1.

void sort(struct Appointment appoints[], int max)

I use this function in here to sort the data of patient’s appointment. This function sorts the Appointment struct array's values in ascending order using bubble sort. Otherwise the data will mess up and won’t show from the small to large number.

Bubble sorting is a simple sorting method. Its main function is to rearrange a set of numbers according to their size so that they appear in ascending or descending order. Although it is not the fastest sorting method, it can be used for teaching, small-scale data sorting and comparison with other sorting methods. Bubble sort can also perform well if the data is mostly sorted, but in most cases other more efficient sorting methods are more appropriate.

void inputString(char\* str, int minLength, int maxLength)

I use in here because I want to print “Invalid %d-digit number!” this line. Everything is the same as inputCString but I just change the output.

This function is used to get the string entered by the user and verify whether the length of the string is within the specified range. If the length does not meet the requirements, the function will ask the user to re-enter until the condition is met. The main purpose of the function is to ensure that the length of the string entered by the user is within the specified range to prevent incorrect data.

int validAppointment(int patientNumber, struct Date date, struct Appointment\* app, int maxAppointments)

I use this function in here because I want to verify that the given patient number, date, and appointment information is valid. Specifically, it searches through an array of appointments to see if there is an appointment matching the given patient number and date. If a matching appointment is found, the function returns the index of the found appointment in the array; if no matching appointment is found, the function returns -1.

2.

The reason why uses less scanf is to let the programmer makes sure that user input is consistent and reliable throughout the program, reducing validation code and making it easier to add further improvements. By putting input validation into these special functions, the code becomes cleaner and easier to maintain.

Scanf is easy to have unexpected problems: such as eating unneeded input characters, or not properly handling mismatched input formats.

Handling failures: Handling scanf errors and input failures can become complicated, making programs difficult to maintain.

Unsafe: Using scanf to read a string may cause a buffer overflow, and this will result in program bugs.

I use more than one scanf, the reason is I already try my best to fix it, but I don’t have enough time to do it. And I don’t want to submit late so that is why I have more than one scanf in my milestone.

3.

void sort

Module Placement: I create this is to solve the problem with the mess up number of data. Therefore, I put the prototype for this function in clinic.c it helps to keep the code organized and easy to maintain.

void inputString

Module Placement: I create this is to solve the problem with the error message of Invalid digit number. Therefore, I put the prototype for this function in clinic.c to make the output is what I want.

int validAppointment

Module Placement: I create this function is to solve the problem with the appointment that make from the user. Therefore, I put the prototype for this function in clinic.c. This feature helps ensure that user input is valid and accurate when they type, which is very important to the overall functionality of the program.