Fifteenth Annual
University of Central Florida

ACM · UPE

High School Programming Tournament

Problems

Problem Name	Filename
Canadian Curling	CURLING
Genetic Algorithms	GENETIC
The Turpentine Avenger	AVENGER
Jumping Monks	MONKS
Forgetful Family	FAMILY
Hope Chest Gone Bad	HOPE
Nihongo No AI (Japanese Language AI)	NIHONGO
Bumper Planes	BUMPER
Combination Lock	COMBO
All Your Base Are Belong To Us!	BASE

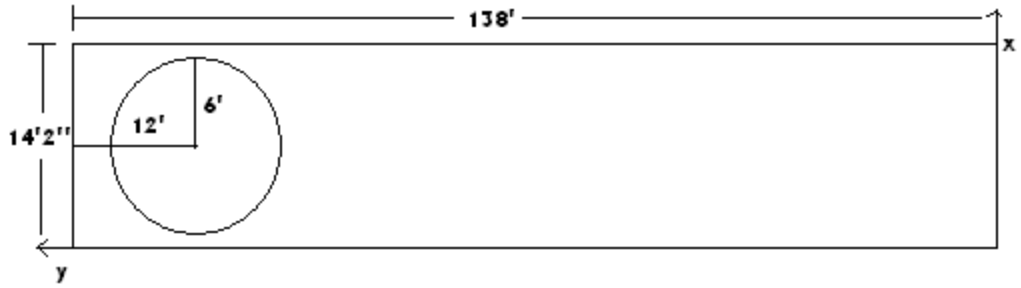
Call your program file: *filename*.pas, *filename*.c, *filename*.cpp, or *filename*.java
Call your input file: *filename*.in

For example, if you are solving Jumping Monks:
Call your program file: monks.pas, monks.c, monks.cpp or monks.java
Call your input file: monks.in

Canadian Curling

Filename: CURLING

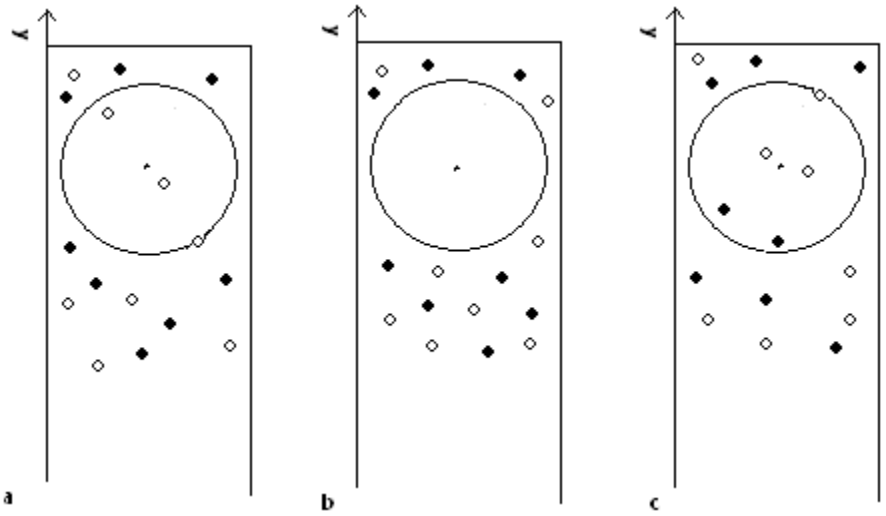
In Canada, the sport of Curling is very popular. Curling is played on a long, thin strip of ice, called a "sheet." The sheet is 138 feet long, and 14 feet, 2 inches wide. At one side is a circle, 12 feet in diameter, called the "house." The center of the house is called the "button," and it is centered in the sheet, 12 feet from the end line. The button is effectively a point for the purposes of this problem. There are other markings on the ice, but they won't affect this problem. There are two teams: White and Black.



