoy the snow. We had all of our gear just to find out there wasn't any snow. But now the weather "experts" are calling for possibly FEET of snow this week. Now it is time to get to the mountain and start having some fun. I am really competitive, so it is time to start racing! I would like to race using snow mobiles. Some of you might prefer to race as cross-country skiers. Or maybe you enjoy downhill skiing? There are so many options.

Program #1 Specifics
For this first programming assignment, you will create a simple application to support races between individuals of a particular mountain sport. We will start by building three different possible sports that you would like to support. These could be skiing, cross country, tobogganing, snowmobiling, inner tubing, snowboarding, ice skating, and many more.

Here are a few of the requirements as you get started:

1. Support three different types of snow sports that could participate in races, such as skiing, snowmobiling, and so on. These are your choice, but you may pick from my list or add other choices. *Be creative and select something that you are interested in.*
2. For each type of snow sport, you will need to select functions that make sense when working with that type when racing. You will need to pick **three** public methods for each class in addition to constructors, destructors, setters, and display functions. For example, with snowmobiling, we could turn on/off the engine, go forward at a particular speed, stop, and find out how long we can run on the current fuel level. Think of these methods as the **job of the class**!
3. All of the types of snow sports will have some kind of similarity such as their name and current location. Such items can be pushed up and the differences derived. ***Inheritance will be discussed in Lecture #2 ***



First Progress Submission: The core hierarchy
The first progress submission will be about setting up these three different types of snow sports that you want to support. Really think of how to work with ONE skier, ONE snowmobiler, ONE inner tuber, etc. At this first step, you are not managing a collective or building data structures.

Look at the common elements of these classes – push them up to a common base class and derive the differences. Anything that is similar between the three different types of sports should be written only once (in the base class). Keep classes small and functions small. A large class or function means that the problem has not yet been broken down into its basic components (objects).

REQUIRED: Your base class and one derived class must both support **ONE** dynamically allocated array of characters (char *'s) for practice. All other use of string data should be represented by the STL's string class. This means char *'s should only be in two classes!

If a data structure is needed in this first stage of the assignment, use the STL (e.g., vector, list, array, etc.).

Second Progress Submission: Creating the Application
For the second progress submission, it is time to create the application and develop the data structures that are needed to hold a race for a particular sport (we wouldn't mix different types of sports together). The user would need to pick which sport they are interested in and then your application will build a race of different entries (objects) of that sport. To race you need more than one entry – so a data structure will be needed of the objects that are racing. For example, you could have a circular linked list of snowmobile objects that are racing amongst each other. Make sure to use a random number generator to assist with running a simple race. Add something fun in the mix, like the snowmobile turns on its side (no one is hurt) or runs out of fuel…..

These are the required data structures. You may decide how best to use them:

- A Linear linked lists (LLL)
- A Circular linked list (CLL)
- A Vector from the STL

Implementation of the LLL and the CLL requires full support of insert, removal, display, retrieval, and remove-all. **All repetitive data structures algorithms must be implemented recursively.**

For the second progress submission, progress should be shown on data structures implementation and towards being able to race. But a completed implementation is not expected until the finished due date.

Important items when programming in C++:

1. Every class that manages dynamic memory must have a copy constructor
2. Every class that manages dynamic memory must have an assignment operator
3. Every class that manages dynamic memory must have a destructor
4. Every class must have a default constructor
5. Your node needs to be a class (no structs in CS302)
6. No global variables allowed
7. No statically allocated arrays allowed
8. Every derived class that implements a copy constructor needs to have an initialization list to case the base class' copy constructor to be invoked
9. When passing class objects to a function, pass as constant references whenever possible.
10. Avoid passing or returning class objects by value except to test copy constructors and assignment operators
11. Make sure to use the returned value when calling a function unless you are implementing exception handling. We always need a "hand shake" when one class is communicating with another.
12. Avoid methods that have no arguments and void returned types; such methods do not communicate with any other classes. In CS302, our abstractions should be working together as a team.
13. Avoid overuse of getters!!
14. **Have fun and be creative!**

Remember, all code submitted must compile and run to gain credit. Progress submissions are submitted using the MCECS gitlab repository and must be shared with karlaf_grader@cs.pdx.edu. The finished assignment must be submitted as a tarball or zip to Canvas for our grader. Upload the 400 word efficiency writeup and 200 word gdb writeup separately (not part of the tarball or zip file).