1. **Introduction / Purpose / Intent**

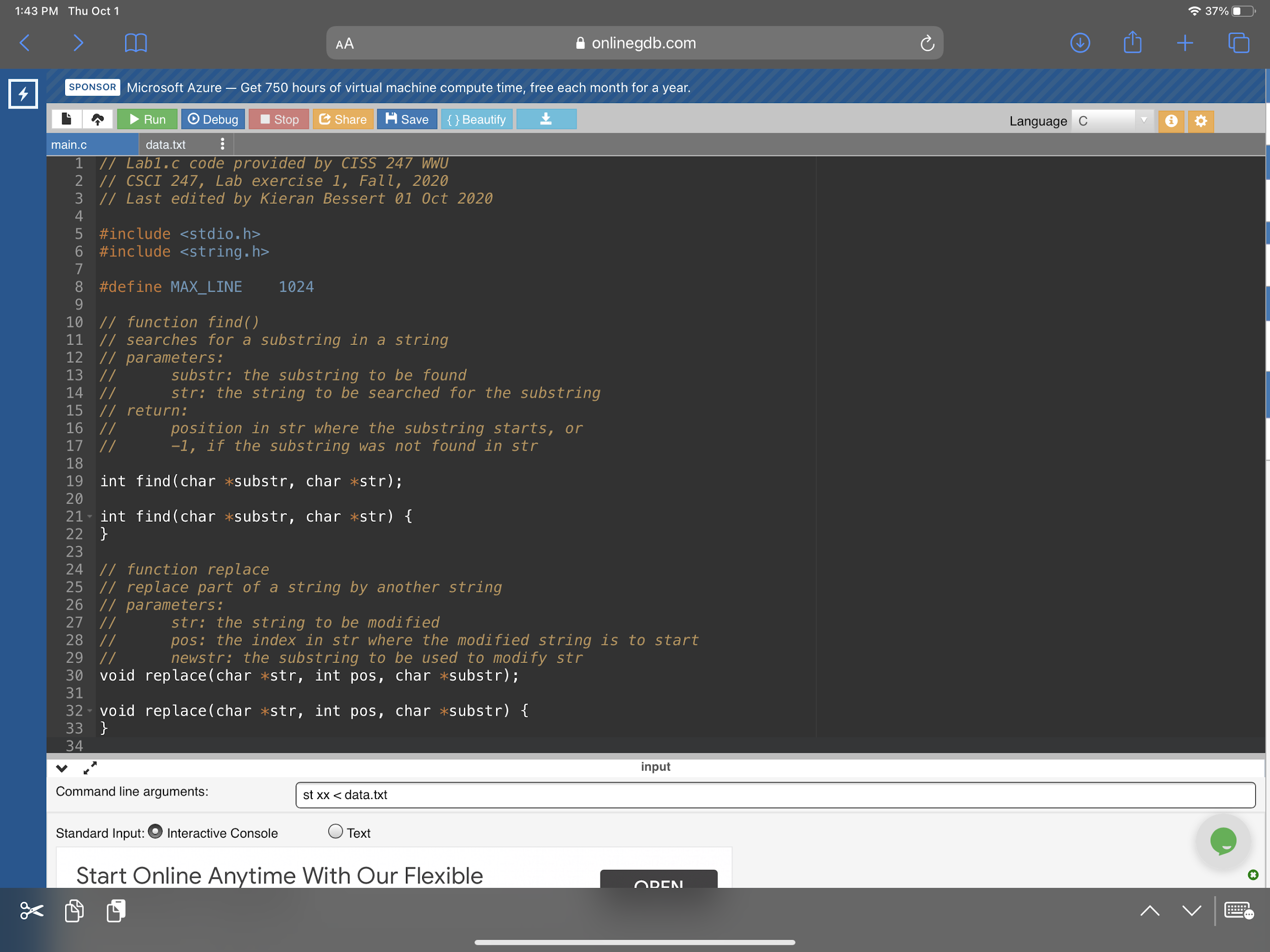
For this assignment we were tasked to write 2 functions for the Lab1.c program provided to us, 1 of which is named find() and the other named replace(). These functions will be utilized inside a pre-written program to find a substring of 2 chars inside of a string and replace the matching portion of the string with the substring.

The former function, find(), takes 2 arguments, both of type character, the first of which is the substring of characters wished to be found and the second is the string to be searched. The function returns the start index of the string where the substring match was found. If no matches found this function returns a -1.