The game consists of 8 to 10 frames, called "ends." In each end, the teams take turns throwing "rocks" down the ice. A "rock" is a circular piece of granite, 1 foot in diameter with a handle on the top, weighing 44 pounds. Each team throws 8 rocks per end.

The shooter slides the rock from one end of the sheet to the other, kneeling and sliding along with the rock. He or she can also give the rock a slight spin, which will cause it to curve, much like a curveball in baseball, but in slow motion. At the shooter's command, his teammates can use brooms to sweep the ice, to affect the course of the rock.

At the conclusion of an end, the score is tallied. Only one team can score in an end, and that's the team that has placed a rock closest to the button. They score one point for every rock they have either in the house OR touching the house, and which is closer to the button than any of the opponent's rocks. Two points are considered the same if they are within 0.01 inches of each other. If the closest rock of each team is the same distance from the button as the other, this is a tie and neither team scores. Note that it is possible for no rocks to be in the house or touching the house, in which case neither team scores.



For example, in the situation presented in diagram *a* above, White has 3 rocks in the house and Black has none, so White scores 3 points. In diagram *b*, neither team has rocks in the house, so neither team scores. In diagram *c*, White has 2 rocks closer to the button than Black's closest rock, so White scores 2 points.

The Problem:

Your job is to compute the score of an end, given the positions of the 16 rocks.

The Input:

The first line of input will contain a positive integer *n*, indicating the number of ends in the input. For each end, there will be 2 lines of 16 integers each. The 16 integers represent the (x, y) positions of the centers of the 8 rocks for each team, with White on the first line and Black on the second. All distances will be given in inches. The first coordinate will be the distance from the left edge of the sheet, and the second will be the distance down the sheet from the shooter. Note that it is possible for rocks to end up out of the field of play. These will be represented by (0, 0).

The Output:

For each end, print one of the following messages, whichever is appropriate:

```
White scores n
Black scores n
Neither team scores
```

Sample Input:

```
1
130 500 85 1490 0 0 100 600 0 0 0 0 60 800 0 0
0 0 100 200 50 350 75 400 0 0 50 250 0 0 75 1400
```

Sample Output:

```
White scores 1
```

Genetic Algorithms

Filename: `GENETIC`

An area of Computer Science that is beginning to get some recognition is that of Genetic Algorithms (GAs). GAs represent potential solutions to a problem as genes, and then "breed" individuals to try to come up with an acceptable solution to a difficult problem. GAs are very useful when the search space is very large and/or too complex for traditional calculus-based search algorithms.

GAs breed individuals by exchanging information between their genes, and attempting to weed out bad answers. The actual "breeding" step is carried out by a set of genetic operators, many inspired by observed actions in real genes.

The Problem:

Given a gene represented as a string consisting only of zeros and ones, give the result of mutation on the gene. Although there are many genetic operators (some useful on only one problem), you will only consider mutation. Mutation "flips" a single bit, changing a 0 to a 1 or a 1 to a 0.

The Input:

Input will begin with a single line containing an integer n, representing the number of cases in the file. Each case will consist of two lines. The first line will contain a single integer b ($1 \le b \le 25$), the bit to be mutated. The second line will be the gene to be mutated. Genes will be a variable length, no longer than 25. No input parameters will be illegal (ie, if the length of the gene is 10, you will not be asked to mutate the 13^{th} bit).

The Output:

Output will consist of the mutated genes, with a blank line between each case.

Sample Input:

```
3
5
00000000000
9
111111111111111
1
0101010010101
```

Sample Output:

```
00001000000

111111110111111
```

```
1101010010101
```

The Turpentine Avenger

Filename: AVENGER

Of all the sucky powers to be born with, The Turpentine Avenger has one of the worst: he can generate turpentine from his fingers at will, up to 100 gallons per day. He found that this didn't stop criminals nearly as well as he hoped, so he gave up crime-fighting and became a door-to-door Used Turpentine Salesman. But one day, when he walked up the front porch of an unassuming suburban house and rang the doorbell, the door was answered by none other than Paint Man! Paint Man was his arch-nemesis, quite by default: Paint Man was the only supervillain that The Turpentine Avenger had any chance of defeating.

