Testing Report for HW2

1. **samePrefix**

This program in the case of test 0, test 1, test many will test when the string has an index of 1, 2 and test the many test cases that the guidelines provide. This is because the string must contain something for the method samePrefix to compare and “” is an empty string. We expect the result to be true or false depending on the number and the string.

> HW2.samePrefix(" "," ",1)

true

This is the first test where it is tested whether the strings contain the same empty space at index 1. It should test true and it does test true.

> HW2.samePrefix("gg","gg",2)

True

This is the simplest step up from the last test, testing whether two exact same strings are the same with the index being the length of the string. It should return true and does indeed return true.

> HW2.samePrefix("this is a test", "this is a trial", 11)

True

This is the many part of the testing. This tests two different strings and tests all the way up to the 11th character of the strings. As you can see, up to the 11th character, the strings are the same, so this input should return true in the program. It returns true, so the program checks out.

> HW2.samePrefix("this is a test", "this is a trial", 12)

False

This is the same as the above test, but they test the 12th character where the strings differ. In the first string, the 12th character is an e. In the second string, the 12th character is a r. So, this test result should be false. In this case, this turns false.

> HW2.samePrefix("this is a test", "This is a trial", 4)

False

Our program is case sensitive. In this case, when we test the first characters of the strings, they differ because one is a lower case t and one is an upper case t. The program should return false just by seeing that and it does indeed return false

> HW2.samePrefix("this is a test", "this is a test", 100)

False

This should return false to the number being higher than the length of the string. This does end up returning false.

1. **trimSpacesFromFront**

So, for this one, the first, last, middle test doesn’t really apply as we are just trimming only the initial white spaces from the string, so any spaces after the initial characters doesn’t really matter. We can actually test this in our program to show that this is the case. In this case, for this test, we can test the string with no spaces before, 1 space before, and a lot of spaces before the string. We expect it to remove all the initial spaces before the first non white space character.

> HW2.trimSpacesFromFront(" Good day!")

"Good day!"

This is the basic trimspaces where we want to return the string without any spaces before the initial character. This should return everything after the G, aka Good day!. We see that it does indeed return the expected result.

> HW2.trimSpacesFromFront("h Good day!")

"h Good day!"

We’re testing here that it doesn’t remove any spaces after the initial character. This should actually return the original string as there is no spaces before the first non white space character. We see that it does return the original string in all its glory.

> HW2.trimSpacesFromFront(" h")

"h"

This is our test of just the basic string and if it returns the h minus the space in the beginning. We see that it does return the h minus the space in the beginning, so the program all checks out.

> HW2.trimSpacesFromFront(" I like pie a lot ")

"I like pie a lot "

This tests whether the program removes the spaces after the string. This should remove the first space before the I, but keep the remaining spaces. We see that it does indeed return what we expected, as it removes the first space, but keeps the spaces afterward, especially the big amount of spaces after the characters end.

1. **countWords**

This program counts words by seeing if there is spaces surrounding a non whitespace character or a set of non whitspace characters. It also counts a word if there is an initial nonwhitespace character/characters and if there is an ending nonwhitespace character/characters. For the test 0, test 1, test many test, we will test a 0 word case, 1 word case, then the many other word cases.

> HW2.countWords(" ")

0

> HW2.countWords("")

0

The above should be 0 word cases due to being an a) empty string or b)having no non whitespace characters. We test this case by inputting an empty string and one with just spaces and see that it does return that it counts no words in the string.

> HW2.countWords(" i ")

1

> HW2.countWords("i ")

1

> HW2.countWords(" i")

1

These are all the 1 word cases that can be covered. The character, regardless whether it is placed at the end, beginning, or middle should return that there is 1 word in order for the programming to be working correctly, aka counting that there is a word when there are two spaces around the character, when the character is at the beginning, and when the character is at the end. It does return 1 no matter where the character is. Therefore, the program checks out.

