SPRING 2016

PROGRAMMING CONTEST

Overview

Welcome to the Spring 2016 ACM Programming Competition! This is a contest wherein you compete amongst your fellow students, and professors, in an effort to solve a set of programming problems as fast as possible.

Problems

There are nine problems of (roughly) increasing difficulty. The team who solves the most problems is declared the winner. In the event of a tie, the total time is used to determine a winner. Total time is the time of a team's last successful submission, with a 20 minute deduction for each failed submission. Each problem has a general description, details on the formats for input and output, and example input and output.

Input is done from standard input and output is done to standard output. *Do not open any files inside of your program.* Output of your program must match that of the correct output *exactly*. You can solve the problems in C, C++, Haskell, Java, PHP, Python, Racket, Ruby, or R. Up to date versions of these languages will be used to judge your entries. You should save your code using the standard extension for your language of choice (.c .cpp .hs .java .php .py .rkt .rb .R respectively).

At the end of your packet, there is also a brief overview of how to perform input and output in C++, Python and Java in case you need a refresher.

Rules

You are allowed to have any printed material such as books or printouts. You are also allowed to access standard library documentation for your language of choice. *No other use of the internet is permitted.* You are also not allowed to copy and paste code from any source be it a thumb drive, or another file that you have. Only one computer is allowed to a team.

Logging In

To log in to the system, direct your browser to `http://bit.ly/1NKMdFr`. You will then need to enter your team name and password which are provided for you. This will bring up the interface for you to submit solutions to the problems, ask questions (which will be broadcast to all participants), and see how your team is doing.

Submitting

To submit a solution, first make sure that you select the correct problem letter on the top of the site. Then click the "Choose File" button. Then choose the file that contains your source code (be sure it has the correct extension). After that it will be submitted for judging. It will receive an automatic judging response which is marked pending. Your solution will then be checked manually and a final response will be assigned.

DO NOT TURN OVER THIS SHEET UNTIL THE CONTEST STARTS.
MAY THE FORCE BE WITH YOU!

Problem A: Shield Testing

The Star Destroyers used by the Empire are large space ships which transport many officers and soldiers, and patrol the galaxy. Like other large ships, star Destroyers have powerful shield generators which protect the ship from most laser attacks.

The shield generators on Star Destroyers have an adjustable strength. Firepower which is less than or equal to the strength of the shield is repelled, while higher firepower will do damage to the ship. Additionally, multiple concurrent laser attacks have an additive effect. For instance, if two X-Wing fighters attack a Star Destroyer with lasers of magnitude 5, then the shield will need to have a strength of at least 10.

Running the shield at higher strengths uses more energy, so ideally the Star Destroyer would set it at, or just over, the minimum strength needed to repel each attack. Your commander, Admiral Piett, has asked you to write a program which will determine the minimum strength the shields need to be in order to repel a Rebel attack.

Input

The first line of input is the number of laser shots N which is between 1 and 100. Following that will be N integers giving the magnitude of each of the shots.

Output

The output of your program should be a single line giving the minimum strength which the shields need to be in order to repel all of the shots. The format should be "The shield strength must be at least S." where S is the minimum strength of the shield.

Sample Input

```
5
5
1
5
```

```
5
2
```

Sample Output

```
The shield strength must be at least 18.
```

Problem B: Attack Distance

The Death Star is a large Imperial space station with a large laser weapon capable of destroying an entire planet. However, the laser only has a limited range, so the Death Star must be quite near a planet in order to attack it (the more powerful Starkiller Base built by the First Order has a much greater range). Grand Moff Tarkin has tasked you with writing a program that will decide whether or not a planet is in range of the Death Star's weapon.

Your program will be given the coordinates of the Death Star's laser weapon, the coordinates of the center of the target planet and the diameter of the target planet. Because space is three-dimensional, the coordinates will consist of an X, Y, and Z component. Recall that the distance between two points is given by the following formula:

$$D = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}$$

Your program will then print out whether the planet is in range of the laser which has an effective range of 10.0 units. The laser will work as long as it can reach the surface of the planet – it doesn't necessarily have to reach the center.