When Paint Man saw who was at the door, he screamed and slammed it shut. Yet The Turpentine Avenger was not to be defeated so easily. He opened the door (since Paint Man had forgotten to lock it) and began a search for his hideous adversary. This turned out to be difficult. Paint Man's house was completely coated in paint, and Paint Man himself looked just like a blob of paint, so there was no way to know where Paint Man was. He decided to just blast the whole house, or as much as he could, in the hopes that he would stumble upon the fiend.

The Problem:

Your job is to find the maximum number of rooms that The Turpentine Avenger can clear. He has 100 gallons of turpentine, and each gallon will clean 50 square feet of paint.

The Input:

The first input line will contain a positive integer *n*, indicating the number of rooms in Paint Man's house. Each of the next *n* input lines will contain a single positive integer representing the square footage of that particular room. There will be multiple data sets. End of input will be represented by a value of 0 for *n*.

The Output:

For each data set, print the maximum number of rooms that The Turpentine Avenger can clear. Leave a blank line after the output for each data set. Only completely cleared rooms will count. For instance, a data set resulting in 2 completely cleared rooms and 1 partially cleared room will output 2.

Sample Input:

```
3
3000
500
2000
0
```

Sample Output:

```
2
```

Jumping Monks

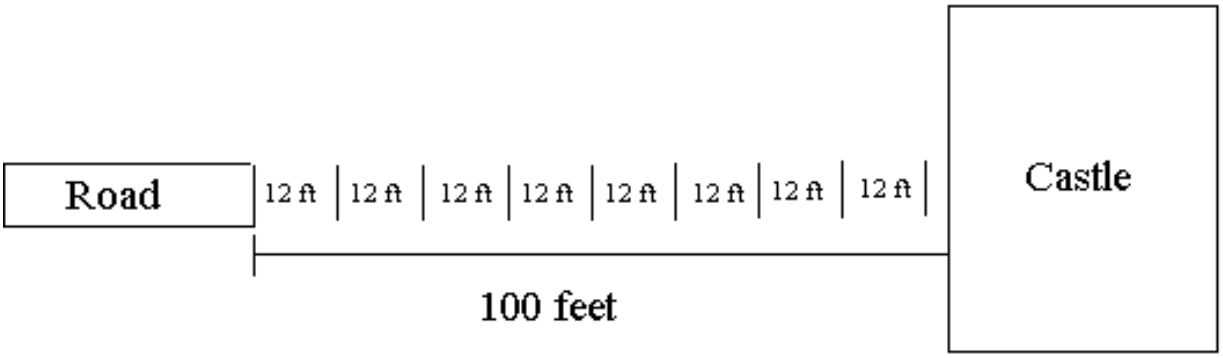
Filename: MONKS

You have awoken from a deep sleep to discover that you have been mysteriously transported back in time to the days of ancient China. Your strange behaviors, mannerisms, and experiences cause you to become cast out from Chinese society. Luckily, a traveling acrobatics troupe takes you in.

But all is not as it seems! After days of hard work learning the acrobatic ways of your new friends, you discover that they are not acrobats at all. They are incredibly powerful warrior monks who follow Ti-Tsang Wang, the Chinese god of mercy. In accordance to their god's merciful teachings, the monks defeat evil not through violence, death, and destruction, but by confusing evil with acrobatic displays, magical shows, and other athletic feats. The evil is then put to sleep with magic powder and confined to a box.

You question them about your discovery of their true identity. Your friends tell you that they are willing to let you join their group, providing that you help them in their latest mission.

