10th Annual NB (English) High School Programming Competition

hosted by UNB Saint John

Friday, May 6, 2016

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 and corresponding outputs for each problem are available through the web shortcut on the desktop.**

There are 8 questions. Good luck!

Problem 1: Financial projections

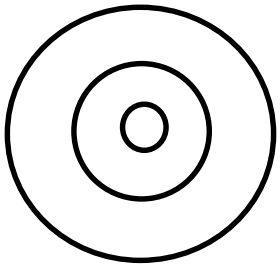
A company has an aggressive target for sales, and hope to double its profits every month for the next 4 months. In order to simplify the management of such target, they want to know the exact amount that this will represent. You decide to write a small program that will perform this for them.

The input to your program is just one positive whole integer, representing the latest monthly profits. You should then calculate and print the predicted profits for the next 4 months, one number per line. For example, if the input is 300, the projections should be: 600, 1200, 2400, and 4800. The example input and output below illustrates this example.

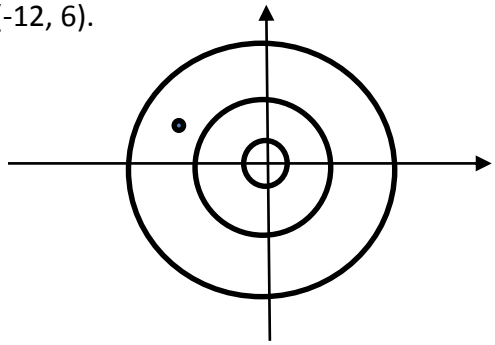
```
EXAMPLE INPUT:
300

EXAMPLE OUTPUT:
600
1200
2400
4800
```

Problem 2: Simple darts game

You like to play the darts game, but you are tired of having to count the points yourself. You decide to write a program to do this for you. For simplicity, the game is going to be played on a simple dartboard that looks like the one below. When a dart hits the small circle in the center, the player gets 20 points. The middle region is worth 10 points and the outer region is worth 5 points. We will assume here that 5 darts are thrown by the player, and the total number of points is the sum of the points related to each of the regions hit by the 5 darts. We will also assume, for simplicity, that the darts never exactly hit the circles separating the regions.



In order to identify the location of the 5 darts, we will use a Cartesian plane where the origin is aligned with the center of the dartboard, as illustrated below. The position of a dart is provided as the x-y coordinates on that plane. For example, if a dart hits the place of the dark circle, its coordinates would be (-12, 6).



The circles defining the 3 regions of the dartboard are at 2 cm, 10 cm, and 20 cm respectively from the origin. Your input consists of the location of the 5 darts, one per line. Each line contains 2 whole numbers separated by a space, corresponding to the X and the Y coordinates of a dart. Using these locations, you should identify the region that they hit, calculate the total points, and print this total as output. Hint: if the position of a dart is (x,y), then the distance between the dart position and the origin is calculated as $\sqrt{x^2 + y^2}$.

```
EXAMPLE INPUT:
  3   0
  1   1
 -8   1
-10 -15
 16  -5

EXAMPLE OUTPUT:
50
```

Problem 3: Rounding cents

When going to a store and paying cash, we rarely provide the exact amount. Instead, we give a dollar bill higher than the amount to pay, and the cashier returns the difference. Since we do not have pennies anymore, the amount to be returned should be "rounded" to the closest nickel. For example:

$1.22 → $1.20
$1.24 → $1.25
$1.26 → $1.25
$1.28 → $1.30

In the problem here, you should read the amount to be paid in the input. This amount will always be smaller than 100. Then, you should identify the smallest dollar bill that covers such an amount (among: $5, $10, $20, $50, and $100), and calculate the amount to return, rounded as described above. For example, if the amount to be paid (provided as input) is $7.34, then the dollar bill to be used is the $10, giving us $2.66 to be returned. After the "rounding", this amount becomes $2.65, and that is what should be printed on the screen.

Your program should handle 5 such cases, as illustrated in the example input and output below. Each line of input has its result printed on the corresponding line of output.

```
EXAMPLE INPUT:
7.34
12.41
87.80
19.93
2.22

EXAMPLE OUTPUT:
2.65
7.60
12.20
0.05
2.80
```

Problem 4: Stained-glass window

You are an artist, and someone asked you to create a beautiful stained-glass window. For simplicity, your piece of art is going to be composed of a number of colored geometric shapes inserted on a transparent window. However, colored glass is more expensive than regular (transparent) glass, so you want to limit the use of colored glass to no more than 30% of the whole window.

In order to help you check that your constraint is met, you decide to write a program to make all the calculations necessary, and tell you the percentage of colored glass used in your design. In the input, the first line contains the dimensions (width and height) of the whole window. On the second line, you have the number of shapes that are going to be used. The description of those shapes are provided, one per line, in the rest of the input. Each line contains 3 numbers (referred to later as code, num1, and num2): the first one is a code for shape, and the other two numbers are dimensions related to this shape (all in cm). The information provided for each of the shapes that your program should handle is provided in the table below. The formula to calculate the area of such shape is also provided, as a reminder.

