Project 5 Report

I encountered a few big challenges during this project. In the cleanupRules method, I couldn’t figure out how to sort the array of modified match rules so that the deleted rules would be moved to the end with the empty strings. I looped from the beginning to the end of the array, swapping elements as necessary; however, I wasn’t able to fix groups of multiple empty strings. I tried again and looped from the end of the array to the beginning, and it worked. In the determineScore method, I first used the strstr function to check whether or not the document contained wordin and wordout. My program compiled on both Xcode and through g31 on Linux, but I received an error when testing assert statements and I didn’t know why. The assert statements would function correctly individually but when I tested multiple at once, I would always get an error on the second statement, regardless of the order of them. Then, I decided to change my approach and try copying the document words into an array, but that didn’t work either. Finally, I revisited my initial thought process of using the strstr function and fixed some overcomplicated parts of my code, and I was successful.

For this program, my design was to modify the wordin and wordout arrays in cleanupRules and make copies of the document and wordin and wordout arrays in determineScore. A big part of determineScore was my use of the strstr function in order to find the rules that matched the document. My pseudocode for this program is:

int cleanupRules(char wordin[][MAX\_WORD\_LENGTH+1],

char wordout[][MAX\_WORD\_LENGTH+1,

int nRules) {

for each rule:

make chars lowercase

if wordin = wordout,

remove rule

for each rule:

if each char isn’t a letter,

remove rule

for each rule:

remove duplicate wordin values

remove duplicate 2-word rules

start count

for each rule:

if wordin isn’t an empty string,

increment count

for each rule:

swap rule with empty string in front of it

return count

}

int determineScore(const char document[],

const char wordin[][MAX\_WORD\_LENGTH+1],

const char wordout[][MAX\_WORD\_LENGTH+1],

int nRules) {

create document copy

if char in document is a letter or space

make lowercase, append to copy

start count

for each rule:

copy wordin, wordout into temp variables

if document contains wordin,

if rule has 1 word,

increment count

else

if document doesn’t contain wordout,

increment count

return count

The test data I used on this function are:

wordin1[][] = {“a”, “a”, “a”, “aA”, “aaa”, “b”, “b”, “cd”, “cde”}

wordout1[][] = {“”, “a”, “a”, “aa”, “aaa”, “bb”, “Bc”, “c”, “c”}

wordin2[][] = {“”, “a”, “hot”, “aa”, “summer”, “beach”, “b”, “c”, “c”}

wordout2[][] = {“a”, “A”, “a”, “cold”, “aaa”, “summer”, “b”, “c”, “c”}

wordin3[][] = {“ice”, “beach”, “day”, “summer”, “summer”, “hot”, “ice”, “water”, “sky”}

wordout3[][] = {“”, “day”, “beach”, “”, “pool”, “sun”, “pool”, “weather”, “”}

wordin4[][] = {“dog”, “cat”, “cow”, “horse”, “cow”, “sheep”, “chicken”, “goat”, “pig”}

wordout4[][] = {“pet”, “animal”, “”, “farm”, “barn”, “”, “milk”, “cheese”, “egg”}

wordin5[][] = {“math”, “school”, “”, “teacher”, “class”, “student”, “grade”, “pass”, “school”}

wordout5[][] = {“subject”, “”, “teacher”, “”, “room”, “”, “pass”, “grade”, “school”}

1. cleanupRules(wordin1[][], wordout1[][], 0) — for nRules = 0
2. cleanupRules(wordin1[][], wordout1[][], -1) — for nRules < 0
3. cleanupRules(wordin1[][], wordout1[][], 3) — for part of the array with similar chars
4. cleanupRules(wordin1[][], wordin1[][], 9) — for the whole array with similar chars
5. cleanupRules(wordin2[][], wordout2[][], 0) — for nRules = 0
6. cleanupRules(wordin2[][], wordout2[][], -2) — for nRules < 0
7. cleanupRules(wordin3[][], wordout3[][], 6) —
8. cleanupRules(wordin4[][], wordin4[][], 9) — for the whole array
9. cleanupRules(wordin5[][], wordin5[][], 4) — for part of the array
10. cleanupRules(wordin5[][], wordin5[][], 7) — for many rules already in clean form
11. determineScore(“abcdefghijklmnopqrstuvwxyz”, wordin5[][], wordout5[][], 1) — for document to have 1 long word
12. determineScore(“math school teacher class student grade pass school”, wordin5[][], wordout5[][], 0) — for document to contain every string in wordin
13. determineScore(“I like going to the beach. It feels good to swim on a hot summer day.”, wordin4[][], wordout4[][], 3)
14. determineScore(“a b c d e f g h i j k l m n o p q r s t u v w x y z”, wordin4[][], wordout4[][], 2) — for document to not contain any string in wordin
15. determineScore(“I like going to the beach. It feels good to swim on a hot summer day.”, wordin3[][], wordout3[][], 5) — to test wordin and wordout with this document
16. determineScore(“ice beach day summer pool sun hot weather water sky”, wordin3[][], wordout3[][], 4) — for document to contain every string in wordout
17. determineScore(“\*\*\*\*2020\*\*\*\*”, wordin2[][], wordout2[][], 7) — document with special chars and numbers, no letters or spaces
18. determineScore(“the year is 2020, 1 year after 2019”, wordin2[][], wordout2[][], 6) — document with letters, spaces, numbers, and other chars
19. determineScore(“I like going to the beach because it feels good to swim on a hot summer day.”, wordin1[][], wordout1[][], 9) — document with only letters and spaces
20. determineScore(“I don’t like going to the beach. It feels good to swim on a hot summer day.”, wordin1[][], wordout1[][], 8) — document with letters, spaces, and other chars