SFT 221

Workshop 5

Name:	Gyeongrok Oh
ID:	119140226
Email:	goh3@myseneca.ca
Section:	NBB

Authenticity Declaration:
I declare this submission is the result of my own work and has not been shared with any other student or 3rd party content provider. This submitted piece of work is entirely of my own creation.

Updated code is below

Line Number	Filename	Updated Line
72	stringhelp.c	result.numLines++;
77	stringhelp.c	if (isNumber(str + i, sp)) {

What Was Wrong	VS Techniques Used
This code should be output based on \n, but the previous code did not use the number of characters properly in \n To explain, it is a function that analyzes the string str to extract information such as words, numbers, and new lines and returns it in the form of structural StringIndex. Increase the result.numLines each time you encounter a space character or a new line character during analysis, and record the number of lines found to date. Therefore, result.numLines++; is the code used to count the number of new line characters in the input string str.	Breakpoint
The code if (isNumber(str + i, sp)) is checking whether the substring of str starting from index i and with a length of sp represents a valid number. The reason why sp is used as the length parameter in isNumber() is that sp is calculated as the length of the next substring of str that contains consecutive non-whitespace characters. It represents the length of a word or number in the given string. By passing str + i as the starting address and sp as the length to isNumber(), the code checks whether the corresponding substring is a valid number. By passing the appropriate starting address and length to the isNumber() function, you ensure that it operates on the correct substring of str, allowing it to determine if the substring represents a number or not.	Debugging toolbar and shortcuts, Breakpoints

What was the most useful debugger tool you used to find the bugs? Why was it more useful than other techniques you tried?

The debugging toolbar/shortcuts and breakpoints are two commonly used and special tools. The debugging toolbar/shortcuts provide quick access to essential debugging actions, allowing developers to control the debugging process efficiently. On the other hand, breakpoints enable developers to pause the program's execution at specific lines of code, helping to identify the cause of unexpected behavior or bugs. By breaking execution and inspecting variables, evaluating expressions, and stepping through code, breakpoints allow for focused investigation and analysis. While these tools are handy, it's important to remember that the effectiveness of debugging techniques may vary depending on the nature of the bug and the developer's familiarity with the available tools. Utilizing a combination of debugging tools and techniques can significantly enhance the debugging process.

Which debugger features did you NOT use? Why did you not use these features? For each feature you did not use, give an example of a problem you would use it for.

The Step Over feature in a debugger allows you to execute the following line of code in your program without stepping into any function calls. It is beneficial to have a function call you to believe it is working correctly and want to concentrate on the surrounding code.
For example, you have a complex sequence of function calls in your code and want to examine the code's behavior outside a specific function. By using Step Over, you can avoid diving into the details of that particular function and instead move directly to the following relevant code snippet. This way, you can continue your analysis without getting caught up in the intricacies of the function call.
Step Over helps streamline the debugging process by allowing you to focus on the parts of your code that require immediate attention while bypassing the details of functions that are already tested and functioning correctly.

Which do you think is a faster way to find bugs:
a. Using the debugger alone,
b. Using print statements alone,
c. Using the debugger and print statements?
Justify your answer.

Using the debugger and print statements together provides a robust and comprehensive approach to debugging. The debugger allows for in-depth analysis of program execution, variable values, and control flow, while print statements offer real-time logging and visibility into specific code sections. By combining these two approaches, developers can quickly identify and fix bugs by leveraging the strengths of each method, leading to faster and more effective debugging. Additionally, the debugger and print statements complement each other, providing different perspectives and insights into the code's behavior, making it easier to uncover subtle bugs and understand the program's runtime dynamics. Together, they form a powerful duo.

How did you test the program to find the bugs and to make sure you had fixed the bugs? Did you use any additional test data other than that supplied? If so, describe the techniques you used to create the test data. How confident are you that you found all the bugs?

When testing and debugging a program, developers employ unit testing, integration testing, functional testing, and more techniques. Additional test data is often created to cover various scenarios and edge cases beyond the supplied test data. This can be done manually based on requirements or using automated tools for generating test data. Bug fixing involves identifying, reproducing, and fixing bugs. After making code changes, tests are rerun to ensure the bug is effectively resolved without introducing new issues. Achieving 100% confidence in finding all bugs is challenging, as bugs can be elusive, and program complexity is a factor. Continuous testing, code reviews, and user feedback improve bug detection and instill confidence in the software's reliability.