13th Annual NB (English)

High School Programming Competition

hosted by UNB Saint John

Friday, May 3, 2019

Competition Problems TEAM

Rules:

1. **For each problem, your program must read from the standard input and write to the standard output. Your programs cannot use files for input or output.**
2. **Your submission to each problem will consist of a single source-code file (i.e., you will create just one source-code file per problem).**
3. **Your source code will be recompiled by the judges prior to testing.**
4. **The output of your programs must correspond exactly with the examples provided, including spelling, spacing and capitalization. Note that in the examples provided, each line (including the final line) is terminated with an end-of-line character.**
5. **The sample inputs for each problem are available in the Contestant folder on the desktop.**

There are 8 questions. Good luck!

Problem 1: Gender bias

With the new advances in artificial intelligence, there are now some software can that help with the pre-selection of candidates for jobs or positions (i.e., identify the most promising candidates). These typically check the applications received and identify the ones that are closer to the ones from people hired in the past. The danger here is that if the past selection process was biased (against a particular gender), then the new system would probably continue having such bias. It is thus important to test that the automatic selection is not biased.

For this problem, you should write a program that will perform such test. We will assume here that an unbiased selection process is defined as one where every candidate has an equal chance of being selected. This would mean that if 40% of the candidates are women, then we would want to have about 40% of the selected candidates to be women as well (more or less).

Your program should test 5 selection cases. For each case, there are 2 lines of input (for a total of 10 lines – see the example input below). The first line of the 2 contains 2 positive integers: the number of applications from male candidates and from female candidates respectively (in this order). The second line (of the 2) also has 2 numbers (zero or positive integer): the number of selected candidates from each gender (male and then female). What you have to do with these numbers is to first calculate what should have been the number of female candidates selected if the process was perfectly bias free. For example, in the first case below, this number should have been 7 (as 20 x 21 / 60). Note: this expected number will always be an integer. Then you should compare it with the actual number of female candidates selected. If the difference is 2 or less, then print "no bias". If the difference is 5 or less (but larger than 2), print "some bias". If the difference is higher than 5, print "heavily biased". Also indicate the gender that is negatively affected.

For the example input below, here are the calculations:

Case 1 (lines 1 & 2): expected=7, actual= 8 ➔ no bias
Case 2 (lines 3 & 4): expected=10, actual=13 ➔some bias against men
Case 3 (lines 5 & 6): expected=10, actual=10 ➔ no bias
Case 4 (lines 7 & 8): expected=15, actual=5 ➔ heavily biased against women
Case 5 (lines 9 & 10): expected=4, actual=10 ➔ heavily biased against men

```
EXAMPLE INPUT:
60 21
20 8
45 45
10 13
10 20
5 10
30 45
10 5
30 40
3 10
```

```
EXAMPLE OUTPUT:
no bias
some bias against men
no bias
heavily biased against women
heavily biased against men
```

Problem 2: Biorhythms

According to the theory of biorhythms, aspects of people's life (physical, emotional, intellectual, etc.) are influenced by rhythmic biological cycles: like a sine wave starting at birth, with cycles of various lengths depending on the aspect: 23 days for "physical", 28 days for "emotional", and 33 days for "intellectual". The following graph shows a wave for the "physical" aspect (the vertical bars show the beginning and end of each cycle):

With this, it is suggested that a person's level of ability in each of these domains can be predicted from day to day. Our goal for this problem is to write a program that given the number of days since birth, indicates when the next high peak and low peak will be in that person's life, for each of the 3 aspects above.

This can be calculated by taking the remainder of the division by 23, 28, or 33 (depending on the aspect), and compare it with what such remainder should be at those peaks:

Physical:	high peak at remainder = 6, low peak at remainder = 17
Emotional:	high peak at remainder = 7, low peak at remainder = 21
Intellectual:	high peak at remainder = 8, low peak at remainder = 25

For the example below where the number of days since birth is 7300 (see the example input), we have the following calculations:

Physical: 7300 = 317 x 23 + 9 (i.e., remainder of division by 23 is 9)
- So the next high peak is in 20 days
  - ((23 – 9) days until the end of the cycle + 6 days in the new cycle)
- And the next low peak is in 8 days
  - ((17 – 9) days until that peak, which will occur in the current cycle)

Similar calculations can be done for the emotional cycle (remainder of 7300 divided by 28 is 20) and the intellectual cycle (remainder of 7300 divided by 33 is 7). The output of your program should be those numbers (high peak then low peak), one per line, for the 3 aspects above in that sequence (see example output – line 1 and 2 for physical, 3 and 4 for emotional, and 5 and 6 for intellectual). Note: the number printed can be zero if we are already on a peak.

