

---

# 1. Detention

**Program Name:** boring.java

**Input File:** <none>

While in your high school programming class, you were recently reprimanded for talking too much and at the wrong time (as usual). The instructor has given you after school detention and tasked you with writing, “I will not talk during this boring class.” two hundred times! Luckily, a substitute teacher is overseeing your punishment and you manage to convince him that your instructor allows students to write programs to meet these requirements.

Now all you have to do is write the program to get out of detention.

## Input

There is no input for this problem.

## Output

Print the line, “I will not talk during this boring class.” two hundred times.

### Example Output To Screen (first 10 lines only)

```
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.  
I will not talk during this boring class.
```

---

## 2. Capitals

**Program Name:** capital.java

**Input File:** capital.in

Your Geography class is currently studying Europe, and the next test covers the capital cities. The test will be 20 fill-in-the-blank questions. Each question will be in one of the following two formats:

Format 1      The capital city of <country> is \_\_\_\_\_.  
Format 2      The capital city of \_\_\_\_\_ is <city>.

Where placeholders <country> and <city> are provided, and you must fill in the blanks. Here is a copy of your study sheet (No, it doesn't cover all of Europe, but you know it will be enough for your test):

Capital	Country	Capital	Country
Dublin	Ireland	London	UK
Lisbon	Portugal	Madrid	Spain
Andorra	Andorra	Paris	France
Bern	Switzerland	Vaduz	Liechtenstein
Brussels	Belgium	Berlin	Germany
Vienna	Austria	Rome	Italy
Valletta	Malta	Oslo	Norway
Stockholm	Sweden	Copenhagen	Denmark
Vatican City	Vatican	Budapest	Hungary
Athens	Greece	Helsinki	Finland
Monaco	Monaco	Sarajevo	Bosnia Herzegovina

You have little interest in Geography and are an accomplished programmer. To skirt the opportunity of genuine learning, you decide to take advantage of your existing abilities and program your calculator to fill in the blanks.

### Input

Input will consist of 20 lines, each of the format given in the description above. Blanks will consist of 5 consecutive underscore characters ('\_').

### Output

Output will exactly mirror the input except that the blanks will be replaced with the correct country or capital names from the study sheet.

### Example Input File (first 3 lines only)

The capital city of Italy is \_\_\_\_\_.  
The capital city of \_\_\_\_\_ is Helsinki.  
The capital city of Bosnia Herzegovina is \_\_\_\_\_.

### Example Output To Screen (first 3 lines only)

The capital city of Italy is Rome.  
The capital city of Finland is Helsinki.  
The capital city of Bosnia Herzegovina is Sarajevo.

---

## 3. The Chase

**Program Name:** chase.java

**Input File:** chase.in

Cats chase mice, but who will win?

Simulate a cat and a mouse in a maze. Each is facing in a cardinal direction (north, east, west, south) and takes turns moving, with the mouse going first. Both the cat and the mouse can see indefinitely far in a straight line in the direction which they are facing.

The mouse moves according to the following rules:

1. If the mouse sees the cat at the beginning of its move, he will turn 90 degrees counterclockwise and then run straight as far as possible until he hits a wall or successfully exits the maze (yay!).
2. If the mouse does not see the cat at the beginning of its move, he will run straight as far as possible until he hits a wall or successfully exits the maze (yay!).
3. Whenever the mouse hits a wall, he turns 90 degrees counterclockwise and waits for the cat to take his turn.

The cat moves according to the following rules:

1. If the cat sees the mouse at the beginning of his move, he runs and eats it immediately (yum!).
2. If the cat saw the mouse run by, he moves to the last position where he saw the mouse, turns to face the direction the mouse was moving, and waits for the mouse to take his turn.
3. If neither 1 nor 2 apply, the cat will run straight until he hits a wall or arrives at the edge of the maze. He then stops, turns 90 degrees clockwise, and waits for the mouse to take his turn.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of mazes that need analysis. Each maze in the input begins with a line containing two integers,  $c$  and  $r$ , indicating the size of the maze. The following  $r$  lines each contain  $c$  printable characters that form the maze, each of which is one of:

'#' - wall

'.' - empty floor

'n', 'e', 's', 'w' – A lower-case letter representing the mouse and the direction it is facing.

'N', 'E', 'S', 'W' – An upper-case letter representing the cat and the direction it is facing.

Each maze has exactly one cat and one mouse.

Each dimension of each maze is in the range 1 to 20, inclusive.

### Output

For each maze in the input, determine whether the mouse escapes, is caught, or if the chase continues forever. If the mouse escapes, print, “The mouse escapes in  $x$  moves.” where  $x$  is the number of moves made by the mouse. If the mouse is caught, print, “The mouse is caught in  $y$  moves.” where  $y$  is the number of moves made by the cat to catch the mouse. Otherwise, since the chase continues without end, print, “The chase never ends.”

