CS 362 Midterm

Joshua Hanna

Q1:

Deliverables:

1. Describe in detail how you would test **paraphrase**. Try to cover as many as different scenarios, including valid (good/positive), invalid (bad/negative), and boundary cases. (**40 points**)

Positive Test Cases:

1. in.txt contains only valid ASCII letters and isn't null, therefore paraphrase can find it in out.txt.
2. out.txt contains a valid string and isn't null. This will make sure paraphrase can find it in when given the string from in.txt
3. When the in.txt string is found in out.txt, then the program will display the lines of where they occurred.
4. When the string from in.txt doesn't match anything from out.txt, then nothing should display on the screen.
5. When there is either nothing in in.txt or out.txt it, then nothing should be outputted onto the screen.
6. When in.txt has uppercase letters and out.txt has lowercase letters in a matched word, then nothing should display on the screen.

Negative Test Cases:

7. Screen doesn't display line when string is found in out.txt.
8. Program displays whole text of out.txt instead of just the lines of where the in.txt string is.
9. There aren't exactly 3 command line arguments; therefore, the program won't know what to do.
10. The file in.txt has a larger number of characters in it then out.txt.

Boundary Cases:

11. in.txt has as the string with space at the end of it.
12. If in.txt has multiple words in the file, then it should be able to find the same set of words in out.txt if they match.

2. Describe in detail how you would check the correctness of the output for each type of test case. (**10 points**)
    1. First, make sure in.txt have valid numbers and letters in their file.
    2. Check to make sure out.txt has valid letters in the file and isn't null. This should trigger a warning if either of these are null.
    3. I would first make sure that the program paraphrase can find any matched word so I would put the same word in both the programs and if I get them to display the word then I know it works. I would then check to see if you can match multiple lines in out.txt and then have the correct lines display on the monitor.
    4. I would put a word in in.txt that doesn't match anything in the out.txt file, to ensure that nothing is display on the monitor when the program is ran.
    5. If there is nothing in either in.txt or out.txt then nothing should be display on the monitor.
    6. I would run the same test as I did in number 3 where I would make put the same word with uppercase and lowercase in in.txt and out.txt. Then I would make sure it could pick up the word and have it displayed on the monitor. After this test, I would make sure try this with multiple lines in the text to make sure that the output is correct.
    7. I would run the same test I did with 3 & 6, If the screen doesn't output the lines at all, the correct lines, too many lines, or not all the lines of the matched word, then I know that this program has bugs inside it.
    8. I would run a test where I had the right match word on 5 out the 7 lines in out.txt. if only 5 out of the 7 lines are display on the screen then I know that this test will pass.
    9. I would try the program with only 2 or 4 command line arguments. If either of them gives me errors, then I'll know that this works correctly.
    10. If I have a string in in.txt than is in the text of out.txt, then nothing should be printed onto the monitor.
    11. I would put the same word in in.txt and out.txt expect I would add a space to the in.txt string and check to see if there is no output. If there is no output then I know that this test will pass.
    12. I will have the same sentence inside both in.txt and out.txt, if the screen can display the string then I know it will pass. Then I will place the same sentence in the out.txt will other sentences to make sure that the program will give me the correct lines display also.
3. Describe in detail how you would check if each of your tests are thorough. (**10 points**)
    - First, I would priority on the making sure the program is running correctly by testing each positive test case and negative test cases. If there are any bugs. You have to check each case with care making sure that your program can catch small things such as

spaces, multiple words, differentiate between uppercase/lowercase letters, and displays the correct lines when the string from in.txt matches a word (or sentence) from out.txt. After I test for the positive and negative test cases then I move onto the boundary cases and make sure that the program can handle these limits too. After all this testing, I would then find program to be thoroughly tested.

Q2:

1-Using the provided function calls above, write **three** different **test cases** that you feel will thoroughly test and verify that **removeAll** properly deletes all copies of the number n from the container c. List the assertions you would make to verify your expected results in your unit test.

Assertions:

1. number n isn't null
2. pointer container c isn't null
3. container is a class that has these functions available to it.

```
//First I would create a pointer container
Container *x = container.newContainer();

//then I would add a bunch of random numbers to the container like 12, 4, 15, & 1
x.add(&x, 12);
x.add(&x, 4);
x.add(&x, 15);
x.add(&x, 1);
x.add(&x, 1);
x.add(&x, 4);
x.add(&x, 2);
```

Test case 1:

```
Print x.size(&x); //this should print 7 because I haven't removed anything yet.
x.removeAll(1, &x); //this should remove all instance of 1
print x.size(&x); // this should print 4 because we got rid of all the instances of 1.
```

Test case 2:

```
Print x.size(&x); //this should print 7 because I haven't removed anything yet.
x.remove(1, &x); //this should remove all instance of 1
print x.size(&x); // this should print 6 because we got rid of one instance of 1.
x.removeAll(1,&x); //removes all 1
x.removeAll(4, &x) //removes 4
x.removeAll(15, &x) // removes 15
x.removeAll(12, &x) // removes 12
```

x.removeAll(2, &x) // removes 2
print x.size(&x); //should be zero.

Test case 3:

Print x.size(&x); //this should print 7 because I haven't removed anything yet.
x.remove(1, &x); //this should remove all instance of 1
print x.size(&x); // this should print 6 because we got rid of one instance of 1.
x.removeAll(1,&x); //removes all 1
print x.size(&x); // this should print 4 because we got rid of all instance of 1.
x.removeAll(4, &x) //removes 4
print x.size(&x); // this should print 3
x.removeAll(15, &x) // removes 15
print x.size(&x); // this should print 2
x.removeAll(12, &x) // removes 12
print x.size(&x); // this should print 1
x.removeAll(2, &x) // removes 2
print x.size(&x); //should be zero.