<see next page for the example input and output>

EXAMPLE INPUT:
7300

EXAMPLE OUTPUT:
20
8
15
1
1
18

Problem 3: Targeted ads

A major company would like to place advertisements at some public places, but they would like to put them where there will be enough of their targeted audience to see it. In order to estimate such targeted audience, they decide to do a quick survey on the street (to people passing by) at each of the locations they have in mind. The idea is that if there are enough people from their target audience (perfect cases or close enough), they will put an ad there.

The target audience is people in their 30s, spending 6 to 10 hours of screen time per week (inclusively), and owning a car. That would be the ideal audience ("perfect" cases). However, they would be content with the following cases ("close enough" cases):

- Persons in their 30s, with a car but with screen time either between 3 and 5 hours, or between 11 and 15 hours (inclusively)
- Persons with a car and with screen time between 6 and 10 hours, but in their 20s or in their 40s

Your program should count the number of people at each location, that match perfectly the target audience or that are close enough, and then decide whether each location is good or not based on given thresholds on those counts.

The input starts with the 2 thresholds on the first line (2 positive integers). The first number is the minimum number of "perfect" cases we should have, and the second number is the minimum number of "perfect" cases + "close enough" cases we should have. Both thresholds should be met in order to consider this location a good one.

The second line of input indicates how many locations we have to handle. Then for each location, we start with the name of that location on one line, followed by the number of persons surveyed on the following line. Then for each person surveyed, we have the following 3 pieces of information (all on one line, separated by spaces): age, number of hours of screen time, and "yes" or "no" indicating if the person owns a car or not.

The output should be, for each location: (1) the words "Good:" or "Not good:"; (2) the name of that location; (3) the number of "perfect" cases; and (4) the sum of the 2 counts (number of "perfect" and number of "close enough"). Note that the 2 numbers in the output are the ones that we compare to the thresholds provided in the first line of input.

<see next page for the example input and output>

EXAMPLE INPUT:

```
3 5
2
mall
9
25 6 yes
31 8 yes
33 10 no
40 4 yes
36 7 yes
39 9 yes
41 6 yes
21 8 no
32 15 yes
uptown
6
35 6 yes
34 7 no
31 8 yes
19 9 yes
38 7 yes
32 13 no
```

EXAMPLE OUTPUT:

```
Good: mall 3 6
Not good: uptown 3 3
```

Problem 4: Green software engineering

Software quality is not only about returning the correct answer or running fast: it can also be about the amount of energy required by the computer or device to run that software. For example, our choice of how to store and access data in the program has an impact on the energy consumption. This is an important aspect to consider when developing software. For example, reducing the energy used by mobile devices leads to longer battery lives. The domain related to this problem is called green software engineering.

A first step in identifying where we can focus our attention for reducing energy consumption is to measure the energy required for each function in the software. Your program should help with this. In particular, assuming that you have the number of watts at every second while running the program (as input), we want to calculate how many joules (unit of energy consumption) were necessary overall. We also want such information for each sub-function ("function", "module", "method", etc.) used throughout the execution of the program.

The input starts with a positive integer N, indicating the number of seconds that the application was running. Then you have N lines (one per second), each having a real number indicating the number of watts at that time. An empty line separates this part from the second part, which is for indicating when each sub-function was running. This part starts with a positive integer indicating the number of sub-functions. Then there is one line of input for each sub-function: first there is an index indicating when this sub-function started running (with index 0 being the first second that the program was running), and then you have the name of that sub-function (no space in the name). It is assumed that the sub-functions were executed in sequence, from the index they began (inclusively) up to the index of the next sub-function (exclusive). The list of sub-functions is always sorted over time in the input, from earliest to latest. So for the example input below, the "Init" function ran from second 0 to 2 (inclusively), the "Main" function ran from second 3 to 7, and finally the "Closing" function ran from second 8 to 9 (i.e., until the end).

Your program should calculate, for each sub-function, the average watts over the corresponding period (e.g., for "Init", take the average of 0.7, 1.0 and 1.5). Then you should multiply this number by the number of seconds the sub-function took, in order to get the number of joules used. Round up this number of joules (i.e., take the ceiling of that number) and print it in the format shown in the example output below. Finally, add the number of joules used by each sub-function and print this. Also indicate which of the sub-functions used the highest number of joules.

