Third Annual University of Central Florida

High School Programming Tournament: Online Edition

Problems

Problem Name	**Filename**
Anti-Anti-Trolling	anti
Ghosts of Contests Past	ghosts
The Floor is Lava 2: The Floor is Still Lava	lava
Ali's Movie Collection	movie
Scrabberific!	scrabble
Sink All the Mayo	sinko
Flint's Umbrellas	umbrella
A Meme Generator	yuno

Call your program file: *filename*.c, *filename*.cpp, or *filename*.java
Call your input file: *filename*.in

For example, if you are solving Ghosts of Contests Past:
Call your program file: ghosts.c, ghosts.cpp or ghosts.java
Call your input file: ghosts.in
Call your Java class: ghosts

Anti-Anti-Trolling

Filename: `anti`

UCF Programming Team members love playing games. One game they play is hangman using the names of algorithms. Fascinated by this game, John and Travis decided to play against each other. The problem is Travis loves to troll people. In this case, that means that he will try and lose the game on purpose. John is pretty clever and eventually caught on to Travis's game. In retaliation, John would try and pick hangman puzzles where it would be difficult or even impossible to lose. After a few games Travis, who is also clever, caught on to what John was doing. Travis wants to make sure John doesn't give him a puzzle where it is impossible for him to lose. That is why he has asked you to write a program to verify games.

What is this magical game called hangman, you ask? The game starts with John selecting a word or phrase. Blank lines are then drawn to indicate the number of letters in each word. Travis then must guess letters. Each time Travis guesses a letter he must use a letter that he has not already guessed. After guessing, John will tell Travis if the letter he has guessed is in his phrase. If the letter is not in the phrase, John must draw a body part or item on the character. When a letter is guessed, all the upper case and lower case forms in the phrase get revealed. Generally, the game is lost when the character starts to look like the stick figure to the right (specifically, Travis loses immediately when he has exhausted all wrong guesses allowed). The game is won when all the letters in the phrase are guessed.

The Problem:

Given a phrase and the number of wrong guesses allowed, tell whether John is cheating.

The Input:

The input will begin with a single, positive integer, n, on a line by itself, representing the number of games you must analyze. Each game there will be a line containing an integer, w ($1 \leq w \leq 10$), and an integer, g ($1 \leq g \leq 100$), representing the number of words in the puzzle and the number of wrong guesses allowed, respectively. The following line will contain w space separated words which is the answer to the puzzle. The length of each word will be no more than 30 characters. Each word will consist of upper case and lower case letters.

The Output:

For each game output the header "`Game #p: `" where p is the number of the game (starting from 1). This will be followed by one of two phrases. Output "`Impossible to lose`" if John is cheating and there is no way to lose the game. If it is possible for Travis to reach his goal of successfully losing, output "`Trolling succeeded`" instead. Both of these phrases are printed without quotes. Print a single blank line after each game analysis.

Sample Input:

```
3
9 2
The quick brown fox jumps over the lazy dog
2 13
Fenwick Trees
3 26
Divide and Conquer
```

Sample Output:

```
Game #1: Impossible to lose

Game #2: Trolling succeeded

Game #3: Impossible to lose
```

Ghosts of Contests Past

Filename: `ghosts`

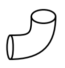
Did you participate in the HSPT: Online Edition of February 2012? If you did, you may remember that there were two problems that were not solved by anyone during the contest. Those problems were called "Eliminating Elbow Macaroni" and "Series of Tubes." In "Eliminating Elbow Macaroni," you were trying to help Pete zap the elbow macaroni from his soup (he was afraid of what might be hiding in there!) In "Series of Tubes," you had to travel through the tubes of the Internetz to help Professor Redwood's lolcatz deliver e-mail without getting stuck.

Since no one helped Pete with the nefarious elbow macaroni, his fear has grown into a fear of all things bent, including his own elbows and knees. He had to be carted off to an insane asylum before he became a danger to himself and others. Poor Pete! Worse yet, since no one helped Professor Redwood and his lolcatz, the lolcatz decided to find their own way through the tubes and got themselves stuck. This caused a huge problem on the Internetz. Remember the blackout of sites like Reddit and Wikipedia? Yep, if you'd only helped the lolcatz, you could have prevented it! Furthermore, if you'd only helped Pete, he wouldn't be stuck in the looney bin right now! It's all your fault!

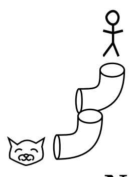
What's that? You have a plan that will solve both problems? You want to use the lolcatz and some giant elbow macaroni to find Pete and bust him out of the asylum? You think making Pete escape through a series of tubes made of giant elbow macaroni will cure his fear of all things bent? That's ridiculous! That's preposterous! That's… That's not bad, actually.

