9th Annual NB (English)

High School Programming Competition

hosted by UNB Saint John

Friday, May 8, 2015

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: odd numbers

In grade 2, children learn to count by twos. Andy has no problem when counting with even numbers, but finds it more difficult with the odd numbers. You decide to write a program to help him practice to count in successive odd numbers. Given two numbers, your program should display all the odd numbers between these two numbers inclusively. The two numbers are not necessarily provided in order. In the output, the numbers should be separated by a space.

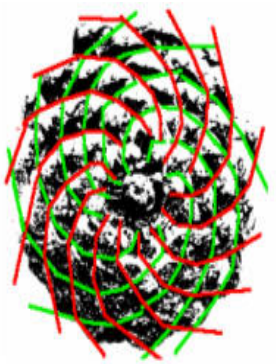
```
EXAMPLE INPUT:
51 42

EXAMPLE OUTPUT:
43 45 47 49 51
```

Problem 2: healthy sunflowers

The Fibonacci sequence is a sequence of numbers where the two first ones are 1, and then each number after that is the sum of the two previous ones in the sequence. Here are the first few numbers of that sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, etc. Although this seems a very theoretical sequence, it is found in nature.

Pinecones and sunflowers have an interesting way of shaping themselves so as to use space in the most efficient way. First, you should count the number of spirals, clockwise and counter-clockwise (when drawn starting in the center). In the example below, one can see 8 and 13 such spirals respectively.



If the two numbers you get are two consecutive numbers in the Fibonacci sequence, then it means that the pinecone or flower is using the space in an efficient way…it is healthy. The problem here is: given 5 such pairs of numbers of spirals (the first one always smaller than the second one), indicate which flowers are healthy and which ones are not.

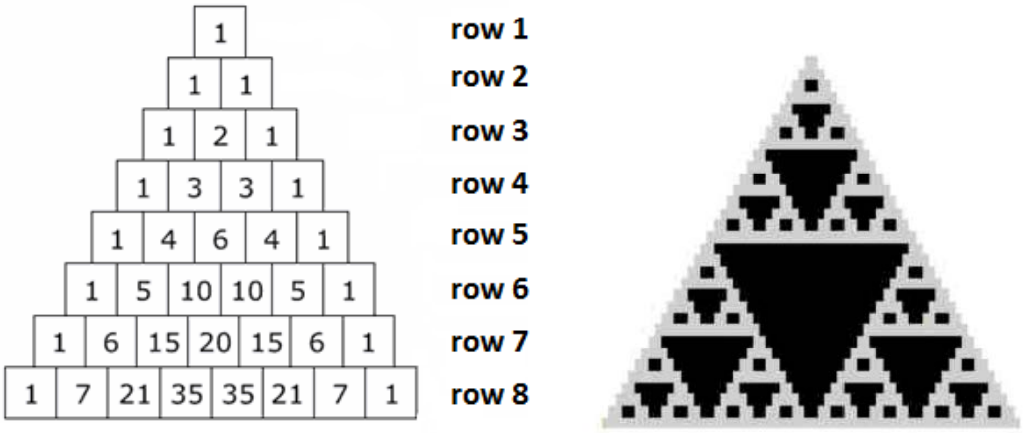
```
EXAMPLE INPUT:
8  13
18 23
8  21
21 34
1  2

EXAMPLE OUTPUT:
healthy
not healthy
not healthy
healthy
healthy
```

Problem 3: triangular garden

You are a professional landscaper, and you have an eccentric client: he loves triangle shapes and arrangements made of various triangles of various sizes. You propose to him to make a triangular-shape garden made of flowers of two different colors, and following the pattern called “Mod-2 coloring of Pascal’s triangle”.

As a reminder, the Pascal triangle is a triangular array that has a very interesting property: every cell is equal to the sum of the two cells in the row above it. The left most and right most cells in every row will always have a value equal to 1. Below, on the left, is a Pascal triangle with 8 rows. The Mod-2 coloring means that you paint each cell containing an odd number with one color, and you paint the cells containing an even number with another color. On the right, you can see a particular pattern made this way with the colors black and white, for 32 rows.