They have been sent to this part of China to rescue a beautiful maiden from an evil demon prince. The clever fiend has placed movable fences between his castle and the road that the monks will have to jump over in order to rescue the maiden. As you can imagine, these great leaps must be well planned in advance! Your friends want you to use your superior intellectual prowess to calculate the number of leaps needed. The only information that they have is the spacing of the walls, and the total distance from the road to the castle.



The Problem:

Your task is to use the spacing information, and the total distance from the first wall to the castle to calculate the total number of walls that the monks will have to jump over. The spacing number that the monks give you is a whole number. They explain that the number represents the distance from the road to the first wall, and the distance from a wall to its neighboring walls. The demon prince likes the number one hundred, so the total distance between the road and the castle is 100 feet. Once the monks clear the final wall before the castle, they do not need to jump again in the short distance to the castle.

The Input:

The first line contains an integer *n*. The next *n* lines contain an integer that represents the distance between the road and the first wall.

The Output:

A whole number representing the number of jumps that the monks have to make.

Sample Input:

```
3
12
3
4
```

Sample Output:

```
8
33
25
```

Forgetful Family

Filename: FAMILY

You have a friend named Maurice who has many siblings. Often times, people ask Maurice how old his brothers and sisters are, but he can't remember their ages. Instead, he remembers how old they are in relation to his age, and does a quick computation in his head to get the answer.

The Problem:

Your task is to write a program to help people like Maurice, so they can put the program on their PDA, and not have to use their heads anymore.

The Input:

The input contains one or more sets (families). The first line of each set contains only a non-negative integer *F* less than or equal to 20. This integer indicates the number of people in the family of this set. A value of zero for *F* indicates end-of-data, and this case should not be processed. Following this line are *F* line(s) describing the people in the family and their ages (this is the family member description section). Each line in this section will be in one of three forms:

1. `name1 name2+n`
2. `name1 name2-n`
3. `name1 n`

Where *name1* and *name2* are strings (with 10 or fewer alphabetic characters each), and *n* is a positive integer. Form 1 indicates that *name1*'s age is calculated by adding *n* to *name2*'s age, Form 2 indicates that *name1*'s age is calculated by subtracting *n* from *name2*'s age, and Form 3 indicates that *name1*'s age is *n*. A name will only appear as a *name1* once in this section, and every *name2* will appear as a *name1* in this section. All ages, when calculated according to these rules, are positive integers less than or equal to 100.

Following these *F* lines is a line containing only a non-negative integer *R*. Following this line, there are *R* lines of age requests. An age request is a string of 10 or fewer alphabetic characters. This string will be one of the names that appeared as a *name1* in the family member description section. Case is important.

The Output:

For each age request in the input, you will calculate the age of the person by the requested name. This number is calculated to be consistent with the information given in the family member description section, according to the rules discussed above.

For a requested name *name* and calculated age *n*, print the message:

name is *n* years old.

Or, if *n* is 1,

name is 1 year old.

Print a blank line in the output after processing each data set in the input.

Sample Input:

```
1
UCFHSCntst 15
1
UCFHSCntst
1
ThirdMlnm 1
1
ThirdMlnm
3
Richard 20
Nancy Richard-6
Jay Nancy-2
3
Nancy
Jay
Richard
0
```

Sample Output:

```
UCFHSCntst is 15 years old.

ThirdMlnm is 1 year old.

Nancy is 14 years old.
Jay is 12 years old.
Richard is 20 years old.
```

Hope Chest Gone Bad

Filename: HOPE

While you were cleaning out your attic, you came across a strange brown box. It was locked, so you spent the rest of the day trying to get it open. Finally, you managed to break it open, and out fell a pack of old love letters, some jewelry, class rings, photographs, and disturbing underwear. You've accidentally stumbled upon your parent's box of high school and college memorabilia! They would be embarrassed and annoyed if they found out that you read their torrid love letters (even though you didn't, of course - ugh).

