11th Annual NB (English) High School Programming Competition

hosted by UNB Saint John

Friday, May 12, 2017

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: Time converter

Time is usually indicated under the 12-hour clock system, meaning that the time is a number from 1 to 12 inclusively, to which we attach “am” or “pm” (e.g., 11am). Of course, minutes can be specified too (e.g., 11:30am).

However, this is not the international format for time. A formal way of specifying time is to use the 24-hour clock system, where the hour is a number between 0 and 23 (with 0 representing midnight). For example, 3:20pm is 15:20 in the 24-hour system. Such format is used for example in the military or on flight itineraries. Some people have difficulties reading time in this format though, and you decide to write a short program for doing the conversion from the 24-hour clock to the 12-hour clock. For simplicity here, we will only look at the hours, not the minutes.

You should write a program that will perform 3 time conversions. The input consists of 3 whole numbers between 0 and 23 inclusively, one per line. For each of these numbers, you should do the conversion and output the result, one per line, in the same order as the input. The example input and output below shows you that 15:00 is in fact 3pm, that 5:00 is 5am, and that 12:00 is 12pm (noon).

```
EXAMPLE INPUT:
15
5
12
```

```
EXAMPLE OUTPUT:
3pm
5am
12pm
```

Problem 2: Chinese Zodiac

If you have gone to a Chinese restaurant, you have most likely seen a message such as "if you are a horse, you would get along best with a dog or a tiger, but you should avoid the rat".

In the Chinese Zodiac, each year is associated with one of 12 animals, and you determine your type of personality by getting the animal associated with your year of birth. Here is a list of these animals and some of their matching years:

- Rat: 1984, 1996, 2008
- Ox: 1985, 1997, 2009
- Tiger: 1986, 1998, 2010
- Rabbit: 1987, 1999, 2011
- Dragon: 1988, 2000, 2012
- Snake: 1989, 2001, 2013
- Horse: 1990, 2002, 2014
- Goat:1991, 2003, 2015
- Monkey:1992, 2004, 2016
- Rooster: 1993, 2005, 2017
- Dog: 1982, 1994, 2006
- Pig: 1983, 1995, 2007

The compatibility between people follows a simple rule: one animal is compatible with the other animals that are 4 "steps" apart, while the opposite animal (i.e., 6 "steps" apart) is the incompatible one. The compatibility with other animals than those is not determined. So for example, someone born in 1998 (Tiger) will get along very well with someone born in 1994 (Dog) or born in 2014 (Horse), but will not be compatible with someone born in 2004 (Monkey).

You should write a program that identifies the animal associated with two people, and indicate their compatibility. This should be done for 4 pairs of people. More specifically, the input contains one line per pair of people (so 4 lines in total). Each line contains 2 years from the lists above (i.e., two numbers between 1982 and 2017 inclusively), separated by a space. The output should have 2 lines per pair of people (i.e., 2 lines for each of the lines in input, for a total of 8 lines). The first of these 2 lines should contain the animal corresponding to each of these two years, in the same order, and separated by a space. Make sure that you capitalize only the first letter of the animal name. The second line should contain one of the 3 possible compatibility outcomes, according to the rule described above: "compatible", "not compatible" or "not determined" (all lower case, and without the double quotes).

```
EXAMPLE INPUT:
1998 2004
1984 2009
2006 2002
2017 2017


EXAMPLE OUTPUT:
Tiger Monkey
not compatible
Rat Ox
not determined
Dog Horse
compatible
Rooster Rooster
not determined
```

Problem 3: Lucky numbers

A number is considered a lucky one if the sum of its digits, as well as the sum of the square of its digits, are prime numbers. For example, the number 120 is lucky because the sum of its digits (i.e., 1+2+0) is 3, a prime number, and the sum of the square of its digits (i.e., (1*1)+(2*2)+(0*0)) is 5, which is also a prime number.

We would like to know how many lucky numbers are in an interval between two given numbers X and Y. You should write a program to count this. The first line of input is a positive integer T, indicating the number of intervals to be tested. Then, there are T lines (one per interval) containing 2 positive integers (X and Y), separated by a space. Note that the first number will always be smaller or equal to the second number.

