# 12th Annual NB (English) High School Programming Competition

## hosted by UNB Saint John

## Friday, May 4, 2018

## 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: Weight divisions in boxing

In boxing, the two players who are fighting must be of similar weight in order for the fight to be fair. More precisely, the players are placed into categories based on their weight (called "weight division"), and they can only fight opponents of the same category. Here are the weight divisions used for men in the last Olympic Games:

| **Name of weight division** | **Weight thresholds** |
|---|---|
| FLY WEIGHT | up to 52 kg |
| LIGHT WEIGHT | up to 60 kg |
| MIDDLE WEIGHT | up to 75 kg |
| HEAVY WEIGHT | up to 91 kg |
| SUPER HEAVYWEIGHT | more than 91 kg |

Note: there were actually 10 weight divisions, but we reduced it to 5 for simplicity.

Your goal here is to identify the weight divisions corresponding to the weights given in input. More precisely, the input contains 3 integers (one per line) corresponding to the weights of 3 boxers. These weights are in pounds though (between 100 and 250), so first you need to convert them into kg (1 kg = 2.2 pounds). For the example input below, here are the corresponding weights in kg:

220 lbs → 100.0 kg
140 lbs → 63.6 kg
185 lbs → 84.1 kg

The output should be the weight division for each of these boxers, one per line, in the same order as they appear in the input.

```
EXAMPLE INPUT:
220
140
185

EXAMPLE OUTPUT:
SUPER HEAVYWEIGHT
MIDDLE WEIGHT
HEAVY WEIGHT
```

**Problem 2: Best fuel consumption**

Fuel consumption for cars depends on the speed the car is going. There is actually a typical pattern: fuel consumption is typically high at very low speeds, and then decreases gradually up to some optimum speed, and then increases again gradually as speed increases past this optimal point. Your goal for this problem is to identify such optimal speed, and print it.

The example input below contains data collected about a group of similar cars over a period of a few years. Each of the numbers is the average consumption (in L/100km) for different speeds: the first line is for 20 km/h, the second line is of 30 km/h, the third line is for 40 km/h, and so on until the last line which is for 120 km/h. As one can see, the numbers are going down until the fifth row, and then they go up again. Since the fifth row corresponds to the average fuel consumption at 60 km/h, the output is 60.

Note: Consumptions numbers are all between 1.0 and 20.0. You are guaranteed to have the pattern described above (i.e., data going down and then up), and no two consecutive numbers are the same (i.e., the solution cannot be more than one speed).

```
EXAMPLE INPUT:
10.5
9.0
8.0
7.5
7.0
7.4
7.7
8.0
8.6
9.3
10.0
```

```
EXAMPLE OUTPUT:
60
```

**Problem 3: Ice cream ordering**

You decide that for your summer job, you will create a small business and sell ice cream cones. You need to order enough ice cream to cover for the whole month of July, based on some daily estimates. For simplicity here, we will have only 3 flavors: vanilla, chocolate, and cookie dough. The cones can be of 1 scoop or 2 scoops. The 2 scoops ones can be of a single flavor only, or be a mix: vanilla-chocolate or vanilla-cookie-dough. In the input (see the example input below), you get the estimate of the number of cones to be sold per day of each type, in the following order:

- # 1-scoop vanilla cones
- # 2-scoops vanilla cones
- # 1-scoop chocolate cones
- # 2-scoops chocolate cones
- # 1-scoop cookie-dough cones
- # 2-scoops cookie-dough cones
- # vanilla-chocolate cones
- # vanilla-cookie-dough cones

Using such estimates, you want to know how many tubs of ice cream of each flavor (vanilla, chocolate, and cookie-dough – in this order) you need to order for the whole month of July. For your information, there are approximately 96 scoops of ice cream in a “3-gallon” tub (approx. 11.3 litres). And just to be on the safe side, you decide to order for 5% more scoops than needed, just in case sales are higher than predicted.

Here is an example of the calculations for the vanilla ice cream, based on the estimates provided in the example input below:

> Number of scoops needed per day: 5 + (2x15) + 22 + 12 = 69 scoops
> 5% of 69 is 3.45, so you need to cover for 72.45 scoops per day (just to be safe)
> For the 31 days of July, this is 2245.95 scoops (72.45 x 31),
> thus we need 24 tubs of vanilla ice cream (2245.95 / 96 = 23.4)

In a similar way, we can calculate the number of tubs needed of the chocolate flavor and the cookie dough flavor, and we get 28 and 19 respectively. These 3 numbers make up the output below.

