**Notable Obstacles:**

While working on this project, I encountered a problem with parsing through c strings. In one of my functions, I was accessing garbage values of the array because I was not checking for the null byte and instead was going for the max possible length of the array. As a result, my program was exhibiting undefined behavior as the elements in the array were filled with miscellaneous values that sometimes triggered certain parts of my code and other times did not. I solved the problem by looping until hitting the null byte instead.

**Pseudocode:**

makeProper:

if nPatterns <= 0 return 0

check for negative separation values

if negative separation value

overwrite elements at that position in all 3 arrays

decrement num of valid elements

decrement loop counter

check for words with no characters

if empty words found

overwrite elements at that position in all 3 arrays

decrement num of valid elements

decrement loop counter

check for word with non-alphabetical characters

if non-alphabetical characters found

overwrite elements at that position in all 3 arrays

decrement num of valid elements

decrement loop counter

break

else

make make all characters lowercase

check for repeat patterns

if repeat pattern found

check separator values at each position

overwrite elements at the position with the lower separation value in all 3 arrays

decrement num of valid elements

break

return num of valid elements

rate:

if nPatterns <= 0 return 0

create a copy of document and remove all non alphabetic characters (except spaces) and make characters lowercase

for each pattern

repeatedly:

get next word in document copy

if that word matches either of the words from the pattern arrays

repeatedly separation times:

get next word in document copy

if that word matches the respective word from the other word array

increment num matches

set is match to true

break

return num matches

**Test Cases:**

**makeProper**:

//should remove repeat patterns and making valid patterns lowercase

const int TEST3\_NCRITERIA = 4;

int a1[TEST3\_NCRITERIA] = {2, 1, 1, 13};

char b1[TEST3\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {"mad", "PLOT", "nefarious", "scientist"};

char c1[TEST3\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", "plot", "mad"};

assert(makeProper(a1, b1, c1, TEST3\_NCRITERIA) == 2);

a1[0] == 1, b1[0] == “plot”, c1[0] == “nefarious”

//should remove patterns with illegal characters

const int TEST2\_NCRITERIA = 4;

int a2[TEST2\_NCRITERIA] = {2, 4, 1, 13};

char b2[TEST2\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {"mad", "half-witted", "nefaRIous", "have"};

char c2[TEST2\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {"scientist", "guy", "plot", "MaD"};

assert(makeProper(a2, b2, c2, TEST2\_NCRITERIA) == 3);

a2[1] != 4, b2[1] != “half-witted”, c2[1] != “guy”

//should remove patterns with negative separation value

const int N1 = 3;

int a3[N1] = {2, -1, 3};

char b3[N1][MAX\_WORD\_LENGTH+1] = {"mad", "PLOT", "nefarious"};

char c3[N1][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", "plot"};

assert(makeProper(a3, b3, c3, N1) == 2);

a3[1] != -1, b3[1] != PLOT, c3[1] !- NEFARIOUS

//non-positive nPatterns should return 0

const int N2 = 3;

const int N3 = -3;

int a4[N1] = {2, -1, 3};

char b4[N1][MAX\_WORD\_LENGTH+1] = {"mad", "PLOT", "nefarious"};

char c4[N1][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", "plot"};

assert(makeProper(a4, b4, c4, N3) == 0);

//repeat pattern with same separation and multiple repeat patterns – should remove patterns with smallest separations

const int N4 = 5;

int a5[N4] = {2, 4, 3, 4, 4};

char b5[N4][MAX\_WORD\_LENGTH+1] = {"mad", "PLOT", “deranged”, “plot”, “plot”};

char c5[N4][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", “robot”, “nefarious”, “nefarious”};

assert(makeProper(a5, b5, c5, N4) == 3);

a5[1] == 4, b5[1] == “plot”, c[5] == “nefarious”