> HW2.countWords("d i d")

3

> HW2.countWords("d i , , asfdoijweoinqcoiwncoqinwecoiqnwe, ")

5

> HW2.countWords("One fish, two fish, red fish , blue fish !")

10

These are all the multiple case wordcount tests. The three word count tests all the cases for there being a word: a character at the beginning, middle and end. The 5 word count test tests whether it recognizes that a space at the end is not a word and whether the punctuation is recognized as a word with the spaces surrounding it as well as throwing in a huge word for it to run through. The 10 word test is the example given and should return 10 based on the number of words. We see that they all return the right amount of words, so we are confident this program works.

1. **truncate**

This truncates the string according to user input. It keeps initial spaces and doesn’t truncate words. In order to test the method, we use the test the first, middle, and last. Really, the first middle and last test doesn’t work because as we modify the string, it must fulfills conditions that prevent us from truncating a single character or a last character. If we truncate in the middle, it will return the same word as if we truncated at the first character or the very last character. Thus, we’ll test the conditions First would be truncating a word from a phrase. Middle would be truncating the middle of the word to test if that work. These should both return the same phrase. Last would be truncating at the last character. This should also return the same character.

> HW2.truncate("Is this homework fun?", 4)

"Is this"

> HW2.truncate("Is this homework fun?", 6)

"Is this"

> HW2.truncate("Is this homework fun?", 7)

"Is this"

This tests the beginning, middle, and end theory where we truncate the string at the beginning, middle, and end of a word. They should all return the same string which is “Is this”. We see that they indeed do all return the same string, checking out the condition that it will not truncate the word.

> HW2.truncate(" Hi ", 1)

" Hi"

> HW2.truncate(" Hi ", 5)

" Hi "

This tests whether it really keeps the spaces and starts counting the desired length after the spaces in order to truncate the string. The first one should output the spaces in front and then the hi while the second one should return the same output but with extra spaces due to the longer desired length. As we see, the return values are as expected.

> HW2.truncate(" excessively long", 9)

" excessively"

This is essentially just testing the same things as before but with a bigger word. It tests whether the word will print out with the spaces in front and whether it will print out the word not truncated despite it being longer than the desired length and not print the other word. The return value should be the string with spaces in front and just the one word. We see that it does return excessively with the necessary spaces before the word, so this checks out.

> HW2.truncate("long", 9)

"long"

> HW2.truncate(" long ", 9)

" long "

This tests what it prints if the desired length is longer than the string. It should just return the string regardless of if it has spaces or anything. As we see, the return values are what we expected, which is just the original string.

1. **padString**

This method adds spaces under a bunch of conditions. The 0, 1, and many test for these would be to test adding no spaces, adding 1 space, and then adding a different combination of spaces to test whether the spaces will be added the correct way.

> HW2.padString("i!", 299)

"i!"

This is testing the case that no extra spaces should be added if there are fewer than 2 words in the string. It should return the original string. We see that it does indeed return the original string.

> HW2.padString("This really is fun!", 19)

"This really is fun!"

This adds 0 spaces to the string and thus, should return the original string without any modifications. We see that it does indeed return the original string and so fulfills the method. We also see that it doesn’t append spaces to the beginning or the end, so that works too.

> HW2.padString("This really is fun!", 20)

"This really is fun!"

This adds 1 space to the string and thus, should return the original string but with one space appended to the last space before the last word. We see that it does indeed return that exact string as Microsoft word is raising an error to me.

> HW2.padString("This really is fun!", 23)

"This really is fun!"

> HW2.padString("This really is fun!", 24)

"This really is fun!"

These both add spaces; the difference is that 24 adds an uneven amount of spaces and this tests if the uneven spaces are added to the end. The first string should return the string with 2 spaces in between all strings and the second string should return the string with 2 spaces in between the 1st and 2nd word and then 3 spaces in between for the rest. We see that it does return the strings as specified.