Input

Input consists of three lines. The first line is three floating point numbers giving the (X, Y, Z) coordinates of the Death Star's laser. The second line is three floating point numbers giving the (X, Y, Z) coordinates of the planet to be destroyed. The third line is a floating point number giving the diameter of the planet to be destroyed.

Output

Output will consist of a single line of text. If the target is in range for the Death Star's weapon, your program should print "Fire!". Otherwise, you should print "Target not in range.".

Sample Input 1

```
105.3 -108.1 9.6
```

```
108.0 -103.2 9.8
0.9
```

Sample Output 1

```
Fire!
```

Sample Input 2

```
-203.6 88.7 213.5
-197.9 95.9 201.1
1.2
```

Sample Output 2

```
Target not in range.
```

Problem C: Bantha Populations

Banthas are large, furry mammals that are native to the planet Tatooine. In the wild, they live in herds that migrate across the sand dunes of their home planet. Banthas are also important to the Tusken Raiders. The Tuskens domesticated the banthas and used them for transport, milk, and meat.

Due to the harsh, dry conditions the bantha live in, and the fact that they are domesticated by the Sand People, survival for wild bantha herds is tenuous. An ecologist who wishes to study banthas and their interaction with Tatooine has observed the following facts about bantha herds:

Each year the number of banthas that die is equal to the ceiling of the population divided by 20. Each year the number of banthas that are born is equal to the floor of the population divided by 16. At the end of the year, local Tusken Raiders take 3 banthas from the herd. If there are fewer than 10 banthas at that time, the Tusken Raiders will not take any. For the first two rules, they are applied to the population of the herd when that year began, not after applying the other rules.

The ecologist has asked you to write a program that will allow him to extrapolate these rules over any number of years. Your program will take the number of banthas in the herd to begin with, and the number of years to study. Your program will then determine the number of banthas, if any, in the herd after that number of years have passed.

Input

Input will consist of two lines. The first line is an integer giving the starting population of banthas. The second line will be the number of years after which you should report the bantha population.

Output

The output should report how many banthas are in the herd after the given number of years. The format should be "After Y years there are B banthas." where Y is the number of years of the study, and B is the number of remaining bantha. If the herd had died off by the time the given

number of years have passed, you should print “After Y years there are no more banthas!”.

Sample Input

```
75
10
```

Sample Output

```
After 10 years there are 43 banthas.
```

Problem D: The Junk Boss

Unkar Plutt runs the junk yard of Niima Outpost on the planet of Jakku. Plutt deals out food rations in exchange for the salvaged remains of the many Imperial ships that crashed on the planet's surface during the Battle of Jakku.

Plutt has commissioned you to write a program that will help him automatically figure out how many rations of food to give scavengers who bring in common parts. The most common parts that scavengers bring in, along with their value in food portions is given in the following table:

Part	**Portions**
Shield Plate	One quarter
Telesponder	One quarter
Flux Converter	One half
Power Damper	Three quarters
Stabilizer	One and a quarter
Hydrobooster	One and a half

Your program will read in a list of parts that a scavenger has brought in, and check if all of the parts are on this list. If so, you should calculate the total value of all of the items, and output it. If the scavenger has brought any items which do not appear on this list, you should just output "Unusual Haul" so that Plutt can take a look himself.

You should also output the total portions only using simplified fractions. For instance, instead of "2 quarter portions", you should output "1 half portion", and instead of "3 half portions", you should output "1 and 1 half portions". Also be sure you use correct plurality.

Input

The first line of input is a positive integer N giving the number of items the scavenger has brought. Following that are N lines each containing the name of an item the scavenger has brought. The item names may contain spaces, but will not exceed 100 characters in length.

Output

You should output either "Unusual Haul", in the case when at least one item is not on Plutt's list, or the number of portions the scavenger has earned. If the scavenger has earned a whole number of portions, you should output "X portions" where X is the number of portions. If the scavenger is awarded less than one whole portion, you should output either "X quarter portions" or "X half portions". In these cases, if X is 1, then you should not pluralize "portions".

