Chase Verbout
CS202 Spring 2022
Program 2

Efficiency Write-Up

A. **Discuss the effectiveness of your design and classes**
    a. If the question is of how effective my design is given the parameters then I'd say it was. Creating the Nodes and DLL with class templates made for a multipurpose structure that could hold any of my reservation derivatives with working operations due to overloading in the base reservation class. I also managed to overload operators in the derivative classes.The DLL itself was effective in managing reservations because I ordered them by time. This way inserting at points was made a tiny bit easier with prev functions. Though this functionality would be best with a BST. The use of STLs (list and deque) was fun but pointless in a way because they can all be used fairly similarly and we don't measure our memory usage or function call times. Overall though, my program gets the job done.
B. **Discuss the validity of your approach**
    a. I think using class templates is the logical way to go about this program, however being asked to use two other STLs for the other derivatives feels illogical. Since the DLL and Node were made to work with any data type it would have been nice to focus more there. Though it is a show of just how convenient it is to use premade data structures. Being able to both trust and understand them quickly by online resources is very helpful. I especially appreciate the iterator capabilities. I'm hoping to use some sort methods next time.
C. **Analyze your results in terms of object oriented programming methodologies**
    a. My program, outside of the generic programming done, follows the layout of OOP really well. Also ignoring the interface cpp I use to demonstrate the classes it is very organized and well encapsulated. I even dipped my toes into the use of error handling with structs. This meant some of my functions were void return calls which felt odd but I really liked being able to just write in a throw. It looks much cleaner than 'true' and 'false' littered throughout. My classes implement copy constructors/assignment operators when working with dynamic memory and I overloaded derivative operators correctly with the base classes in mind.
D. **What major changes did you have to make in your design and explain why**
    a. I played around with unique_ptrs for a long while before I realized that they simply just would not work for what was being built. A DLL innately has two ptrs to a node excluding head and tail. A unique_ptr points to ONLY one node. So I shifted and played around with shared and weak _ ptrs before making it an only shared_ptr system with a created destructor to ensure deletion.

E. Would a different data structure work better? Which one and why...

a. There weren't many specifications for what the reservations had to do and so this is purely based on my ideas but I would have made a generic BST. Ordering by time in a DLL means that in the worst case we have a linear runtime. Ordering by time in a BST we would be able to guarantee a worst case runtime with a Big Oh of log(n).

GDB Write-Up

A. Discuss the effectiveness of the debugger

a. I love GDB. It has now saved me time on numerous occasions. It isn't the most user friendly interface and half the time I don't know what is on my screen but following my functions through their process and printing variables is immeasurably useful. I actually have been using it in CS201 too for assembly and it's crazy being able to see registers and their values as the program moves along. I look forward to unlocking more of the secrets it holds.

B. Discuss what problems it helped you solve

a. A specific example of what GDB helped me figure out this week was in working with my DLL. I found that the shared_ptrs, while solving my needing multiple pointers conundrum, were causing a memory leak. So I followed them and was able to see the way in which they worked. They hold a counter for how many ptrs point to that object. And so after some reading online and observing that each ptr had two (because next and prev) as it was trying to delete itself, it was only removing one of the ptrs thus not deleting the object because it still had something pointing to it. I decided the solution for me was to ensure the DLLs destruction with a destructor function that unwound the DLL calling reset on every object. This felt right to me because it meant that there would be no doubt afterwards if it was gone.

   I had another similar issue where my operator for comparing time ('<' and '>') was not working the way I intended so I followed it through my program and was able to see the values being compared in real time. From there, I fixed the issue quickly. Very helpful this week.