1. A brief description of notable obstacles

In development of this project, the first obstacle I encountered is how can I divide the function into sub-functions, and what is the job of each sub-function. Then, after having a brief outline and to-do list, I need to determine the return type and parameters of each sub-function. Another obstacle that I encountered is how to use C strings to achieve do the job of normal C++ strings. For example, I need to replace “=“ with “strcpy”, and “==“ with “strcmp”. And finally, I need to thoroughly test my program to cover every case I can predict.

1. A description of the design of the program

int cleanupRules(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

call the function *eliminateNonLetter* to eliminate all characters that is not a letter

call the function *toLowerCase* to convert all characters to lower case

call the function *eliminateEmptyWordin* to eliminate all empty wordin

call the function *eliminateSameWordinWordout* to eliminate the word-rule that wordin and wordout are the same

call the function *eliminateSameOneWordRule*, for the one-word case, eliminate all the one-word rule and two-word rule that has the same wordin

call the function *eliminateSameTwoWordRule*, for the two-word case, eliminate all the two-word rule with the same wordin that has the same wordout

call the function *eliminateEmptyMatchRule* to eliminate all the empty match rules in the document

return the value of *getRuturnValuereturn*

int determineScore(const char document[], const char wordin[][MAX\_WORD\_LENGTH + 1], const char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

if nRules is zero or negative, return 0

create a local sting called ducumentString

store document into ducumentString

call the function *convertToAllLowerAndAllLetter* to convert the string to all lower case and eliminate the nonletter characters

create a local array of strings called documentArray

call the function *convertDocumentToArray* convert a string to a array of strings, with each element seperated by a space character

return the value calculated by the function *countScores*

void eliminateNonLetter(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

Go through every string in the array wordin

Go through every character in the current string

if the current string contains a nonletter, replace the current string to empty string

Go through every string in the array wordout

Go through every character in the current string

if the current string contains a nonletter, replace the current string to empty string

void toLowerCase(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

Go through every string in the array wordin

Go through every character in the current string

Call the function *tolower* to convert every character to lower case

Go through every string in the array wordout

Go through every character in the current string

Call the function *tolower* to convert every character to lower case

void eliminateEmptyWordin(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

Go through every string in the wordin array

If the current string in wordin is empty, assign its corresponding string in wordout to be empty

void eliminateSameWordinWordout(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

Go through every string in the wordin and wordout array

If the current string in wordin array and wordout array is the same

Assign both the wordin and wordout array to be empty

void eliminateSameOneWordRule(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

declare a local string called current

Go through every string in wordin

If the current word rule is a one-word rule:

search through the strings in wordin

if there is a string in wordin that is the same as current, set both the string and its corresponding string in wordout to be empty

void eliminateSameTwoWordRule(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

Go through every string in wordin

If the current word rule is a two-word rule:

Search through the strings in wordin and wordout

If there is a string that has the both the same wordin string and the wordout string as the current one, set its wordin string and wordout string to empty

void eliminateEmptyMatchRule(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

declare a string called tempIn

declare a string called temptOut

Use bubble sort to move all the empty strings to the back of the array

int getRuturnValue(char wordin[][MAX\_WORD\_LENGTH + 1], char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

go through all the strings in both wordin array and and wordout array

if the current wordin and wordout are both empty

return the index of this array

void convertToAllLowerAndAllLetter(char documentString[])

declare a new empty string called stringWithOnlyLetters

Go through documentString

If the current string is a letter or blank space:

Convert the current letter to lower case

Append it into the string stringWithOnlyLetters

Assign stringWithOnlyLetters to documentString

void convertDocumentToArray(char documentString[], char documentArray[][MAX\_WORD\_LENGTH + 1])

set the 2D array documentArray to empty

Go through evert string in the documentString

Find the index of the blank space

Store the substring before the black and after the previous blank to documentArray

Store the last substring in the documentString to documentArray

int countScores(const char documentArray[][MAX\_WORD\_LENGTH + 1], const char wordin[][MAX\_WORD\_LENGTH + 1], const char wordout[][MAX\_WORD\_LENGTH + 1], int nRules)

declare a int called score and set it to 0

declare a bool called match and set it to false

Go through both the wordin and wordout array

Go through the words in documentArray

if the current wordin is the same as the current word in documentArray:

set match to true

Go through the words in documentArray

If we can find the string in the documentArray that is the same as the current wordout

Set match to be false

If the match is true, add 1 to score, and reset match to false

Return score

1. A list of test data

int cleanupRules(char wordin[][MAX\_WORD\_LENGTH+1],

char wordout[][MAX\_WORD\_LENGTH+1],

int nRules)

Test for clean rules:

wordin[5][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, “nice”, “email”, “Live”};

wordout[5][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”};

nRules=5;

Test for repeated one-word rule

wordin[7][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, “nice”, “email”, “Live”, “Big”, “Big”};

wordout[7][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”, ““, “tiny”};

nRules=5;

Test for repeated two-word rule

wordin[7][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, “nice”, “email”, “Live”, “Big”, “Big”};

wordout[7][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”, “small”, “tiny”};

nRules=5;

Test for rules containing non-letter

wordin[5][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, “nice”, “e-mail”, “Live”}

wordout[5][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”};

nRules=5;

Test for rules has zero nRules

wordin[5][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, “nice”, “email”, “Live”};

wordout[5][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”};

nRules=0;

Test for rules has negative nRules

wordin[5][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, “nice”, “email”, “Live”};

wordout[5][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”};

nRules=-1;

Test for two-word rule has same wordin and wordout:

wordin[7][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, “nice”, “email”, “Live”, “Big”, “Big”};

wordout[7][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”, “big”, “tiny”};

nRules=5;

Test for one wordl rule that has empty wordin:

wordin[5][MAX\_WORD\_LENGTH+1]={“Big”, “Hi”, ““, “email”, “Live”};

wordout[5][MAX\_WORD\_LENGTH+1]={“Small”, ““, “love”, “hi”, “five”};

nRules=5;

Test for complex cases:

wordin[12][MAX\_WORD\_LENGTH+1]= { “confusion”,”FAMILY”,”charm”,”hearty”,”house”,”worn-out”,”family”,”charm”,”ties”,”“,”charm”,”Family” };

wordout[12][MAX\_WORD\_LENGTH+1]= { ““,”TIES”,”confusion”,”hearty”,”intrigue”,”younger”,”first”,”“,”family”,”frightened”,”“,”tIeS” };

nRules=12;

int determineScore(const char document[],

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

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

int nRules)

char wordin[nRules][MAX\_WORD\_LENGTH+1] = {"family", "unhappy", "horse"};

char wordout[nRules][MAX\_WORD\_LENGTH+1] = { "”, "horse",""};

Containing both one-word rule and two-word rule

char document[] = “Happy families are all alike; every unhappy family is unhappy in its own way.”

Containing one-word rule only

char document[] = “Happy families are all alike; every unhappy family is sad in its own way.”

Containing the two word rule only

char document[] = “Happy people are all alike; every unhappy people is unhappy in its own way.”

Don’t match the two-word rule

char document[] = "Happy horses are all alike; every unhappy horse is unhappy in its own way."

Don’t match any rules at all

char document[] = "HAHAHAHAHA NO MATCH"

Containing non letter

char document[] = "this is a black-horse"

The substring of a word can match rules

char document[] = "this is a blackhorse"