<see next page for the example input and output>

```
EXAMPLE INPUT:
5
15
8
25
3
20
22
12

EXAMPLE OUTPUT:
24
28
19
```

## **Problem 4: Perfect paragraph structure**

When writing a text in plain English, a good paragraph structure is to have the first sentence somewhat summarizing the idea in the paragraph, and then the rest of the paragraph just adds the details to the main point in the first sentence. Fancy text editors or word processors could evaluate how well this rule is followed by simply counting how many words (of at least 4 characters – to avoid words such as “the”, “an”, etc) of the first sentence are repeated at least 2 more times in the rest of the paragraph (not including the first sentence). For simplicity, we check only for words re-spelled exactly in the same way (including uppercase/lowercase characters).

The input is a single paragraph to be evaluated. Your program should identify all long-enough words (i.e., at least 4 characters long) in the first sentence, and count how often each of those appear in the rest of the paragraph. For the example input below, there would be 7 such words (“This”, “programming”, “competition”, “students”, “from”, “high”, and “school”). The counts of each of these words are (respectively): 0 (“this” is not considered the same as “This”), 3, 2, 1, 0, 0, and 0. The output should indicate how many of those words are repeated at least twice, out of how many long-enough words we have. So the output here would be “2 out of 7” (you should use this wording exactly).

Notes on the input:

- The whole paragraph is on a single line of text, for simplicity. In the example below, the text has been “wrapped” in order for you to be able to read it all on this page, but that is not how it is stored in the input file used for testing.
- The paragraph (or line of text) does not contain more than 500 characters.
- The long words in the first sentence are never repeated within that first sentence.
- The first sentence always terminates with a period (“.”, no other characters such as “?” or “!”), and does not have other punctuation marks (i.e., all words are simply separated by a single space).

```
EXAMPLE INPUT:
This programming competition is for students from high schools.
Do you like competition? Do you like programming? If you like
both and you are a student in New Brunswick, this competition is
for you. You will meet many other students with the same passion
for programming as yours, and hopefully you will make new
friends! Let’s have fun programming!

EXAMPLE OUTPUT:
2 out of 7
```

## **Problem 5: Sustainable fishing**

Determining fishing quotas for each species depends on how healthy the fish population is. More precisely, we look at whether the population is growing or shrinking in general, over a number of years. We also look at the population of young fish, as an indication of future overall population. Indeed, if the overall population has been stable or slightly decreasing, but that the young population is growing, then we would be less worried about the population of that species in the next few years. So your goal here is to identify how healthy the population is (overall and young one) for various species, so as to help with quotas determination.

The first line of input contains the number of species to be analyzed. Then for each of them, there are 2 lines: one for the overall (adult) population and one for young population. Each line contains the estimated population in a given fishing area for the past 5 years (as 5 positive integers separated by a space). Your program should identify how healthy the populations are based on the average growth rate. The growth rate between two consecutive years is the difference between the two (second number minus first number) divided by the first number. For example, the growth rate for the two first years in the first line of such data in the example input below (i.e., 300 and 350) is (350-300)/300. You need to calculate such rate for each of the 4 pairs of consecutive numbers within the line, and then take the average of those rates.

Such calculation has to be done for each line of estimated populations. For the data below (example input), the average rates are: 0.0145, -0.061, -0.080, and 0.392. Then, for each species, we have one line of output with 2 words, separated by one space. The first word is related to the average rate of the overall population:

> "healthy": rate is > 5%
> "danger": rate is < -5%
> "stable": otherwise

The second word is related to the young population:

> "promising": rate is > 5%
> "concerning": rate is < -5%
> "stable": otherwise

So for the first specie in the example input below, we get "stable" (0.0145 is between -0.05 and 0.05) and "concerning" (-0.061 is smaller than -0.05), while the second specie is in "danger" (-0.080 is smaller than -0.05) but "promising" (0.392 is larger than 0.05).

<see next page for the example input and output>

```
EXAMPLE INPUT:
2
300 350 305 290 310
100 80 85 70 75
500 400 425 375 350
50 60 100 150 180

EXAMPLE OUTPUT:
stable concerning
danger promising
```

## **Problem 6: Detecting activity swings**

Activity tracking devices (also called fitness tracking devices) are used for tracking the number of steps walked over certain periods of time. To a certain extent, those could be used to track mood swings (including initial phase of depression), as activity swings might be good indicators of mood swings. What we are interested in, here, is to look at the level of activity every day over a given period (e.g., one week), and compare it with daily averages we have over a similar period, to highlight differences from normal times.

The input starts with a number on the first line, indicating the length of the periods being analyzed (in number of days). For the example input below, this number is 7, meaning that we will check how the level of activity was different over each day of the week. Such number of days is always between 1 and 30. Then, on the second line, you have the sequence of averages (as positive integers separated by a space) for each of those days. For the example below, this could be interpreted as: this person typically walks 6450 steps on Sundays, 7321 on Mondays, and so on for the 7 days of the week. The rest of the input contains the data for the specific “weeks” (or other periods as defined in the first line) to be analyzed. The third line indicates how many “weeks” should be analyzed, and then we have one line per “week” containing the number of steps every day of that “week” (in the same format as the second line).