You need to repair this box and get all the stuff back in it! But your prying has ruined the hinges. What to do, what to do? You have only ten minutes before the hardware store closes, and your parents are too cheap to get you a car, so you grab your skateboard and hitch a ride into town holding onto the back of passing vehicles, like that guy did in *Back To The Future*. You reach the hardware store with one minute left to go, and unable to decide which hinge to buy, you quickly grab several of every size hinge they have.

When you get home, you realize that you still don't know which hinges to use. It's not as easy as it sounds because drunken carpenters made some of these hope chests with a few too many screw holes. This makes your life a little harder as you'll need a program to figure out the ones to use.

The Problem:

Given that you have two hinges of every size possibly needed, pick the two that when combined cover the largest portion of the back of the hope chest. In the event of a tie, select the pair of hinges that are farthest apart on the back of the hope chest, for added stability. If you have more than one set of hinges that fit these criteria, choose the set with the largest single hinge in it. Hinges cannot share a single hole and because of their shoddy design cannot cover unused screw holes. Given the layout of the screw holes on the back of each hope chest, repair the hope chest and report what size hinges had to be used to fix it.

Each line will have an ASCII description of the back of the hope chest. For example:

```
=======o==o======o======o=====
```

Where '=' are where the top and bottom of the box fit together, and the 'o' is a screw hole where a hinge once went. In the example above, you need a 2-space hinge and a 6-space hinge to fix the chest.

```
======o==o==o===o===o====o====
```

In the example above, several of the screw-holes are extra and unused. Since you want to use the biggest hinges that will fit, you would use a 4-space hinge and a 3-space hinge, leaving the two holes on the far left empty and uncovered.

For the given box description, decide the largest two hinges that can be used.

The Input:

The first line of the input is a non-negative integer, *N*, representing the number of hope chests in the input. For each hope chest, there is a line consisting of 30 characters, either '`=`' or '`o`', where '`=`' is the crease between the lid and the bottom and '`o`' is a screw hole for a hinge.

The Output:

The output for each hope chest will be a single line, consisting of "`Hope Chest #`*N*`: `*X* *Y*", where *N* is the number of the hope chest being fixed and *X* and *Y* represent the size of the two hinges used, *X* being the larger of the two hinges, and *Y* being the smaller.

Sample Input:

```
5
=======o==o======o======o=====
======o==o==o===o===o====o====
====o====o==========o====o====
===o=o=====o=======oo======o==
=======o=====o=======o=o======
```

Sample Output:

```
Hope Chest #1: 6 2
Hope Chest #2: 4 3
Hope Chest #3: 4 4
Hope Chest #4: 7 6
Hope Chest #5: 5 1
```

Nihongo no AI (Japanese Language AI)

Filename: NIHONGO

What a lucky programmer you are! You've just been hired by CircleSoft, a Japanese game company best known for the *Ultimate Fantasy* line of games, and assigned to work on a top-secret project that will revolutionize console gaming. This project, codenamed "Cid", is so secret that you don't even know what it is. All you know is that every morning you will be given a task by your superior, and that task must be completed by day's end. You show up to work on your first day, and are given a sheet of paper that says:

The Problem:

Your program will read in "yes or no" questions in the Japanese language, and answer each one in the affirmative.

Since you are one of the several hundred Americans that haven't learned Japanese yet, it will be helpful for you to know that Japanese sentences of this form always end with a verb and *ka* (a Japanese language sentence particle which indicates a question). For example, in the sentence *umi de oyogimashita ka* (meaning "did you swim at the ocean?"), *oyogimashita* is the verb.

For purposes of this problem there is only one exception to this rule, the copula *desu*. When the next-to-last word in the sentence is *desu*, the "verb" of the sentence consists of *desu* and the word immediately before it. For instance, in the sentence *ogenki desu ka* ("are you doing well?"), the "verb" of the sentence is *ogenki desu*. The word *desu* will never be immediately preceded by *desu*. Furthermore, it will always appear in the input file in all lower-case characters.