How about this, we'll only use two different elbow macaroni shapes (which you cannot rotate), like this:

```
XX           X
X    and    XX
```



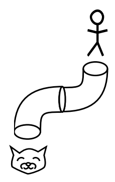
Using these shapes, we can link them together in a series of tubes to create a path from a safe location to reach Pete. To keep it simple, we'll just be lining these macaroni shapes up on the ground (no going 3D with the pasta!) Note that we can use as many of each shape as we want, but we must create a continuous path with the open ends of the elbow macaroni touching. Otherwise, Pete will either get stuck, or he'll emerge out in the open and get spotted by the orderlies. They'll drag him back inside, and we'll never see Pete again. Here are some examples to make things clear (S is the safe location, and P is where Pete is located):



```
   P
   X
  XX
  X
SXX
```

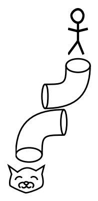
No Good

Open ends don't match up



```
   P
   X
XXXX
X
S
```

Good



```
  P
  X
 XX
XX
X
S
```

No Good

Open ends don't match up

The Problem:

Given Pete's location relative to the safe location, determine if it is possible to reach Pete using any number of elbow macaroni. Pete may be held in any number of locations around the insane asylum, so you should be able to check multiple different scenarios.

The Input:

The first line of input will contain a positive integer, *n*, on a line by itself. This will be followed by *n* scenarios. The input for each scenario will consist of a single line containing two integers, *x* and *y*, separated by a single space ($1 \le x,y \le 100$). These represent the *x* and *y* coordinates of Pete's position. The safe location is always at position (0,0). Note that each piece of giant elbow macaroni is two units long and two units wide (each character representing part of the giant elbow macaroni is one square unit).

The Output:

For each scenario, print a line containing "`Scenario #n: `" where *n* is the number of the scenario (beginning with 1). Follow this by "`Yes, we can save Pete!`" if it is possible to construct a path using any number of the given two shapes of giant elbow macaroni to reach Pete, or "`We needz more pasta!`" if it is not possible to reach Pete using only the two shapes of giant elbow macaroni. Leave one blank line after the output for each scenario.

Sample Input:

```
2
1 1
3 4
```

Sample Output:

```
Scenario #1: We needz more pasta!

Scenario #2: Yes, we can save Pete!
```

The Floor is Lava 2: The Floor is Still Lava

Filename: `lava`

It's been almost two years since the floor of Roger and Pete's home underwent some molten modifications. Though it was a painstaking process, with the help of some brilliant programmers they've finally discovered safe routes to any room in the house. It's time they struck back at the ludicrous levels of lava, and they have just the plan to do so! A rudimentary course in fluid dynamics revealed to the boys that when a toilet is flushed, it will drain the surrounding area of all lava (and other, grosser stuff)! The brothers have decided to flush away their troubles using the abundance of toilets scattered throughout the household. Though the aforementioned routes guarantee that the two can get into any room (and to any toilet) and do so without causing any lava to leave each room, Roger and Pete would like to minimize the number of toilets they have to visit in the interest of safety.

The Problem:

Given a tile-based map of the lava-filled house and the tile locations of various toilets, determine the minimum number of toilets the boys must flush to rid their home of the volcanic vexation. When a toilet is flushed, all lava that can reach that toilet will be flushed away. The lava on a tile can reach a toilet if there is a path from that tile to the toilet that does not cross a wall. Lava may flow through an area already occupied by a toilet but, oddly, lava does not start on tiles containing a toilet (the toilets simply have super suction powers when flushed!). The lava on a tile can flow into any of its four adjacent cells, but not across a diagonal.

The Input:

Input will begin with a positive integer, *t*, the number of floors in the house. Each floor will begin with a single line containging two positive integers, *r* and *c* ($3 \leq r, c \leq 50$), the numbers of rows and columns in the floor map, respectively. The next *r* lines will contain *c* characters each (one character per tile), where '`~`' denotes lava, '`T`' denotes the location of a toilet, and '`#`' represents a wall. All floors are bounded by a wall and you are guaranteed to have at least one tile with lava.

The Output:

For each floor, first output "`Floor #i: `" where *i* is the floor number (starting at 1). Immediately follow this by "`Call a plumber!`" if there is at least one lava tile that cannot reach a toilet, or "`Only need to flush` *`x`* `toilet(s).`" where *x* is the smallest number of toilets Roger and Pete will have to visit to clear the floor of lava if there is not.