For each “week” (or period), you should compare the number of steps of each day with the corresponding number in the average sequence. You should count the number of times that the numbers are almost the same (i.e., within 500 steps of the average), more, or less than the average (both by more than 500 steps). In the example below, using the first week to be analyzed (i.e., line 4), such counts would be 2 “same” (6950 and 4030), 4 “more” (8000, 7706, 8912 and 5214), and 1 “less” (3885). Those counts are printed in the output, on the same line, separated by a space. Finally, if the count of “more” or the count of “less” is more than half of the period length (i.e., 4 or more for a week of 7 days), this word is added at the end of the line. The output contains one such line per “week” analyzed, in the same order as the input, and terminates with one line indicating which “week” had the highest total number of steps overall (using the exact same wording as what is used below in the example output).

```
EXAMPLE INPUT:
7
6450 7321 4040 6708 8411 5998 4537
3
6950 8000 4030 7708 8912 3885 5214
3400 6190 4567 6077 5634 5432 6000
6450 9999 3511 9452 8077 5444 7890
```

```
EXAMPLE OUTPUT:
2 4 1 more
0 2 5 less
2 3 2
Period 3 is the most active one
```

**Problem 7: Data security**

We would like to implement a system for monitoring data access requests (to our server), and identify potential threats. As a first version of such a system, you just want to highlight the "heavy" users (or IP addresses), both in terms of number of access requests and in the total size of their downloads. Those heavy users would then be further investigated, but that part is outside the scope of your program here.

The input to your program is a list of accesses that were handled over a certain period of time, as the user making such access and the total size downloaded during such access. More precisely, the input starts with a single number on the first line, indicating the number of accesses in the list. Then for each access, there is one line of input with the data about it: first a "user ID" (as a single letter between 'a' and 'z' for simplicity), and then the number of MegaBytes (MB) downloaded during that access (always a positive integer). Note: it is possible that there are no access recorded for a particular letter (or user).

Using this data, your program should count the number of accesses made by each user, and the total number of MB downloaded. For the example input below, here are those counts:

| user | # accesses | Total MB downloaded |
|---|---|---|
| a | 1 | 60 |
| b | 8 | 80 |
| c | 4 | 25 |
| d | 3 | 100 |
| e | 6 | 30 |
| f | 5 | 70 |

The numbers in shaded cells are the top 3 numbers in each of the columns ("# accesses" and "Total MB downloaded") over all users. So your program should identify such top 3 values, and then it should print the letters (one per line, and in alphabetical order) corresponding to the users having both their # accesses and their total MB downloaded in the top 3 (i.e., lines in the table above where both values are shaded). Note: we guarantee that no two values will be the same, in each of the columns above.

<see next page for the example input and output>

```
EXAMPLE INPUT:
27
d 25
a 60
b 10
b 10
c 10
b 10
e 5
b 15
b 5
b 15
e 5
b 5
c 5
c 5
d 50
d 25
e 5
f 10
e 5
e 5
e 5
f 20
f 10
f 15
f 15
b 10
c 5

EXAMPLE OUTPUT:
b
f
```

## **Problem 8: Business trip planning**

You are going on a business trip, bringing a collection of things that could be sold there. Each object has a weight (in kg) and the price that the object can be sold for. You are limited to only one luggage though, which is flexible but can only take a maximum weight (also in kg). So you cannot bring all objects you have, as they would not fit all. Your question is: assuming that all objects brought will be sold, which objects should you pick in order to maximize your total sales, but with the constraint that the total weight of those objects should not exceed the maximum space you have in your luggage? (Note: you cannot take one object more than once) Write a program for answering that question.

The input to your program is the following. First, you have the maximum weight for the luggage, as a positive integer. The next line indicates the number of objects you can pick from. Then there is one line per object, with the following information about it: first, the name of that object (as only one word), then its weight and finally the price we expect to sell it (all separated by a space). For simplicity, both the weight and the price are positive integers. Your program should identify which subset of objects would maximize the total sales (subject to the total weight constraint), and print those objects, one per line, in the output. The selected objects should be printed in the same sequence that they appear in the input. Finally, your program should print the total amount of money you will get if you can sell all the selected objects. In the example below, the best subset of items fitting in a 3kg-luggage is the ipod and the knife, for a total sale of $500 ($350 + $150).

```
EXAMPLE INPUT:
3
7
Wine 3 30
Pillow 1 10
Razor 2 50
Soaps 2 10
Ipod 1 350
Knife 2 150
GPS 3 450
```

```
EXAMPLE OUTPUT:
Ipod
Knife
500
```