Souriya Khaosanga

CS202 HW3

HW#3 Write up

The purpose of this program was to create a blog using the doubly linked list as the data structure. In addition to using a doubly linked list, this program also implements operator overloading and dynamic binding. This program has the ability to write a post, edit the post, add reply/comments, and remove all reply/comments. These functions are implemented within the class blog, which performs all these functions.

The data structure used for this programming assignment works well because the use of a doubly linked list means that the data is always being modified and data is being removed in no particular order. For my design, I used an index number, which kept track of the location of the reply/comments. This data structure could have easily been done using simple linear linked list because the previous pointer was not utilized in this program. Looking at performance, it runs fairly well, but we have extra memory that is not used in the program.

Looking back at the program, I would have used a linear linked list instead of a doubly linked list. This would have allowed me to reduce about of memory used because of the previous pointer. I still would have been able to do all the functions in the required, which included write, post, reply and remove comments. In addition, I would have created objects of replies instead of pointers. This might have been better for run time because I would just have direct access to the array instead of having to traverse the array of pointers. However, I would have sacrificed memory for run time performance.

Some of the efficient items in my code are the dynamic binding. Dynamic biding was efficient because it allowed me to have the program dynamic decide which type of derived class to create. This reduced the need for extra functions if each class was not part of a hierarchy. The dynamic binding was quite nice for reducing the total number of functions needed to be implemented.

Items which were not efficient would be when used RTTI to check to see what type of dynamic memory I had. This was used for the copy function because in order to create a copy, I would need to identify the type of derived class to create. I ran into issues with this part because creating this was not as simple as I had thought. If I wanted to make the overload for the assignment operator for the blog class, this would have required me to have copy functions for each of my derived classes in the data structure.

Given more time, I would have done quite a few things. First I would have completed my operator overloading for the blog class. This is actually quite complicated because my thought is that I needed copy functions for each class depending on the data structure. Where in the class, which only contained a string, it was quite simple for overloading the operator. Secondly I would have done more testing of functions. In my code I was able to check that it performed the base 4 requirements for the assignment, however I would have liked to incorporate operator overloading at the blog class. Third, I would have created a user interface for the program that allowed input from a user. For testing purposes, I found that user input should be the last and final addition because of the time used for inputting data from the user. Given that every time I want to check my code, I have to input from the user, I found this way too time consuming. So given more time I would implement the user interface for user input.

GDB Debugger

I used gdb at various locations of my code. The main place I used it was when I was trying to figure out what my code was doing. For example, one part of my code I needed to add comments/replies. Instead of indexing those comments and replies, I had a counter, which was incremented each time I added. However, one issue I had was I was incrementing at my Boolean value flag. My adding reply was functioning but it was only adding at the index pointer value of true (one). So my code was functioning properly, but it was replacing that one index.

So when I ran it through gdb, I found that I was incrementing the Boolean value. Without gdb, I might have been trying to decode this for days because the code was running and compiled, but it was not functioning properly.

Another location which I found gdb was helpful was when I needed to keep track of the variables. For example, I needed to see at which the pointer was pointing. In gdb, I am able to dereference at a particular step in the function, which was extremely useful in identifying what I was pointing to.

In addition, gdb was helpful in figuring out where I segmentation faulted in my code. Most of the time, I would just go until my code segmentation faulted and once I found out where I segmentation faulted, I would insert a statement to check if I it exist then go there. This has been a great solution for segmentation fault is to always check before I go. Most of my code does not really segmentation fault because of gdb.

It allows me to figure out where I am segmentation faulting though the use of the gdb. If I didn’t have gdb, I would have to individually comment out certain spots of my code to have my code not segmentation fault. This however is extremely time consuming and rather a chance of luck to allow my program not to segmentation fault.

The gdb debugger was also useful for setting the break points and stepping through each of the functions. What I would do is set the break point at any particular function, which was not functioning properly. The gdb allowed me to only step through those functions and I could easily follow through each step. It also flows into the go to functions, which helps identify all possible problems.