**WORKSHOP-1**

In this documentation, I would like to explain some bugs that are identified from the code that is provided in Workshop 1‘s pdf file.

To begin with, I would like to highlight the most important factors that can cause some debugs in this code. In this code “ char buf[20]; , strlen(prefix); and strlen(suffix); “ are used in order to restrict but char string size and count the length of prefix and suffix arguments that are used in the functions as parameters. While examining in detail them, I figured out some bugs that were caused because of these structures. In the below, I would explain how these bugs happen step by step with provided screenshots.

1. “ char buf[20];

* This is a character string that is restricted to a maximum of 20 characters. If the user tries to input more than 20 characters for the prefix, the program cannot respond accurately because the size of the string is limited by 20 characters. Let’s picture how to work on Visual Studio.
* Add more than 20 characters inside of the main function.

A screenshot of a computer

Description automatically generated

* Getting crush and error after running this code.

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

* In case of inputting so many characters from the user, the program does not run properly, so this bug should be solved to create more reliable code for the users. The back-end developer can discard the restriction of the char buff string, and create a more accurate structure that allows more characters.

1. Special Characters

* Some keyboards might contain different types of characters such as İ , Ş , Ğ, Ü, Ç, Ö. Those are used in other alphabetical characters which can be used by the user because the program should be universal, and it should support other special characters while servicing. In this case, when I use those characters, the program gives me weird output as I picture below.
* Add “üÇşALİ” characters, inside of the main function as a prefix.

A screenshot of a computer

Description automatically generated

A computer screen with a black screen

Description automatically generated

* In this case, the output is different than what we expect because üÇşALİ characters cannot be identified by the program. The expected output should match with aimed input, so this is the other bugs which should be fixed.

1. strlen(prefix);

* This function is special a function that provides the length of the string until the null terminator (‘\0’). This function is quite useful to compare strings in terms of their length.
* In addition, strleng () function can identify so many characters that we can use in our many reel cases, so it is useful to the main goal of these specific characters. However, this function can underperform some specific characters, that’s why I examine this function in the code over and over because this is the weak part of the code. Let’s picture how the bug occurs in the code.
* Add “ é “ character inside of the main code as a prefix.

A screenshot of a computer

Description automatically generated

* Getting crush and error after running this code.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* As you can see, “é” is not identical to getting the correct output, and also after trying to run this code, the system gives an error and gs\_report.c file, but in any possible real scenario, the user might add “é” character, so this cause the bug.

1. Null terminator ( '\0')

* strleng() function provides the length of the string until the null terminator (‘\0’), but if the user put \0 as an input, the program cannot work accurately, so this bug affects the program negatively. Let’s explain how the bug occurs below.
* Add \0 characters inside of the main function.

A screenshot of a computer

Description automatically generated

* The program gives incorrect output rather than what I added in the main function

A screenshot of a computer

Description automatically generated

* In this case, \0 can be an issue in the case of input by the user, because strleng() can not identify other characters after ‘\0’. This is another bug which should be fixed.

**REFLECTION**

1. Actually, the most effective methods for identifying and resolving issues in code are frequently testing and inspection. when inspecting can concentrate on the code's quality and any design flaws, testing can assist in locating faults when the programme is in use. However, I discovered from my earlier findings that testing was a better method for trying out various strategies because it allowed me to enter different inputs and observe what happened as the algorithm was being executed. In order to improve the program's stability and maintainability, both approaches must be applied.
2. Yes, I do find it fairly difficult to find programme problems. This could be explained by the fact that this is my first time testing and I'm still working to fully grasp what components of the programme can be changed to allow for in-depth testing and examination. I anticipate that as I become more accustomed to the codebase, my ability to identify and fix problems will advance.