Practice Problems for PE06

For **PE04**, you'll complete one problem, which will ask you to **use the stream classes** to process text files. You don't have to worry about opening or closing the files. You will simply write a function that takes an input stream and an output stream as parameters, and use them for your processing.

Below you'll find a few of the sample problems. Note, these are NOT necessarily the problems that appear on the test, but they should give you a feel for the structure and type of problems.

1 The `boyGirl` Problem

Write a function named `boyGirl` that accepts an input stream and an output stream as arguments. The input stream contains a series of names followed by integers. The names alternate between boys' names and girls' names. Compute the absolute difference between the sum of the boys' integers and the sum of the girls' integers. The input could end with either a boy or girl; you may not assume that it contains an even number of names.

If the input file `tas.txt` contains the following text:

```
Steve 3 Sylvia 7 Craig 14 Lisa 13 Brian 4 Charlotte 9 Jordan 6
```

then your function could be called in the following way:

```
ifstream input("tas.txt");
boyGirl(input, cout);
```

and should produce the following output, since the boys' sum is 27 and the girls' sum is 29:

```
4 boys, 3 girls
```

```
Difference between boys' and girls' sums: 2
```

Here is some extra test data:

```
Steve 3 Sylvia 7 Craig 14 Lisa 13 Brian 4 Charlotte 9 Jordan 6
4 boys, 3 girls
Difference between boys' and girls' sums: 2

Steve 3 Sylvia 7 Craig 14 Lisa 13 Brian 4 Charlotte 9 Jordan 10
4 boys, 3 girls
Difference between boys' and girls' sums: 2

Frank 12 Alexandra 8 Xandros 6 Kyrie 9 Luther 5 Electra 13 Benandre 7
4 boys, 3 girls
Difference between boys' and girls' sums: 0

Jason 1
1 boys, 0 girls
Difference between boys' and girls' sums: 1
```

2 THE `stripHtmlTags` PROBLEM

Write a function `stripHtmlTags` that accepts an input stream and an output stream as arguments. The input stream contains an HTML web page as its parameter. Output the text with all HTML tags removed. A tag is any text between the characters < and >.

For example, consider the following text:

```
<html>
<head>
<title>My web page</title>
</head>
<body>
<p>There are many pictures of my cat here,
as well as my <b>very cool</b> blog page,
which contains <font color="red">awesome
stuff about my trip to Vegas.</p>

Here's my cat now:<img src="cat.jpg">
</body>
</html>
```

If the file contained these lines, your program should output the following text:

```
My web page


There are many pictures of my cat here,
as well as my very cool blog page,
which contains awesome
stuff about my trip to Vegas.

Here's my cat now:
```

You may assume that the file is a well-formed HTML document and that there are no < or > characters inside tags.

3 THE dupeCount PROBLEM

Write a function dupeCount that accepts an input stream and an output stream as arguments. The input stream contains a series of lines as shown below. Examine each line looking for consecutive occurrences of the same token on the same line and output each duplicated token along how many times it appears consecutively.

For example, if the input stream contains the following text:

```
hello how how are you you you you
I I I am Jack's Jack's smirking smirking smirking smirking smirking revenge
   bow  wow wow yippee yippee   yo yippee   yippee yay  yay yay
one fish two fish red fish blue fish
It's the Muppet Show, wakka wakka wakka
```

Your function would produce the following output:

```
how*2 you*4
I*3 Jack's*2 smirking*5
wow*2 yippee*2 yippee*2 yay*3

wakka*3
```

Only output repeated tokens; the ones that only appear once in a row are not shown. Place a single space between each reported duplicate token; respect the line breaks in the input. This is why a blank line appears in the expected output, corresponding to the fourth line of the input that did not contain any consecutively duplicated tokens. You may assume that each line of the input contains at least 1 token of.