The later function, replace(), takes 3 arguments, the first and last of type char and the middle of type int. The first variable is the string to be altered, the second variable is the starting position of the array to be altered and the third variable is the characters to be placed at the starting position.

1. **Process**

To begin, I downloaded the provided files ‘Lab1.c‘ and ‘data.txt‘ from the courses canvas files page. I then Imported these files to the compiler I will be using, <https://www.onlinegdb.com/online_c_compiler#>.



I then read and analyzed the code in ‘int main{}’ to gain an understanding of how the provided program functioned and utilized the functions I would be writing. I began with the pseudocode for the find() function;

* Compare each character, one at a time, in the string to the first character in the substring.
* If a match is found, compare the next character in the string to the second character in the substring
* If the second character matches, return the position of the first character

I needed my return variable of type int for the function, so I first declared **tempPos** and set it to -1 and followed it with a return statement of that variable. This will be the case for no matches of the substring inside the string. I then tackled my first line of pseudocode, deciding that a ‘for’ loop would work best to increment variable **i** to parse through the string array. A ‘for’ loop also provided a nice way to stop my comparisons once every character had been checked, with “**i** != strlen(**str**)”, stopping once **i** was equivalent to the total length of **str**. Now that I had a method for looking through the entire string, I moved to designing the comparison. I chose an ‘if’ statement nested inside the ‘for’ loop to encapsulate both of my required comparisons at the same time with “( (**substr**[0] == **str**[**i**]) && (**substr**[1] == **str**[**i**+1]) )”. This way if a match was found for the first character, the second character is immediately tested without any additional code, and any action requires both characters to match. Finally, I moved to my third line of pseudocode, which simply sets **tempPos** to the position within the array of the string that the 2 matching characters were found, **i** .

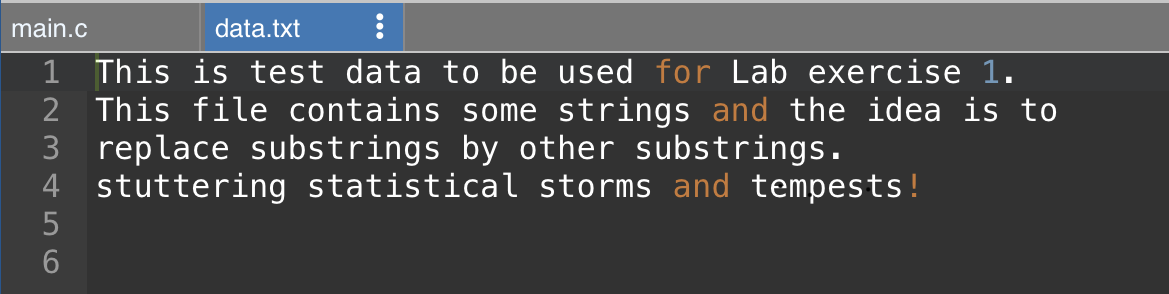
I then moved onto designing the pseudocode for the replace() function;

* Replace the 2 matching characters with the 2 substring characters

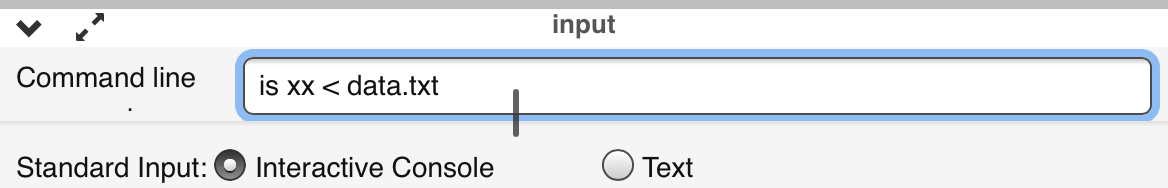
This was very simple to accomplish with 2 assignment statements, assigning the variables **str**[**pos**] and **str**[**pos**+1] to **substr**[0] and **substr**[1] respectively.