For purposes of this problem, a word is a sequence of one or more alphabetic characters terminated by a space or new-line character.

The Input:

The input file will consist of multiple lines. Each line, except for the last, will contain a "yes or no" question in the Japanese language. These lines will contain 70 or fewer characters, which are alphabet symbols or spaces. Words are separated by one space. The first word of a sentence starts in the first column of the line.

The last line of data in the input file will consist of only the word `yame`, in lower-case characters and starting in column 1. Do not process this line. (*Yame* means "finish" in Japanese.) The 4-character sequence `yame` will not occur anywhere else in the input.

The Output:

For each line of input, except for the last, extract the verb from the sentence according to the rules given above, and produce output in the following form:

```
Hai, verb
```

Print the *verb* portion exactly as it appeared in the input. Note that there is one space between the comma and the *verb*.

Sample Input:

```
Umi de oyogimashita ka
Ogenki desu ka
yame
```

Sample Output:

```
Hai, oyogimashita
Hai, Ogenki desu
```

Bumper Planes

Filename: BUMPER

The lovely country of Kent has a problem. Its military airplanes keep wandering too close to other airplanes, mountains, and whatnot. They have asked a large military contractor, General AeroDyne, to create a system that can keep the military aircraft a safe distance from objects that might damage them.

The hard working brilliant physicists at General AeroDyne quickly created a marvelous device. The Close Encounter device will be capable of keeping an airplane a safe distance away from all sorts of objects.

Well, it will as soon as the software is done.

Unfortunately, all of General AeroDyne's programmers were lured away to the land of Fiji by unscrupulous Dot Com recruiters. Now that Dot Com is Dot Dead, they are stranded there, and General AeroDyne has no one who can write the software.

That is, no one but you.

The Problem:

The amazing Close Encounter hardware monitors the military plane's surroundings and converts all threatening objects into an X,Y location and a radius on a Cartesian grid. All of the input numbers are given in integers. All X,Y values will be in the range of –100 to 100. The radii will be in the range of 0 to 100. The military plane is always at location 0,0 and its radius is always 10. Your task is to calculate whether any of the threatening circles touch or intersect the military airplane's circle.

The Input:

The first line of the input is an integer that gives the number of threatening objects, N. There are then N lines, each of which contains three integers. The numbers are the X, Y, and radius of the threatening object.

The Output:

The output is N lines of text, one for every input case. If the input circle touches the military airplane's circle print the string "`Danger`". Otherwise, print "`Ok`".

Sample Input:

```
5
20 20 10
19 5 5
40 70 1
100 40 25
20 0 10
```

Sample Output:

```
Ok
Ok
Ok
Ok
Danger
```

Combination Lock

Filename: `COMBO`

Herb has forgotten the combination of his locker at school. He desperately needs to get the mp3 player in his locker because Napster is being shut down tomorrow. He decides to try every combination he can until he opens the locker. Unfortunately the combination locks are old and rusty so each number will only move 3 numbers at a time, so getting from 4 to 5 requires 7 steps – from 4 to 7 to 0 to 3 to 6 to 9 to 2 to 5. On the other hand, because the locks are so pitiful, you can hear a click when a digit is found correctly, so when you change a number to the correct digit and pull, a click can be heard. Herb is a very linear guy, so he always starts with the leftmost digit and moves to the next digit on the right only after he hears a click.

The Problem:

The school combination locks consist of digits from 0 to 9. The maximum number of digits in the combination is nine. Given the starting combination, output a series of combinations that result in the correct combination being reached and the locker being open. Starting from left going right, rotate each digit until it is correct and a click can be heard when pulling on the lock. Do this for each of the digits until you can report that the lock is open and how many steps it took.

