Notable obstacles:

In the cleanupRules function, I had initially written all of the conditions to check for invalid rules together, and the function was unsuccessful when I tested it at the end. I, then, went through and tested each condition/loop within the function individually, while commenting out the remaining portions of the function. Evidently, incrementally developing the program would’ve been helpful, instead of going back and having to isolate portions of my function. Furthermore, since I used rotateLeft to move invalid rules out, I had an issue where I was skipping the rule after one that I removed because the loop was moving past that index. I solved this by decrementing the loop counter whenever an element was rotated left, so that it’d check the same element again.

In determineScore, I had initially only used strstr to search for rules within the document, but this caused it to come up with matches even when a word was contained within another (i.e. “an” in “another”). The solution to this issue complicated my function a lot more, but I decided to create temporary rules to check for additional spaces surrounding a word. While this worked to solve the issue with words contained in other words, it was not matching rules that appeared at the beginning or end of a document, since there aren’t spaces surrounding those words. In order to solve this, I modified the copy of my document and added a space both at the beginning and the end. Furthermore, I tried to use strlen of each rule in order to make the temporary rules, but this relied on undefined values. While this did not produce an error in xcode, it did in the linux server. I solved this problem by making them the max length of a rule + 2 extra indices for the spaces I added surrounding a rule.

Pseudocode:

Main

Assert testers

**rotateLeft function**

Check if pos is last element

Temp variable with element at position

Repeatedly:

copy elements from right to left element

Set last element to temp

**cleanupRules function**

check valid nRules #

Repeatedly:

Make characters in wordin lowercase

Repeatedly:

Make characters in wordout lowercase

Check for empty wordin → repeatedly:

If empty, rotate all rules left

Subtract from # of valid rules / Move loop counter back

Check if there’s 0 valid

Check for non alphabetical characters in wordin → repeatedly:

Check if character is alphabetic

If not, rotate all rules left

Subtract from # of valid rules / Move loop counter back

Check if there’s 0 valid

Check for non alphabetical characters in wordout → repeatedly:

Check if character is alphabetic

If not, rotate all rules left

Subtract from # of valid rules / Move loop counter back

Check if there’s 0 valid

Check if wordin and wordout are the same → repeatedly:

Compare corresponding wordin and wordout

If same, rotate all rules left

Subtract from # of valid rules / Move loop counter back

Check if there’s 0 valid

Check if no repeated wordin as a one word rule → repeatedly:

Check if it’s a one word rule

If so, check for repeats of its wordin

If repeated, rotate all rules left,

Subtract from # of valid rules / Move loop counter back

Check if there’s 0 valid

Check for duplicate two word rules → repeatedly:

Search for corresponding same wordin and wordout of elements

If repeated two word rules, rotate all rules left

Subtract from # of valid rules / Move loop counter back

Check if there’s 0 valid

Return the # of valid rules

**determineScore function**

Check valid nRules #

Create cstring to copy doc

Put space at beginning of copy

Repeatedly:

Copy alphabetical and space characters into copy of document

Put space at end of copy

Repeatedly loop through all rules:

Create copy of the rules

Start copy with space and append space to end of copy

Check if it’s a one word rule

Search for copy of rule in doc

If found, add 1 to counter for # of valid rules

If it’s not a one word rule

Search for wordin copy in doc and check that wordout copy is not in doc

If both conditions satisfied, add 1 to counter for # of valid rules

Return counter for # of valid rules

**Test Cases for cleanupRules:**

wordin = {“ABCD”, “XYZ”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, 0);

* Check that cleanupRules recognizes 0 nRules and returns no clean rules

wordin = {“ABCD”, “XYZ”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, -10);

* Check that cleanupRules recognizes negative nRules and returns no clean rules

wordin = {“ABCD”, “XYZ”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, 400);

* nRules > the number of rules // code wouldn’t be able to handle correctly

wordin = {“ABCDasdsdsdsdsdsdsadkjalsdflkklasdqwkejdsa”, “XYZ”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, 0);

* Match rule that’s longer than 20 characters // code wouldn’t be able to handle correctly

wordin = {“ABCD”, “XYZ”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, 2);

* Check that cleanupRules function works to lowercase all uppercase characters in both wordin and wordout

wordin = {“”, “XYZ”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, 2);

* Check that cleanupRules works to check empty wordin

wordin = {“”, “”, “”};

wordout = {“ABCD”, “XYZ”, “asdf};

cleanupRules(wordin, wordout, 3);

* Checking for several empty wordins/ 0 valid rules

wordin = {“a-b”, “XYZ”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, 2);

* Checking for non alphabetical characters in wordin

wordin = {“a-b”, “.....”}

wordout = {“ABCD”, “XYZ”};

cleanupRules(wordin, wordout, 2);

* Checking for several non alphabetical characters in wordin

wordin = {“ABC”, “DEF”}

wordout = {“\*\*\*”, “asd”};

cleanupRules(wordin, wordout, 2);

* Checking for non alphabetical characters in wordout

wordin = {“ABC”, “DEF”}

wordout = {“\*\*\*”, “..”};

cleanupRules(wordin, wordout, 2);

