MyString Class Implementation
Cpt S 223 Homework Assignment
Washington State University

1. Create source files

Create the source file main.cpp. Use the existing MyString.h file included in the assignment zip file. You can optionally add MyString.cpp if desired, or you can implement the entire MyString class in the header file and then you will only need main.cpp and MyString.h.

In main.cpp you will have the command processing loop, similar to prior assignments. In main.cpp you may use C and C++ standard library string functions and the std::string class. But only in main.cpp.

Pay attention to additional directions, listed in comments, in MyString.h. In the MyString.h (and potentially MyString.cpp as well) files, you must use only C-string library functions and not the C++ std::string class.

2. Implement the "rule of 3" items in the MyString class:

If this is unfamiliar, refresh yourself on this concept and why it is important. Many of the other operations will not work without these 3 important items implemented.

3. Set up the main function to read commands from stdin:

Much like in prior assignments, in this assignment you will read simple commands from standard input. Display your name and ID number on a single line to start with and then enter the command processing loop. There will be one command per line.

Command strings will be much like C function calls in terms of syntax. You must support the following commands:

set(string_value) : Sets the current string. All subsequent commands that you process will operate on this string. Set can be called at any time to change to a new string. Do NOT display anything in response to this command, just update the current string value. Also note that the string value may consist of any characters except parentheses, line breaks, and non-displayable characters.

substr(startindex,length) : Displays a substring from the current string. The starting index will be a non-negative integer that is less than the length of the string. The length will be a non-negative integer that represents the length, in characters, of the substring. Zero is a valid length. If the length would cause you to go beyond the end of the current string then simply go up to the last character (which equates to truncating the length to stay within the string bounds).

indexof(substr,startindex) : Displays an integer that is the starting index of the substring within the current string.

- The search starts at the specified starting index (second parameter), which will be a valid non-negative integer. The substring value (first parameter) may consist of any characters except commas, parentheses, line breaks, and non-displayable characters.
- Display -1 if the substring is not found.
- You are not required to use Boyer-Moore for this, but can if you want to.

bad_char_table() : Builds and displays the bad character rule table for the current string. (This is the bad character rule portion of the Boyer-Moore algorithm if that wasn't otherwise obvious). Recall that this is a table of 256 integer values that starts out with all entries initialized to -1. Then it is populated with rightmost occurrences of character indices. Include the null terminator as well. Then display ALL of the table values on a single line with a comma after each one, including the last number on the line. Do not add any spacing.

split(char_separator) : Splits the string based on the specified separating character and displays each "piece" (substring) on the same line with a comma after each one, including the last one on the line.

- If the separator never occurs in the current string then the split function is meant to create a list with just one string and that string is equal to the current string. So, one string with a comma after it should be displayed in this case.
- The separating character parameter may be any character except parentheses, line breaks, and non-displayable characters.
- Pieces must be displayed in order of their occurrence within the current string.

reverse() : Reverses the current string AND then displays it on one line.

is_integer() : Displays either "true" or "false", lower-case, without quotes, based on whether or not the current string is an integer value. The following are requirements for a string to represent an integer:

- If it starts with a minus character, then it must contain at least one numeric character after this, and satisfy all the other requirements as well.
- Other than the optional minus character at the start and the null-terminator at the end, every character in the string must be a numeric digit, [0-9]. Note that this implies that not even whitespace is allowed.

starts_with(str) : Displays either "true" or "false" (without quotes) based on whether or not the current string starts with the specified string parameter. This does NOT change the current string. Note that the string parameter could potentially be longer than the current string, in which case you would display false. This can be done in 1 line of code (hint: use IndexOf functionality).

anagrams(str1,str2,str3,…) : Compares all strings against the current string and displays only the ones that are anagrams of the current string.

- This function must support any number of parameters >= 1.
- A string S1 is an anagram for S2 if you can rearrange the letters of S1 to get S2. Anagrams involve simply rearranging letters, not adding or removing any. You can look up anagram examples online for more information and examples.
- This is a problem-solving part of the assignment. You must come up with the algorithm that determines whether one string is an anagram of another. As long as the execution happens in no more than a few milliseconds for 10 strings then there are no time or space complexity requirements. But just for your reference there is a way to do this in O(n) time, where n is the sum of all the string lengths (including the length of the current string).
- Display all anagrams on one line. Follow each one, including the last one on the line, with a comma. Display the strings that are anagrams in the order they appeared in the command and do NOT display any strings that are not anagrams.

quit() : Breaks the command processing loop and terminates your program. Display the line "Done" as the very last thing before returning from main.

Note: For each command your app processes, the output produced is on a single line followed by a line break at the end (except for "set", which displays nothing).

Final notes/requirements:

- See the input and output files included in the assignment zip file to get examples of command lines and corresponding output lines.
- Put comments in your code! Give a brief explanation for each function at a minimum. Points WILL be taken off if your code has poor commenting.
- Only display information on the screen when the command says to. An automated grading application will be used to grade these assignments and the output must exactly match the assignment requirements for it to be properly recognized and graded. Take a good look at the sample input and output files included with the assignment to make sure your application produces the same results for each input.
- Also, for commands that you cannot get working, you must still process the command so that the number of lines in your output matches up with the expected number of lines. Simply process the command and display something like "not implemented" if you did not implement the command completely. This will ensure that the grading app can grade your output line-by-line with no problems.