Circle	Code = 1 Num1 = radius of the circle Num2 = always zero (not necessary in the calculation) Area = 3.14 x $(radius)^2$
Square	Code = 2 Num1 = size of the side of the square Num2 = always zero (not necessary in the calculation) Area = $(side)^2$
Rectangle	Code = 3 Num1 = size of the base of the rectangle Num2 = height of the rectangle Area = (base) x (height)
Isosceles triangle	Code = 4 Num1 = size of the base of the triangle (the side of different length than the others) Num2 = height of the triangle Area = (base) x (height) / 2

The sample input below (see next page) thus represent a window of size 100 cm X 100 cm, that will contain 4 colored shapes inserted, covering a total area of 1009.66 cm^2 (100 + 7.5 + 452.16 + 450). The percentage covered is thus 10% (1009.66 / (100 x 100), rounded). Note: the input will not necessarily contain every possible shape, and there could be more than one shape with the same code. Also, all numbers in the input are positive whole numbers, separated by spaces when on a single line.

EXAMPLE INPUT:

```
100  100
4
2  10  0
4  3  5
1  12  0
3  30  15
```

EXAMPLE OUTPUT:

```
10
```

Problem 5: Encoded e-mails

You suspect your parents of reading your e-mails. Indeed, they often show up at the places where you go meet your friends. This has to stop! So together with your friends, you come up with a simple way to encode your message in a text: the message is composed the following words in the text:

- No word from the first line
- First word or the second line
- Second word of the third line
- Third word of the fourth line
- And etc.

Write a program that reads the text, and prints the encoded message. The first line of the input indicates the number of lines of text that should be read. For simplicity, words are a sequence of characters (letters or other punctuation marks) separated by a space.

EXAMPLE INPUT:
6
Kim is in town! It would be so great to finally
meet her! I am sure that
seeing me would make her happy too. She
would smile, at the very least. She has
family living on Georges street. I think I will wait
for her at that place and surprise her.

EXAMPLE OUTPUT:
meet me at Georges place

Problem 6: Facial recognition of terrorists

In national security agencies across the world, facial recognition software are used to identify known terrorists on images from security cameras. There is still a lot of research devoted to this area, though. In simplistic terms, such recognition is based on various metrics (or distances) between the main features in the face. The measures from a new image are collected and then compared with the ones from a database, searching if the new image would match, with high probability, to one (or more) from that database.

For the problem here, you have to implement a simple version of such facial recognition software. In particular, we will be using only 3 metrics for the identification, for simplicity: the distance between the eyes, the nose height, and the length of the mouth (when not smiling, like on passport photos). All will be measured in millimeters. Since there can be some errors when collecting the metrics, the probability will be based on the metrics being within a certain margin error. For each metric (separately), you should assign one of the following probabilities: 1 if the measures are exactly the same, 0.75 if the difference in the measures is 1 mm, and 0.5 if the difference is 2mm. It is assumed that if the difference is more than 2 mm, you definitely do not have a match. The overall probability of a match is calculated as the product of these 3 individual probabilities. We will consider that an overall probability of 0.5 or higher is considered a potential match. For example, when comparing an image with the characteristics [eyes=30, nose=50, mouth=45] to another one with [eyes=30, nose=51, mouth=44], the overall probability of a match is 0.5625 (calculated as 1 x 0.75 x 0.75). Since this number is higher or equal to 0.5, this is flagged as a potential match.

The input begins with a single line containing N, a positive whole number, indicating the number of cases (or images) in the database. Then there are N lines. Each consists of the 3 measures related to that image, as 3 positive whole numbers separated by spaces. This is followed by an empty line, and then a positive whole number M (on a single line) indicating the number of new images to be searched in the database. There are then M lines. Each consists of the measures of each of these images, in the same format as the ones in the database.

Your program should take each of the new images in turn, and compare it with each of the images in the database. The output should contain one line per new image, in the same order as in the input. The line should contain the list of images from the database (i.e., their index) that are a potential match, separated by spaces. Note: indexes start at 1. If there are no matches at all for a particular new image, the words “no match” (all lower case) should be printed instead. Within each line, the indexes should be displayed in ascending order.