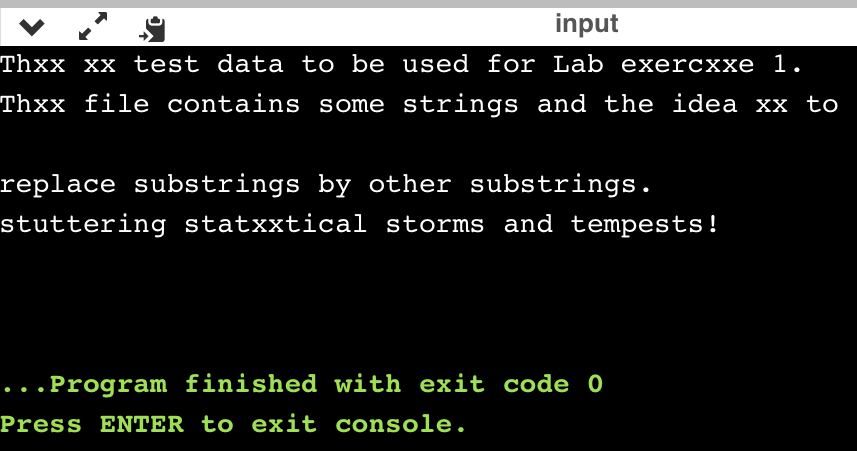
1. **Testing**

To start testing, I planned to test 2 conditions; known positive matches and known negative matches. The substring ‘is’ appears 6 times in the provided data.txt, so this was the substring used for the known positive test. The substring ‘ss’ appears 0 times inside data.txt, however instances of a single ‘s’ do occur multiple times allowing us to test for sequential matching, so this would be the known negative test.

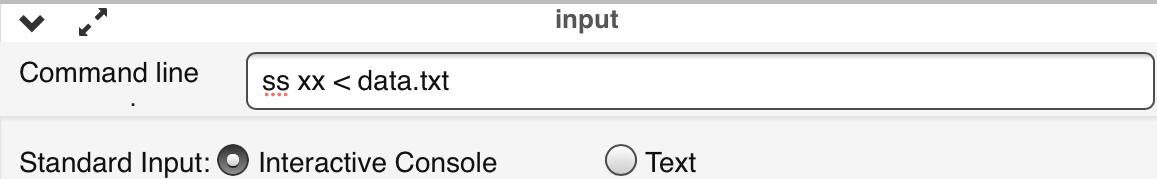


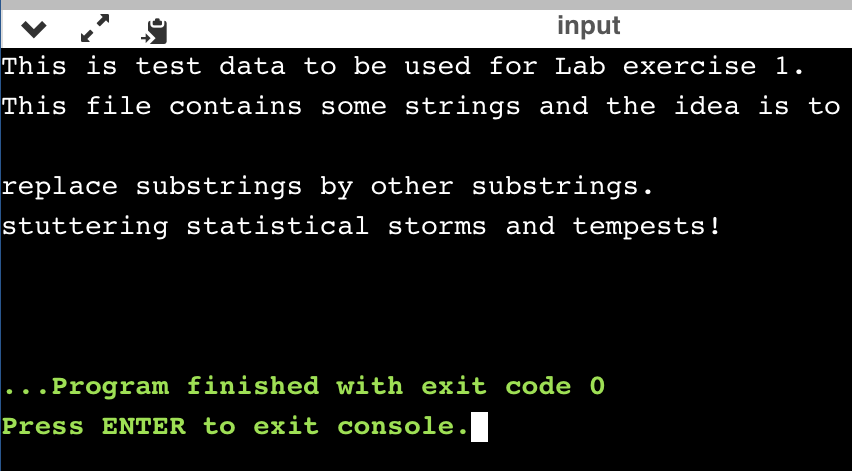
The test for known positive matches concluded successfully, finding all instances of ‘is’ and replacing them with ‘xx’. No additional or erroneous text was replaced, meaning only matches of sequential ‘i’ and ‘s’ were replaced, and no instances of ‘is’ were missed. The program finished with an exit code 0, indicating proper execution.





The test for known negative matches concluded successfully as well, finding no instances of ‘ss’ and replacing no text inside the string. This indicates that the comparison test for a match is working, and only matches of both characters result in a replace.