> HW2.padString("This really is fun! omg i love tacos and burritos please help me", 24)

"This really is fun! omg i love tacos and burritos please help me"

This tests if the dlength is less than the amount of characters scenario. This should return the original string without any modifications. We see that it does.

> HW2.padString("This really is fun! omg i love tacos and burritos please help me", 278)

"This really is fun! omg i love tacos and burritos please help me"

This is just a silly example where we add 278 – string1.length() spaces. This should add whatever amount of spaces in between the string’s words as that possibility does. I can’t double check if this is right, but I feel confident that it checks out.

1. **prettyPrint**

This one is supposed to basically indent according to the user input and make it a nice little rectangle. The testing first, middle and last means testing according to the user input being 1, a middle value, and a huge value. The method I made sorta works and sorta doesn’t. It prints out the strings, but fails to line up the left and right margins with non characters.

> HW2.prettyPrint("This homework is demonstrating how word processors make the text in a document line up so neatly.", 90)

This homework is demonstrating how word processors make the text in a document line up so neatly.This one prints out one line, but the word doesn’t allow us to see that. This one is supposed to print out just the original string as the character limit is fairly large. It does return the original string.

> HW2.prettyPrint("This homework is demonstrating how word processors make the text in a document line up so neatly.", 23)

This homework is demonstrating

how word processors make

the text in a document line

up so neatly.

So, this is where it started to get tricky. The output should have been lined up with the right and left margins being non whitespaces and the margins all being the same. They also should have padded the last line. However, my output didn’t do that, so something went wrong with the code there. Testing in the future would involve actually debugging the code so that it would all line up nicely.

> HW2.prettyPrint("This homework is demonstrating how word processors make the text in a document line up so neatly.", 1)

This

homework

is

demonstrating

how

word

processors

make

the

text

in

a

document

line

up

so

neatly.This one actually works. The user input of 1 renders it so that only one word can be printed per line and it is assumed that there are spaces to make it line up to the margins. The output returns the predicted string, so this works.

1. **isAnagram**

Ok, I know whoever reading this doesn’t want to hear how I did it but I’m super proud of this. I made everything lowercase and removed anything that is non-letters and non-digits using for loops. This was converted to two strings. Then, I made two arrays with 36 0’s, one for each letter and digit, and one for each string. Then, I ran through each string and each time the letter/digit was found, it added a 1 to the value of the respective index. Then, I compared each element of the two arrays using a for loop and voila, I found if they were anagrams or not. THE EXTRA CREDIT SAID THIS:

“The method returns true if the two strings have exactly the same letters (case insensitive) and digits, possibly re-arranged. Any non-letter or digit characters are ignored.” This contradicts itself so I included digits.

Ok, for the test 0, 1 and many, we first test the empty string and then, we test the one word string. Then, we test the huge anagrams for the many. We expect them to return true if they are anagrams and false if they are not anagrams.

> HW2.isAnagram(" ", " ")

true

> HW2.isAnagram("", "")

True

These are the empty arrays and they should return true because they are technically anagrams. They return true so they check out.

> HW2.isAnagram("dog", "dog")

true

> HW2.isAnagram("dog", "god")

True

The most obvious anagram is when the words are the exact same. Then, we can actually rearrange the letters to see if it checks whether it is an anagram or not. They both should return true. They both return true.

> HW2.isAnagram("Case Western Reserve University", "We're an ever icy, nerve, stress suit!!")

true

> HW2.isAnagram("Case Western Reserve Univer56sity", "We're an ever icy, nerve, stre56ss suit!!")

True

This tests if the anagram checker ignores the punctuation but still keeps note of digits. The first and second should both return true as they are anagrams. They both return true despite the punctuation in the first one and the digits in the same one.

> HW2.isAnagram("Case Western Reserve Univer56sity", "We're an ever icy, nerve, st2re56ss suit!!")

False

This is clearly not an anagram. It should return false due to the extra 2 in the second string. It returns false, so it checks out.