One of the greatest obstacles that stood in my way of completing the project was figuring out a design for determineQuality. I tried mapping things out by hand and took many different approaches but couldn’t get my programs to work. I overcame this by talking to an upperclassmen and he suggested that I create a new array of cstrings to store each individual cstring in document. Another obstacle that stood in my way was the fact that there were so many different requirements the program had to account for. I overcame this by developing incrementally.

Pseudocode:

int standardizeRules(int distance[], char word1[][MAX\_WORD\_LENGTH + 1], char word2[][MAX\_WORD\_LENGTH + 1], int nRules){

*loop through every character in word1 and make it lowercase*

*loop through every character in word2 and make it lowercase*

*set a bool, valid, representing whether a rule is standardized*

*loops through the rules*

*if the distance is nonpositive or either of the cstrings are empty, valid is false*

*loops through each character in the current index of w1 and w2 to make sure every character is an alphabetic character. If it isn’t valid is set to false*

*loops through the rest of the elements in w1 and w2 to see if there are duplicates. If there is a duplicate with a greater distance, valid is false*

*if valid is false,*

*shift every element over one to the left.*

*Decrement the current index and nRules*

*return nRules*

*}*

int determineQuality(const int distance[],

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

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

int nRules,

const char document[]){

*initialize integers score and copyLength to be 0*

*initialize cstring documentCopy*

*Loops through document and copies document into documentCopy with only lowercase letters and no special symbols. Also changes copyLength to become the length of documentCopy*

*initialize integers temp, placement, and wordListLength to be 0*

*initialize an array of cstrings wordlist*

*initialize a cstring tempString to be an empty cstring*

*loops through document copy and tracks where a word has ended. Add that word to wordlist until wordlist consist of an array of cstrings representing the words in document*

*loop through each rule and see if the rule appears in wordlist. If it appears and the distance between them is less than or equal to the distance allowed, we increment score and move onto the next rule*

*return score*

*}*

**Tests:**

**standardizeRules…**

// This test checks for non-alphabetic characters and makes sure everything is lowercased

int  a1[] = { 5,3,1,6,2,7,12 };

char b1[][MAX\_WORD\_LENGTH + 1] = { "alpha", "phi", "rho-rho", "omega", "PSI", "GaMmA", "Chi!!" };

char c1[][MAX\_WORD\_LENGTH + 1] = { "leadership", "friendship", "volunteer", "service", "APO-values", "", "Pledging" };

int  d1 = 7;

int n= standardizeRules(a1, b1, c1, d1);

cout<< “number of standardized rules: “<< n <<endl;

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

cout<< a1[i] <<”, “ << b1[i] << “, “<<c1[i] << endl;

}

// This test checks for non-positive inputs in the int array

int  a1[] = { 5,-3,1,-6,0,-7,12 };

char b1[][MAX\_WORD\_LENGTH + 1] = { "alpha", "phi", "rhorho", "omega", "PSI", "GaMmA", "Chi" };

char c1[][MAX\_WORD\_LENGTH + 1] = { "leadership", "friendship", "volunteer", "service", "APOvalues", "APO", "Pledging" };

int  d1 = 7;

int n= standardizeRules(a1, b1, c1, d1);

cout<< “number of standardized rules: “<< n <<endl;

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

cout<< a1[i] <<”, “ << b1[i] << “, “<<c1[i] << endl;

}

// This test checks for duplicates

int  a1[] = { 5,3,1,6,2,7,7 };

char b1[][MAX\_WORD\_LENGTH + 1] = { "alpha", "phi", "leadership", "omega", "PSI", "GaMmA", "Chi" };

char c1[][MAX\_WORD\_LENGTH + 1] = { "leadership", "friendship", "alpha", "service", "APOvalues", "chi", "gamma" };

int  d1 = 7;

int n= standardizeRules(a1, b1, c1, d1);

cout<< “number of standardized rules: “<< n <<endl;

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

cout<< a1[i] <<”, “ << b1[i] << “, “<<c1[i] << endl;

}

**determineQuality…**

const int TEST1\_NCRITERIA = 4;

int test1dist[TEST1\_NCRITERIA] = {

2, 4, 1, 13

};

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

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

};

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

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

};

// Regular test

assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,"The mad UCLA scientist unleashed a deranged evil giant robot.") == 2);

//Tests multiple spaces

assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA, "The mad UCLA scientist unleashed a deranged robot.") == 2);

//Tests non alphabetical characters

assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,"\*\*\*\* 2014 \*\*\*\*") == 0);

// Tests non alphabetic characters and caps

assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA," That plot: NEFARIOUS!") == 1);

// Tests repeats

assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,"deranged deranged robot deranged robot robot") == 1);

// Tests distances

assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"Two mad scientists suffer from deranged-robot fever.") == 0);

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