During the first set of tests of the functions I came across the line of code in main{};

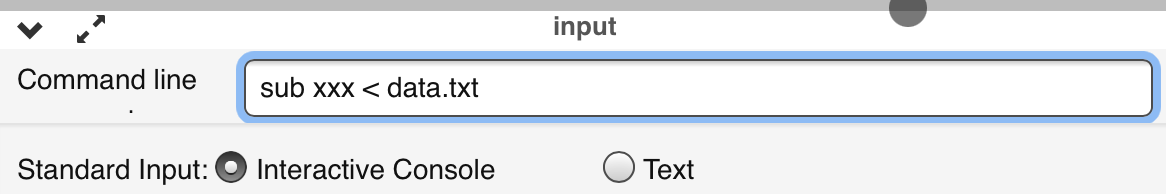
*if (strlen(argv[1]) != strlen(argv[2])) {*

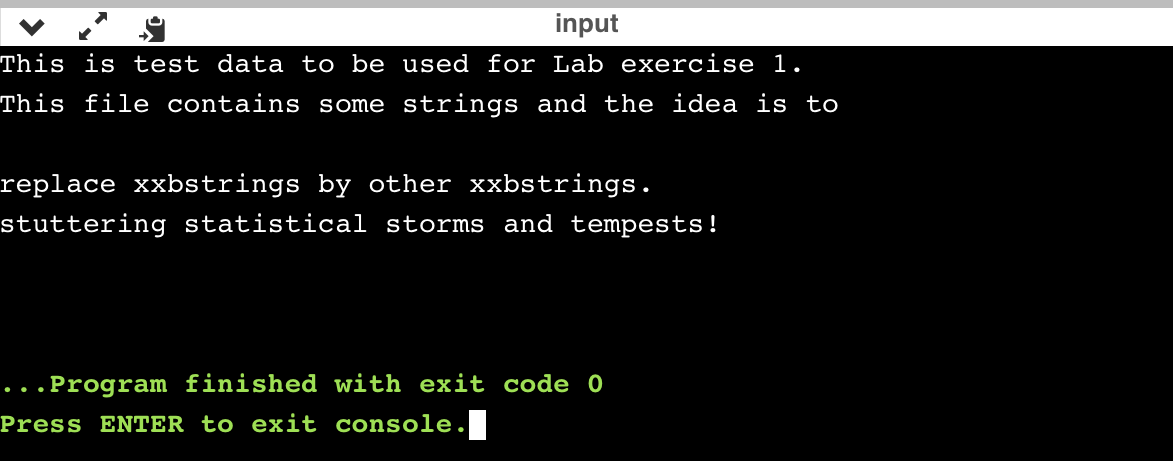
*puts("The two words must have the same length\n");*

*return 1;*

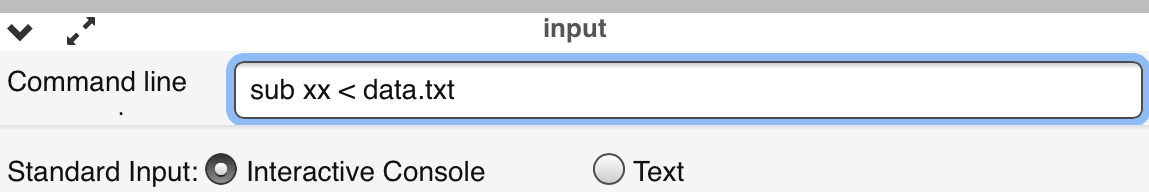
*}*

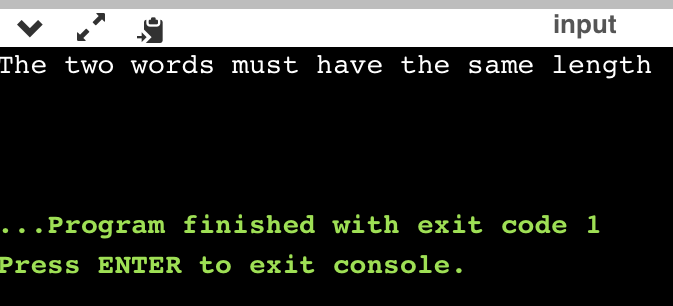
This code tests that the inputs are of the same length, not that they are of any particular length. My initial interpretation of the assignment was to only replace instances of 2 matching characters in a substring from the example in the lab. The function find() is written in a way that only the first and second characters in the substring are tested against the current position of **i** in the string and the sequential character respectively. I decided to test what would happen if multiple characters were provided, so I used ‘sub‘, which occurs 2 times in data.txt, as the value to find and ‘xxx’ as the replace.





