1. My biggest obstacle throughout the project was finding out how to deal with an empty string in word1 or word2 in the editStandards function. I was able to overcome this by checking if the first character of each word is ‘\0’ because that would mean that the cstring is technically empty and only filled with a null character and thus should not be iterated through and should be removed from consideration as a matching standard.

**editStandard**

* declare variable to count incorrect match standards
* Repeatedly:
  + check if the distance is zero or negative and thus does not match
    - increment the amount of incorrect standards
  + if distance is positive
    - if the first char in either word is null increment notWorking and don't continue and increment incorrect standard counter
    - If not
      * Repeatedly:
        + if there is a char in the first word that is not a letter

increment notWorking, indicate that the first word is not functioning and break out of the loop

* + - * + Repeatedly:

if there is a char in word2 that is not a letter

increment the count of the match standards that don’t work and break out of the loop

* Repeatedly:
  + initialize needing to switch as false so the while loop will be broken out of if none are needed to switch
  + Repeatedly:
    - switch values if current distance is less than the next distance
    - switch values if first character in the first word is null
    - switch values if first character in the second word is null
    - Repeatedly:
      * if there is a nonalpha character in the c string, switch that to the next position
      * if the alpha character is uppercase, make it lowercase
      * iterate through the first c string in the second word
        + if there is a nonalpha character in the c string, switch that to the next position
        + if the alpha character is uppercase, make it lowercase
* Repeatedly:
  + initialize the biggest position to the first
  + Repeatedly:
    - if the distance of biggestPosition is larger than that of the current position, send values to the end so they are not counted
    - if the distance of biggestPosition is smaller than that of the current position, switch the values at currentPosition with that at biggestPosition
    - if the current position is not the end then send the values at the current position to the end so they are counted as invalid standards
  + find the working match cases which are represented by the difference of the total standards and the cases that don't work and return it

**determineMatchLevel**

* Create new cstring to store the working values of jeet
* Repeatedly:
  + if the value of jeet is alpha and uppercase input the lowercase value into the new cstring and increment the current position counter
  + if the value of jeet is alpha and is lower input the value into the new cstring and increment the current position counter
  + if the value of jeet is a space
    - check if it is at the beginning of the current position and do nothing if so
    - check if the character at the position before the current of the new cstring is a space and if so do nothing
    - else add a space at the current position and increment the position counter
  + check if the current value of jeet is not alpha and is not a space and if so skip over it
* Repeatedly:
  + check new c string if there is at least 1 alpha character if not return 0 for working match standards as it is empty
* create an array of cstrings and iterate through it to set all values to null
* Repeatedly:
  + assign the separate words of the cstring to different positions in the array
* Repeatedly:
  + iterate through nStandards to search for word1 and word2 in the array
  + Repeatedly:
    - iterate through the words in the array
    - if the first word has not been found yet in the array and the current word is the first word, store the position in the array of the first word
    - if the first word has been found and the current word is the second word being searched for, store the position in the array of the second word and break
  + if the positions were found, check if the distance between the positions is less than or equal to the distance given in the distance array and if so increment match standards
* return the value of matchStandards

c. Test Cases

const int TEST1\_NSTANDARDS = 5;

int test1dist[TEST1\_NSTANDARDS] = {

2, 1, 62, 4, 34

};

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

"dog", "eccentric", "eccentric", "sp@ce", "dog"

};

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

"house", "billionaire", "billionaire", "capsule", "house"

};

assert(editStandards(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS) == 2 && test1dist[0] == 62);

*This tests for handling 2 different cases of replicated word1 and word2s by making sure the numbers have the highest duplicates at the beginning in addition to incorrect characters in word1*

const int TEST1\_NSTANDARDS = 5;

int test1dist[TEST1\_NSTANDARDS] = {

2, -3, 62, 4, 34

};

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

"", "eccentric", "eccentric", "space", "dog"

};

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

"house", "billionaire", "billionaire", "capsule", "h34ous"

};

assert(editStandards(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS) == 2);

*This tests for negative distances not being counted in addition to empty word1 and number in word2*

const int TEST1\_NSTANDARDS = 7;

int test1dist[TEST1\_NSTANDARDS] = {

2, 1, 62, 0 , 4, 34, 3

};

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

"dog", "big", "little", "taxi" , "space", "cat" , "racing"

};

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

"house", "billionaire", "billionaire", "cab", "capsule", "house" , ""

};

assert(editStandards(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS) == 5);

*Tests if the same second word affects the number of correct match standards (it shouldn’t) and if it can handle an empty second word in addition to getting rid of a 0 direction*

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS, "The eccentric outspoken billionaire launched a space station cargo capsule.") == 1);

*Tests if the function can handle changing the string when there is a capital and punctuation and counting the working match standards*

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS, "The eccentric outspoken billionaire launched a space capsule.") == 1);

*Tests if the function can handle changing the string when there is a capital and punctuation and weird spacing and counting the working match standards*

assert(determineMatchLevel(test1dist, test1w1, test1w2, TEST1\_NSTANDARDS, "\*\*\*\* 2022 \*\*\*\*") == 0);

*Tests if the function can handle changing the string when there are no letters (just numbers) and there is weird spacing*

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

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

*Tests if the function can handle changing the string when there is many capitals in a row and punctuation and 2 spaces at the beginning of the phrase*

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

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

*Tests if the function can handle repeats of the same 2 words and not count them*