<see next page for the example input and output>

EXAMPLE INPUT:
10
0.7
1.0
1.5
1.3
1.7
1.8
2.0
1.9
1.1
1.5

3
0 Init
3 Main
8 Closing

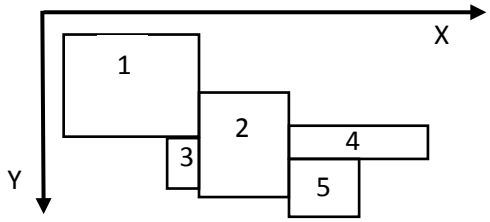
EXAMPLE OUTPUT:
Init used 4 joules
Main used 9 joules
Closing used 3 joules
total joules: 16
worst: Main

Problem 5: Fencing animal enclosures

A farmer asks for your help in planning the construction of some animal enclosures. For each enclosure, you have its coordinates (as upper-left and bottom-right corner). With such information, you should calculate the length of fence needed for this entire construction. Note: when 2 enclosures share a boundary (in part or in full), you should add the length of that boundary only once. Note: there will never be an overlap in enclosures (i.e., 2 enclosures sharing the same surface – in part or in full).

The input starts with a positive integer indicating the number of enclosures. Then for each enclosure, there is one line of input with 4 positive integers (separated by a space): the X and Y coordinates of the upper-left corner, and then the X and Y coordinates of the bottom-right corner.

Here is a drawing (approx.) representing the enclosures specified in the example input below:



The output should be the length of fence needed overall. For the example below, here is the calculation:

- Perimeter of enclosure 1 is 400 (2 x (120 – 10) + 2 x (100 – 10))
- Perimeter of enclosure 2 is 310 less the shared boundary with enclosure 1 of length 40
- Perimeter of enclosure 3 is 80 (the 2 sides not shared with enclosures 1 or 2)
- Perimeter of enclosure 4 is 250 (the 3 sides not shared with enclosure 2)
- Perimeter of enclosure 5 is 100 for the sides not shared with any other enclosures + 15 for the part of the side not shared with enclosure 2

For a total of: 400 + 310 – 40 + 80 + 250 + 100 + 15 = 1115

```
EXAMPLE INPUT:

5
10 10 120 100
120 60 190 145
80 100 120 140
190 90 300 120
190 120 250 160

EXAMPLE OUTPUT:
1115
```

Problem 6: Word count with stemming

When you write an essay, it is important to use a diverse vocabulary (i.e., not always the same words). Having an application that simply count each word is not enough though: sometimes words are very similar (e.g., "friend" and "friendly"), and that should count as the same word.

One simple approach that can be used is "stemming by suffix stripping". The idea is simply to remove the end of the word (suffix) if it matches one of the recognized suffixes. For our problem here, we will limit ourselves to the following suffixes: ly, ed, ing, ion, and s. Note: in the case of the s (which is basically about transforming the word from plural to singular), once it is removed, you should check again for other suffixes. For example, the word "televisions" would first be stemmed to "television" and then to "televis".

The input is one line of text to be analyzed. In the example input below, the text has been "wrapped" in order for you to be able to read it all on this page, but it is stored on a single line in the input file used for testing. This line of text does not contain more than 1000 characters. Spaces are used to separate words, and there is no space after the last word. Here is what you should do for each word:

- Make it all lower case. Note: only the first letter could be upper-case.
- Remove the trailing punctuation mark if any. Those can be: period (.), comma (,), semi-colon (;), colon (:), exclamation mark (!), or question mark (?).
- If the word has 3 letters of less, do not consider it (i.e., stop at this step). We do not want to be limited in the number of times we use words like "the", "an", "and", etc.!
- Remove any suffix if present

Keep a count of all words that went through all steps above, and print those words and counts as output (see below for the format). The words should be printed in the order that they first appear in the text. Note: there will not be more than 50 such words to be counted.

```
EXAMPLE INPUT:
My best Friend reminded me to be friendly with their friends and
their relatives, who are visiting during their vacations. Best
visit ever!

EXAMPLE OUTPUT:
best: 2
friend: 3
remind: 1
with: 1
their: 3
relative: 1
visit: 2
dur: 1
vacat: 1
ever: 1
```

Problem 7: Limiting options

When you buy a plane ticket online, you are offered various alternatives for the same origin & destination on the same day. Each alternative has a different price and a different overall travel duration (including the stops). For example, here are a few options to travel from Saint John to Orlando on a particular day (let's dream a bit…):

$519 for a flight taking 6 hours and 5 minutes
$448 for a flight taking 7 hours and 50 minutes
$453 for a flight taking 8 hours and 8 minutes
$448 for a flight taking 8 hours and 20 minutes
$844 for a flight taking 14 hours and 33 minutes
$409 for a flight taking 15 hours and 13 minutes

You can expect that someone will look at the alternatives and pick something that has a reasonable travel time for a reasonable price. Note: the concept of "reasonable" here may vary from one person to another. However, it is strange to offer the third flight above as option, because it will probably not be picked: there is another option (the second flight) that takes less time for a lower cost. We have the same situation with the fourth and the fifth option above. You should write a program that will take these options and eliminate the ones that do not make sense.

The input starts with the number of options available, and then you have one option per line. For each option, you have the following information: price, number of hours and number of minutes (all integers separated by a space).

