1. I encountered two major obstacles while I was doing this project. The first major obstacle is figuring out how to weed out the bad rules that does not fit the rather strict requirements of a “normalized” rule. Since there are so many different requirements to the normalized rule, I realized I can separate the normalizeRules function into many shorter functions. The second major obstacle I encountered was how to properly read the document C-string in calculateSatisfaction. In order to easily count the distance between each word in the document, I ended up separating each word in document into elements in and an array of C-strings. Therefore, to calculate the distance between words, I only have to find the different between there indexes.
2. normalizeRules:

repeatedly through all the rules:

find empty strings in word1&word2 & negative ints in distance

mark these indexes for deletion

find rules that are the same as the current rule

mark the rules with shorter distance for deletion

delete the rules marked for deletion

decrease the number of total rules accordingly

turn all the words in the rules to lowercase

calculateSatisfaction:

set nRules to zero if it is negative

repeatedly through every character in the sentence:

if encountered a letter,

add it to a slot in a c-string array

if encountered a space,

skip to the next slot in the array

turn all the words in the array to lowercase

repeatedly through all the rules:

find words in the array that match the words in the rule

find the absolute distance between the words

if fits the rule,

increment satisfaction score

return satisfaction score

1. test cases:

For normailizeRules:

char word1[4][MAX\_WORD\_LENGTH + 1] = {

"mad", "scientist", "Deranged", "nefarious", "Have", "abe", ""

};

char word2[4][MAX\_WORD\_LENGTH + 1] = {

"scientist", "mad", "robot", "Plot", "mad", "lincoln", "a"

};

int distance[4] = {

2, 1, 4, 1, 13, -1, 3

};

int nRules = 7;

normalizeRules(word1, word2, distance, nRules);

//make sure the function makes all rules lowercase, deletes repeated rules, //deletes rules with empty strings, and deletes rules with negative //distances.

For calculateSatisfaction:

char word1[4][MAX\_WORD\_LENGTH + 1] = {

"mad", "deranged", "nefarious", "have"

};

char word2[4][MAX\_WORD\_LENGTH + 1] = {

"scientist", "robot", "plot", "mad"

};

int distance[4] = {

2, 4, 1, 13

};

int nRules = 4;

assert(result = calculateSatisfaction(word1, word2, distance, nRules, "The mad UCLA scientist unleashed a deranged evil giant robot.") == 2);

//make sure calculateSatisfaction returns the correct number of satisfied //rules.

assert(result = calculateSatisfaction(word1, word2, distance, nRules, "The mad UCLA scientist unleashed a nefarious plot.") == 2);

//make sure the program recognizes multiple consecutive spaces as just one //space.

assert(result = calculateSatisfaction(word1, word2, distance, nRules, "Th3 m@d UCLA sc13nt1st unl3@sh3d a deranged 3vil g1@nt robot.") == 1);

//make sure the function ignores non-alphabetical characters.

assert(result = calculateSatisfaction(word1, word2, distance, nRules, "mad scientist a a mad a scientist mad scientist") == 1);

//make sure the function only triggers once when there are multiple sets of //words that fits the same rule