1. What is your favourite string function from the string library that you have used in this application? Describe why?

* my favorite function is “strcmp”.
* Very efficient 🡪 pointer arithmetic
* It stops as soon as it reaches the end of the string or there is mismatch
* It compares each index of a string array, character by character 🡪returns value relative to the first array
* Don’t have to write/test/debug code, no headers or definitions 🡪readily available with one line
* Example of comparing the passwords.

My favorite function is “strcmp” mainly because of its usability. It compares the two arguments character by character for each index, returning a value relative to the first string which describes if the arguments match or not. By using this error proof library function, it saves me from the tedious responsibility of creating a lengthy function with the same complex functionality,thus, improving the readability of the code. For example:

// prototype

int stringCompare(const char\* strA, const char\* strB);

// definition

int stringCompare(const char\* strA, const char\* strB)

{

// compare strA to strB (result: relative to strA compared to strB)

int i;

char a, b;

// Loop: length of strA or until strA is different from strB

for (i = 0; strA[i] != '\0' && strA[i] == strB[i]; i++);

a = strA[i];

b = strB[i];

return (a < b) ? -1 : (a == b) ? 0 : 1;

}

When using string library, the above code is reduced to a simple line:

if (strcmp(str, account[index].login.loginName) == 0)

Furthermore, it is extremely useful for validating strings entered by the user which is a fundamental component of every program. For example, when I was authenticating the user input during the login process in milestone 2, this function allowed me to focus on other aspects of the login process like printing the error messages when there was a mismatch, instead of focusing on creating the comparison logic. As a result, its functionality eased the task of comparing strings.

Moreover, this string function is extremely efficient because it uses pointer arithmetic which is an optimal technique for comparison. More importantly, it stops comparing the strings as soon as it reaches the end of the string (null byte) or there is a mismatch between the string characters. Therefore, it returns the value that represents the differences as soon as any of the above conditions are met.

In milestone 2, I created a total of 8 functions after spending a great amount of time considering the factors of cohesion and coupling when creating the modular design of my program. Among the functions, only “getZeroPositiveInt” was included in the “commonHelpers” module because this module is totally independent and doesn’t require the existence of an account or ticket structures; similarly, this function could be reused in other modules where the general purpose of taking input of an integer that is greater than or equals to zero is required.

“displayAgentTicketsHeader” and “displayCustomerTicketsHeader” were included in the “accountTicketingUI” module because they perform one specific task of displaying the ticket table header to the agent’s or the customer’s screen. These were created to keep the table header separate from the contents of the table which helped to improve the readability of my source code.

“displayCustomerTickets” and “displayAgentTickets” were included in the “accountTicketingUI” module because they both have a single task of displaying the details of the customer’s or agent’s tickets to the screen. “displayCustomerTickets” was created for handling the customer’s tickets based on the matching account number while “displayAgentTickets” is a multi-functional function that displays either new or active tickets based on the menu selection of the agent.

“displayDetailedMessages” was included in the “accountTicketingUI” module because it performs a single task of displaying the messages between the customer and the agent. This function was created due to its reusability because it is called every time a customer or an agent wants to view the messages. “validateTicketNum” was included in “accountTicketingUI” because it validates the ticket number entered by the user to ensure the customer or agent doesn’t access other tickets which are not displayed. This process must be done before the displaying the messages to the screen.

“loginProcess” was included in the “accountTicketingUI” module because it requires user interaction for entering the details of the account based on the prompts on the screen. It performs a single task of authenticating the entered details and gives the user access to the menus if the user is authenticated.

All the above functions in the “accountTicketingUI” module are not very data specific processes, and only display the data; therefore, I included them in this module.