4 THE `coinFlip` PROBLEM

Write a function `coinFlip` that accepts an input stream and an output stream as arguments. The input represents sets of coin flips that are either heads (H) or tails (T) in either upper or lower case, separated by at least one space. Consider each line to be a separate set of coin flips and output the number of heads and the percentage of heads in that line, rounded to the nearest tenth. If this percentage is more than 50%, you should out a "You win" message.

For example, consider this input:

```
H T H H T
T t   t T h  H
   h
```

For the input above, your function should produce the following output:

```
3 heads (60.0%)
You win!

2 heads (33.3%)

1 heads (100.0%)
You win!
```

The format of your output must exactly match that shown above. You may assume that the input stream contains at least 1 line of input, that each line contains at least one token, and that no tokens other than h, H, t, or T will be in the lines.

5 THE `tokenStats` PROBLEM

Write a function `tokenStats` that takes an input stream and an output stream as arguments. Report the number of lines in the file, the longest line, the number of tokens on each line, and the length of the longest token on each line. Assume at least one line of input and that each line has at least one token.

For example, if an input stream contains the following text:

```
"Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch."
```

Then the function should produce the following output:

```
Line 1 has 5 tokens (longest = 11)
Line 2 has 8 tokens (longest = 6)
Line 3 has 6 tokens (longest = 6)
Line 4 has 3 tokens (longest = 14)
Longest line: the jaws that bite, the claws that catch,
```

6 THE `percentPlus` PROBLEM

Write a function `percentPlus` that accepts an input stream and an output stream as arguments. The input stream represents student records; each student record takes up two lines of input. The first line has the student's name and the second line has a series of plus and minus characters. Below is a sample input:

```
Kane, Erica
--+-+
Chandler, Adam
++-+
Martin, Jake
+++++++
Dillon, Amanda
++-++-+-
```

The number of plus/minus characters will vary, but you may assume that at least one such character appears and that no other characters appear on the second line of each pair. For each student you should produce a line of output with the student's name followed by a colon followed by the percent of plus characters.

The above input should produce the following output:

```
Kane, Erica: 40.0% plus
Chandler, Adam: 75.0% plus
Martin, Jake: 100.0% plus
Dillon, Amanda: 62.5% plus
```

7 THE `leetSpeak` PROBLEM

Write a function `leetSpeak` that accepts an input stream and an output stream as arguments. Your function should convert the input file's text to "leet speak" (aka 1337 speak), an internet dialect where various letters are replaced by other letters/numbers. Output the leet version of the text to the given output stream. Preserve the original line breaks from the input. Also wrap each word of input in parentheses.

Perform the following replacements:

Original character	'Leet' character
o	0
l (lowercase L)	1
e	3
a	4
t	7
s (at the end of a word only)	Z

For example, if the input file `lincoln.txt` contains the following text:

```
four score and
seven years ago our

fathers brought forth on this continent
a new nation
```

And your function is called in the following way:

```
ifstream input("lincoln.txt");
ofstream output("leet.txt");
leetSpeak(input, output);
```

Then after the call, the output file `leet.txt` should contain the following text:

```
(f0ur) (sc0r3) (4nd)
(s3v3n) (y34rZ) (4g0) (0ur)

(f47h3rZ) (br0ugh7) (f0r7h) (0n) (7hiZ) (c0n7in3n7)
(4) (n3w) (n47i0n)
```

You may assume that each token from the input file is separated by exactly one space.

