Report for LAB 2

Leticia Dupleich Smith

Deadline: October 16, 2024 23:59

Task description

The objective of this project was to simulate a doctor's office waiting room. The program offers four actions: adding a new patient (N), treating a patient (T), displaying the list of patients (L) and quitting the program (Q). When adding a new patient, details such as name, age, and priority are collected about them. The priority helps to determine which patient to treat - patients with a higher priority get treated first. Moreover, when printing the list of patients in the waiting queue, the order is also from highest to lowest priority. When the list of patients is printed all of their details must be included as well. Therefore, the main objective of this project was to ensure that the doctor's waiting room operated effectively, that is, that the patients were treated in the correct order and that their data was properly stored and displayed.

Structures and Algorithms

The main data structure used in this assignment was a linked list. Typically, linked lists save integers in each node, so the challenging part of this assignment was to figure out how to make the linked list save the names, ages and priorities of the patient all in one node. To make this possible, I decided to create a custom data structure called *patients*. This structure includes the name, age and priority of the patient as well as a pointer to the next node (patient). Essentially, each structure corresponding to a patient was used as a node in the linked list. This was an appropriate method since a linked list allows for the storage of a lot of separate nodes of data which in this case could be used as patients. Furthermore, a linked list also allows for the simple insertion and removal of patients.

Aside from linked lists and the *patients* structure, I also implemented functions that would aid in the addition, treatment and displaying of the patients. The first function *addPatient*, called by the command "N" is used to add a new patient to the list. To avoid having to search for the highest priority when a patient had to be treated, the *addPatient* function sorts patients when they are being added to the list. The function traverses through the list to find where the new patient should be inserted according to its priority levels. This way the patient with the highest priority is always at the start (head) of the list. The *treatment* function, triggered by the command "T", treats the patients with the highest priority. Since the list is already sorted when a new patient is added this function simply prints the name of the patient at the start of the linked list and frees the memory allocated to the storage of that patient. The function *printList* is another important function that runs when the user inserts "L". This function recursively prints the details of all the patients who are currently in the waiting queue. Additionally, the function *freeList* ensures that all the memory used to store patients' information is freed when the program exits - this prevents memory leaks. The *main* function was also crucial for the program. The function creates a linked list called *doctor*. Moreover, this function asks the user to input a command ("N", "T", "L", or "Q") which calls the corresponding helper function.

Evaluation of the program

To test this program, I used the “make test”, random inputs and CodeGrade. Passing the “make test” command was my first priority because that way I would know that the code did not have any warnings or errors. Passing this command also ensured that the program was working for the two test cases that we were provided. After it had passed this test, I was putting random inputs to verify that the program would also function for extreme cases like empty lists or long names. Lastly, I put the program through CodeGrade where it would be tested against more examples, memory leaks, and formatting.

Nonetheless, when testing the program there were bugs and errors that were encountered. The biggest challenge was adding new patients in the correct order. Initially the function only worked starting from the patient with the second highest priority. For instance, if there were four patients (patient 1 with priority 10, patient 2 with priority 7 and patient 3 with priority 5 and patient 4 with priority 1) the code would treat patients 2-4 even though patient 1 has the highest priority and should have been treated first. I was able to pinpoint the issue to the fact that the program was not taking into consideration that the new patient was of highest priority, in other words, it was not taking into consideration that the head of the list could change. Thus, the program would continue to point to the old head, which had already been freed. This issue was resolved by re-assigning the head of the list each time a patient had to be treated by using the command “doctor = treatment(doctor)” rather than “treatment(doctor)”.

Conclusion and Self-Reflection

This assignment was really helpful to me because it helped me to better familiarize myself with linked lists and structures. I learned that linked lists can store multiple pieces of data per node which was extremely interesting and could be helpful for future projects.

Throughout this process, I really struggled when trying to figure out how to save the patient’s data. I did not fully understand how it was possible for a linked list to save three different pieces of information about each patient (name, age, and priority) in a single node. In order to make this clearer, I created a paper diagram where I had a *patient* structure in each node and then the pointer to the next node. By creating this I was able to understand what I had to program and how to program it. The easy part came afterwards when I had to code the program. Once I knew how I wanted to save information, coding the functions was relatively easy, as they were similar to exercises from weekly assignments.

For future assignments or projects I would like to know what else can be done with linked lists. I am interested in learning what other structures and functions can be combined with linked lists to create more complex and efficient programs. Furthermore, I would also like to learn what other ways this program could have been made; for example, perhaps this program could have been made with trees rather than linked lists.