//repeat pattern with different separation and multiple repeat patterns – should remove patterns with smallest separations

const int N5 = 5;

int a6[N5] = {2, 4, 3, 2, 5};

char b6[N5][MAX\_WORD\_LENGTH+1] = {"mad", "PLOT", “deranged”, “plot”, “plot”};

char c6[N5][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", “robot”, “nefarious”, “nefarious”};

assert(makeProper(a5, b5, c5, N4) == 3);

a5[1] == 3, b5[1] == “deranged”, c5[1] == “robot”, a5[2] == 5, b5[2] == “plot”, c5[2] == “nefarious”

//inverted pattern with multiple patterns and different separation – should remove patterns with smallest separations

const int N6 = 5;

int a7[N6] = {2, 4, 3, 2, 5};

char b7[N6][MAX\_WORD\_LENGTH+1] = {"mad", "PLOT", “deranged”, “nefarious”, “nefarious”};

char c7[N6][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", “robot”, “plot”, “plot”};

assert(makeProper(a5, b5, c5, N4) == 3);

a5[1] == 3, b5[1] == “deranged”, c5[1] == “robot”, a5[2] == 5, b5[2] == “nefarious”, c5[2] == “plot”

//only one word from a pattern appears more than once – should not remove

const int N7 = 4;

int a8[N6] = {2, 4, 3, 2};

char b8[N7][MAX\_WORD\_LENGTH+1] = {"mad", "PLOT", “deranged”, “mad”};

char c8[N7][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", “robot”, “plot”};

assert(makeProper(a5, b5, c5, N4) == 4);

//empty string in word pattern array – should remove the pattern with empty string

const int N7 = 4;

int a8[N6] = {2, 4, 3, 2};

char b8[N7][MAX\_WORD\_LENGTH+1] = {"", "PLOT", “deranged”, “mad”};

char c8[N7][MAX\_WORD\_LENGTH+1] = {"scientist", "NEFARIOUS", “robot”, “plot”};

assert(makeProper(a5, b5, c5, N4) == 3);

**rate:**

//more than one pattern instance in document (w1 first & w2 first) – should only count each pattern once

const int TEST1\_NRULES = 4;

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

"mad", "deranged", "nefarious", “deranged”

};

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

"scientist", "robot", "plot", “deranged”

};

int test1dist[TEST1\_NRULES] = {

1, 3, 0, 12

};

assert(rate("deranged deranged deranged deranged deranged", test1w1, test1w2, test1dist, TEST1\_NRULES) == 1);

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

//illegal characters in document should be removed

const int TEST1\_NCRITERIA = 5;

int \_test1dist[TEST1\_NCRITERIA] = {

2, 4, 1, 13 , 3

};

char \_test1w1[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

char \_test1w2[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

assert(rate(" That plot: NEFARIOUS!", \_test1w1, \_test1w2, \_test1dist, TEST1\_NCRITERIA) == 1);

assert(rate("\*\*\*\* 2014 \*\*\*\*", \_test1w1, \_test1w2, \_test1dist, TEST1\_NCRITERIA) == 0);

//nPatterns is less than or equal to 0 should return 0

const int TEST1\_NCRITERIA = 5;

int \_test1dist[TEST1\_NCRITERIA] = {

2, 4, 1, 13 , 3

};

char \_test1w1[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

char \_test1w2[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

assert(rate("The mad UCLA scientist unleashed a deranged robot.", test1w1, test1w2, test1dist, -2) == 0);

assert(rate("The mad UCLA scientist unleashed a deranged robot.", test1w1, test1w2, test1dist, 0) == 0);

//empty document should return 0

const int TEST1\_NCRITERIA = 5;

int \_test1dist[TEST1\_NCRITERIA] = {

2, 4, 1, 13 , 3

};

char \_test1w1[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

char \_test1w2[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

assert(rate("", \_test1w1, \_test1w2, \_test1dist, TEST1\_NCRITERIA) == 0);

//more than 1 space in between words and illegal characters should be handled properly

const int TEST1\_NCRITERIA = 5;

int \_test1dist[TEST1\_NCRITERIA] = {

2, 4, 1, 13 , 3

};

char \_test1w1[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

char \_test1w2[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

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

};

assert(rate("The mad UCLA $scientist unle@asheD a deranged robot.", \_test1w1, \_test1w2, \_test1dist, TEST1\_NCRITERIA) == 2);