Constraint: $1 \leq X \leq Y \leq 10^9$

The output of your program should display only one number per interval: the count of lucky numbers found within that interval.

In the example below, there are 2 intervals to be tested. In the first one (between 1 and 20 inclusively), there are 4 lucky numbers: 11, 12, 14, and 16. In the second one (from 120 to 130), there is only one lucky number: 120.

```
EXAMPLE INPUT:
2
1  20
120  130
```

```
EXAMPLE OUTPUT:
4
1
```

Problem 4: Where is UNBSJ?

This problem is just a simple game of word searches, with a single word to be found: UNBSJ.

The input contains a character array, containing only upper-case letters (not separated by spaces). The dimensions of this array are provided in the first line, as the number of rows followed by the number of columns (2 positive integers separated by a space).

The word UNBSJ is guaranteed to be in the array exactly once. It can be read only from left to right or from top to bottom. Your output should contain a single line, with 2 numbers separated by a space. These numbers should represent the coordinates (row number followed by column number) of the 'U' in UNBSJ. It is assumed that the top-left corner of the array (the letter 'T' here) is at coordinates (1,1).

```
EXAMPLE INPUT:
6 14
THECOMPETITION
ISATUNBSJEVERY
YEARANDITISLOT
OFFUNANDFRIEND
COMINGFROMTHIS
ENTIREPROVINCE
```

```
EXAMPLE OUTPUT:
2 5
```

Problem 5: Analyzing extreme effects

Data analytics encompasses the identification of effects of one variable over another one in a large dataset. For example, one may want to know how the outside temperature is affecting the level of physical activity of people. One problem is that very often, there are many factors that may come into play. For our example here, the level of physical activity of people may be affected by other things than the temperature, for example if the person has the flu, or did not have a chance to sleep enough recently. In this case, we can still detect an effect “at the extreme”: we can check for example if on really hot days, people are getting significantly more or less active than usual. You goal here is to write a program to perform this kind of “analysis at the extreme”.

In particular, you will be analyzing a dataset related to a person who is exercising by running outside. This dataset contains pairs of numbers: the temperature at a certain time, and the speed of the runner at that time. We are interested to see if the runner is typically running faster or slower on really hot days (or really cold days).

The first line of input provides the thresholds to be used, for considering some value as “extreme”. These are specified as 4 positive integers, separated by a space. The first two numbers represents the thresholds for the temperature. In our example input below, it means that temperatures below 20 degrees are considered “really cold” and temperatures above 30 degrees are considered “really hot”. The last two numbers are the thresholds for the speed. Looking again at the example input below, a speed slower than 12 km/h is considered “really slow” while a speed higher that 15 km/h is considered “really fast”.

The rest of the input contains the actual data. First, you have a value N (positive integer on its own line), indicating the number of entries in the dataset. Then there are N lines, containing two positive integers separated by a space: the first one is the temperature, and the second one is the speed. As the dataset is being read, you should keep track of the following counts:

- Number of times where the speed was really fast, on a really hot day
- Number of times where the speed was really slow, on a really hot day
- Number of times where the speed was really fast, on a really cold day
- Number of times where the speed was really slow, on a really cold day

Your output should contain those 4 counts, one per line, in this order. In the example below, you will see that the runner being analyzed is generally affected negatively by the heat, but is not really affected by the cold (same number of “fast” than number of “slow”).

```
EXAMPLE INPUT:
20  30  12  15
15
32  10
25  13
17  11
16  17
22  10
33  11
29  16
15  11
31  10
19  16
31  13
32  16
34  9
29  10
30  17
```

```
EXAMPLE OUTPUT:
1
4
2
2
```

Problem 6: Maximum paper folding

Origami is the art of paper folding, where the goal is to transform a flat sheet into some kind of "object" or "sculpture". Some sculpture design are particularly fancy, and require a large number of folds. One issue is the feasibility of the design: a piece of paper can be folded only a certain number of times before such operation becomes impossible. For example, if you try to fold a regular letter-size piece of paper in half multiple times, you will see that you cannot do this more that 6 or 7 times (depending on how thick your paper is).