---

### Example Input File

```
3
6 5
#####
#....#
#.#.#.#
#E..n#
#.#.###
6 5
#####
#....#
#.#.#.#
#...W#
#n#####
6 5
#####
#....#
#.#.#.#
.E..n#
#####
```

### Example Output To Screen

```
The mouse escapes in 3 moves.
The mouse is caught in 2 moves.
The chase never ends.
```

---

## 4. Decryptonomicon

**Program Name:** `decrypt.java`

**Input File:** `decrypt.in`

Encryption can be simple, complex, or just plain annoying. Luckily, for this problem, the encryption is very straightforward. Here is the encryption algorithm:

- For each line of text to encrypt

- For each character of the line to encrypt

- determine the 8-bit binary ASCII value of the character

- flip all the bits in the ASCII value (i.e., 0 becomes 1 and 1 becomes 0)

- convert the bit-flipped ASCII value to a hexadecimal representation

- print the hexadecimal value to standard output followed by a single space

- print a newline character ('\n') to standard output

For example, the ASCII value of the character 'A' is 65, which is 0100 0001 in binary. Flipping the bits gives you 1011 1110, which is represented as 'be' in hexadecimal.

Not so luckily, you have to write a program to reverse the encryption process.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of lines of encrypted input to decrypt.

### Output

For each encrypted input line, output the decrypted line.

### Example Input File

```
2
```

```
b7 9a 93 93 90 df a8 90 8d 93 9b de
```

```
b6 df 90 88 91 df cd df 9c 9e 8b 8c d1
```

### Example Output To Screen

```
Hello World!
```

```
I own 2 cats.
```

---

## 5. Disco Stew

**Program Name:** disco.java

**Input File:** disco.in

Disco Stu is a cannibal and wants to capture you for his next batch of, you guessed it, disco stew.

You must escape the dance floor, but the rotating colors cast by the disco ball make it difficult to move. In fact, you can only reliably move within a region of the floor that is all one color. If you want to move further, you must wait for the disco ball to rotate (90 degrees clockwise) and try again.

Note that a colored 'region' of the floor only consists of same-colored squares that are horizontally or vertically adjacent. Regions that are only diagonally adjacent (i.e., they meet at a corner) are separate.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of discos that you must attempt to escape from. For each disco, the first line of the input will have two integers,  $x$  and  $y$ , representing the row and column of the dance floor that you start on (rows are numbered 1-9 from top to bottom, columns are numbered 1-9 from left to right). The next nine lines in the input represent the dance floor and its coloring with the disco ball in its starting position. Note that all discos are 9x9 with the disco ball located above the center of the floor (5,5).

### Output

For each disco in the input, determine the minimum number of moves it takes to escape to the disco door (which is always in the upper left corner, at row 1 column 1).

In the first example below, you can immediately move from the starting position to the door because the entire room is blue. When you can escape, print a message, "I escaped from the disco # $s$  in  $z$  turn(s) of the disco ball." where  $z$  is the smallest number of moves possible to get you from your starting position to the disco door and  $s$  is 1 for the first disco, 2 for the second, etc. Since you can escape the disco in example 1 without the disco ball turning at all,  $z$  is 0.

In example 2, you are trapped because you must cross the green region to get to the exit, but you have no way to get into or out of it. When it is not possible to reach the exit in any number of rotations of the disco ball, print the message, "I'm the next ingredient in disco stew # $s$ ." where  $s$  is as described above.

To solve example 3, you must understand how the disco ball rotates. For purposes of this problem, the disco ball rotates clockwise in 90 degree increments. Here is how the dance floor would look for 0, 1, 2, and 3 rotations of the disco ball:

BBBBBBBBB	BBBBBBBBB	BBBBBBBBB	BBBBBBBBB
BGGGGGGGB	BGGGGGGGB	BGGGGGGGB	BGGGYGGGB
BGBBBBBGB	BGBRRRBGB	BGBBBBBGB	BGBRYRBGB
BGRRRRRGB	BGBRRRBGB	BGRRRRRGB	BGBRYRBGB
GRRR <b>R</b> YYYB	BGBRRRBGB	BYYYRRRGG	BGBRRRBGB
BGRRRRRGB	BGBR <b>Y</b> RBGB	BGRRRRRGB	BGBRRRBGB
BGBBBBBGB	BGBRYRBGB	BGBBBBBGB	BGBRRRBGB
BGGGGGGGB	BGGGYGGGB	BGGG <b>G</b> GGGB	BGGGGGGGB
BBBBBBBBB	BBBBBBBBB	BBBBBBBBB	BBBBBBBBB

In the beginning, you're in the red region, but if you move to (6, 5) and wait for the next rotation, you're in the yellow region and can move to (8,5) which after the next rotation puts you in the green region. You can then move to (5,9), which after one final rotation will land you in the blue region where you can move directly to the exit. In total, you made it with only three rotations of the disco ball. (Starting positions for each turn in this example are in bold.)

