**Project 5 report**

1. **A brief overview of notable obstacles:**

I initially declared: int positions[nStandards]; to create an array of positions so that I can just set everything that should be removed to 0 step-by-step. While this compiled on Xcode, it didn't compile on g31 as I used a variable for the size of the array. To get around this, I have kind of "cheated" the system by setting distance to 0 here whenever we needed to set positions to 0. This is allowed as 0 distance is anyway not counted as a standard match.

I also found it difficult to implement the second function. It wasn’t until very recently that I discovered the strstr function. In hindsight, I could have used this to simplify the second function. But instead, I decided to split the jeet into an array of “words” i.e. lower case letters only separated by spaces. The motivation for this was that it would then be extremely simple to check for words being a certain distance away from others. However, the implementation of this conversion to an array was slightly difficult and has been documented in next section.

1. **Int editStandards(){**

if nStandards is negative, set it to 0

iterate using i from 0 to nStandards – 1

if distance[i] is less than 0, set it to 0

if word1[i] or word2[i] are the empy string, set distance[i] to 0

if either of them have a non-letter, set distance[i] to 0

iterate through all characters in word1 and word2 and make them lower case

iterate using i from 0 to nStandards – 1

if distance is not 0 (i.e if we haven’t set it to 0 because of a bad argument)

for all words in word1, check the words after it to see if there’s a duplicate

if those positions also have the same words in word2

keep the word with the greater distance and set the other’s distance to 0

// now we have distance = 0 for all the bad standards

Define and initialize an r = 0 to count the good standards remaining

Iterate through the 3 arrays and wherever distance is not 0, move it to the front of the array and increment r accordingly.

At the end, return r (which should now be the total number of good standards)

**Int determineMatchLevel()**

If nStandards is negative set it to 0;

Create a new jeet which is equivalent (according to the definition in the spec)

Iterate through the characters of the original jeet and add it to the new jeet only if it is a letter or a space if it’s a letter also convert it to upper case.

Create a variable to store the number of “words” in this new jeet by counting the number of runs of spaces in it (make adjustments accordingly if the spaces are at the start or end of the jeet)

Create a new array (of cstrings) that stores these words. Define its length to be MAX\_JEET\_LENGTH + 1 as there can’t be more words in a jeet than the number of characters it allows.

Iterate through the good jeet using i

For all characters stored between spaces, move those characters to the array. After each occurrence of a run of spaces, increment the index of the word array to which the characters are being copied.

// this is done by starting a new iterator at j and increment it till it hits a space. then copy the characters from i to j – 1 to the array and after the copying, increment word\_index. Also set i to j at the end of each word has been copied.

Also make sure that for any position of the stored words, no more than 20 characters are being copied. If more are, break it there (which shortens the word but it’s fine because we are promised no more than 20 char words in the match standards) and continue to the next word being copied.

After this array has been created, it is easy to determine the match level

Iterate i from 0 to nStandards – 1

If word1 is found in jeet array, let i be its position, else break

If word2 is also found in the rest of the jeet array let j be its position else break

If the distance between them is less than d then increment matchLevel

Return matchLevel;

1. **Test cases used (pasted from main routine in assert form):**

// test set 1

const int TEST1\_NSTANDARDS = 4;

int test1dist[TEST1\_NSTANDARDS] = {

1,2,3,4

};

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

"nakul", "nakul", "nakul", "wordone"

};

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

"khambhati", "khambhati", "khambhati", "wordtwo"

};

int test1result[TEST1\_NSTANDARDS] = {3,4};

int r1 = editStandards(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS);

assert(r1 == 2); // works with a triplicate as well

for (int i = 0; i < r1; i++){

assert(test1result[i] == test1dist[i]);

}

// test set 2

const int TEST2\_NSTANDARDS = 8;

int test2dist[TEST2\_NSTANDARDS] = {

-1, 0, 1, 3, 2, 1, 13, 6

};

char test2w1[TEST2\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"eccentric", "space", "ELECTRIC","tunnel-boring", "space", "Electric", "were", "electric"

};

char test2w2[TEST2\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"billionaire", "capsule", "CAR", "equipment", "capsule", "car", "eccentric", "car"

};

int test2result[TEST2\_NSTANDARDS] = {2, 13, 6};

int r2 = editStandards(test2dist, test2w1, test2w2, TEST2\_NSTANDARDS);

assert(r2 == 3); // works for the test case in the project spec

for (int i = 0; i < r2; i++){

assert(test2result[i] == test2dist[i]);

}

// test set 3

const int TEST3\_NSTANDARDS = 5;

int test3dist[TEST3\_NSTANDARDS] = {

1, 2, 3, 4, 5

};

char test3w1[TEST3\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"nakul", "angad", "arnav","ani-rudh", "angAD"

};

char test3w2[TEST3\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"khambhati", "kapOOr", "mierd0", "rao", "kapoor"

};

int test3result[TEST3\_NSTANDARDS] = {1, 5};

int r3 = editStandards(test3dist, test3w1, test3w2, TEST3\_NSTANDARDS);

assert(r3 == 2); // checking that upper is converted, non alpha is not counted

for (int i = 0; i < r3; i++){

assert(test3result[i] == test3dist[i]);

}

// test set 4

const int TEST4\_NSTANDARDS = 0;

int test4dist[TEST4\_NSTANDARDS] = {

};

char test4w1[TEST4\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

};

char test4w2[TEST4\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

};

int test4result[TEST4\_NSTANDARDS] = {}; // if n standards is 0, everything should be 0

int r4 = editStandards(test4dist, test4w1, test4w2, TEST4\_NSTANDARDS);

assert(r4 == 0); // returns 0 retained elements cuz there are 0 elements

for (int i = 0; i < r4; i++){

assert(test4result[i] == test4dist[i]);

