Ryan Lam

A. One notable obstacle I overcame was making sure that I did not go out of bounds on C-string indexes while copying various strings. I had trouble as my program would skip over the null byte character at the end of a string and begin accessing incorrect memory locations. Another notable obstacle I overcame was designing algorithms that did not use variable length arrays and did not alter the input arrays and strings. I had to copy the input strings in order to manipulate them myself.

B. Program Design

makeProper:

if number of patterns is negative

return 0

loop through pattern arrays

if a pattern contains empty word or negative separation

remove pattern from arrays and decrement num. of patterns

else

loop through characters in word1 and word2 arrays of pattern

change uppercase letters to lowercase

if word contains non-letters

remove pattern from arrays and decrement num. of patterns

loop through arrays of proper patterns

if a pattern repeats in the arrays

remove pattern with smaller separation and decrement num. of patterns

if separations are the same size, remove either one

return number of patterns

rate:

if number of patterns is negative

return 0

if document is empty

return 0

create c-string to copy document

loop through characters in document

if character is a letter or space

convert to lowercase and add character to copy of document

create c-string array to store individual words in document copy

loop through characters in copy of document

if character is a letter

add characters to word in c-string array and increment loop until a space is reached

move to next index in c-string array

create result integer

loop through pattern arrays

loop through c-string array of document words

if pattern in word1 or word2 matches word in array

if corresponding word in pattern matches and separation is valid

increment result

move to next pattern

return result

C. Test Data

For rate function:

const int nPatternsTest1 = 4;

char word1Test1[nPatternsTest1][MAX\_WORD\_LENGTH+1] = {

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

};

char word2Test1[nPatternsTest1][MAX\_WORD\_LENGTH+1] = {

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

};

int separationTest1[nPatternsTest1] = {

1, 3, 0 ,12

};

assert(rate("The mad UCLA scientist unleashed a deranged evil giant robot.",

word1Test1, word2Test1, separationTest1, nPatternsTest1) == 2);

//Rate works correctly

assert(rate("The mad UCLA scientist unleashed a deranged robot.",

word1Test1, word2Test1, separationTest1, nPatternsTest1) == 2);

//Document with multiple consecutive spaces

assert(rate("\*\*\*\* 2018 \*\*\*\*",

word1Test1, word2Test1, separationTest1, nPatternsTest1) == 0);

//Document with characters and numbers but no letters

assert(rate(" That plot: NEFARIOUS!",

word1Test1, word2Test1, separationTest1, nPatternsTest1) == 1);

//Document with mix of characters and letters, and uppercase letters

assert(rate("deranged deranged robot deranged robot robot",

word1Test1, word2Test1, separationTest1, nPatternsTest1) == 1);

//Repeated occurrence of patterns

assert(rate("That scientist said two mad scientists suffer from deranged-robot fever.", word1Test1, word2Test1, separationTest1, nPatternsTest1) == 0);

//Document with no matching patterns

assert(rate("The noMADic unscientistics eat pmadie every Thaveursday.",

word1Test1, word2Test1, separationTest1, nPatternsTest1) == 0);

//Document with patterns embedded in words

assert(rate("deranged deranged deranged deranged deranged",

word1Test1, word2Test1, separationTest1, nPatternsTest1) == 0);

//Document with repeated word

assert(rate("", word1Test1, word2Test1, separationTest1, nPatternsTest1) == 0);

//Empty document

For makeProper function:

const int nPatternsTest2 = 8;

char word1Test2[nPatternsTest2][MAX\_WORD\_LENGTH+1] = {

"mad", "deranged", "nEfArIoUs", "bl%ue42", "plot", "have", "mad", "mad"

};

char word2Test2[nPatternsTest2][MAX\_WORD\_LENGTH+1] = {

"sciENTIst", "robot", "plot", "UCLA", "nefarious", "mad", "scientist", "scientist"

};

int separationTest2[nPatternsTest2] = {

1, -2, 3, 4, 5, 6, 7, 7

};

assert(makeProper(word1Test2, word2Test2, separationTest2, nPatternsTest2) == 3);

//MakeProper works correctly

//Makes all letters lowercase

//Removes patterns with non-letters

//Removes repeated patterns in either order

//Removes one of multiple identical patterns

const int nPatternsTest3 = -4;

char word1Test3[4][MAX\_WORD\_LENGTH+1] = {

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

};

char word2Test3[4][MAX\_WORD\_LENGTH+1] = {

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

};

int separationTest3[4] = {

1, 2, 3 ,4

};

assert(makeProper(word1Test3, word2Test3, separationTest3, nPatternsTest3) == 0);

//Negative number of patterns

const int nPatternsTest4 = 4;

char word1Test4[nPatternsTest4][MAX\_WORD\_LENGTH+1] = {

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

};

char word2Test4[nPatternsTest4][MAX\_WORD\_LENGTH+1] = {

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

};

int separationTest4[nPatternsTest4] = {

1, 2, 3 ,4

};

assert(makeProper(word1Test4, word2Test4, separationTest4, nPatternsTest4) == 2);

//Empty words are removed from patterns

const int nPatternsTest5 = 4;

char word1Test5[nPatternsTest5][MAX\_WORD\_LENGTH+1] = {

"have", "have", "have", "have"

};

char word2Test5[nPatternsTest5][MAX\_WORD\_LENGTH+1] = {

"mad", "mad", "mad", "mad"

};

int separationTest5[nPatternsTest5] = {

1, 2, 3 ,4

};

assert(makeProper(word1Test5, word2Test5, separationTest5, nPatternsTest4) == 1

&& separationTest5[0] == 4);

//Only keeps highest separation value of repeated patterns