// Keaton Spiller

// CS302

// 02/02/2024 Winter 2024

// Program # 1

Efficiency Writeup

Each Class has it’s job to complete, with similarities pushed up and differences derived. The Snow Race class has the similarities of the distance of each individual competitor of each type of snow sport, and the racing name of who each competitor was. However for each unique type of Snow Sport, they each have a unique data member about that class. The Ski’s have different types of ski such as racing skis, the Sled dog has the number of dogs and dog breed, and the Snowmobile had the percentage of gas, and mph. Each of these has a job to derive the differences between the types of classes. The CLL Node and LLL Node’s job are to act in place of a struct, allowing up to navigate between the nodes, and the data structures, Vector of Snowmobiles, CLL of Sled dogs, and LLL Skiers access this information to insert, display, and update the states of the races to compete in a simulated race.

My approach consisted of applying the different coding techniques shown in Lab1 and Lab2. This showed me how to simplify the Base class, and understand what each classes job is responsible for. Initially I had combined the Node and Data structure which added too much complexity, and when I started deriving and simplifying the abstract pieces of the classes I was able to have a cleaner and more organized hierarchical structure. My approach consisted of using random increments, and updating the racers in states of distances away from the finish line. When Are Racer surpassed the finish line, I would return a positive value that would end the race and display the racer with the largest distance.

Instead of using getters I would have the classes that manage the data perform actions needed in their classes. Display functions would cout the data up the chain to visualize the data members instead of getting the data back to the user. Compare functions to return comparisons of the data members, which will become extremely helpful when we get to graphs the future assignments.

Major changes I could have made in the project could be using a generic template to incorporate many types of Snow Sports with the same type of data structure. This would allow flexibility to what the sport could be about and could be applied to many more different cases. I could also add more exception handling, especially with repetitive error checks such as (!head). When the User enters a non integar, cin will give an error, and I would like to have exception handling to check cin.fail and cin.clear to reset the buffer. The winner function currently doesn’t account for ties within the cases and would need extra cases to return multiple winning user’s upon a tie.

The runtime performance of my algorithm is not as fast as it could be. I created random values to increment the racers without sorting the Data Structures. And while updating the states of each racer I would sum up a count of racers of have crossed the finish line, with a winner function in the base class to return 1 or 0 if the distance has surpassed the finish line. After a winner has been found I have a separate function to loop back through the racers and display the winning racer with the largest distance. If the racers were instead sorted, then we could reduce this last loop to directly find the winning racers with the largest distances.