// Keaton Spiller

// CS302

// 02/19/2024 Winter 2024

// Program # 2

Efficiency Writeup

My Design in this assignment is clean and sophisticated, and a large improvement from assignment 1. The generic nature of writing a Node and Double Linked List that can work for multiple different types of data is a powerful tool, and adding in a menu system alongside exception handling brings me closer to writing professional code that can be used in industry.  
 Since we were supposed to have sorted data in the Doubly Linked list the best approach was to insert data into the list sorted, this way the data is always sorted, and as data is taken out of the list, the data will stay sorted. I initially had the Doubly Linked List insert at the end of tail, which was very simple and effective, yet I would’ve needed to do a lot of re-shuffling of the list to get the list into sorted order which would have required some kind of sorting algorithm.

To fulfill the overloader requirements I created all of the required overloaders in the base class as friends which were accessible from all the other classes. In the comparisons I used the age of the animals to compare integars {< <= > >=}, and strings of the animal names to compare equality operators {==, !=}, and added the ages of the names alongside concatenating the names of the animals {+ and +=}.

After these were written I wrote exception handling to try and catch every cin overloader that handles the input of data, to make sure that all data coming into the DLL was able to be compared into the overloaders and fit the type of data. However, I also added error checking into the overloaders themselves to make sure that any anomalous data was caught because overloaders themselves don’t always know how to catch errors. Any non-integers and non-float values inserted by the user that were supposed to be float or integer were asked to be re-entered. Age was a special case checked to be positive values, because animals can’t have a negative age.

In order to avoid the overuse of getters I would make create any function necessary the handle specific data handle it in their own functions close to the class of the data. This is especially important for doing different comparisons and avoid the use of returning copies of values. In this assignment because the nodes and DLL weren’t inheriting from the data we had to use a getter in the Node to return the data to change, and update in the Node of the Doubly Linked list.

In my code major changes I had to make were in the use of using too many unique pointers. Originally I had every char \* using a unique pointer, and every node using a unique pointer, and it made it a lot harder to understand where I was making mistakes. Because I was unfamiliar with the unique pointer functionality I couldn’t tell if the program was breaking because of the pointer functionality or some other mistake of logic.

Additionally, I had the Node deriving from the animal object class, and I had a really difficult time understanding how we change the data that is nested in the containing relationship. I ended up using multiple scope operator calls and static casting to push data up to get the hierarchy working.

Since my program inserted as the Nodes were added to the program the runtime performance was at worst case big O N. Worst case being if we needed to go the very end of the list to insert. This was great because there wasn’t a need to go back through the Doubly Linked List to move Nodes Around and Sort, and because they are in sorted order, if we wanted to find A node by the age, we would be able to use searching algorithm that take advantage of a sorted list.  
  
  
  
12 Pet

13 / | \

14 Working Competitive Breeding

15     \            |           /

16 DLL Node Template

17 |

18        Data Structure DLL Template