8 THE evenStats PROBLEM

Write a function `evenStats` that accepts an input stream and an output stream as arguments. The input stream represents a series of integers; you may assume that there is at least one integer in the input. Report the total number of numbers, the sum of the numbers, the count of even numbers and the percent of even numbers. For example, if an input stream contains the following text:

```
5 7 2 8 9 10 12 98 7 14 20 22
```

Then the function should produce the following output:

```
12 numbers, sum = 214
8 evens (66.67%)
```

9 THE negativeSum PROBLEM

Write a function `negativeSum` that accepts an input stream and an output stream as arguments, and returns a true/false value. The input stream contains a series of integers. Your function should determine whether the running sum, starting from the first number is ever negative. For example, if the input consists of the following:

```
38 4 19 -27 -15 -3 4 19 38
```

The first number is 38 (not negative), the sum of the next (38 + 4), the sum of the next (38 + 4 + 19), and so on up to the sum of all of the numbers. None of these sums is negative, so the function would produce the following output and return false.

```
no negative sum
```

If the file instead contains the following numbers, the function finds that a negative sum of -8 is reached after adding 6 numbers together (14 + 7 + -10 + 9 + -18 + -10):

```
14 7 -10 9 -18 -10 17 42 98
```

It should output the following, indicating that a negative sum can be reached, and return true.

```
-8 after 6 steps
```

10 THE countCoins PROBLEM

Write a function `countCoins` that accepts an input stream and an output stream as arguments. The input stream represents a person's money grouped into stacks of coins. Add up the cash values of all the coins and print the total money at the end. Input is arranged as pairs of tokens, where each pair begins with an integer and is followed by the type of coin, which will be either "pennies" (1 cent each), "nickels" (5 cents each), "dimes" (10 cents each), or "quarters" (25 cents each), case-insensitively. A given coin might appear more than once on the same line.

For example, if the input file contains the following text:

```
3 pennies 2 quarters 1 pennies 3 nickels 4 dimes
```

3 pennies are worth 3 cents, and 2 quarters are worth 50 cents, and 1 penny is worth 1 cent, and 3 nickels are worth 15 cents, and 4 dimes are worth 40 cents. The total of these is 1 dollar and 9 cents, therefore your function would produce the following output if passed this input data. Notice that the function should show exactly two digits after the decimal point, so it says 09 for 9 cents:

```
Total money: $1.09
```

Here is a second example. Suppose the input file contains the following text. Notice the capitalization and spacing:

```
12   QUARTERS      1   Pennies      33
PeNnIeS

  10    niCKELs
```

Then your function would produce the following output:

```
Total money: $3.84
```

You may assume that the file contains at least 1 pair of tokens. You may also assume that the input is valid; that the input has an even number of tokens, that every other token is an integer, and that the others are valid coin types.

11 The `printStrings` Problem

Write a function `printStrings` that takes an input stream and an output stream as arguments. The input stream contains a sequence of integer/string pairs and that prints to the output stream one line of output for each pair with the given `string` repeated the given number of times. For example if the input contains the following data:

```
6 fun. 3 hello  10   <> 2      25   4 wow!
```

your function should produce the following output:

```
fun.fun.fun.fun.fun.fun.
hellohellohello
<><><><><><><><><><>
2525
wow!wow!wow!wow!
```

Notice that there is one line of output for each integer/string pair. The first line has 6 occurrences of "`fun.`", the second line has 3 occurrences of "`hello`", the third line has 10 occurrences of "`<>`", the fourth line has 2 occurrences of "`25`" the fifth line has 4 occurrences of

"wow!". Notice that there are no extra spaces included in the output. You are to exactly reproduce the format of this sample output. You may assume that the input values always come in pairs with an integer followed by a `String` (which itself could be numeric, such as "25" above). If the input stream is empty (no integer/string pairs), your function should produce no output.

12 THE `triathalon` PROBLEM

Write a function `triathlon` that accepts an input stream and an output stream as arguments. The input stream represents triathlon race results for athletes. Add up the swimming, biking, and running times for each athlete and report the total time for each athlete and their time difference from the winner's. Each athlete's data is represented by four tokens in the following order: athlete's first name, swimming time, biking time, and running time (all times are given in minutes). The data for an athlete may or may not span multiple lines, but you are guaranteed the athletes will come in the order they finished the race (the winner's data is first, the second place triathlete's data comes next, etc.).

