10 points.
Soft Due Date: December 17, 2021 at 11:59 pm
Hard Due Date: December 23, 2021 at 5:00 pm

Section 1 Problem Statement

In this final program of the semester, we'll use regular expressions to help decode a message.
Please take note of the due dates listed above. Unlike previous assignments, assignment 7 has "soft" and "hard" due dates. If you turn your assignment in by the soft due date, you will be given 2 points of credit towards any points lost during grading *of this assignment*. There is no penalty for turning in the assignment after the "soft" due date, however we can not accept submissions after the "hard" due date.
Please also note that the hard due date is at **5:00 pm** on the 23rd. Plan to submit your assignment sooner than later, and **do not** wait to do the assignment until the last day. While this program is simpler than the others this semester, we need your submission by the deadline to ensure it is graded included in your final course grade.

Section 2 Description

Your program will read in two files: a message file named `message.txt`, and a "word swap" file named `swaps.txt`. The message file will contain a large amount of text, encoding a simple message. However, there are multiple layers in the decoding process, requiring different uses of regular expressions to extract the original text.

The first line of `swaps.txt` will contain an integer, indicating the number of word pairs. Then, each line after that will contain two words, separated by a space. The first word is a "search" word, the second a "replace." Your program should read the "word swap" file first. Then, for each line of the message file, use regular expressions and `regex_replace(...)` to perform all swaps. This is the first layer of the code.

The results from the first layer will be a bunch of lines of text, in an XML/HTML-style format. That is, each line will start with an "opening tag," end with a "closing tag," and contain some plain text in the middle. A "tag" is made up of a single word, surrounded by < and > symbols (note that true XML/HTML would include a '/' in the closing tag, we will not).
An example line: `<apple>bananas are good too<apple>`
For the second layer of the decoding process, write a regular expression to check if each line is "valid." A line is valid if its opening tag matches its closing tag. Discard any invalid lines (those with non-matching tags).

Hint: use capture groups (parentheses around sections of the expression) to get the word inside each tag. Remember, you can use a `smatch` object to get the results from a `regex_match`, including each "group" match. You can then compare the text from the tags, and if they match, store the inner text for the final step.

After decoding the second layer, you should have a series of "inner" strings from the previous step. For each inner string, the total number of characters (including spaces and punctuation) in the string represents a letter: 1 character for 'a', 2 for 'b', 3 for 'c', etc. If a string has 27 characters, treat that as a ' ', the space character. Convert the number from each line to the appropriate character, in order, and it will spell out the final message.

Section 3 Output

In principle, the output from this program is just the final message. However, in order to most easily grade for partial credit, we'll have you output a few segments of text:

1. After decoding the "first layer," your program should output the lines of text with their swaps completed.
2. After completing the "second layer," your program should output the "inner" text from each valid line in that step (do not output anything from the invalid lines).

3. Finally, your program should output the decoded message.

You should separate each of these output segments with a single line containing three asterisks (`***`).

Section 4 Example

Below are examples for a message.txt and swaps.txt, with the decoded message at bottom:

message.txt:

```
<tag>I am Groot<tag>
<whale>This man is dead, Jim.<dolphin>
<telephone>Happy<xylophone>
<jack>Never will <jack>
<html>give you up.<xml>
<pal>Never will <lap>
<dog>let you down<days>
<span>Call him Ishmael<span>
```

swaps.txt:

```
6
xml html
will gonna
jack diane
him me
Groot Bob
xylophone telephone
```

message:
hello

Section 5 Deliverables

You should submit a single .zip file, named `[wiscID]_asg7.zip`, where [wiscID] is replaced by your Wisc ID (aka Net ID, or the user name you use to log into Canvas). Inside the zip file, there should be a single file called `main.cpp`.

Section 6 Rubric

Item	Points
Program builds and runs without compile errors	2
Program uses regular expressions to perform word swaps and/or validity checking, instead of custom parsing code	1
Program correctly replaces all "swap word" pairs	2
Program correctly finds all "valid" lines with matched tags	1.5
Program discards all "invalid" lines with non-matching tags	1.5
Program correctly converts characters per line to decoded characters to find the message	2
Total	**10**