}

// test set 5

const int TEST5\_NSTANDARDS = 4;

int test5dist[TEST5\_NSTANDARDS] = {

1, 2, 3, 4

};

char test5w1[TEST5\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"", "naku1", "kham bhati", "0"

};

char test5w2[TEST5\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"word", "word", "word", "word"

};

int test5result[TEST5\_NSTANDARDS] = {};

int r5 = editStandards(test5dist, test5w1, test5w2, TEST5\_NSTANDARDS);

assert(r5 == 0);

for (int i = 0; i < r5; i++){ // the empty string is not allowed

assert(test5result[i] == test5dist[i]);

}

// \*\*\* determining match level for jeets \*\*\*

const int TEST6\_NSTANDARDS = 4;

int test6dist[TEST6\_NSTANDARDS] = {

2, 4, 1, 13

};

char test6w1[TEST6\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

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

};

char test6w2[TEST6\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

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

};

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

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

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

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

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

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

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

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

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

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

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

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

// works for the test cases in the spec

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

"The eCcen8tric weird BILLION999AIRE") == 1); //caps, nonalpha and spaces don't matter

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

"electric c car") == 0); // doesn't work for distance of 2 if it's meant to be 1

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

"eccentric billionaire space capsule electric car were eccentric") == 4); // match levels stack up

assert(determineMatchLevel(test6dist, test6w1, test6w2, TEST6\_NSTANDARDS,

"eccentric billionaire space capsule electric car were eccentric were eccentric were eccentric") == 4); // doesn't double count

const int TEST7\_NSTANDARDS = 4;

int test7dist[TEST7\_NSTANDARDS] = {

2, 4, 1, 13

};

char test7w1[TEST7\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"nakul", "nakul", "and", "smartest"

};

char test7w2[TEST7\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"barkha", "crocs", "the", "alive"

};

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"nakul and barkha see nakul wearing crocs and the consensus is that he is the smartest man alive.") == 4); // checking that all 4 work even when there are words that occur twice in the jeet

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"barkha and nakul see the the the crocs wearing nakul the and consensus is that he is the alive smartest man.") == 0); // reversing order doesn't count

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"nakul and barkha nakul and barkha") == 1); // checking that a double match isn't counted twice

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"nakul and barkha and crocs and the smartest thing to do is to check whether the living people are still alive") == 4); // spacing each out at its maximum

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"nakul khambhati and barkha and crocs and I think the smartest thing to do is to check whether the living people are all still alive") == 0); // exceeding each's maximum by 1

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"") == 0); // empty jeet

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"nakul nakul and smartest") == 0); // all just from w1

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"barkha crocs the alive") == 0); // all just from w2

assert(determineMatchLevel(test7dist, test7w1, test7w2, TEST7\_NSTANDARDS,

"nakul 3957203&$9035- and//4530790 barkha se4590e nakul wear343ing 3434 cr3o4cs and the conse!!nsu!3s 35is t53hat5 3he 5is t32552he 513smartest ma-0325\_)\*0-5831n ali351-0ve.") == 4); // putting random non-alphabets in the middle has absolutely no effect

const int TEST8\_NSTANDARDS = 0; // with nStandards = 0, absolutely everything returns 0 as the match level

int test8dist[TEST8\_NSTANDARDS] = {

};

char test8w1[TEST8\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

};

char test8w2[TEST8\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

};

assert(determineMatchLevel(test8dist, test8w1, test8w2, TEST8\_NSTANDARDS,

"nakul and barkha see nakul wearing crocs and the consensus is that he is the smartest man alive.") == 0);

assert(determineMatchLevel(test8dist, test8w1, test8w2, TEST8\_NSTANDARDS,

"nakul and barkha nakul and barkha") == 0);

assert(determineMatchLevel(test8dist, test8w1, test8w2, TEST8\_NSTANDARDS,

"nakul and barkha and crocs and the smartest thing to do is to check whether the living people are still alive") == 0);

assert(determineMatchLevel(test8dist, test8w1, test8w2, TEST8\_NSTANDARDS,

"nakul khambhati and barkha and crocs and I think the smartest thing to do is to check whether the living people are all still alive") == 0);

assert(determineMatchLevel(test8dist, test8w1, test8w2, TEST8\_NSTANDARDS,

"") == 0);

assert(determineMatchLevel(test8dist, test8w1, test8w2, TEST8\_NSTANDARDS,

"nakul nakul and smartest") == 0);

assert(determineMatchLevel(test8dist, test8w1, test8w2, TEST8\_NSTANDARDS,

"barkha crocs the alive") == 0);

const int TEST9\_NSTANDARDS = 4; // trying things where swaps are in the standard

int test9dist[TEST9\_NSTANDARDS] = {

2, 3, 2, 3

};

char test9w1[TEST9\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"happy", "sad", "up", "down"

};

char test9w2[TEST9\_NSTANDARDS][MAX\_WORD\_LENGTH+1] = {

"sad", "happy", "down", "up"

};

assert(determineMatchLevel(test9dist, test9w1, test9w2, TEST9\_NSTANDARDS,

"I'm happy and sad. Happy because things are looking up. Down bad though because down is sometimes up") == 4); // including both ends isn't an issue

assert(determineMatchLevel(test9dist, test9w1, test9w2, TEST9\_NSTANDARDS,

"happy x x sad x x happy. up x x down x x up") == 2); // distances aren't getting mixed up (out of bounds still works)