Andy Franck - CS 163
Program 2 - Efficiency Writeup

Similarly to the first assignment, there was not much freedom in regards to the data structure. The type was already given, with the freedom being in regards to using a struct/class for event data.

I chose to use a struct to store this data. Although the information was more complex than the first program, I could not find any reason to use a class. The struct was simpler, and I had no issue with reading any of my data. I used two structs for my queue and stack nodes, and they also worked well.

In regards to the stack, the linear linked list holding an array of elements was an interesting way of solving the problem. I think it was considerably better than using just a basic linked list, because of the challenges with large amounts of apartments. Although this method is not much better, the memory overhead is smaller and therefore I think it is better overall. Because it is a stack, there is little traversal needed and therefore it is a very fast data structure.

Again, my knowledge in data structures is limited, but it appears that from my knowledge this is the best way to store the data in stack form. We have the benefits of faster access with the arrays, but also can store an unlimited amount of data with the linked list. Because traversal is not necessary, I did not have to worry much about traversing through the array and nodes, besides in my destructor and display functions. The top of the stack was extremely easy to access because it was wherever the top_index of the array was in the current node, so I could read or remove that data directly when I was on that node.

An efficient part of my program were the management functions for both my stack and queue user interactions/menu. This was extremely beneficial for organization and modularity on my end, however I might have made these into a different file, because front end devs would have an easier time accessing them. With that said, however, it made main extremely easy to navigate and I suspect with a more complicated main function this change would be even more beneficial.

Similarly to program 1, I was able to shorten and simplify my code by using strlen when allocating memory. This made it extremely easy to peek/pop, and enqueue/dequeue the data structures. I could easily copy the data into a passed-in function for my peek functions, so those functions were extremely condensed/simplified.

On a similar node, I also was able to condense my pop stack considerably by noticing that when I did not need to go to the previous node, I could just subtract one from my index. This works because if I add another item, it will automatically overwrite the previous data in that compartment of the array, and if it is never overwritten, then the destructor will automatically deallocate that memory when I close the program. I suppose that this could cause unnecessary use when popping large numbers of events, but the scale of this program is small enough that I decided it was not an issue.

The main difficulty I faced was properly deleting memory in the array for my stack. Because of how I was popping the stack, it was necessary to possibly go past the top_index to make sure I was cleaning out every item, so I needed to check the entire node for data. This meant I had to first check if they were null or not, because I was getting references to uninitialized variables. This stumped me for a while, but I was able to add a few checks that fixed the issue. In the queue, I needed to figure out how to properly break out of a circular linked list. First, I forgot that I could not use a while temp != null, because it was circular, so I had to store my original rear and check for that comparison instead. Doing this recursively was an extra challenge, since I needed to then add checks for if I was back at the start.

If I had more time to work on the project, I would have divided up my .cpp file into multiple files for my queue and stack, and separated them a bit more. Because I haven't done this in my program, my .cpp file is very long, and pretty difficult to navigate. Dividing these up into two files would have made my program more modular and convenient in the long run. I was having trouble navigating my massive file in vim, so I think some more advance planning to divide would have been ideal. I have been hesitant for some reason to create extra files, but I believe I should get more comfortable with it in the future.