<see next page for the example input and output>

EXAMPLE INPUT:

```
4
30  51  44
30  50  45
32  52  45
34  50  45

3
30  50  45
32  50  45
30  55  45
```

EXAMPLE OUTPUT:

```
1  2
2  3  4
no match
```

Problem 7: Best flu prevention treatments

When deciding what to propose as a way to avoid getting the flu, health professionals look at published experiments and compare results. For example, there could be experiments where groups of people try various approaches, over a number of years. We then count the number of flu cases among each group, on each of those years, to identify the best approach. The best one is considered the one that has the smallest number of cases over the entire period covered by the experiment. This best approach is considered to be significantly better than the others only if it wasn't beaten by any other more than a certain number of times. Your goal here is to write a program that analyzes such experimental data in order to identify this best treatment, and provide information about how significantly better it is.

The input begins with a single line containing N, a positive whole number, indicating the number of treatments being analyzed. Then there are N pair of lines: one line containing the name of the treatment, and the second line containing the yearly counts of flu cases for the past 5 years (as 5 positive whole integers, separated by spaces). Your work is then to calculate the sum of all flu cases over the 5 years, for each flu treatment, and print the name of the treatment having the lowest sum. Then, on a separate line, you should print how many of the 5 years were in the situation where all other treatments had more flu cases than this best one. Note: there will not be a tie for the best treatment. Also note that the data below is totally fake, and is not meant to promote any actual treatment!

EXAMPLE INPUT:
5
Do nothing
20 15 20 15 10
Vaccine
10 10 25 10 5
Use hand sanitizer
15 10 20 15 7
Antiviral flu drugs
19 14 19 14 10
Immune system booster
19 14 19 14 10

EXAMPLE OUTPUT:
Vaccine
3

Problem 8: Binary puzzles

You have probably noticed recently that Sudoku puzzle lovers are moving to a new type of puzzle: Binary puzzles. This puzzle is done on a 10 x 10 grid, where each cell has to be filled with the value 0 or 1. Some cells are already filled at the beginning. The remaining cells are then filled using the following rules:

1. Each row and each column should have the same number of 0s as the number of 1s (i.e., 5 of each).
2. There cannot be more than 2 consecutive 0s or 2 consecutive 1s, in any of the rows or columns.

Using the rules above, solving the puzzle (when it is quite simple) can be done by applying the following strategies:

1. When you find the pattern 0—0 (i.e., two consecutive 0s), add a 1 on each side, to get 1—0—0—1. Similarly, when finding the pattern 1—1, you can add 0s to get 0—1—1—0.
2. When you find two 0s separated by an empty cell, you can fill this cell with a 1 (i.e., 0—?—0 becomes 0—1—0). Similarly, when you find two 1s separated by an empty cell, you can fill this cell with a 0 (i.e., 1—?—1 becomes 1—0—1).
3. When you have 9 out of the ten numbers identified in a row or column, you can easily identify the missing symbol.

As an example, starting with the grid below, the first 2 empty cells can be filled with 1s, to surround the double-0 sequence in the second and third column. With this, we then see that this first row has five 1s and four 0s, so the last empty cell in this row (next to last column) should be a 0. Then, looking at the first column, the last empty cell (8th row) should be a 0, because of the two 1s around it. This kind of work continues (not necessarily in the sequence illustrated here), until all empty cells are filled with either a 0 or a 1. This example is the same as the example input below, with its solution in the example output.

	0	0		0	1	1	0		1
0	1	0	1	1	0	1	0	0	1
1	0	1	0	1	0	0	1	1	0
	1		0			1	0		0
	0			1			0	0	1
0	1	1	0	1	0	0	1	1	0
1	1		1	0	1	0	1		0
	0	1		1	0			1	1
1	0		1	0	1	0	1	1	0
0	1	1	0	0	1	0	1		1

You should write a program that reads an initial binary puzzle that can be solved by the strategies given earlier, and apply the strategies until all cells are filled. The input contains 10 lines, with 10 numbers each, separated by spaces. A value of 2 is used to indicate an empty cell. The resulting puzzle should be printed in a similar format, with exactly one space between each number, but no space at the end of the line before the return character. There should not be any 2s left, only 0s and 1s.

EXAMPLE INPUT:

```
2 0 0 2 0 1 1 0 2 1
0 1 0 1 1 0 1 0 0 1
1 0 1 0 1 0 0 1 1 0
2 1 2 0 2 2 1 0 2 0
2 0 2 2 1 2 2 0 0 1
0 1 1 0 1 0 0 1 1 0
1 1 2 1 0 1 0 1 2 0
2 0 1 2 1 0 2 2 1 1
1 0 2 1 0 1 0 1 1 0
0 1 1 0 0 1 0 1 2 1
```

EXAMPLE OUTPUT:

```
1 0 0 1 0 1 1 0 0 1
0 1 0 1 1 0 1 0 0 1
1 0 1 0 1 0 0 1 1 0
0 1 1 0 0 1 1 0 1 0
1 0 0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 1 0
1 1 0 1 0 1 0 1 0 0
0 0 1 0 1 0 1 0 1 1
1 0 0 1 0 1 0 1 1 0
0 1 1 0 0 1 0 1 0 1
```