CS31 Project 5 Report

1. Removing duplicates as instructed in the first function is tricky.

Finding matches number in the second function is extremely hard, and there were so many restrictions to consider.

Both functions require lots of modifications that I have spent days to figure out.

2. Project description:

**void** DeleteNonalpha (**char** input[]);

// my own code that used in function 2 to delete nonalpha characters except spaces.

**…trace through the string…**

**…Store all the alpha-characters and spaces into the string itself…**

**…Put an ‘\0’ at the end of operation…** // Ignore all the characters after the one we stored since they don’t comply the standard.

**int** editStandards(**int** distance[],

**char** word1[][MAX\_WORD\_LENGTH+1],

**char** word2[][MAX\_WORD\_LENGTH+1],

**int** nStandards);

// Return 0 if nStandards is nonpositive.

if ( nStandards is smaller or equals to 0 ) return 0 , and we don’t care more about those three arrays distance, word1, and word2.

// Remove modules with negative numbers in the array.

Trace through position 0 to nStandards-1; (i starting from 0 going to nStandards-1)

Save all the lines i with positive number and word1 and word2 at that line.

Remove all the lines i with negative number to the end and decrement nStandards for each of them.

// Remove the standards with words who are actually empty strings.

Trace through position 0 to nStandards-1;

If (word1 has an empty string on line i or word2 has an empty string on line i) {remove line i and decrement the size of nStandards by 1.}

// Remove standards that contain non-alpha character for two arrays.

for (i from 0 to nstandrads-1)

for(j from 0 to strlen(word1[i]-1)) //traces through each char of word1.

If (some position word1[i][j] is nonalpha) {remove that line as above.}

{Also remove line i in the distance array and word2 array;

Decrement the size of nStandards by 1.}

Do the Same things for word2.

//Convert all characters into lowercase in word1 and word2.

In word1:

for (i from 0 to nstandrads-1)

for(j from 0 to strlen(word1[i]-1))

{convert each char to its lowerform.}

In word2:

Do the same.

//Finally, remove all the duplicates and leave one with the biggest corresponding int.

for (i from 0 to nStandards-1){

int max = distance[i];

for (int j = i+1; j < nStandards; j++){ compare the elements starting from the next of the array to the ith, if the number of the latter is bigger, delete the former line and other arrays at that postion; if the former Is bigger, delete the latter line and other arrays at that position.}

{maneuver the position so that no lines missed the comparison and no line get compared many times.

For each deletion, decrement the size of nStandards by 1.}

**int** determineMatchLevel(**const** **int** distance[],

**const** **char** word1[][MAX\_WORD\_LENGTH+1],

**const** **char** word2[][MAX\_WORD\_LENGTH+1],

**int** nStandards,

**const** **char** jeet[])

{

//First make a char array copy that takes in the string in char jeet[] since we can’t modify the jeet inputted. It size is 280 specified by spec.

…make a copy of jeet…

…Call the function DeleteNonapha… //Delete any nonalpha characters except space.

// Modify the string so that each word separated only by one space.

…Trace through the string…

If posi is letter …store it…

If posi is space …store it and ignore all the space comes after the first one until the next letter…

…attach a space to the end for later use…

// Make a loop that loop through elements 0 to nStandards-1 in distance, word1, word2 which check the position of occurence of ith element of word1 in copyOfJeet, ith element if word2 in copyOfJeet, and see if the spaces between them are <= distance[i].

**int** count = 0; That check how many matched standards in the string can be found.

**for** (trace through all the standards) {

**int** numSpace = 0;

**if** (there’s no word1 in the copy of jeet or no word2 in the copy of jeet) {do nothing so that count won’t change.}

**else** **if** (position of the word2 is prior to the position of word1) {do nothing so that count won’t change.}

**else** {

take down the first occurrence of word1 using pointer as first, and the first occurrence of word2 as second.

…from the first position to the second, check the number of spaces..

