# Programming Assignment #1
# CS 202 Programming Systems

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

*We do not accept late work beyond the late due date. There are no exceptions.*

*Always backup your files PRIOR to using tar*
*Double check that the arguments specified with for tar are correct!*
*Late penalties will apply to all submissions incorrectly archived/uploaded*

**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. We will be working in situations where there are 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. With these, think about how to design the classes such that they reduce the amount of work another class needs to do. The idea is if we have "robot" like classes doing the smaller tasks or "jobs", that by the time we get to a larger class that has more to manage – it will have little left to do! We can achieve this by delegating. Often the over-use of "getters" can cause the opposite to happen – and instead of delegating the managing class has to fundamentally do all of the work itself.

**Overview:**
Social games have really become popular, whether they are game we watch (e.g., Survivor, the Great Race) or games that we play (e.g., Pokemon Go). The advantage of these games is the unpredictable results. Because there are interactions, we may or may not be able to predict how to win the game, which makes them more interesting to watch and/or play.

**Program #1**
For Program #1, you will be creating an object oriented program that will create a CS Maze Game that builds upon the social nature of games today. The fundamental rules of this game are outlined below. You are allowed to add more rules to make it as interesting as possible:

1. Multi player game
2. Each player must be part of a group called a "friends list"; each list must have at least 2 players. Each player within a group moves independently of the group, but information will be shared among group members at times.
3. Each player will navigate through a Maze beginning from a starting place and ending when the first two players reach the end goal or they are ready to quit
4. The Maze must be different each time the player plays the game – which means you will want to use a random number generator to assist when generating the maze. It needs to be different at the beginning of each new game (the mazes cannot be pre-built).
5. Everyone starts at the same place in the game
6. Players take turns to progress from one checkpoint to the next through the Maze
7. There is one end goal (everyone that reaches the winning checkpoint wins)
8. At each turn, a player will move forward to the next checkpoint. There can be at most 3 different directions possible; the user gets to decide which direction to take.
9. At each checkpoint, the player spins for a reward. The following are some sample rewards (you may add others). A reward may be used now or saved for later. They can only be used when it is the player's turn. Sometimes, when the player spins, they will not get a reward.
    a. The player is allowed to ask a friend which direction to take next
    b. The player loses their turn and joins the closest friend that is ahead of them
    c. The player loses their turn and joins a friend that is closest behind
    d. The player gets to take another turn
    e. The player takes a reward from a friend
    f. The player gives a saved reward to a friend
10. Some checkpoints will be at a dead end (there are no directions forward). In that case, the only choice is go back to the previous checkpoint on the next turn.
11. One checkpoint will be the winning checkpoint. The first player to reach this checkpoint is in first place!

The following represents some ideas on the design – the first step is to plan what classes might make the most sense! Some of the basics that are part of an OO program for a CS Maze **could** be:

1. CS_Maze game – this would manage who is playing the game (the list of Players)
2. CS_Maze – this would manage the paths from one checkpoint to another and indicate where the players start and where the winning checkpoint exists.
3. Checkpoint – A location in the maze where we can go forward to up to 3 directions or go backward. Each checkpoint has a name. One checkpoint is the winning checkpoint. It is always a dead end. A dead end checkpoint means that we cannot go forward.
4. Reward – the prize that is obtained at a particular checkpoint. A reward may be received, played, or saved for later use
5. Player – A player may have a list of rewards, they have a name, and a list of friends. A player is located somewhere in the Maze at all times (unless they are a winner!)

6. Friends – A list of players that are in the same group. We need to find the closest friend forward in the maze or behind. We may want to take a friend's reward or give them a reward.

- Limit the number of checkpoints to no more than 20.
- Provide at least limited support to cyclical relationships for paths between checkpoints (e.g., It should be possible to go from Checkpoint 10 to checkpoint 11, then 12, and it has a path back to checkpoint 10)
- Limit the number of dead ends to no more than 5.

**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 problem. Some relationships should be hierarchical, others can be containment. With hierarchies always push the common elements up to the base class. Remember to avoid classes with only setters and getters! And Nodes will need to be classes instead of structs.

Here are some suggestions to start with:

1. Player IS its location plus name and friends list
2. The Map HAS multiple paths connected to Checkpoints
3. A Friends list HAS Players
4. Each list of players, friends, rewards IS a List

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. 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).

**Data structures**
This program should implement the following data structures:

1. A Graph to handle the maze, paths and checkpoints should be implemented as an Adjacency List. All traversal functions should be implemented **recursively**.
    a. An adjacency list for this assignment is to be implemented as a dynamically allocated array of checkpoints with head pointers to edge lists. Each head pointer points to a DLL of checkpoints that they are adjacent to
2. Implement the List as a linear linked list of arrays (of rewards, friends, or players)