Here is an example input. Notice the spacing and that the order the athletes appear are in the same order that they finished the race:

```
Meghan 12 40 23 Bryan 16
42 20 Lori 14 41 29 Jessica 18

    37 30 Toni 19   43
29 Tamara 17 42 34
```

For this input, you function should produce the output below. For example, Meghan swam for 12 minutes, biked for 40 minutes, and ran for 23 minutes so it took for Meghan 75 minutes to finish the race. Notice that for the athletes that did not win the race, in addition to reporting their total race time, the difference in their time from the winner's is also reported. For example, Bryan swam for 16 minutes, biked for 42 minutes, and ran for 20 minutes so it took Bryan 78 minutes to finish the race, which is 3 minutes longer than Meghan's winning time of 75 minutes.

```
Meghan: 75 min
Bryan: 78 min (+3 min)
Lori: 84 min (+9 min)
Jessica: 85 min (+10 min)
Toni: 91 min (+16 min)
Tamara: 93 min (+18 min)
```

You may assume that the input stream contains data for at least one athlete. You may also assume that the input is valid; that the input has four tokens per athlete, the first token for an athlete is a string and the following three are integers.

13 THE mostUnique PROBLEM

Write a function `mostUnique` that accepts an input stream and an output stream as arguments. The input stream contains integer quiz scores separated by spaces. Each class period is on its own line and contains a different number of students. Your function should return the highest number of unique scores given in a single class period. The function should also output information about the unique scores given in each period, in a format described below.

On a given line, all occurrences of any particular number value will occur consecutively in the input. For example, if there is a sequence of occurrences of the number 8, there will not be any other occurrences of the number 8 on that same line outside of that sequence.

Given an input stream that contains the following text:

```
10 10 10  9 9   8    3
3 3 8 10 9 7   7 6 6
4  1 9  9 10  7 7
10  10   10 10
```

The function should return `6` and generate the following output:

```
Period 1: 4 unique scores
Period 2: 6 unique scores
Period 3: 5 unique scores
Period 4: 1 unique scores
```

On the first line, there are 4 unique scores: 10, 9, 8 and 3. The second line contains 6 unique scores: 3, 8, 10, 9, 7 and 6. The third line contains 5 unique scores: 4, 1, 9, 10 and 7. The fourth line only has one unique score: 10. The value returned is 6 because it is the highest number of unique scores given in a class period.

Assume that the input contains at least one line of data and that each line contains at least one score.

14 THE wordStats PROBLEM

Write a function `wordStats` that accepts an input stream and an output stream as arguments. The input stream contains a sequence of words. Report the total number of words (as an integer) and the average word length (as an un-rounded real number). You may assume that the input stream isn't empty. For example, suppose the input stream contains the following:

```
To be or not to be, that is the question.
```

We will use whitespace to separate words. That means that some words include punctuation, as in `"be,"`. (This is the same definition that the input stream uses for tokens.) For the input above, your function should produce exactly the following output:

```
Total words    = 10
Average length = 3.2
```

15 THE `pizza` PROBLEM

Write a function `pizza` that accepts an input stream and an output stream as arguments. Imagine that a college dorm room contains several boxes of leftover pizza. A complete pizza has 8 slices. The pizza boxes left in the room each contain either an entire pizza, half a pizza (4 slices), or a single slice. Your function's task is to figure out the fewest boxes needed to store all leftover pizza, if all the partial pizzas were consolidated together as much as possible.

Read lines from input where each line represents the contents of the pizza boxes in one dorm room. These contents are written as `whole`, `"half"`, or `"slice"` in either upper or lower case, separated by at least one space. You should print to output the number of pizza boxes necessary to store all the slices of pizza out of the total. You must use a whole number of boxes. For example, if there are 10 total slices of pizza in a dorm room, 2 pizza boxes are needed: one for the first whole pizza, and one for the final 2 slices. Note that some lines might be blank, meaning that the dorm room contains no pizzas; output for such a case is shown below.