If the number of portions is larger than 1, but not a whole number (i.e. a mixed number), then you should output either "X and Y quarter portions" or "X and Y half portions" where X is the number of whole portions and Y is the number of quarter or half portions.

Sample Input 1

```
4
Shield Plate
Flux Converter
Shield Plate
Stabilizer
```

Sample Output 1

```
2 and 1 quarter portions
```

Sample Input 2

```
4
Shield Plate
Flux Converter
Coupling Motivator
Stabilizer
```

Sample Output 2

```
Unusual Haul
```

Problem E: Bothan Codes

During the Battle of Korriban, Bothan spies were able to steal plans to the second Death Star. This enabled the Rebel Alliance to analyze the structure for weaknesses and plan their assault on the battle station accordingly. The Bothans are well-known for their aptitude and skill in all manner of subterfuge.

When seding messages to one-another, even over relatively mundane topics, Bothans often use some form of encryption to prevent others from reading their messages. When transmitting stolen plans, they use complex, secure encryption schemes, but for less important matters will use simpler schemes. One such simple method of encryption is called an XOR cipher. This is based on the XOR binary operation which is defined in the following table:

A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0

With an XOR cipher, each character in the text to be encoded is XOR'ed with a value called the key. Then, when it is time for the other party to decode the message, each character in the encoded text is also XOR'ed with the key producing the original text. For each character, the Imperial Standard Code for Information Interchange (ISCII) code is used which, by a coincidence, works exactly like ASCII. For example, the text "Ewok" is encoded and then decoded with the key 'G' (ISCII code 71) as follows:

```
   E          w          o          k
   01000101   01110111   01101111   01101011
⊕  01000111   01000111   01000111   01000111
-------------------------------------------
   00000010   00110000   00101000   00101100


   00000010   00110000   00101000   00101100
⊕  01000111   01000111   01000111   01000111
-------------------------------------------
   01000101   01110111   01101111   01101011
   E          w          o          k
```

Because using only one character for the key is very easily broken, the key can be as many as 8 characters long. When a longer key is used, the first character of the key is used to XOR the first character of the message, then the second for the second and so on. If the key is shorter than the message, then the key repeats when it runs out.

For example if “Ewok” was encoded with a key of “Han”, then the E would be XOR’d by the H, the ‘w’ by the ‘a’, the ‘o’ by the ‘n’, and the ‘k’ by the ‘H’ again.

You will write a program that will decode text that Bothans have encrypted using this XOR cipher.

Input

The first line of input is the key that should be used for the decoding. If there are multiple letters, they will be separated with spaces. The second line is the message to be decoded, again each letter will be separated by spaces. The message will consist of fewer than 1000 characters.

Because some characters may not be printable once they have been encoded, each character of the input (the key and message) will be given as a decimal value for its ISCII code.

Output

The output should consist of one line giving the decoded message.

Sample Input

```
72 97 110
5 0 23 104 21 6 45 65 40 39 19 13 45 65 12 45 65 25 33 21 6 104 24 1 61 64
```

Sample Output

```
May the Force be with you!
```

Problem F: Droid Control

The Empire uses many MSE-6-Series repair droids (pictured on the right) to perform cleanup, basic repair and patrol duties. The droids are quite small at only 25 centimeters high and run on four wheels. The droids are used to patrol Star Destroyers and the Death Star.

They are typically used to patrol or clean a specific path in the Death Star over a certain period of time. Because the Death Star is of such a uniform, repetitive structure, the droids movements are controlled by programs. Each droid runs a control program that keeps it moving throughout the Death Star.

The droid operates as a stack-based machine where each instruction in the control program causes the droid to take some action. The stack of the droid is initially empty and can hold up to 100 values. The MSE-6-Series droid supports the following instructions:

Instruction	**Meaning**
push [value]	Pushes the given integer value onto the stack.
add	Pops the top two values off the stack and adds them, putting the result on the stack.
move	Pops the top value off the stack and moves the robot by that number of meters. If the value is positive, the droid should move forward; if negative, it should move backward.
turn	Pops the top value off the stack and turns the robot by that angle. The value will always be a multiple of 90. Positive values turn the robot to its left and negative values turn it to its right.
branch [line]	If the top value on the stack is non-zero, control transfers to the given line of the program. If the top value is zero, the instruction does nothing. This instruction does not modify the stack. Lines are numbered starting from 0.
halt	Stops the droid and ends the control program.