As a first step, you should sort these options in increasing duration. For the input below, the result would be the example shown above. Then, you should eliminate all options that have a higher price than any of the options above it in the sorted list, and print the ones kept as output (see below). The order of the printed options should match the order in the sorted list.

```
EXAMPLE INPUT:
6
844 14 33
519 6 5
448 8 20
409 15 13
453 8 8
448 7 50

EXAMPLE OUTPUT:
519 6 5
448 7 50
409 15 13
```

Problem 8: Flood risk map

With global warming and the recurring episodes of flooding, it will be important to create flood risk maps that indicate the probability of flooding in each location. Your program here will be a first attempt at doing this (note: of course, the real assessment would be way more complex than the calculations described below!).

We will assume that the map of an area is a square and that it is divided into cells like in a 2-dimension array. In the input (see the example input below), we first have the dimension of this array as a positive integer N (so N=6 in the example below). Then we have 2 N-by-N matrices as input. The first one indicates where the land is and where the rivers are: each cell contains either the letter 'L' for land or 'R' for river (separated by a space). The second matrix shows the height of each piece of land or river (as positive integers separated by a space). Note: the idea would be that this height is relative to the sea level, but we simplified the numbers here to keep them small and manageable.

Our analysis of such data will be limited to 3 aspects of flood risk only. First, it is assumed that land closer to a river is riskier. So as a first step, you should identify the shortest distance of each land cell to a river cell, as the number of horizontal or vertical moves to get to a river cell. You should print your result matrix as output (the "—" are for cells corresponding to a river).

Once you have identified the closest point to a river, you should take the difference in height between the river at that point and the land cell being considered (i.e., land that is not much elevated compared to the river has higher chances of flooding). Note: the land will always be higher than the closest river point. Also, if there are 2 points (of the river) for which the distance is the same, the difference in height should be the smallest one (e.g., first cell of third row below is at a distance of 1 to both the river cell on the left and the one down). Print your resulting array similarly to the distance one above, keeping an empty line in between.

The last risk aspect is related to the higher grounds around a particular land cell: when snow is melting, the water runs down on the land cell before reaching the river. So you should calculate the number of land cells that are higher than the cell being considered, which are accessible by a path of cells of increasing height (i.e., not on the other side of a mountain). Also print this resulting array in the same way as the other 2 above.

Use these 3 calculated arrays to estimate the risk level of each cell, in the following way:

Distance factor = distanceToRiver / maxDistance
where maxDistance is the number of steps between 2 opposite corners of the array (10 here, as 2 x (size – 1) where size=6)

Height factor = heightDifferenceWithRiver / maxDifference
where maxDifference is the largest height in the input minus the smallest height (36 – 1 here)

High grounds factor = 1 – (numberOfHigherGroundCells / numberOfLandCellsInArray)

Final factor = average of the 3 factors above

The final factor is used to categorize the cell as:

H (i.e., high risk of flooding): if the final factor is less than 0.25

L (i.e., low risk of flooding): if the final factor is greater than 0.5

M (i.e., medium risk of flooding): if the final factor is between 0.25 and 0.5

Finally, print your resulting array with these letters, with no space between the letters.

```
EXAMPLE INUT:
6
L L R L L L
L R R L L L
L R L L L L
R R L L L L
L L L L L L
L L L L L R
9 10 6 21 24 25
5 4 5 20 26 30
3 3 6 11 15 36
1 2 3 10 30 35
9 10 8 29 25 20
15 20 28 26 15 10

EXAMPLE OUTPUT:
2 1 - 1 2 3
1 - - 1 2 3
1 - 1 2 3 3
- - 1 2 3 2
1 1 2 3 2 1
2 2 3 2 1 -

3 4 - 15 18 19
1 - - 15 21 25
0 - 1 6 10 26
- - 1 8 20 25
8 8 6 19 15 10
14 18 18 16 5 -

1 0 - 5 4 2
2 - - 6 2 1
3 - 11 10 5 0
- - 18 12 2 1
4 2 4 0 4 5
2 1 0 2 7 -

MM-MLL
M--MLL
M-HMML
--HMLL
MMMLMM
LLLLM-
```

Example calculation for the land cell in 3rd row & 4th column:

- Distance to river = 2 (as going left twice, or going left and then up)
- Difference in height is either 8 (11 – 3) or 6 (11 – 5). We take 6 as it is the smallest value.
- Number of cells as higher ground is 10: the last 3 cells of rows 1 & 2, and the last 2 cells of rows 3 & 4
- Distance factor = 2 / 10 = 0.5
- Height factor = 6 / 35 = 0.17
- High ground factor = 1 – 10 / 29 = 0.66
- Final factor = (0.5 + 0.17 + 0.66) / 3 = 0.44

➔ M (final factor between 0.25 and 0.4)