…however if the first appearance of word1 is inside another word… { do nothing so that count won’t change.}

…same if the first appearance of word2 is inside another word… { do nothing so that count won’t change.}

if( those two conditions are not reached and the number of space counted is smaller than the number specified in distance[]) {increase the amount of count}

}

**Test cases for editStandards:**

//A simple test case that didn’t require the function to do any modification and return the expected number. It was used to test if the function can run correctly without fatal mistakes.

const int TEST1\_NSTANDARDS = 4;

int test1dist[TEST1\_NSTANDARDS] = { 2, 4, 1, 13 };

char test1w1[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "electric", "space", "electric", "were" };

char test1w2[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "billionaire", "capsule", "car", "eccentric" };

assert(editStandards(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS) == 4);

// A test case that examined whether the function has the ability to delete the bad standard containing the negative number and the empty string.

int test2dist[TEST1\_NSTANDARDS] = { -2, 4, 1, 13 };

char test2w1[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "electric", "space", "", "were" };

char test2w2[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "billionai4re", "capsule", "car", "eccentric" };

assert(editStandards(test2dist, test2w1, test2w2, TEST1\_NSTANDARDS) == 2);

// A similar test case that examined whether the function has the ability to delete the bad standard containing the negative number the empty string and the string with nonalpha character.

int test3dist[TEST1\_NSTANDARDS] = { 2, 4, 1, -13 };

char test3w1[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "electric", "", "electric", "were" };

char test3w2[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "billionaire", "ca4psule", "car", "eccentric" };

assert(editStandards(test3dist, test3w1, test3w2, TEST1\_NSTANDARDS) == 2);

// A test case that examine whether the function has the ability to remove duplicate.

int test4dist[TEST1\_NSTANDARDS] = { 2, 4, 1, 13 };

char test4w1[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "electric", "electric", "electric", "were" };

char test4w2[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "billionaire", "billionaire", "car", "eccentric" };

assert(editStandards(test4dist, test4w1, test4w2, TEST1\_NSTANDARDS) == 3);

// A test case that examine whether the function has the ability to remove multiple duplicate and save the pair with the highest corresponding int.

int test5dist[TEST1\_NSTANDARDS] = { 2, 4, 1, 13 };

char test5w1[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "space", "space", "space", "space" };

char test5w2[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH + 1] = { "capsule", "capsule", "capsule", "capsule" };

**Test cases for determineMatchLevel:**

**const** **int** TEST1\_NSTANDARDS = 4; // A very simple test case that test if the function correctly compare the matchlevel (when comparing words apart to limited distance).

**int** test1dist[TEST1\_NSTANDARDS] = {

2, 4, 1, 13

};

**char** test1w1[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"eccentric", "space", "electric", "were"

};

**char** test1w2[TEST1\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"billionaire", "capsule", "car", "eccentric"

};

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS,

"The eccentric outspoken billionaire launched a space station cargo capsule.") == 2);

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS,

"The eccentric outspoken billionaire launched a space capsule.") == 2); //Same as the first, a simple case that test the general function of the function.

// A test case that examine if the program delete all the characters in the array and return 0 as result since there’s no matched pair.

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS,

"\*\*\*\* 2022 \*\*\*\*") == 0);

// A test case that examine if the function delete the exclamation point and transfer the ElECTRIC into lower case and correctly turn out the match level which is 1 in this case.

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS,

" It's an ELECTRIC car!") == 1);

// A test case that examine if one match standard get counted several times. It check if the function is correct by seeing if the count turn out is bigger than 1.

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS,

"space space capsule space capsule capsule") == 1);

// A test case that examine if the function remove the “-“ and discard the space capsule and eccentric billionaires as good standard matches since they are not exactly the same as the ones in the word1 and word2 arrays.

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS,

"Two eccentric billionaires were space-capsule riders.") == 0);

cout << "All tests succeeded" << endl;