You client loves it and you get the contract for a few such triangles. Now you need to plan on the number of flowers to buy of each color. Your program should read in input a size for the garden (as number of rows), and return the number of flowers needed to cover the “odd” cells (which will be blue) and the number of flowers needed to cover the “even” cells (which will be pink). In the example above (on the left), we would need 27 blue flowers and 9 pink flowers. The input starts with the number of triangular gardens to make, followed by the number of rows for each garden, one per line. The output should contain the same number of lines as the number of gardens, and each line should contain the number of blue flowers and pink flowers needed (separated by a space).

```
EXAMPLE INPUT:
2
8
6

EXAMPLE OUTPUT:
27  9
15  6
```

Problem 4: casino

You friend is managing a casino, containing a number of dice gambling tables. He suspects that the dice used at some tables are biased, and he asks your help to detect which ones are potentially biased. He can give you the numbers that were rolled at each table over 24 rolls. This means that if the dice were not biased, then you would expect each value (1 to 6) to appear approximately 4 times each. Of course, there could be some variations, and thus you decide that when the frequencies for each value is between 2 and 6, that could simply be by chance. When some values appear more than 6 times though, chances are that the dice are biased.

Your problem is to read the values rolled at a number of tables, and identify which tables potentially have biased dice. The input starts with the number of tables to check, followed by one line per table. For each table, the 24 numbers rolled are provided, separated by a space. In the output, you should have the table numbers (the first table is #1), one per line, corresponding to tables with potentially biased dice.

```
EXAMPLE INPUT:
3
6 6 6 6 5 6 6 6 1 6 6 6 6 5 5 6 6 6 6 6 6 6 6 6
3 4 1 2 4 5 6 3 1 2 1 5 4 6 3 2 4 3 1 6 3 2 1 5
1 4 2 1 5 6 3 1 4 1 2 3 1 5 4 2 1 6 1 3 5 2 4 1

EXAMPLE OUTPUT:
1
3
```

Problem 5: room painting

It is spring, and it is time to paint your house. You have a number of rooms to paint inside, both the walls and the ceiling. You expect you will need two coats of paint on the walls, but only one coat on the ceiling. Before starting this job, you want to evaluate how much paint you will need, so you write a program to do the calculations for you.

As input, you receive the dimensions of each room (assume it is rectangular, and that you do not need to take into account the windows and doors). For example, you may have 3 rooms to paint: one that is 2 m x 4 m x 3 m high, one that is 3 m x 5 m x 3 m high, and one that is 3 m x 4 m x 3 m high. The total surface to cover here would be 35 m^2 of ceiling and 126 m^2 of walls, for a total of 287 m^2 to cover (2 x 126 + 35). You know that typically it requires a 4 L of paint to cover 37 m^2, so here you would need 32 L of paint (31.03 exactly, but you cannot buy fractions of containers of paint).

Paint comes in 3 different formats (1 L, 4 L, and 19 L), at a cost of $24, $46, and $205 respectively. You need to figure out which containers to buy, so that you have enough paint to cover the entire surface (you may have more than what is truly needed) at the lowest possible cost. In the problem above, you figure out that you actually need 1 large container, 3 medium ones, and 1 small one (19 L + 3 * 4 L + 1 L = 32 L), for a total cost of $367. Other options (e.g., 8 * 4 L, or 2 * 19L, or 1 * 19 L + 4 * 4 L) all have a total cost that is larger.

So here is what your program should do: using the sizes of each room, it should return the total number of containers of each size that you will need. The sample input below and corresponding output matches the situation described above. Note that the first number in the input is the number of rooms. For simplicity, the dimensions will always be integers, with the last one always being the height of the room. The output should show the number of containers necessary in the following order: first the large ones, then the medium ones, and finally the small ones.

