1. In cleanupRules() function:

Since I decide to convert all the strings that are not in clean form to empty string, the final array I get will contain many empty spaces, and all the valid strings separate in the array. The biggest task is how to make them consecutive. I design a for loop to detect if there are both empty spaces in wordin and wordout. Once detected, finding the first non-empty string behind them and make a copy to them. Them eliminate those strings at that position. Keep repeating this process, and the final array I get is an array which contains all the valid stings in front and all empty strings in back.

In determineScore Function:

One obstacle is how to determine whether the string in wordin really exist in the sentence in document. The first methods I came up with is suing strstr() function. It works well at first, but there is a serious problem. For example, I want to find “jump” in sentence “jumping every day is good for our health”. Even though it does not exist (jump and jumping are two different words), strstr() will still return a pointer to the “j” of “jumping”. Therefore, I turn to test the data in document manually. I cut off each word from document and store it in a temporary string, then compare this string with the string in wordin. Keep repeating this process until every string in wordin is finished.

1. cleanupRules()

{

Set nRules to zero if it is negative

repeatedly in wordin and wordout

repeatedly in each character in string in wordin

eliminate non-alphabetic string

repeatedly in each character in string in wordout

eliminate non-alphabetic string

repeatedly in wordin and wordout

repeatedly in each character in string in wordin

convert to lower case

repeatedly in each character in string in wordout

convert to lower case

repeatedly in wordin and wordout

eliminate identical strings at the same position

eliminate the strings in wordout on empty position in wordin

repeatedly in wordin and wordout

find the one-word rule string

repeatedly in wordin

eliminate the identical strings

repeatedly in wordin and wordout

repeatedly in wordin and word out

eliminate the identical two-word rule strings

repeatedly in wordin and wordout

find empty strings in wordin and wordout

repeatedly find non-empty strings behind them

make copy of them

eliminate them

repeatedly in wordin and wordout

find the empty string position

return position

}

determineScore()

{

Create a cstring called document2

Repeatedly in document

Find the alphabetic character and space

Make copy to dcocument2

Create an integer called countScore

Repeatedly in wordin and wordout

Find one-word rule

Repeatedly, store each word in cstring temp

if temp and wordin are identical

Increase countScore by one

Find two-word rule

Create two bool values in and out

Repeatedly, store each word in cstring temp

if temp and wordin are identical

set in to true

if temp and wordout are identical

set out to false

if in and out are true

increase countScore by one

return countScore

}

c.

cleanupRules():

const int nRules = 12;

char wordin[12][21] = { "conf@@sion", "FAMILY", "charm", "hearty", "house", "worn-out", "family", "charm", "ties", "", "charm", "FaMiLy" };

char wordout[12][21] = { "", "TIES", "confusion", "hearty", "intrigue", "younger", "first", "", "family", "frightened", "", "tIeS" };

(wordin, wordout, nRules)

This test data test following conditions: eliminate stings contain non-alphabetic characters, convert upper-case characters to lower case, eliminate strings if they are the same in wordin and wordout, eliminate strings only in wordout, keep only one-word rule if there are many copies in wordin, eliminate identical two-word rules.

(wordin, wordout, -3)

Negative nRules will be treated as zero

determineScore():

const int TEST1\_NRULES = 3;

char test1win[TEST1\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"family", "unhappy", "horse",};

char test1wout[TEST1\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"", "horse", "",};

("Happy families are all alike; every unhappy family is unhappy in its own way.", test1win, test1wout, TEST1\_NRULES) normal test

("Happy horses are all alike; every unhappy horse is unhappy in its own way.", test1win, test1wout, TEST1\_NRULES - 1) zero score returned

("Happy horses are all alike; every unhappy horse is unhappy in its own way.", test1win, test1wout, TEST1\_NRULES) different document sentence

("A horse! A horse! My kingdom for a horse!", test1win, test1wout, TEST1\_NRULES) == 1) sentence that contains other punctuation

("horse:stable ratio is 10:1", test1win, test1wout, TEST1\_NRULES) test if “horse” is not equal to “horsestable”

("\*\*\*\* 2020 \*\*\*\*", test1win, test1wout, TEST1\_NRULES) invalid sentence

("Happy families are all alike; every unhappy family is unhappy in its own way.", test1win, test1wout, -3) treat negative nRules as zero