When the droid begins operation, it starts by executing the first line in the command program, and continues on executing them one by one until it halts. Along the way, the move and turn instructions will cause the droid to move throughout its environment. As the control programs are developed, the programmers would like to know, for a given control program, where the droid will eventually end up. This will make it easier and faster to test the control programs as the droids are quite slow and can take a lot of time to finish one program.

You must write a program that will read in the control program, simulate it as the droid will execute it, and report the final position of the droid relative to its starting position. You may

assume that the control program will consist of fewer than 100 lines, and will always eventually halt. You may also assume that each branch instruction will specify a valid line number.

Input

The first line of input will consist of an integer giving the number of lines in the program. Each subsequent line is one line of the control program in the format given above.

Output

The output should report the final position of the droid relative to where it started. It should be given in the following format: "The droid halts [number] meters [left/right] and [number] meters [forward/backward]." If the droid ends up where it started in either direction, only report the direction it has moved in. For example "The droid halts [number] meters [left/right/forward/backward]." If the droid finishes back in the same location it started in, the output should be "The droid halts where it began."

Sample Input

```
9
push 6
push 100
move
push -90
turn
push -1
add
branch 1
halt
```

Sample Output

```
The droid halts 100 meters right and 100 meters forward.
```

Problem G: Runoff Voting

After the Battle of Endor, the leadership of the Rebel Alliance formed a government referred to as the "New Republic" to restore peace and prosperity to the galaxy. They implemented many changes from the Old Republic to prevent a singe person or group from acquiring too much power as Chancellor Palpatine did. These changes included a reduction in the powers of the Chancellor, rotating the location of the senate amongst member worlds, and the implementation of a runoff voting system.

In a runoff voting system, rather than selecting only one candidate, voters rank their preferred candidates in order. For example, when voting for the next Chancellor, delegates are presented with a list of candidates such as:

1. Lanever Villecham
2. Thena Barakis
3. Mon Mothma
4. Nomar Truden
5. Davessi Jaxx

A vote is cast by typing the numbers of the candidates in the preferred order. For example, if a delegate's first choice is Nomar Truden, with Lanever Villecham second, but they do not care about the other three candidates, they would vote:

```
4 1
```

Each integer corresponds to the number preceding a valid candidate on the ballot. No number should appear more than once on a single ballot. Also each number should be between 1 and the number of candidates. Ballots which do not meet these rules are called "spoiled ballots". Spoiled ballots are ignored when tabulating a winner.

When polling is complete, the outcome of the election is calculated as follows:

First, all the first-preference votes (on valid ballots) are counted. After this count, if the number of votes for any candidate is more than half of the number of valid ballots, that candidate is declared the winner. Otherwise, the candidate(s) with the fewest number of votes is (are) eliminated. Ballots on which an eliminated candidate is mentioned are effectively modified by

deleting that candidate, thereby "promoting" any lower-preference non-eliminated candidates. If, after the elimination, there are any remaining candidates, the first-preference votes are counted again to determine a winner. If, after the elimination, no candidates are left, the election is declared indecisive.

This runoff voting system is in place to avoid the emergence of a small number of powerful political parties gaining undue influence over the political system. You are to write a program to calculate the outcome of elections using this system.

Input

The first line of input contains two integers separated by a space. The first is the number of candidates, and the second is the number of ballots. There will be between 1 and 25 candidates, and between 1 and 1,000 ballots.

After that will come the list of candidates, one on each line. Each of these lines will have the candidate's number (starting at one and increasing), then a space, then the candidate's name. Each candidate's name will be at most 25 characters long.

Next comes the list of ballots. Each ballot will consist of between 1 and 25 integers representing the order the voter prefers the candidates. These numbers refer to the numbers identifying each candidate in the ballot. The numbers will be separated by spaces. You must check that the ballots are not spoiled before using them.

