**SFT221- Reflection on Unit Testing**

In this project, I implemented a C program to gather customer information, ensuring that valid inputs were provided for each field, including first name, last name, street address, city, province, and postal code. The primary objective was to validate user inputs and handle incorrect or blank entries by re-prompting the user until valid data was entered. Throughout this process, I aimed to develop a robust program that adheres to good programming practices, including input validation and error handling. To accomplish the task, I leveraged several key concepts and functions in C:

Input Handling with fgets: The fgets function was used to read strings from the standard input. This function allows for safer input handling compared to gets, as it prevents buffer overflow by specifying the maximum number of characters to read.

String Validation: Functions like isalpha and isdigit from the ctype.h library were utilized to check whether characters were alphabetic or numeric. This helped in validating names and postal codes.

String Manipulation: The program used basic string manipulation techniques to trim newline characters, convert characters to uppercase, and ensure proper postal code formatting.

The function to gather customer information, enterData, was designed to handle various edge cases, including:

* Ensuring that no blank lines were entered.
* Validating that names only contain alphabetic characters.
* Formatting the postal code correctly and converting it to uppercase.

Here is a detailed assessment of how I ensured the correctness of the program:

1. **Blank Line Handling:**

For each input field, the program checks if the first character of the input is a newline ('\n') or if the string is empty. If so, it prompts the user again until a valid input is provided.

if (data->firstName[0] == '\n' || data->firstName == '\n' || !checkalpha(data->firstName))

1. **Alphabetic Validation:**

The checkalpha function was implemented to ensure that the strings for names, city, and province contain only alphabetic characters.

int checkalpha(char string[]) {

int check = 0;

int i;

for (i = 0; string[i] != '\0'; i++) {

if (isalpha(string[i])) {

check = 1;

}

}

return check;

}

1. **Postal Code Formatting and Validation:**

The checkCode function validated the postal code format and converted it to uppercase. It handled cases with or without spaces and ensured the correct format.

for (int i = 0; data->postalCode[i] != '\0'; i++) {

data->postalCode[i] = toupper(data->postalCode[i]);

}

Testing the function to ensure that blank lines are not entered involved the following steps:

I ran the program multiple times and entered various types of inputs, including blank lines, strings with digits, special characters, and correctly formatted inputs. This helped verify that the program correctly re-prompted for invalid inputs. It also broadened my view on which inputs I can actually put in, what will crash the software as well as which ones will scrape just by the edge and not break.

In conclusion, by reflecting on the implementation and thoroughly testing the program, I ensured that the function to gather customer information was robust and handled invalid inputs appropriately. Further improvements could involve automating the testing process to enhance efficiency and coverage.

**While automated testing was beyond the initial scope, it is possible to automate input testing using techniques like input redirection or writing unit tests with mock inputs. Testing frameworks such as CUnit or writing scripts to simulate user inputs could be used in a more advanced setup.**