Sample Input:

```
2
5 16
################
#~~T~##T~~~~~~~#
#~~###~~#~T~##~#
#~~T~#~~#~~~~#~#
################
4 7
#######
#~#~~##
#~T#~~#
#######
```

Sample Output:

```
Floor #1: Only need to flush 2 toilet(s).
Floor #2: Call a plumber!
```

Ali’s Movie Collection

Filename: `movie`

Ali is an avid movie collector. He loves all sorts of movies, from action to comedy, historical to science fiction. However, he has a small problem: Ali's nemesis, Ila, snuck into Ali's room and threw all of his prized DVDs on the ground in a scrambled mess. Now Ali wants your help to reorganize his beautiful disc collection (and he wants your solution to work for multiple uses in case Ila tries this again).

The Problem:

Ali has some special requirements for you while you help him put his collection back together, they are as follows:

The DVDs should be placed on the shelf from left to right in standard lexicographical order. For this problem, two strings are sorted in lexicographical order if, for the first character which they differ, the string with the character that comes earlier in the alphabet, or is a white-space character, comes first. The case of the letters in the title does not matter.

If a movie begins with the word "The", its title when sorting should omit this word. For example, "The Gladiator" will be sorted as "Gladiator".

Some movies are sequels, but the numbering for sequels is inconsistent. Sometimes the number of the sequel is given as an Arabic numeral, and other times it is given as a word. Ali wants you to make sure that any titles that end with a number are in their proper order in their series. If there is no number, you may assume that the number is one. For example, if Ali owns "Terminator", "Terminator Two", and "Terminator 3", they should be listed in the order shown here.

The Input:

The first line of input contains a positive integer, t ($t < 100$), representing the number of collections to re-sort. On the first line of each collection is a positive integer, n ($n \leq 100$), the number of DVD's in this instance of Ali's collection. Each of the following n lines contains the title of a DVD. This string will optionally begin with the word "The" (not necessarily the same case) followed by a space, then 1 or more space separated words consisting of only letters, denoting the movie title, and finally will end with an optional space and number, given as either an Arabic numeral or a word. It is guaranteed that this number, if present, will be between one and ten inclusive. No line will contain more than 50 characters. You may assume that Ali does not own any duplicate movies, and that each movie has at least one non-numeral word in its name.

The Output:

For each collection, output on a separate line the header “`Movie Collection #`*`i`*`:`” where *i* is the number of the collection (starting with 1). Then, output *n* lines, one DVD title on each line. The lines should be in the same order as on Ali's shelf, with the first line as the left most DVD. Separate each test case with a blank line.

Sample Input:

```
2
5
Land before time 2
The Land before time
The Land before tim 6
The Land Before Time three
land before time 4
3
Terminator 2
Harry Potter 5
The Illusionist
```

Sample Output:

```
Movie Collection #1:
The Land before tim 6
The Land before time
Land before time 2
The Land Before Time three
land before time 4

Movie Collection #2:
Harry Potter 5
The Illusionist
Terminator 2
```

Scrabberific!

Filename: `scrabble`

Dr. Orooji's sons, Mack and Zack are at it again. They've now developed a friendly competition in the game of Scrabble. Recently, however, Mack has been consistently defeating Zack. But Zack has gotten an idea! He's hung around the programming team long enough to know that they could probably write a program to help him. Zack has enticed you into writing a program for him by allowing you to use his DS for a couple hours. (As a poor high school student, you have no money for such a gaming system, so you've taken the bait, naturally.)

In order to solve the problem, you need a list of the value of each letter in Scrabble, which is included in the chart below:

Letter	Value	Letter	Value	Letter	Value	Letter	Value
A	1	H	4	O	1	V	4
B	3	I	1	P	3	W	4
C	3	J	8	Q	10	X	8
D	2	K	5	R	1	Y	4
E	1	L	1	S	1	Z	10
F	4	M	3	T	1		
G	2	N	1	U	1		

The Problem:

Your job is to write a program that takes in Zack's tiles, and determines the word that can be formed with those letters (without using any letters on the board) of maximum score (excluding any special scoring tiles). To simplify the problem, no input cases with a blank tile will be presented. The score of any word will simply be the sum of the scores of the tiles that form the word. If there are multiple words which produce the same score, the word that comes first alphabetically of all of the maximum score words should be selected.

(Problem continues on next page)

The Input:

The first line of the input file contains a single positive integer, *n (n < 100,000)*, representing the number of words in the dictionary. The following *n* lines will contain one of these words each, in alphabetical order. Each word will contain uppercase letters only and no word will exceed 7 letters.

