Programming Assignments #1 and 2
CS 202 Programming Systems

***** Make sure to read the Background Information first!**
It applies to all programming assignments this term***

LATE work will be accepted for 5% deduction – pay close attention to the late due dates, since they are shorter in Summer!

We do not accept late work beyond that. No exceptions.

Background:
When beginning with 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 and the class construct facilitated this. Instead, this term we will be focusing on how to create Object Oriented Solutions. An ADT may be part of that solution – but it certainly shouldn't be the primary focus. Instead you want to strive for classes to have specific "jobs" and have classes derived from more general classes, whenever appropriate.

Overview:
For the first two programs you will be applying OO techniques to creating a game. Did you see the movie Maze Runner? It is a post apocalyptic dystopian science fiction movie where they must travel through a Maze if they are going to escape. In the movie there was a code by which the maze was based. It wasn't random.

Imagine that your job has been assigned to create a computer game that represents a Maze. The game has at least two levels. In Phase 1 of the assignment (program #1), you will be implementing the first level of the maze. In Phase 2 of the assignment (program #2), you will be implementing the second level of the maze.

Program #1 - Level 1:
In the first level of the maze, all players begin at the same start point and at any intersection that are given the choice of (a) going left, (b) going right, or (c) going backwards (without knowing what is to the left or right yet). At any intersection, they may encounter prizes (e.g., points or coins) or they may lose their life. A player begins with at least 3 lives and they must start over when all lives are lost. The maze itself must be randomly generated each time a player starts. Use a binary search tree as the data structure. Only one leaf will be the finish line, so the player must choose correctly! If a dead-end is reached, a life is lost! You may use prizes creatively to change the game.

Program #2 - Level 2:
In the next level, the player has up to 5 different directions at any intersection. This time a graph is used as the data structure, implemented as an adjacency list. In program #2 the idea is to support the notion of dynamic binding, where the same function call can be used to call one of two (or more) different functions. This way, the client program will not have to be changed when moving from level #1 to level #2. The code for the operations will be different but the client program calling the functions will not. **At least one function must be pure virtual so that you experience abstract base classes in your design.**

Just like the previous level, the maze is generated randomly, with random number of directions this time (up to 5), random points where prizes are won, and random pitfalls where a life is lost.

To make this Object Oriented:
You will want to first think about breaking this down into a series of classes and create them independent of the entire game. The design for the ENTIRE game is due prior to the first program being due. Pay close attention to the Course Outline for the due date for the design and programs.

Here are some suggestions to start with. You may alter this set of classes but you will need to implement 5 classes at a minimum.

1. Prize – what is a prize and what benefit does a prize (e.g. points or coin) have on the game
2. Life – a player begins with three lives and then must start over after the last one is lost.
3. Player – a player is their name plus how may prizes they have won and what their life is. Think about how best to use inheritance. Players should have background information about who they are. For example "Alby was the eldest and leader of the Gladers" or "Teresa was the only girl to enter the Glade".
4. Maze – a maze is a series of paths; players can go left, right, or backwards for level 1. In level 2, they can go up to 5 different directions or backward. The Code is the secret that "seeds" the random number generator.
5. Node – Yes, your data structures will need to use a node class! Examine the code used for labs 1 and 2 to get some ideas of how a node can be used as a class

Anything that is similar between these or other classes that you write should be pulled up to be part of a base class. For example, classes that manage collections of items may be derived from a common base class that manages the collection.

When Implementing Data Structures;
Implementation of the data structures requires full support of insert, removal, display, retrieval, and remove-all.

Writeups Required: (per style sheet)

I. Written Design (turned in early)

- The design writeup has a separate due date and may **not** be turned in late
- This must be written in English using complete sentences.

1. 600 word minimum
2. It should cover the major design considerations
3. Discuss what classes you are intending to create
4. Discuss the relationship between those classes (using, containing, hierarchical)
5. Discuss what methods are needed to avoid excessive use of "getters"
6. Outline the functions that you need for each class and how they will be used by other classes in your design
7. UML diagrams are highly encouraged but they do not replace the need for a writeup!

II. Analyzing your Design (after programming):

- This must be written in English using complete sentences.

1. 400 word minimum
2. Discuss the effectiveness of your design and classes
3. Discuss the validity of your approach
4. Analyze your results in terms of object oriented programming methodologies
5. What major changes did you have to make in your design and explain why
6. Describe the efficiency of the approach and the efficiency of the resulting code
7. **Think in terms of analyzing your solution!**

III. Debugger Writeup (after programming):

- This must be written in English using complete sentences.

1. 400 word minimum
2. Discuss the effectiveness of the debugger
3. Discuss what problems it helped you solve
4. Did you discover how it could be used to enhance the programming experience
5. Discuss features that you would like to learn about so that you could use them the next time you program