**Chelsea Biala**

**CS31 Smallberg**

**Fall 2020**

1. My greatest obstacle in this project was figuring out the c-string functions and making sure my function could handle words with spaces around them. The cleanupRules function was also difficult, because I had to write a helper function in order to rotate the word correctly.
2. (Rotate function puts the rule passed in at the end of the array and changes the size pointer, effectively eliminating the rule from the array) cleanupRules:

Iterating through the array of rules

Checks for if rule is the empty string

Rotates to end

Checks for if rule has any non-alphabetical characters

Rotates to end

Converts all characters in the rule to lowercase

Iterating through array of remaining rules twice

Checks if two win only rules are the same

Rotates to end

Checks if any two rules have the same win and wout

Rotates to end

Returns number of rules

determineScore:

Creates new array

Copies document into array with all lowercase letters, no special characters

Iterating through new array

Creates a new C string with each rule

Checks if there’s a match to that rule in the document

Iterates score

Returns document score

c. char in[14][MAX\_WORD\_LENGTH+1] = {"confusion", "FAMILY", "charm", "hearty", "house", "worn-out", "", "family", "charm", "ties", "", "charm", "FaMiLy"};

char out[14][MAX\_WORD\_LENGTH+1] = {"", "TIES", "confusion", "hearty", "intrigue", "younger", "", "first", "", "family", "frightened", "", “tIeS"};

assert(cleanupRules(in, out, 14) == 6);

Empty string as win, rules with uppercase letters, rules with special characters, rules with the same win as wout, one-word win rules with the same win as other rules, and two-word win rules with the same win and wout as another rule

assert(cleanupRules(in, out, -1) == 0);

Negative nRules

char in1[1][MAX\_WORD\_LENGTH+1] = {“confusion”};

char out1[1][MAX\_WORD\_LENGTH+1] = {“hello”};

assert(cleanupRules(in1, out1, 1) == 1);

All rules correct

char in2[1][MAX\_WORD\_LENGTH+1] = {“confusion1”};

char out2[1][MAX\_WORD\_LENGTH+1] = {“he llo”};

assert(cleanupRules(in2, out2, 1) == 0;

All rules incorrect

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(determineScore(“Hi Horse man”, test1win, test1wout, TEST1\_NRULES) == 1);

Rule match is capitalized

assert(determineScore(“Hi Horse man”, test1win, test1wout, -1) == 0);

Negative nRules argument

assert(determineScore(“Nothing”, test1win, test1wout, TEST1\_NRULES) == 0);

No rules match

assert(determineScore(“I’m unhappy rn”, test1win, test1wout, TEST1\_NRULES) == 0);

Two word rule matches

assert(determineScore(“Unhappy horse”, test1win, test1wout, TEST1\_NRULES) == 1);

Has win and wout of two word rule

assert(determineScore(“Unhappy family”, test1win, test1wout, TEST1\_NRULES) == 2);

When matches are at beginning and end of document

assert(determineScore(“Family-horse”, test1win, test1wout, TEST1\_NRULES) == 0);

Matches not separated by space

assert(determineScore(“Horse horse”, test1win, test1wout, TEST1\_NRULES) == 1);

More than one instance of one rule

assert(determineScore(“LLL horse jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj”, test1win, test1wout, TEST1\_NRULES) == 1);

Document with max amount of characters