Output

As each candidate is eliminated, a message to that effect must be printed. If more than one candidate is eliminated in the same round (because they each had the minimum number of first-preference votes at that stage) then their eliminations should be output in the order in which the candidates originally were input. An elimination is recorded as "[candidate] with [number] votes is eliminated.".

Next your program should either declare the winner, as "The winner of the election is [candidate]." or declare that the election was indecisive by printing "The election was indecisive.".

Sample Input

```
5 11
1 Lanever Villecham
2 Thena Barakis
3 Mon Mothma
4 Nomar Truden
5 Davessi Jaxx
1 5 3
3 1 2
1 4 1
2
2 5 3
1 6
3 1 5
4 3
1 3
1 3 4 1
1
```

Sample Output

```
Davessi Jaxx with 0 votes is eliminated.
Nomar Truden with 1 votes is eliminated.
Thena Barakis with 2 votes is eliminated.
Lanever Villecham with 3 votes is eliminated.
The winner of the election is Mon Mothma.
```

Problem H: Breaking Booma

During the Trade Federation's assault on planet Naboo, the Gungan Grand Army used a type of weapon called a booma. A booma consists of blue plasma energy locked in a thin shell which breaks if thrown hard enough. Booma are very effective when used against electronic foes such as droids because the plasma energy burns out their circuits.

Because the weapon discharges when, and only when, the shell is broken, it is important that the shell be thick enough to avoid breaking prematurely, and thin enough to break on impact with a target. In order to test that the shell is an appropriate thickness, Gungan engineers take a number of booma out of every batch and see what force is needed to discharge them. The way they do this is by dropping the booma off of different stories of a tall building and noting which story the booma breaks from. If the booma breaks when dropped from a given story, it would also break from every story above that. If a booma does not break when dropped from a given story, it would also not break from any story below that. If a booma is dropped and does not break, it may be reused in testing. If it breaks, however, it must be thrown away. The engineers must test the booma to determine which story (if any) is the first one they will break from.

The engineers want to do the testing with the least number of drops possible. If they were allowed to break as many boomas as they wanted, they would use a binary search strategy by dropping a booma from the middle story, and then seeing if the top half or bottom half should be tested next based on whether or not the first drop broke. If, however, they are only allowed to break one booma, then they must try dropping it off every story from the first until it finally breaks.

You must write a program that reports how many drops are needed to test a batch of booma based off how many booma are allowed to be used and the number of stories to be tested. The number of drops needed must be the minimum number needed in the worst case scenario, that is, the number of drops in which you can guarantee that you will find the answer.

Input

The first line of input is a number N giving the number of test cases. Following that are N lines, one for each of the test cases. Each test case line contains 2 integers. The first is the number of boomas available for the testing and the second is the number of floors to test.

Output

Output will consist of one line for each test case. Each line will be of the form "Minimum number of drops with B boomas and F floors is D." where B and F are the number of boomas and floors for the test case and D is the minimum number of drops in the worst case needed for the test case.

Sample Input

```
2
1 100
2 100
```

Sample Output

```
Minimum number of drops with 1 boomas and 10 floors is 100.
Minimum number of drops with 2 boomas and 10 floors is 14.
```

Problem I: Pilot Matching

On the Rebel Alliance Echo Base, which is on the frozen planet of Hoth, rebels patrol and defend the base in part with T-47 Snowspeeders. These speeders sit both a pilot and a gunner. The pilot flies the vehicle while the gunner mans the back of the vehicle and fires the speeder's weapons. For the vehicle to be most effective during a conflict, both the pilot and the gunner have to work well together.

Each pilot has a list of the gunners he would be most comfortable working with. Likewise, each gunner has a list of the pilots he would be most comfortable with. A pairing of a pilot and gunner who both want to work together is called an "optimal pairing". If necessary, of course, the rebels will work with whoever, but they would like to maximize the number the number of optimal pairings when assigning partners.

General Rieekan, who is in charge of Echo Base, has commissioned you to write a program to find the maximum number of optimal pairings which can be made from amongst the pilots and gunners at Echo Base. Your program will get as input a list of the preferred partners for each pilot and gunner in the base. There are an equal number of pilots and gunners. Your program should output the maximum number of optimal pairings which can be made.