For our problem here, you will have to determine the maximum number of times that a given sheet of paper can be folded in half, based on its original dimensions and its thickness. We will assume here that the total thickness of the folded paper should be no more than a third of each of the dimensions. Here is an example, for a regular letter-size paper:

	Dimensions	Thickness
Original dimensions:	216 mm X 280 mm	0.1 mm
After 1st fold:	216 mm X 140 mm	0.2 mm
After 2nd fold:	108 mm X 140 mm	0.4 mm
After 3rd fold:	108 mm X 70 mm	0.8 mm
After 4th fold:	54 mm X 70 mm	1.6 mm
After 5thfold:	54 mm X 35 mm	3.2 mm
After 6th fold:	27 mm X 35 mm	6.4 mm
After 7th fold:	27 mm X 17.5 mm	12.8 mm

From this calculation, one can see that folding 6 times is possible: when multiplying the thickness by 3 (i.e., 6.4 x 3), we get 19.2, which is smaller than both dimensions (27 and 35). However, folding 7 times is not possible: multiplying the thickness by 3 gives us 38.4, which is larger than at least on of the dimensions (actually, larger than both 27 and 17.5 here).

You should write a program that returns the maximum number of times that a sheet can be folded in two, given the original dimensions and thickness of the sheet. More specifically, the input contains 2 lines. The first one contains two positive integers, separated by a space, representing the dimensions of the sheet. The first of these two numbers will always be smaller or equal to the second number. On the second line, there is one positive real number that represents the thickness of the sheet. The output should contain only one positive integer, representing the maximum number of folds. The example below is the input and output for the example calculated in the table above.

```
EXAMPLE INPUT:
216  280
0.1


EXAMPLE OUTPUT:
6
```

Problem 7: Loading cargo boat

A cargo boat is transporting 2N containers. Each container has a weight. We need to load the boat into 2 rows of exactly N containers, in such a way that it is as balanced as possible (i.e., the difference between the total weight of each row is as small as possible). You should write a program that will help in planning how to best load this boat, given the weight of each container. Actually, for simplicity here, we will just ask that minimal difference, not where to put the containers to obtain such best balance.

In the first line of input, you have the value N specified, as a positive integer. The second line contains exactly 2N positive integers (separated by a space) representing the weight of each container. Note that there could be two (or more) containers having the same weight. Also, the weights are not sorted in any way.

The output should contain only one line, containing the minimal difference. In the example below, the best way to load the boat is to put the first 3 containers on the same row (for a total weight of 15), and to put the last 3 containers on the other row (for a total weight of 16). The difference between 15 and 16 is 1, which is the output provided below.

```
EXAMPLE INPUT:
3
7  3  5  2  13  1
```

```
EXAMPLE OUTPUT:
1
```

Problem 8: Game recommendations

When you go to an online store to buy some new games, you probably noticed that you also get recommendations about other games that could also be of interest to you. For example, when you go to the page describing the game “Haunted Hotel”, at the bottom you have the following recommendation: People who bought this game also bought “Legends of Fate”.

(note: the example here is not meant to advertise any particular game)

The goal of the program to be written here is to make such recommendations based on previous sales. To simplify the input, we will name the games “A”, “B”, “C”, … “Z” (so 26 possible games). Also, we will assume that when someone buys some games, it is always as a pack of exactly 5 games, because there is a rebate when doing so. Using such data on previous sales, the recommendations associated with a particular game (requested in the input) should be the list of other games for which at least 3 people bought it in the same pack of 5.

The first line of input indicates the number of previously-sold packages of games (as a positive whole number). Then, for each of those packages, you have one line of input containing 5 letters (always upper-case) representing the games in that package, all separated by a space. The rest of the input indicates the games for which we seek recommendations. The first line in that section contains a number N (a positive whole number), indicating the number of games to be handled. Then there are N lines, each containing one letter (upper-case) representing the game. The output should contain N lines, one per game in the second part of the input. Each line starts with the name (letter) of the game for which recommendations are being made, followed by a colon (“:”) and a space. Then, you should put the list of recommended games, separated by a space. This list should be in alphabetical order.

```
EXAMPLE INPUT:
6
B A D F Z
E I B Z D
C D A B I
B I C E X
E I A B C
A B C D E
2
A
C
```

```
EXAMPLE OUTPUT:
A: B C D
C: A B E I
```