Problem 1 – Bar Graphs

PEG is back in action for the 2013-2014 school year at Woburn (you better believe it)! Your new student leaders for this year, Alex and Ben, are giving you this diagnostic because they would like to know the level that you're at. There is only one problem — after being told by Alex to report everyone's scores on the test, Ben was a bit cheeky. He didn't just report a straight percentage that Alex could record. Instead...

...for every member who completed this test, he first recorded **two numbers**:

1. the number of points that was **earned** by the member, and
2. the number of points that was **missed** by the member.

E.g. if someone scored 80/100, Ben would record 80 and 20. Note that the original scores may not necessarily have been out of 100.

Then, because Ben was really mischievous, for certain members, he multiplied the above two values by some random factor such that the new values are integers and their ratios remained the same (hence, when recalculated as a percentage, the final score does not change). E.g. if the numbers were 50 and 50, Ben may have changed them to 17 and 17; or, if the numbers were 25 and 75, Ben may have changed them to 1 and 3.

Alex is reasonably irritated by the data that Ben reported because it is difficult to analyze visually. Being the most loyal and hard-working PEG member, you decide to write a program that converts the scores to a readable bar graph for Alex.

Input Format

The first line contains the positive integer *N* (at most 100), the number of members that completed the test. There will be *N* lines to follow. Each of these lines will contain two non-negative integers (each one at most 100), the values that Ben reported to Alex for that member. These two values will never both be zero.

Output Format

N bars of each member's score. Each bar will be 10 characters long, where asterisks '*' represent the points earned and periods '.' represent the points missed. Each bar should be rounded to the **nearest 10%** (e.g. 85% should round to 90%, while 82% should round to 80%).

Sample Input

```
5
0 1
1 9
32 68
90 10
1 0
```

Sample Output

```
..........
*.........
***.......
*********.
**********
```

Problem 2 - Ben's Quota

Since PEG had just started, Ben realized that he needed to increase his programming calibre. As we all know, solving problems on the judge is the best way to do that.

On his to-do list, there are **10** problems of varying difficulty. A certain amount of points will be awarded for solving each of these problems. He must solve the problems **in order** as they appear on his list, **without skipping over problems**. In the end, Ben wants the total points he earns to be **as close to 100 as possible** (e.g. he will aim for 97 over 105, but 101 over 98). If there are two possible final totals that are equally close to 100 (e.g. 97 and 103), he will aim to get the **larger** one (103 in this case). Once he has achieved this total, he will stop programming for the day.

Write a program that helps Ben work out the number of points he will score.

Input Format

The input will contain 10 lines. Each line contains one positive integer - the point values of the problems, in the order as they appear on Ben's to-do list. The point values will be no greater than 100 (Rubik's Cube Solver and Good Hunting Will, duh).

Output Format

Output one number, the total points he will score in the end.

Sample Input

```
3
5
7
10
15
20
20
20
30
100
```

Sample Output

```
100
```

Sample Input

```
2
3
3
5
5
7
12
20
30
40
```

Sample Output

```
87
```

Sample Input

```
15
15
15
15
15
15
15
15
15
15
```

Sample Output

```
105
```

Problem 3 - Ranklist

While Ben is working on recording everybody's marks from this test, he noticed that the ranklist of users employs a system that we're all probably familiar with, but never really studied in particular.

To rank a list of people by the amount of points they have, we first sort them by their points from greatest to least. Then, we calculate their ranks using the standard competition ranking system, as described below, for dealing with ties.

When there are ties, the tied persons receive the same ranking number, and then some ranking numbers are skipped. This system means that if multiple persons tie for a position, the position of all those below them is unaffected (i.e., a person only comes 2nd if exactly one person scores better than them, 3rd if exactly 2 people score better than them, 4th if exactly three people score better than them, etc.).

Thus during a contest, if Alex is ranked ahead of Ben and Tristan (who ties) which are both ranked ahead of Gennady, then Alex gets rank 1 ("first"), Ben gets rank 2 ("joint second"), Tristan also gets rank 2 ("joint second") and Gennady gets rank 4 ("fourth"). Hence, this system is also called the "1224" ranking system.

In addition to the system above for dealing with ties, tied competitors should be in lexicographically order. E.g. Ben should come before Tristan if both of them are tied. To check if the string `s1` comes before `s2` lexicographically, use the expression `s1 < s2` in C++/Pascal and the expression `s1.compareTo(s2) < 0` in Java.

Given a list of people's names and their point totals, you are to write a program that generates a ranking list so that Ben can use your program for ranking things in general.

Input Format

The first line contains the positive integer *N* (at most 100) — the number of people being ranked.
The next 2*N* lines will contain the data for each person. For each person's entry:

- The first line will be their name (a string consisting of at most 20 alphanumeric characters). There will be no duplicate names in the input.
- The second line will be a positive integer (at most 9999), the number of points they have.

Output Format

The output should consist of *N* lines — the ranking list you have calculated. Each line should contain the person's rank, a space, followed by the person's name.

Sample Input

```
7
trizzytrey
80
Daniel
100
Logical1111
50
jargon
40
benben829
50
HelloMello
100
Alextrovert
9999
```

Sample Output

```
1 Alextrovert
2 Daniel
2 HelloMello
4 trizzytrey
5 Logical1111
5 benben829
7 jargon
```

Problem 4 - Ben's Quota v2

Again, Ben wants to improve his skills by solving problems on the judge. His to-do list contains *N* problems, each with an associated point value that will be awarded for solving it. This time, he realized that solving problems in the order of his arbitrarily arranged to-do list may not be the best way to better his programming abilities. Every good student knows that efficient practice on their own means to not waste time doing every single problem if you already know the concept

(although it is still important to do every problem in the homework that PEG assigns).

He decides that he will start at the top of his list and move down, only solving some of the problems. That is, if he deems a problem too lame for him, **he may skip it**. By the end, he wants to gain **at least** *M* total points.

Given the list of point values for the problems, you must help Ben determine the number of ways that he can gain at least *M* points.

Input Format

The first line of input will contain the positive integers *N* (at most 10) and *M* (at most 1000).
The next line will contain *N* positive integers no greater than 100, the point values for each of the problems.

Output Format

One integer - the number of ways that he can work through the problems to gain at least *M* points.

Sample Input

```
4 20
2 3 10 15
```

Sample Output

```
5
```

Explanation

There are 4 problems on his to-do list. To earn at least 20 points, Ben has the following 5 ways:

- solve problems 3 and 4 (10 + 15 = 25)
- solve problems 2, 3 and 4 (3 + 10 + 15 = 28)
- solve problems 1, 3 and 4 (2 + 10 + 15 = 27)
- solve problems 1, 2 and 4 (2 + 3 + 15 = 20)
- solve problems 1, 2, 3 and 4 (2 + 3 + 10 + 15 = 30)