Input

The first line of input will be a single integer, N, giving the number of pilots which is equal to the number of gunners. The next N lines will give the preferences for the pilots. The next N lines after that will give the preferences of the gunners.

Each preference line consists of a set of 0 to N integers, each of which will have a value from 1 to N. These numbers represent that the given rebel would like to work with the rebel with the given number in the opposite group. The rebels in each group are numbered from 1 to N. If there are multiple numbers in a preference line, they are separated by spaces.

Output

Output should consist of one line displaying the maximum number of optimal pairings which can be made in the format "The largest number of optimal pairings is P".

Sample Input

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

Sample Output

```
The largest number of optimal pairings is 3.
```

Doing Input and Output

This guide contains the basic way of doing input and output in C++, Java, and Python, in case you need a quick refresher.

In programming contests, you should always do input from the terminal screen (e.g. using the keyboard), and do output to the terminal screen. You should not open any files for input or output.

You should also not output any prompts. If a problem instructs you to read a line containing a number, just read it in without any kind of prompt saying "Enter a number" or similar.

C++

Input

1. To skip over whitespace (spaces or new lines), and read in a single value (such as an integer or string), use `cin >> value;`. For example:

   ```
   // read one integer
   int number;
   cin >> number;

   // read one character string
   char str1[100];
   cin >> str1;

   // read one string object
   string str2;
   cin >> str2;
   ```

2. To read in one entire line of input, which may contain spaces, use `cin.getline`. For example:

   ```
   // read in a character string of up to 100 characters
   char str1[100];
   cin.getline(str1, 100)

   // read in a string object
   string str2;
   getline(cin, str2);
   ```

Output

Output is done with `cout <<` in C++ which can take any built-in data type. For example:

```
// print a message, then an integer, then a new line
int x;
cout << "X is equal to " << x << endl;
```

Python

Input

1. The `input` function in Python read in one line of input and returns it as a string. For example:

   ```
   # read in a string
   line = input()
   ```

2. In order to convert from a string to a number, you can use the `int` function for integers, or the `float` function for real numbers:

   ```
   # read in an integer by passing the input string to int()
   number = int(input())
   ```

3. In order to break a line of input into multiple strings, separated by spaces, you can use the `.split()` function which returns a list of strings:

   ```
   # read in a whole line
   line = input()

   # split it into separate strings based on spaces
   words = line.split()
   ```

Output

1. Output in Python is done with the `print` function which outputs all of its arguments, separated by spaces, and puts a newline at the end:

   ```
   # print a message, a number and a new line
   print("X is equal to", x)
   ```

2. In order to prevent `print` from putting spaces between each item, pass `sep=""` as an argument:

   ```
   # now there is no space between message and value
   print("X is equal to", x, sep="")
   ```

3. In order to prevent `print` from putting a new line at the end, pass `end=""` as an argument:

   ```
   # now there is no new line added
   print("X is equal to", x, end="")
   ```

Java

Input

Input in Java can be done with the `Scanner` class which must be imported first:

```
import java.util.Scanner;
```

Then a `Scanner` object must be created:

```
java.util.Scanner in = new java.util.Scanner(System.in);
```

1. To read in one line of input into a string, use the scanner's `nextLine` method:

   ```
   String line = in.nextLine();
   ```

2. To read in a single word into a String, stopping at a space, use the `next` method:

   ```
   String word = in.next();
   ```

3. To read a numerical value use the `nextInt` method for integers, or the `nextDouble` method for real numbers:

   ```
   int number1 = in.nextInt();
   double number2 = in.nextDouble();
   ```

Output

1. To output a string constant or variable, use the `System.out.println` function which takes one argument, prints it to the screen, then prints a new line:

   ```
   // print a message
   System.out.println("This will be printed")

   // print a value
   int x;
   System.out.println(x);
   ```

2. To output something without a new line at the end, use the `System.out.print` function which outputs its argument with no newline:

   ```
   // print a message with no new line
   System.out.print("The value of X is ")

   // print a value with no newline
   int x;
   System.out.print(x);
   ```