John Gill

Dr. Tala Talaei Khoei

CS 5008

10 August 2024

Final Project

My final project was creating a Healthcare Information System (HIS) in the programming language C. There were three levels of operations in my approach to building the code. Those levels, in order, were: (1) making a series of pages that ask the user if they want to work with patient or employee data, then create, add to, view, or delete that data from specific locations in my file explorer; (2) creating functions that automatically scan those folders holding existing patient and employee data from previous runs of the program into a balanced AVL tree so that they could more efficiently be handled in working memory for other, future functions; and (3) creating a priority queue patient check-in function, as well as a dequeue function that accounts for severity, creates new patient profiles for patients that aren’t already in the system, and shows the entire current queue as it is standing. I didn’t intend to include the queue while working on step 1, so some of the functionality of how the queue works isn’t how I’d prefer it to be. For instance, it’ll become apparent that the queue’s new patient check-in uses the same function for adding a patient into the system without the queue, and because of that, also prompts for a new visit. New visits essentially open a .txt file that allows the user to put in the relevant information for that visit. I might be able to correct that before the presentation on Tuesday, however.

Writing to, reading from, and iterating through files of patient data uses the windows.h and direct.h preprocessors in the preprocessor directives section, since all handling of patient and employee data happens within AVL trees. That section also includes the standard stdio.h, stdlib.h, and string.h preprocessors for handling strings, scanning functions, and pointer functions. Also, the medical history of patients is loaded into a linked list under each patient upon loading the program so that it can be more efficiently read through. As for the code, from a UI standpoint, the user will see a menu that has four options: (1) processing patient or employee data; (2) checking in a patient for a priority queue; (3) processing the queue; (4) and exiting. Option 1 will ask if we’re working with patient or employee data. Choosing patient will prompt the user to view all, add one, search for one, or delete one. Viewing all will show the user all patients saved. Adding one will ask for a first name and last name, then check if one with that name already exists, and if not, will create a new patient with all the patient information, as well as make a visit 1. Searching for one will prompt for first and last name, pull up the patient, then ask if we want to display their information or handle visits. Displaying information will show all the information of their profile, and handling visits will prompt the user if they want to view previous visits, or add a new one. This works by reading from file paths in my “HIS information” folder, then looking at the “patients” folder within that. Each patient has their own folder that’s automatically created when a new patient is added to the system, and in that folder, a .txt file with their information, as well as a “Visits” folder that reads the amount of visits, and if they choose to add a new visit, creates a new .txt file called “Visit #”, with “#” being replaced by the amount of files in that folder plus one, which contains all the information written to it. All the same can be said for employee information, except employees don’t contain visits, and it also has a function that lists employees by department.

Choosing option 2 prompts the user to input the first and last name of a patient, alongside severity of their visit, then checks to see if the patient is already in the system by checking for a file with their name in the “Patients” folder filepath. If it exists, it prompts the handle visits function that asks if they want to view previous visits or add a new one. Whichever they choose, after they either view the previous visit of their choice, or add a new one, the patient will be checked into the queue. If the patient doesn’t exist, it prompts the user to input all of their information, then automatically creates “Visit 1” with the information the user inputs – at which point, the patient will be added to the queue. Severity determines where in the queue the patient will be put, and new patients with the same severity as already checked-in patients will be put to the back of that specific severity choice (on a scale of 1-10). Option 3 of the main menu prompts the user to process the patient queue. Once they do, it takes them to another menu that asks if they want to view the queue, or dequeue the next in line. Viewing the queue shows everyone in the queue, with their first and last names, as well as their severity level. And dequeue simply pops off the next in line.

Option 4 is the step that frees all the memory of the AVL trees of patients, and the linked list of patient medical history. Everything I’ve listed here is everything I intend for the program to do, though I’m still debugging. Currently, it appears I’m having trouble getting all patient and employee data to load properly into the necessary AVL trees and linked list. Trying to add existing patients to the queue doesn’t work at the moment; it doesn’t see them in the proper filepath, and it also doesn’t create the .txt file in their folder holding all of their personal information. Also, since deciding to use trees and linked lists, viewing all patients or employees doesn’t show anything, even though they’re located in the proper filepaths as well. These seem like fairly minor inconveniences that I should be able to sort out before presenting. Again, I think most of my issues arise from the introduction of trees and linked lists loading the data into working memory upon launching the program. But I will continue debugging.