Following the words, the next line of input will contain a single positive integer, *r*, representing the number of racks of tiles Zack would like you to evaluate. Each of the following *r* lines will contain one rack of tiles to evaluate. These lines will each contain a string of seven uppercase letters repsenting the rack of tiles. Each string will only contain a set of letters that are possible to receive in the game of Scrabble. (For example, there is only one tile with a Z on it in Scrabble. Thus, no input rack will contain more than one Z.) Also, each input rack will have the ability to form at least one word from the list of valid words.

The Output:

For each rack of tiles, output a single line with the following format:

`Tile Rack #`*`k`*`: Playing `*`w`*` earns `*`p`*` points, the max possible.`

where *k* represents the number of the tile rack, starting at 1, *w* is the word that comes first alphabetically of all the words that provide the maximum score, and *p* is the maximum score for the rack.

Print a blank line after the output for each rack.

Sample Input:

```
5
ABACUS
DATA
DISK
DRIVE
SERVER
2
TAADSIK
VEERRST
```

Sample Output:

```
Tile Rack #1: Playing DISK earns 9 points, the max possible.

Tile Rack #2: Playing SERVER earns 9 points, the max possible.
```

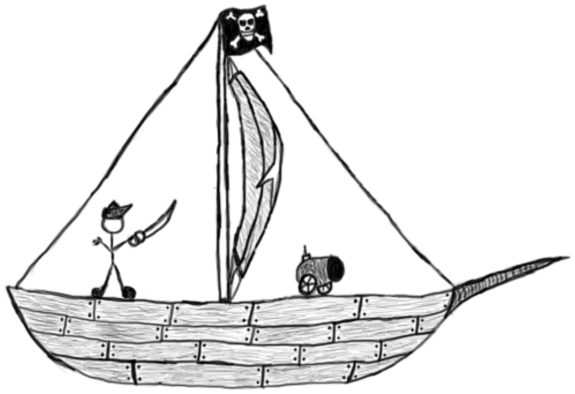
Sink All the Mayo

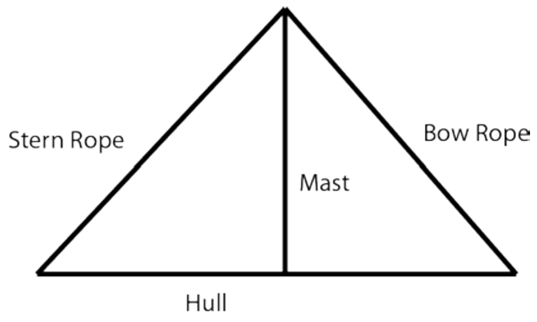
Filename: `sinko`

Captain Ali is very excited, because his favorite holiday, Cinco de Mayo, is coming soon. On this day, he builds a special ship so that he may participate in the famous sport, Sink all the Mayo, where each participating captain takes his ship, fills the hull with mayonnaise, and does their best to sink the ships of all the opposing captains. Captain Ali is certainly going to win this year, because he is designing his new cannon, the Circlotron 9000. Unfortunately, designing this ultimate mayo sinking cannon requires his full attention, so he needs you to finish ordering the parts for his ship and creating the assembly instructions for his crew.

The Problem:

As you know, a battleship consists of four main parts, the hull, the bow and stern ropes, and the mast. Luckily for you, Captain Ali already has the bow and stern ropes and the hull, and needs you to order the mast and tell his crew where to place it. You know that the bow rope must connect the front of the ship to the top of the mast without slack, and the stern rope connects the back of the ship to the top of the mast without slack. You must determine the height of the mast you need to order, as well as its position in relation to the front and back of the ship (note that it is acceptable if the mast is on an edge of the ship). In the event that there is no mast of positive length which satisfies this arrangement, you should tell Captain Ali the bad news.





The Input:

The input will begin with a single positive integer, *n*, on a line by itself, representing the number of ships to build. Each ship description will consist of three numbers: *d*, the distance from the bow to the stern of the hull, *b*, the length of the bow rope, and, *s*, the length of the stern rope. All input values are positive integers no greater than 1,000.

The Output:

For each ship output the "`Ship #`*`q`*`: `" where *q* is the number of the current ship (beginning with 1). If a mast of length greater than zero cannot be placed on the ship with the given part lengths, print "`Ali is sunk!`" Otherwise, three numbers should be printed with a single space between them, each rounded and printed to two decimal places. The first number should be the length of the mast. The second number should be the distance between the front of the ship and the bottom of the mast. The third number should be the distance between the back of the ship and the bottom of the mast.