Consider the following input file representing 5 dorm rooms (note that the fourth is blank):

```
slice half slice whole whole half half half
whole HALF   WhoLE half WHOLE  WhOlE  Slice         half sLICe halF
WHOLE slice   WHOLE SLICE whole SLICE    whole slice WHOLE half

slice
```

For the input above, your function should produce the following output:

```
5 / 8 pizza boxes used
7 / 10 pizza boxes used
6 / 10 pizza boxes used
0 / 0 pizza boxes used
1 / 1 pizza boxes used
```

The number following the slash is the number of original boxes, the number preceding it are the minimum number of boxes required. For instance on line 1, the dorm room had 8 boxes, but by combining, we can store all the pizza in 5 boxes.

The format of your output must exactly match that shown above. You may assume that the input stream contains at least 1 line of input, and that no tokens other than whole/half/slice.

16 The countWords Problem

Write a function `countWords` that accepts an input stream and an output stream as arguments. The function outputs the total number of lines and words found in the input as well as the average number of words per line. For example, consider the following input:

```
You must show: your Student ID card

to 1) a TA or 2) the instructor
before

leaving the room.
```

For the purposes of this problem, we will use whitespace to separate words. That means that some words might include punctuation, as in "show:" and "1)". (This is the same definition that the input stream uses for tokens.) For the input above, your function should produce the following output:

```
Total lines = 6
Total words = 19
Average words per line = 3.167
```

The format of your output must exactly match that shown above, including rounding the words per line to 3 decimal places. Notice that some input lines can be blank. Do not worry about rounding the average words per line. You may assume that the input stream contains at least 1 line of input.

17 The halfCaps Problem

Write a function `halfCaps` that accept an input stream and an output stream as arguments. The input stream contains a sequence of words. Your function should output the same sequence of words with alternating casing (lowercase, uppercase, lowercase, uppercase, etc). The first word, third word, fifth word, and all other "odd" words should be in lowercase letters, whereas the second word, fourth word, sixth word, and all other "even" words should be in uppercase letters. For example, suppose the input contains the following words.

```
The QUick brown foX jumPED over the Sleepy student
```

For the purposes of this problem, we will use whitespace to separate words. You can assume that the sequence of words will not contain any numbers or punctuation and that each word will

be separated by one space. For the input above, your function should produce the following output:

```
the QUICK brown FOX jumped OVER the SLEEPY student
```

Your output should separate each word by a single space. The output may end with a space if you like. Note that the input stream may contain no words or may contain an even or odd number of words.

18 The `runningSum` Problem

Write a function `runningSum` that accepts an input stream and an output stream as parameters. The input stream represents an input file holding a sequence of real numbers. The function outputs the running sum of the numbers followed by the maximum running sum. In other words, the nth number that you report should be the sum of the first `n` numbers in the input stream and the maximum that you report should be the largest such value that you report. For example if the input stream contains the following data:

```
3.25 4.5 -8.25 7.25 3.5 4.25 -6.5 5.25
```

your function should produce the following output:

```
running sum = 3.25 7.75 -0.5 6.75 10.25 14.5 8.0 13.25
max sum = 14.5
```

The first number reported is the same as the first number in the input stream (3.25). The second number reported is the sum of the first two numbers in the input stream (3.25 + 4.5). The third number reported is the sum of the first three numbers in the input stream (3.25 + 4.5 + -8.25). And so on. The maximum of these values is 14.5, which is reported on the second line of output. You may assume that there is at least one number to read.

19 The `flipLines` Problem

Write a function named `flipLines` that accepts an input stream and an output stream as parameters. The input stream represents an input file. Your function writes to the output stream the same file's contents with successive pairs of lines reversed in order.