EXAMPLE INPUT:

```
3
2  4  3
3  5  3
3  4  3
```

EXAMPLE OUTPUT:

```
1  3  1
```

Problem 6: profiling

When trying to find criminals, investigators can rely on many scientific techniques. For example, if they try to identify the author of an anonymous letter, they can apply what is called "authorship profiling", to extract hints about the author based on the text written: the age, gender, level of education, etc. One simple approach under authorship profiling is to count the number of words per sentence, and the average size of those words. Indeed, kids or people with very low education level tend to write short sentences and using short words. On the other hand, people with a much higher level of educations will use more complex (and lengthy) words inside longer sentences.

For our purposes, we will be the following formula to come up with a score for a particular text: take the square of the average length of the words, and multiply this number by the number of words in the sentence. If this number is below 100, we will assume that the author is very young. If this number is between 100 and 250 (inclusively), we will assume that the person has an average education. If this number is higher than 250, we will assume that the person is well educated. Your program should analyze some sentences, and indicate the category it belongs to.

The input file starts with the number of sentences contained in the file (one per line). You should read each line, remove all punctuation marks (period and commas only), cut the sentence in words, and calculate the score as described above. For the examples below, those scores are 42.3, 371.6, and 144.5 respectively. Then, in the output, you should specify the category the corresponding sentence belongs to, based on that score. The following sample input and corresponding output is for the example above.

EXAMPLE INPUT:

```
3
The cat is black.
The mathematics behind atmospheric predictions is complex.
Tomorrow will be a great day for students.
```

EXAMPLE OUTPUT:

```
very young
well educated
average education
```

Problem 7: parking

UNB Saint John wants to modernize by displaying in real time, on an electronic panel at the entrance of each parking lot, the number of available spaces to park one's car. It is assumed that each parking lot is rectangular and a panel displays the character '0' to indicate a lane, '1' for a free space, and '2' for a space that is occupied. We have the following constraints: the entrance to the parking lot is always located at the top left corner; one cannot park in a lane; there cannot be a free space surrounded by 4 other occupied spaces.

You are asked to write a program that calculates, for a given parking lot, the number of free spaces accessible from the entrance at the top left corner. A parking space is accessible if there is a path of free adjacent cells (i.e., either lane or free space, but not occupied space) from the top left corner to that parking space. "Adjacent cells" means that they share a side, either at the top, bottom, left or right (not diagonally).

In the input, the first line contains two numbers C and R, the number of columns and the number of rows respectively. The constraints are: $5 \leq C \leq 100$ and $1 \leq R \leq 100$. Then the grid is provided, using the codes indicated above. The output should have a single integer, corresponding to the number of free accessible spaces.

EXAMPLE INPUT:

```
19 7
0000000000000000000
2002220022200222222
1002220012100221001
1002220022100121002
1001210022100121002
2002210021100122001
1002220022100122001
```

EXAMPLE OUTPUT:

```
18
```

Problem 8: summer jobs

You have a number of opportunities to get some money, by taking some short jobs. Each of those jobs has a start time and an end time, as well as the amount of money to receive if you decide to do this job. For simplicity, all of the jobs are on the same day, and their time is a whole number between 0 and 23 (i.e., time on the 24-hour clock system, for example 10 would mean 10:00 or 10am, and 14 would mean 14:00 or 2pm). Of course, you cannot take two jobs that are overlapping in time. But you are allowed to take up to 3 jobs if they are not overlapping.

You decide to write a program that will take those jobs as input, and return the set of jobs that you should pick in order to make as much money as possible. For example, let's assume that 5 jobs have been posted:

A: from 9 to 11, with a pay of $20
B: from 10 to 15, with a pay of $50
C: from 11 to 13, with a pay of $15
D: from 12 to 15, with a pay of $25
E: from 15 to 16, with a pay of $10

In this case, the best would be to take jobs B and E, for a total of $60.

The sample input file below, with the corresponding output file, matches the example above. The input file first contains the number of jobs available. Then, there is one line per job, containing the following information: the name of the job (as a single letter), the start time, the end time, and the pay (all 3 as integers). The output shows the list of jobs taken, one per line. They should be presented in the order that they appear in the input file.

EXAMPLE INPUT:

```
5
A  9  11  20
B  10  15  50
C  11  13  15
D  12  15  25
E  15  16  10
```

EXAMPLE OUTPUT:

```
B
E
```