Rahul Reddy

CS 31

Professor Smallberg

Report for Project 5

A brief description of notable obstacles you overcame:

* I think that understanding the project spec itself was an obstacle I overcame. Being able to distinguish between what was clean and what was not as well as what was a valid score when looking at 2 arrays and the rules. Once I understood this, it was easier to plan out how I was going to write the program.
* Knowing when to create a separate/helper function was challenging, yet when I figured it out, it made writing the program a lot smoother and simpler.
* In addition, I got an error towards the end ONLY on G31 stating an “Address Sanitizer stack buffer overflow” which was an out of bounds error, but it was tricky since it was not exactly the same error described in the Project 5 FAQ. It was hard to figure out especially since I had to look through the entire code to see where I may have been accessing out of bounds.

A description of the design of your program. You should use [pseudocode](https://web.cs.ucla.edu/classes/fall20/cs31/pseudocode.html) in this description where it clarifies the presentation:

void lowercase()

* Loop through everything in the word[]
  + In the loop, check if the elements in the C string are not alphabetical
    - If they are not alphabetical, break and increment the variable notInAlphabet
  + Make the word[i] lower case

void removeRule()

* Loop through up to nRules -1 (to avoid the null byte)
  + Use strcpy and switch the 1st array’s elements with the array in front of it
  + Use strcpy and switch the 1st array’s elements with the array in front of it
  + Increment I

int cleanupRules()

* Make Rules Lower Case
  + Check if theyre in the alphabet
    - Remove rule
* Use two for-loops to iterate through the arrays
  + Check for 1 word rules that are not clean
    - Remove Rule
  + Check for two word rule
    - If win = wout
      * Remove Rule
    - If there are multiple rules with same win or wout
      * Remove Rule
* Return nRules

bool checkDocument()

* Loop through the length of the documents
  + If wordin equals the documents
    - Check if the index is – and if its not alphabetical
      * Increment index
  + If index equlas the length of wordin – 1
    - Return true
  + Else
    - Index is 0
  + Return false

int determineScore()

* Loop through length of document
  + Check for if the elements of the C string are alphabetical or spaces
    - Copy them onto a copy of document
* Loop through to nRules
  + Check if it’s a 1 word rule
    - If its in copy
      * Matched++
  + Check if it’s a 2 word rule
    - If in document
      * Matched

int main()

* Test Cases

A list of the test data that you could use to thoroughly test your functions, along with the reason for each test:

\*\*\*\*cleanupRules TESTS\*\*\*

const int TEST1\_NRULES = 3;

char test1win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"family", "unhappy", "horse"};

char test1wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "", "horse", "",};

assert(cleanupRules(test1win, test1wout, 3) == 3); //checks to make sure cleanup rules returns the correct value (general)

assert(cleanupRules(test1win, test1wout, 2) == 2); //checks to make sure cleanup rules returns the correct value (general)

char test2win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"", "unhappy", "horse"};

char test2wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "hi", "horse", "",};

assert(cleanupRules(test2win, test2wout, 3) == 2); //checks to make sure it can handle an empty string

/\*char test3win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"", "unhappy", "horse"};

char test3wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "hi", "horse", "",};

assert(cleanupRules(test3win, test3wout, 2) == 1); //empty string with smaller nRules than there are positons \*/

char test4win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"hi", "unhappy", "horse"};

char test4wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "hi", "horse", "",};

assert(cleanupRules(test4win, test4wout, 3) == 2); //same win and wout

char test5win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"", "unhappy", "horse"};

char test5wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "hi", "unhappy", ""};

assert(cleanupRules(test5win, test5wout, 3) == 1); //checks to make sure it can handle an empty string followed by win=wout

char test6win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"hello", "unhapp4y", "horse"};

char test6wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "hi", "horse", "",};

assert(cleanupRules(test6win, test6wout, 3) == 2); //checks to see if a non-alpha character is not included in the # of rules

char test7win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"43dh", "unhapp4y", "horse"};

char test7wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "hi", "horse", "",};

assert(cleanupRules(test7win, test7wout, 3) == 1); //checks to see if mutliple non-alpha character is not included in the # of rules

\*\*\*determineScore TESTS

char test8win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {"hello", "hi", "bye"};

char test8wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = { "", "", "",};

assert(determineScore("Hi my name is Rahul", test8win, test8wout, 3) == 1); //general

assert(determineScore("Hihello bye", test8win, test8wout, 3) == 1); //checks to see putting rules next to each other

char test9win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {

"family", "unhappy", "horse",};

char test9wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {

"", "horse", "",};

assert(determineScore("Happy families are all alike; every unhappy family is unhappy in its own way.", test9win, test9wout, TEST1\_NRULES) == 2); //general

assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.", //checks to test9win, test9wout, TEST1\_NRULES-1) == 0); //checks for if the function generally works with slightly more complex sentences

assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.", test9win, test9wout, TEST1\_NRULES) == 1); //checks for if the function generally works with slightly more complex sentences

assert(determineScore("A horse! A horse! My kingdom for a horse!",

test9win, test9wout, TEST1\_NRULES) == 1); //checks if non-letters plus multiple spaces in between will hinder the result

assert(determineScore("horse:stable ratio is 10:1",test9win, test9wout, TEST1\_NRULES) == 0); //checks to see if a non letter connecting two words will cause it to work properly

assert(determineScore("\*\*\*\* 2020 \*\*\*\*", test9win, test9wout, TEST1\_NRULES) == 0); //checks to see if all non characters will work