1. In normalizeRules, the most notable obstacle that I overcame was figuring out if two of the rules were the same. I ended up using a for loop to compare every word1 to each other, every word2 to each other, and every word1 to every word2. Then, if there was any with the same rule, I would compare their distances. If the smaller distance was later in the array than the larger one, I left it be. If the smaller distance was earlier in the array, I switched the locations of the two distances. Then, I removed that rule from the array by shifting everything else after it to the left.  
   In calculateSatisfaction, my greatest obstacle was going through the document and figuring out if word1 and word2 were within distance of each other. I ended up making a function to put one word at a time into a c-string. I would then search the document for the word, and if it came up, I searched distance after it to see if the complementary word came up. I did this for both word1 and word2 so that I would not have to deal with searching backwards. In addition, I created a bool array so that when a rule was satisfied, I would switch the value of the element in the bool array to false. Then, I would not search for that rule again.
2. normalizeRule:

change any nRule < 0 to 0

repeatedly:

change word to lowercase

flag any rules with non-alphabet characters

Check if any word1 match with word1

Flag if word2 also match

Check if any word1 match with word2

Flag if other word1 and other word2 match

If rules are same

Move smaller distance to the later position

if the rule is flagged or part of the rule is bad

repeatedly

move every rule after that one position to the left

return amount of rules

Basically, in my normalizeRule, I first change everything to lowercase and then check to make sure that none of the rules match. I check every case of rules matching (ie. Word1 matches word2, word1 matches word1). Then, I delete the smaller distance so that the rules will not repeat.

calculateSatisfaction:

change any nRule < 0 to 0

repeatedly:

change char to lowercase

if any char is not a space or alphabet character

move every char after it one position to the left

repeatedly:

put individual word in doc into a new c-string

repeatedly:

if individual word matches word1 and rule hasn't been satisfied

repeatedly:

Check word up to the rule's distance after word1

If going up to distance leads to out of bounds

Break

If a word matches word2

Increment satisfaction up

change rule to satisfied

Do the same thing as above, but check if word matches word2 instead

Return the number of rules satisfied

findWord

repeatedly:

if char is a whitespace

move up one space

repeatedly:

if char is an alphabetical character

save the chars in a c-string until a whitespace appears

return the length of the c-string plus the length of whitespace before it.

1. **normalizeRules Tests:**

char w1[20][MAX\_WORD\_LENGTH+1] = {"scientist", "deranged", "nefarious"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"mad", "robot", "assistant"};

int distance [20] = {1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests normal case

char w1[20][MAX\_WORD\_LENGTH+1] = {"scientist", "DERANGED", "nefarious"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"mad", "robot", "assistant"};

int distance [20] = {1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests case with caps

char w1[20][MAX\_WORD\_LENGTH+1] = {"scientist", "DERANGED", "nefarious"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"mad", "robot", "assistant"};

int distance [20] = {-1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests case negative distance

char w1[20][MAX\_WORD\_LENGTH+1] = {"robot", "DERANGED", "nefarious"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"deranged", "robot", "assistant"};

int distance [20] = {1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests case when two rules are equivalent

char w1[20][MAX\_WORD\_LENGTH+1] = {"deranged", "DERANGED", "nefarious"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"robot", "robot", "assistant"};

int distance [20] = {1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests different case when two rules are //equivalent

char w1[20][MAX\_WORD\_LENGTH+1] = {"robot", "DERANGED", "ROBOT"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"deranged", "robot", "deranged"};

int distance [20] = {1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests case where all three rules are equal

char w1[20][MAX\_WORD\_LENGTH+1] = {"robot", "DERANGED", "Mad"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"deranged", "robot", "scientist"};

int distance [20] = {3, 1, 5};

normalizeRules (w1, w2, distance, 3); //Tests case where distance[1] < distance[2]

char w1[20][MAX\_WORD\_LENGTH+1] = {"robot-now", "DERANGED", "ROBOT"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"deranged", "robot", "deranged"};

int distance [20] = {1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests case where rule has non-alpha char

char w1[20][MAX\_WORD\_LENGTH+1] = {"", "DERANGED", "ROBOT"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"deranged", "robot", "deranged"};

int distance [20] = {1, 3, 5};

normalizeRules (w1, w2, distance, 3); //Tests case where rule has no entry

**calculateSatisfaction Tests:**

char w1[20][MAX\_WORD\_LENGTH+1] = {"robot", "mad", "evil"};

char w2[20][MAX\_WORD\_LENGTH+1] = {"deranged", "scientist", "assistant"};

int distance [20] = {1, 3, 5};

char doc1[] = {"The deranged robot killed the mad scientist and his evil assistant"}; //normal case

char doc2[] = {"The deranged bad robot killed the mad bad scientist and his evil dumb assistant"};

//Checks if distances work properly

char doc3[] = {"The mad robot killed the evil man without empathy"};

//No rules apply

char doc4[] = {"The robot deranged robot killed the mad scientist mad"};

//checks if it applies rules more than once

char doc5[] = {"The deranged robot killed the deranged robot"};

//Another case to check if the rules apply more than once

calculateSatisfaction (w1, w2, distance, 3, doc1)

calculateSatisfaction (w1, w2, distance, 3, doc2)

calculateSatisfaction (w1, w2, distance, 3, doc3)

calculateSatisfaction (w1, w2, distance, 3, doc4)

calculateSatisfaction (w1, w2, distance, 3, doc5)