Assignment #1
ECE 422C (Prof. Edison Thomaz) - The University of Texas at Austin – Fall 2023

Assignment Due Date: Sept 7th 2023

Problem 1 (30 points) - Consider following input:

```
73167176531330624919225119674426574742355349194934
96983520312774506326239578318016984801869478851843
85861560789112949495459501737958331952853208805511
12540698747158523863050715693290963295227443043557
66896648950445244523161731856403098711121722383113
62229893423380308135336276614282806444486645238749
30358907296290491560440772390713810515859307960866
70172427121883998797908792274921901699720888093776
65727333001053367881220235421809751254540594752243
52584907711670556013604839586446706324415722155397
53697817977846174064955149290862569321978468622482
83972241375657056057490261407972968652414535100474
82166370484403199890008895243450658541227588666881
16427171479924442928230863465674813919123162824586
17866458359124566529476545682848912883142607690042
24219022671055626321111109370544217506941658960408
07198403850962455444362981230987879927244284909188
84580156166097919133875499200524063689912560717606
05886116467109405077541002256983155200055935729725
71636269561882670428252483600823257530420752963450
```

The 4 adjacent digits in the 1000-digit number that have the greatest product are 9 × 9 × 8 × 9 = 5832.

Write Java code to find the n adjacent digits in the 1000-digit number that have the greatest product, where *n* is between 2 and 20. Your code should take as input the value of *n* (e.g., 4) and output the greatest product using **at most** *n* digits (e.g., 5832 or the sequence "0100" evaluating to 1). You can assume no overflow issues will happen.

Inputs
n (int): the number of **maximum** adjacent digits
s (String): A 1000-digit number to search through, (not a matrix)
See above for example input. n will be on its own line, followed by s on the next line. It will be entered via standard input.

Constraints
s.length() = 1000

$$2 \leq n \leq 20$$

Output
Print the largest product formed by **at most** *n* adjacent numbers on its own line to the console.

Problem 2 (35 points) - Write a Java program that takes as input a paragraph as a String and identifies if any of the words in the paragraph is a $1.00 word. The value of each word is calculated by adding up the value of each

one of its characters. Each letter in the alphabet is worth its position in pennies i.e. a = 1, b=2… y=25, z=26. Capital and lowercase letters have the same values. All non-alpha characters have a value of $0.00. Words will be delimited using spaces.

Example Input:
The wicked wizard`s wily wraith garnishes his master`s pasta with garlic.

Example Output:
wizard`s
garnishes

Inputs
s (String): the input string which contains a paragraph in English. It will be entered via standard input.

Constraints

$$0 \leq s.length \leq 10^6$$

Output
Print any $1.00 words to the console as they are in input to the console with each word on its own line. Note that you have to print out the word exactly as it appears in the input sentence.

Problem 3 (30 points) - Write a Java program that takes as input a sentence as a String, parses each word in the sentence, and identifies what part of speech each word is. For this assignment, you will need to do some research to learn more about part-of-speech tagging. Additionally, you will have to choose and download a Java-based Part-of-speech tagger and incorporate into your own program. For example, you could try to use the Stanford Log-linear Part-Of-Speech Tagger or do some research and find another suitable library. For this problem, create and submit an executable JAR file that includes the tagger you used in addition to the source code.

Example Input:
I was slowly walking to the park with my over enthusiastic dog when he bit me, and I shouted, Ouch!

Example Output:
I_PRP was_VBD slowly_RB walking_VBG to_TO the_DT park_NN with_IN my_PRP$ over_IN enthusiastic_JJ dog_NN when_WRB he_PRP bit_VBD me_PRP ,_, and_CC I_PRP shouted_VBD ,_, Ouch_NNP !_.

Inputs
s (String): the input string which contains a paragraph in English. It will be entered via standard input. See above for example input.

Output: A printed and tagged line to the console. Your output may not look exactly like this depending on what library and models you use in your speech tagger. See above for example output.

Problem 4 (5 points) – What do you expect (and like) to learn in this class? If we have time at the end of the semester to go through additional topics related to software engineering, Java or object-oriented programming, what would you like us cover? Please write your answers in a text file and include it in your zip file.

Some clarifications about problems 1-3:

1. For each of the problems 1-3, arguments should be read through the console as described in the problem statements. In your main function, you should then call another method which solves the problem. You can name the methods whatever you like.
2. You may assume that all inputs will be entered correctly.

Instructions for writing code and submitting your files:

Please follow these instructions exactly. Failure to do so may result in points being deducted.