* Checking for several non alphabetical characters in wordout

wordin = {“...”, “DEF”}

wordout = {“\*\*\*”, “..”};

cleanupRules(wordin, wordout, 2);

* Checking for non alphabetical characters in both wordin and wordout

wordin = {“ABC”, “DEF”, “...”}

wordout = {“asdd”, “asdf”, “\*\*”}

cleanupRules(wordin, wordout, 3);

* Checking for non alphabetical characters at the end of wordin and wordout
* Should return unchanged

wordin = {“a”, “DEF”}

wordout = {“a”, “asdfgh”};

cleanupRules(wordin, wordout, 2);

* Checking for same wordin and wordout

wordin = {“a”, “asds”, “b”}

wordout = {“a”, “asds”, “f”};

cleanupRules(wordin, wordout, 3);

* Checking for several same wordin and wordouts in a row

wordin = {“kasd”, “asdf”, “b”}

wordout = {“ooweiw”, “asdfsd”, “b”};

cleanupRules(wordin, wordout, 3);

* Checking for same wordin and wordout at the end of wordin and wordout

wordin = {“kasd”, “kasd”, “b”}

wordout = {“”, “asdfsd”, “b”};

cleanupRules(wordin, wordout, 3);

* Checking for a one word rule that’s repeated in a two word rule

wordin = {“kasd”, “kasd”, “b”}

wordout = {“”, “”, “b”};

cleanupRules(wordin, wordout, 3);

* Checking for a one word rule that’s repeated in another one word rule

wordin = {“kasd”, “kasd”, “kasd”}

wordout = {“”, “”, “skdasd”};

cleanupRules(wordin, wordout, 3);

* All repeats of one word rule // all invalid rules

wordin = {“abc”, “kasd”, “abc”}

wordout = {“asd”, “”, “asd”};

cleanupRules(wordin, wordout, 3);

* Checking for duplicate two word rules

wordin = {“\*\*\*”, “kasd”, “abc”}

wordout = {“abc”, “”, “abc”};

cleanupRules(wordin, wordout, 3);

* Non Alphabetical characters as well as same wordin/wordout

wordin = {“aaaa”, “aaaa”, “abc”}

wordout = {“abc”, “”, “abc”};

cleanupRules(wordin, wordout, 3);

* Repeated wordins and same wordin/wordout

wordin = {“aaaa”, “aaaa”, “abc”, “aaaa”}

wordout = {“abc”, “”, “abc”, “abc”};

cleanupRules(wordin, wordout, 4);

* Repeated wordin, same wordin/wordout, and duplicated two word rules

**Test Cases for determineScore:**

Document = “apple dog peanut”;

Wordin = {“apple”, “dog”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 0)

* Checks that program responds correctly to 0 rules

Document = “apple dog peanut”;

Wordin = {“apple”, “dog”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, -15)

* Checks that program responds correctly to negative rules

Document = “apple dog peanut”;

Wordin = {“asdlajdlksadjlsdlkjadljadlkjaslkdadsdas”, “dog”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 0)

* Match rule that’s longer than 20 characters // code wouldn’t handle correctly

Document = “apple dog peanut”;

Wordin = {“asdsads”, “asdasd”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* No matched one word rules

Document = “apple dog peanut”;

Wordin = {“apple”, “asdasd”} ;

Wordout = {“dog“, “apple”};

determineScore(document, wordin, wordout, 2)

* No matched two word rules

Document = “apple\*\*\*\*\*\*\*dog & peanut”;

Wordin = {“appledog”, “asdasd”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Checks that non alphabetical characters are ignored in document

Document = “apple239483dog & peanut”;

Wordin = {“appledog”, “asdasd”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Checks that numbers are ignored in document

Document = “APPLES and peanut butter”;

Wordin = {“apples”, “asdasd”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Checks that case is not regarded when looking for matches

Document = “apples and peanut butter”;

Wordin = {“apples”, “and”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Match of one word rule at very beginning of document

Document = “apples and peanut butter”;

Wordin = {“butter”, “asdasd”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Match of one word rule at end of document

Document = “another dog peanut”;

Wordin = {“an”, “asdasd”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Checking that it’s not matching a rule when the rule is contained within a larger word in the document

Document = “another dog peanut”;

Wordin = {“another”, “dog”} ;

Wordout = {“laugh“, “apple”};

determineScore(document, wordin, wordout, 2)

* All two word rules matched

Document = “another dog peanut”;

Wordin = {“another”, “dog”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* All one word rules matched

Document = “”;

Wordin = {“another”, “dog”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Checking that an empty document will have no matches

Document = “asdlasjdkdklasjdasjkdlladsdkdajsdakdskjdda”;

Wordin = {“another”, “dog”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Document with longer than 20 character words

Document = “another dog peanut”;

Wordin = {“an\*\*\*\*\*other”, “dog”} ;

Wordout = {““, “”};

determineScore(document, wordin, wordout, 2)

* Match rules that aren’t in clean form// code wouldn’t handle correctly