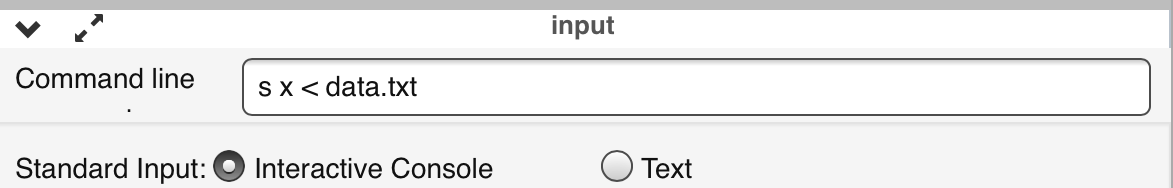
The test completed as expected, where only the first 2 characters of the provided substring were compared and replaced to the position **i** and the matching sequential character. The ‘b’ in ‘sub’ was not compared or replaced. The test in main{} passed because both of the sets of characters were the same length. I then moved to test if values of different lengths beyond 2 chars were provided. For this test ‘sub’ was used again as what to find and ‘xx’ would be the replacement.

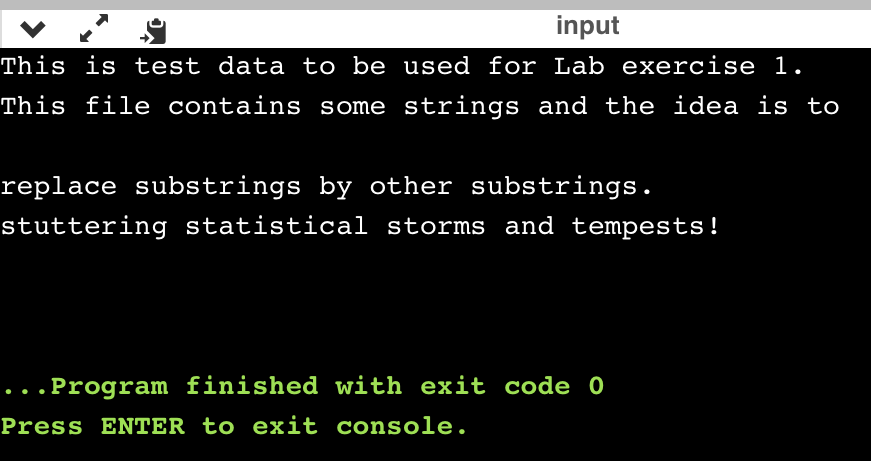




This test completed successfully, and the input was not accepted into the find() function due to the conflicting string lengths.

The final test to be conducted would be that of single character replacement. The character ‘s’ appears multiple times through data.txt, so this is the character chosen to conduct this test. I hypothesize the test will fail, as the second half of the && statement inside the nested ‘if’ will never evaluate to true, so the ‘if’ will always be false and the function will always return a -1.





This test concluded as hypothesized, with a failure to replace any instances of ‘s’.

1. **Results**

The results of the lab accomplish the intent of Lab1, which is to design 2 function definitions to find and replace text within a string. The first function, find(), succeeds in parsing through a provided string and matching a substring of 2 characters with a character and it’s sequential character inside of the string. It also succeeds in returning the position of the first character that initiates the match in the event of a match, and a -1 in an event of no matches, as instructed by the Lab1 documentation on Canvas. The second function, replace(), succeeds in changing the character stored in the position returned by the find() function and the sequential character. This function does not replace data upon a partial match, only on successful match of both characters in substring to sequential characters in the string.

1. **Conclusions**

Based on the results and intent of this lab, I conclude that more code can be added to expand the capabilities of both functions. Instead of using an && statement inside of the nested ‘if’ statement on line 24 to conduct comparisons of specific string lengths, this test could be written in a way to test for any length of supplied substrings and test with equally sized sequential characters. This change would require a major change to the replace() function as well, as currently it is only written to replace 2 characters inside of the substring. Such a change would require extensive testing of variable sized inputs. However the intent of the Lab has been met, as the functions find() and replace() have been completed to the specifications made in the Lab1 assignment in canvas;

* Find() - *This function returns the start index of the string substr in the string str. If substr cannot be found in str, find() must return -1.*
* Replace() - *This function replaces the characters in the string str, starting at index pos, by the string newstr.*

1. **References / Acknowledgements**

C Programming Language, B. W. Kernighan & D. M. Ritchie, 2nd Edition, Prentice Hall, 1988.

C Programming: A Modern Approach, K.N. King, Norton, 2008.

Nathan Burns, Scott Wedge, and Jared Larson are acknowledged to have been present during the Lab1 breakout room where we discussed the intent of the assignment and the pseudocode to accomplish each task.