For example, if the input file contains the following text:

```
Twas brillig and the slithy toves
did gyre and gimble in the wabe.
All mimsey were the borogroves,
and the mome raths outgrabe.

"Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch."
```

The program should print the first pair of lines in reverse order, then the second pair in reverse order, then the third pair in reverse order, and so on. Therefore your function should produce the following output to the output stream:

```
did gyre and gimble in the wabe.
Twas brillig and the slithy toves
and the mome raths outgrabe.
All mimsey were the borogroves,
"Beware the Jabberwock, my son,

Beware the JubJub bird and shun
the jaws that bite, the claws that catch,
the frumious bandersnatch."
```

Notice that a line can be blank, as in the third pair. Also notice that an input file can have an odd number of lines, as in the one above, in which case the last line is printed in its original position. You may not make any assumptions about how many lines are in the input stream.

20 THE `matchIndex` PROBLEM

Write a function named `matchIndex` that accepts an input stream and an output stream as parameters. The input stream represents an input file. Your function should compare each neighboring pair of lines (the first and second lines, then the third and fourth lines, and so on) looking for places where the character at a given 0-based index from the two lines is the same. For example, in the strings `"hello"` and `"belt"`, the characters at indexes 1 (`'e'`) and 2 (`'l'`) match. Your code should be case-sensitive; for example, `"J"` does not match `"j"`.

For each pair of lines, your function should print output showing the character indexes that match, separated by spaces in the format shown below. If no characters match, print "none" instead as shown below.

For example, suppose the input file contains the following text. (Line numbers and character indexes are shown around the input and matching characters are shown in bold, but these markings do not appear in the actual file.)

```
     01234567890123456789012345678901234567890
 1   The quick brown fox
 2   Those achy down socks
 3   Wheels on the school bus go round
 4   The wipers go swish swish swish
 5   His name is Robert Paulson
 6   So long 'n thanks for all the fish
 7   Humpty Dumpty sat on a wall
 8   And then he also had a great fall
 9   booyakasha
10   Bruno Ali G Borat
```

When passed the above file, your function would produce the following output:

```
lines 1 and 2: 0 1 7 12 13 14 15 17
lines 3 and 4: 1 2 13 14 23
lines 5 and 6: none
lines 7 and 8: 4 14 20 21 22
lines 9 and 10: none
```

Notice that lines are not generally the same length. You may assume that the file contains an even number of lines.

21 THE `mostCommonNames` PROBLEM

Write a function named `mostCommonNames` that accepts an input stream and an output stream as parameters. The input stream represents an input file whose data is a sequence of lines, where each line contains a set of first names separated by spaces. Your function should print (to the output stream) the name that occurs the most frequently in each line of the file. The function should also return the total number of unique names that were seen in the file. You may assume that no name appears on more than one line of the file.

Each line should be considered separately from the others. On a given line, some names are repeated; all occurrences of a given name will appear consecutively in the file. If two or more names occur the same number of times, print the one that appears earlier in the file. If every single name on a given line is different, every name will have 1 occurrence, so you should just print the first name in the file.

For example, if the input file contains the following text:

```
Benson Eric   Eric  Marty Kim  Kim Kim   Jenny  Nancy Nancy  Nancy  Paul  Paul
Stuart Stuart Stuart Ethan Alyssa Alyssa Helene Jessica Jessica Jessica Jessica
Jared  Alisa Yuki   Catriona  Cody   Coral   Trent Kevin  Ben Stefanie Kenneth
```

On the first line, there is one occurrence of the name Benson, two occurrences of Eric, one occurrence of Marty, three occurrences of Kim, one of Jenny, three of Nancy, and two of Paul.

Kim and Nancy appear the most times (3), and Kim appears first in the file. So for that line, your function should print that the most common is Kim. The complete output would be the following. The function would also return 23, since there are 23 unique names in the entire file.

```
Most common: Kim
Most common: Jessica
Most common: Jared
```

You may assume that there is at least one line of data in the file and that each line will contain at least one name.