Sample Input:

```
3
5 3 4
5 4 3
7 1 1
```

Sample Output:

```
Ship #1: 2.40 1.80 3.20
Ship #2: 2.40 3.20 1.80
Ship #3: Ali is sunk!
```

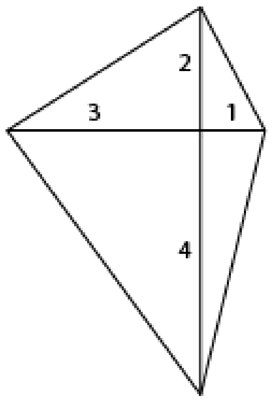
Flint's Umbrellas

Filename: `umbrella`

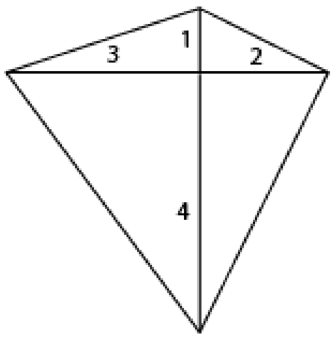
Flint Lockwood is from a small island underneath the "A" in "Atlantic" and he loves to invent things! You may have heard of some of his inventions such as Spray-on Shoes, Hair Unbalder and the Monkey Thought Translator. The latter allows him to communicate with his best friend and trusted colleague, Steve the Monkey. His most recent invention turned water into food and resulted in an unfortunate result after it went flying into the clouds and then got out of hand; however, Flint is determined to continue inventing! One thing his most recent glitch taught him was that he should always be prepared. Unfortunately, he has lost his only umbrella and he does not want to take any additional risks by walking around without one.

You told Flint, that he shouldn't invent anything dealing with atmospheric effects to begin with, but he won't listen, as usual. In order to support multiple experimental trials, he has gathered multiple sets of metal rods and some fabric, and he is determined to make some amazing umbrellas! However, he has run into another problem. He wants to construct each umbrella with the largest area for blocking rain, and he has no clue what that largest area is.

For example, given four rods, one each of length 1, 2, 3 and 4, the figure below shows two possible umbrellas that might be made (since there are four rods, the angle between each pair is 90 degrees). Umbrella A has a total area of 12 while Umbrella B has a total area of 12.5 (and note that Umbrella B is the largest area that may be made in this case).



Umbrella A



Umbrella B

Lockwood knows you're a good ~~programmer~~ friend so he wants you to write a program that will determine the best possible area for each new umbrella.

The Problem:

Determine the maximal area that can be covered by an umbrella using the metal rods given. To make the umbrella look nice all of the rods will be coplanar in the umbrella. To make the umbrella sturdy each pair of adjacent rods in the umbrella will have the same angle between them as any other adjacent pair, and finally Lockwood does not want to be wasteful, so he will use all the rods.

The Input:

The first line of input will contain a single integer, *t*. Following this will be *t* umbrellas. Each umbrella will begin with a positive integer, *n* ($3 \le n \le 8$). The next line will contain *n* positive integers separated by spaces l_1 l_2 … l_n, such that $1 \le l_i \le 10^3$ for all *i* between 1 and *n*.

The Output:

For each umbrella output a single line in the form of "`Umbrella #`*`k`*`: `*`a`*" where *k* is the number of the umbrella, starting with 1, and *a* is the largest possible blocking area of the umbrella rounded to 3 decimal places.

Sample Input:

```
2
3
1 1 1
4
1 4 2 3
```

Sample Output:

```
Umbrella #1: 1.299
Umbrella #2: 12.500
```

A Meme Generator

Filename: `yuno`

The Problem:

Generate "Y U NO" memes.

The Input:

The first line of input contains a positive integer, which is the number of memes to generate using the rest of the input. Data for each meme occupies 2 lines, starting on the next line. The first line is a string representing an *object of frustration*. The second line is a string containing the *expected action* for that object of frustration. Each line will contain at least one character and no more than 80 characters. There are no blank lines in the input.

The Output:

Output each meme on 2 lines as follows. On the first line, print the *object of frustration* followed by an ellipsis ("`...`"). On the second line, print "`Y U NO`" followed by the *expected action*, followed by a question mark. Capitalize all letters in the output. Print a blank line after the output for each meme.

Sample Input:

```
2
wi-fi
free at Starbucks
this program
compile on first try
```

Sample Output:

```
WI-FI...
Y U NO FREE AT STARBUCKS?

THIS PROGRAM...
Y U NO COMPILE ON FIRST TRY?
```