The Input:

Each data set will consist of three lines. The first line is a number, *N*, between 0 and 9 representing the number of digits that make up the combination on the lock. The second line is a series of *N* numbers between 0 and 9 representing the combination at which the lock is currently set. The third line is also a series of *N* numbers between 0 and 9 that represent the correct combination for this particular lock. Every digit in the starting combination will have to be changed. The data ends when *N* for a combination lock is zero.

The Output:

For each combination lock, the output should start with a line indicating the lock number. The next line will be the starting combination for the lock given in the input. Each line after that should show the next combination after rotating one digit to the next possible number, and a "`-click`" after the combination if that number matches in the final combination. After all numbers are found correctly, print out a line that shows how many different combinations had to be attempted before the lock was opened. Each lock should be separated from the previous one by a single empty line. See the Sample Output.

Sample Input:

```
2
0 1
4 6
5
1 2 3 4 5
4 5 2 6 2
0
```

Sample Output:

```
Lock #1
0 1
3 1
6 1
9 1
2 1
5 1
8 1
1 1
4 1 - click
4 4
4 7
4 0
4 3
4 6 - click
Locker opened in 13 steps

Lock #2
1 2 3 4 5
4 2 3 4 5 - click
4 5 3 4 5 - click
4 5 6 4 5
4 5 9 4 5
4 5 2 4 5 - click
4 5 2 7 5
4 5 2 0 5
4 5 2 3 5
4 5 2 6 5 - click
4 5 2 6 8
4 5 2 6 1
4 5 2 6 4
4 5 2 6 7
4 5 2 6 0
4 5 2 6 3
4 5 2 6 6
4 5 2 6 9
4 5 2 6 2 - click
Locker opened in 18 steps
```

All Your Base are Belong to Us!

Filename: BASE

We are at war with the evil Zig Empire. General Ali is trying to formulate an attack plan that will hopefully win the war for us. Fortunately, we've just intercepted a transmission from the Zig commander:

> "How are you gentlemen. The enemy is win too many battles up to now. We move now all the main base to new coordinate to keep enemy from finding at where are we. From new place, we find some enemy base and set them up the bomb. This hope make us win the war, if enemy no find any our base. To stay hide, I send you list of where to put all your base. Go move base! For great justice!"

A list of coordinates followed this message. General Ali has asked you to write a program that will simulate a bombardment of the target area mentioned in the Zig message. This will enable the general and his officers to plan a successful attack.

The Problem:

After translating the cryptic Zig language, our intelligence staff has determined that the target area given by the signal can be divided into a square grid. Given a list of bases occupying that grid and a list of bombing targets on the grid, determine which bases will not be destroyed by the bombs. The area of effect of the bombs is a single grid square, so if two bases are next to each other on the grid and a bomb is dropped on one of the bases, the other base is not destroyed.

The Input:

There will be multiple data sets. Each set will begin with a single non-negative integer *N* on a line by itself, representing the number of bases. On the next *N* lines will be two non-negative integers separated by a space, representing the X and Y coordinates of a square occupied by an enemy base. Following this will be a single non-negative integer *M* on a line by itself, representing the number of bombs dropped. On each of the next *M* lines will be two non-negative integers separated by a space, representing the X and Y coordinates of a square that is bombed. If a base occupies the given square, the base is destroyed. Input is terminated by a negative value for *N*.

The Output:

After removing all bases destroyed by bombs, print the bases that remain standing. For each remaining base, print "`We missed the base at (`*X*`, `*Y*`)`" where *X* and *Y* represent the coordinates of the remaining base. Print the remaining bases in ascending order by Y coordinate, followed by X coordinate (so (2, 3) would come before (1, 4), but after (1, 3)). If there are no bases remaining, print "`All your base are belong to us!`" Leave one blank line after the output for each data set.

Sample Input:

```
5
2 3
1 3
3 2
5 5
1 4
2
5 5
3 2
1
1 1
1
1 1
-1
```

Sample Output:

```
We missed the base at (1, 3)
We missed the base at (2, 3)
We missed the base at (1, 4)

All your base are belong to us!
```