Review 3

CS202 Programming Systems, Summer 2020

Armant Touche

Date: August, 11, 2020

Design Analysis

The task for this assignment was to create a slack-like system called fish bowl which was assigned for use to experience abstract data type with operator overloading. For my abstract base class, I had one function in total with a default constructor and destructor. One of my function was a pure virtual function so I did not have to use run-time type identification for which of the derived classes. First, I choose to implement the derived classes first to experience operator overloading, lvalues, and rvalues. The insertion and extraction operators were the first to implement because I never used friend function before so I decided why not. The two operators were pretty straightforward. The trickiest operator was the compound addition operator because of the nature of the operands and the overloading explicitly needs only one operand. I kept receiving syntax errors because I couldn't understand what the requirement was but after consulting the diagram in lab six, I was to easily implement them. I just kept the diagram handy while implementing each of the two derived classes. After overloading some operators, I moved on the user class because I wanted to get them stored into the array of users and that did not take too long because now I feel comfortable with hash tables. I hashed each user by their password which was just a dynamic array of characters. Default size of the table was seven because I felt that good enough for the assignment since the graders spend most of their time reading the code instead of debugging the code. For the user class, I used the insertion and extraction operator but that's it. I had the usual constructor and destructor. Also, implemented a copy constructor. The user also doubled as nodes in a linked list which means that I had to implement getter's and setter's for the self-referencing pointers. I didn't the equality operator because I did not have enough time to implement it. Honestly, I wish the two-ways of communication requirement wasn't required. The threading could have been good enough for program three but it is whatever. Once I was able to create and store each user into the table of users correctly, then I moved onto to creating and sending messages between existing users. I had another main for testing at each step of the way. I found that doing small implementation and then testing was very beneficial and helped save a lot of time when debugging small problems. Once I was able to successfully send messages between each user, then I moved on to reading received messages. The implementation for reading messages was also straightforward. I pulled an audible by placing the thread called fish bowl into the user-table because I realized that each user does not have a forum but the "all" the users have a forum to post to. I implemented small parts of posting and reading post function because that was much easier because the data structure was more central versus the inbox present in each of the users in the user table. The tricky part with the inbox and fish bowl was figuring out how to the store them in a binary search tree because the number of class scopes I had to go through to store them. The storing of data into the two binary search trees both possess the same logic. Once the logic was good for the inbox, I just copy and pasted the code from the inbox implementation into the thread class. Of course there were small differences but overall, the logic was the same. Overall, the program took a lot of time because of the requirement of having two ways to communicate but I did learn a lot with this program. Now on to Java...

Debugger

GDB was perhaps the biggest time saver because because working with different data structure was a task. Stepping through GDB really solidified how operator overloading worked and was especially important when tracing insertion and extraction operator in both of the derived classes that needed to be implemented. Another debugging program that aided me was valgrind, which helped with memory leaks. I used watch, breakpoints, and backtrace a lot to aide in debugging program one. Working the base class and it's virtual methods in GDB really helped solve issues like reading incorrectly or displaying data members from the base class and not any of the derived one. I realize a little to late into programming that dynamic binding was only useful for function and if you needed to access the derived classes data, that wasn't going to happen. I had to pull an audible a little late in the game but I think I am pretty set up for program number two. I just need to learn return type identification and abstract base class concepts which will be covered next week. I will there was a straightforward way to access data members in a derived class via dynamic binding but I need to practice anyway. No worries.