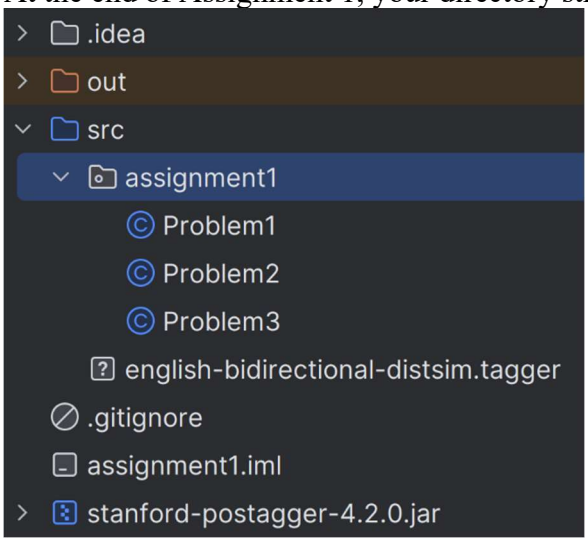
Formatting:

1. You must use a package name (`assignment1`). For each problem, your top-level java file (which contains your main ()) should be called `ProblemX.java`, where X is the problem number. Like most programming languages, Java is case sensitive. Do not change the case of the package name or top-level driver.

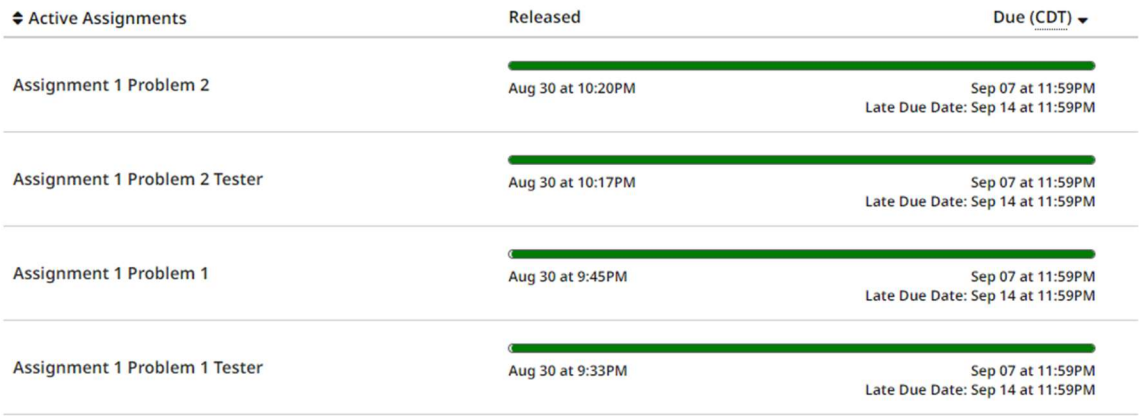
2. Ensure that your code contains a line with your name in the header comments. Insert the following line in the header of your code: /* Student Name: <your name>, Lab Section: <your lab section #> */. This will allow the TA to identify your submission.

Submission:

1. At the end of Assignment 1, your directory structure will look something like this.



2. Go to the src folder in your computer's file explorer, and zip the assignment 1 folder that's inside the src folder (highlighted above). **IMPORTANT:** Please make sure you do not have any tagger or jar files inside the assignment1 zipped folder.
3. Files that are required to be in there are the Problem1 and Problem2 java files. It is okay to have Problem3.java in there but not any tagger or jar files. *Gradescope will not allow you to submit the zip if it includes the tagger or jar files.*
4. For testing, upload your zip file to 'Assignment 1 Problem 1 Tester' to test problem 1 against sample test cases, and 'Assignment 1 Problem 2 Tester' to test problem 2 against sample test cases.
5. When you are ready to submit, submit the final zip to **BOTH** 'Assignment 1 Problem 1' **AND** 'Assignment 1 Problem 2'.



6. IT IS IMPORTANT THAT YOU SUBMIT THE ZIP TO BOTH GRADESCOPE ASSIGNMENTS.

7. You will need to create a separate zipped folder with Problem3.java, Problem3.jar, and Problem4.txt to canvas.

8. Your canvas zip file should be named Project1_eid.zip

Additional considerations:

Your program should be well tested to make sure it works correctly. Remember that your program will be compiled, run and graded on the official lab configuration by your TA. If it doesn't compile on that compiler, you should expect to get 5 points maximum for the program.

If you come to us for help, we will try to help you formulate **your** algorithmic solution; whatever that might be. Remember that there are many different designs that can solve this problem – not just one – and one that works is all that is required. If you need help setting up your dev and JDK environment properly, please see the TAs during lab or office hours for assistance. If you would like any clarifications about the wording of the assignment or if anything was not specified please post it on discussion group.

Please comment your program so that its logic could be explained to your TA (for example). We will not be grading this aspect on this assignment (but might do so on later assignments).

Grading assignments is done by us using a defined rubric, so that all assignments are graded fairly and uniformly. The rubric is derived from the stated and implied requirements, as well as, any requirements clarification that is published on the class discussion group page. We do not share this rubric in advance of the due date.