---

### Example Input File

```
3
9 9
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
5 5
BBBBBBBBB
BGGGGGGGB
BGBBBBBGB
BGRRRRRGB
BGRRRRRGB
BGRRRRRGB
BGBBBBBGB
BGGGGGGGB
BBBBBBBBB
5 5
BBBBBBBBB
BGGGGGGGB
BGBBBBBGB
BGRRRRRGB
GRRRRYYYB
BGRRRRRGB
BGBBBBBGB
BGGGGGGGB
BBBBBBBBB
```

### Example Output To Screen

```
I escaped disco #1 in 0 turn(s) of the disco ball.
I'm the next ingredient in disco stew #2!
I escaped disco #3 in 3 turn(s) of the disco ball.
```

---

## 6. Factor Faster

**Program Name:** factor.java

**Input File:** factor.in

Write a program to factor integers or indicate they are prime.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of integers to factor.

The next  $n$  lines will each contain a single positive integer less than 100000 (one hundred thousand).

### Output

Factor each integer,  $m$ . If it is prime, output the message “ $m$  is prime”. Otherwise, output the prime factorization of  $m$  with factors listed in descending order and with repeating factors noted using exponents. For example, the number  $72 = 3 \times 3 \times 2 \times 2 \times 2 = 3^2 \times 2^3$  (see the third example).

### Example Input File

```
3
13
99
72
```

### Example Output To Screen

```
13 is prime
99 = 11 x 3^2
72 = 3^2 x 2^3
```



---

## 7. Flower Power

**Program Name:** flower.java

**Input File:** flower.in

When planning a garden, it's important to space plants far enough apart to keep their roots from interfering with one another (as well as the above ground portions). Each type of plant has a typical radius for its root structure and another typical radius for the above ground portion.

Write a program that, given a list of plants, their typical radii, and their positions in the garden, will determine whether or not all of the plants have enough room to grow.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of gardens requiring analysis. For each garden, the first line will contain a single integer,  $m$ , indicating the number of plants in the garden. The following  $m$  lines each represent one plant in the garden with four non-negative integers,  $a$ ,  $b$ ,  $x$ ,  $y$ , where  $a$  is the typical radius of the root system,  $b$  is the typical radius of the above ground portion, and the coordinates  $(x,y)$  denote the plant's position in the garden.

### Output

For each garden in the input, determine whether or not all plants have sufficient room to grow (both above and below ground). If so, print, "yes". Otherwise, print, "no". After printing the verdict for each garden, output a string with the total count of "yes" gardens but encrypt the string using the algorithm given in problem 4.

Note: In the example, the unencrypted version of the last line would be the number 1.

### Example Input File

```
3
3
5 5 0 0
1 1 6 0
12 1 0 16
3
5 5 0 0
1 1 6 0
1 12 0 16
2
5 1 0 0
1 5 6 0
```

### Example Output To Screen

```
no
no
yes
ce
```

---

## 8. Mmmmmmath

**Program Name:** math.java

**Input File:** math.in

The mean, or average, of a set of integers is the sum of all the integers of the list divided by the number of integers in the list. For example, the mean of the set  $\{1,1,4,9,9\}$  is  $(1+1+4+9+9)/5 = 4.8$ .

The median of a set of integers is the middle value in the set if it were sorted lowest to highest. If there are two middle values (i.e., there are an even amount of integers in the set), then the median is the average of the two middle values. For example, the median of the set  $\{1,1,4,9,9\}$  is 4 and the median of the set  $\{1,1,1,4,9,9\}$  is 2.5 (the average of the two middle values, 1 and 4).

The mode of a set of integers is a set of the elements that occur most often, sorted in ascending order. For example, the mode of the set  $\{1,1,1,4,9,9\}$  is  $\{1\}$  and the mode of the set  $\{1,1,4,9,9\}$  is  $\{1,9\}$ .

The meanode of a set of integers is the mean of its mode. For example, the meanode of the set  $\{1,1,4,9,9\}$  is 5, since the mode of the set is  $\{1,9\}$  and the mean of that set is 5.

The mediode of a set of integers is the median of its mode. For example, the mediode of the set  $\{1,1,4,4,9,9\}$  is 4, since the mode of the set is  $\{1,4,9\}$  and the median of that set is 4.

Note: The terms meanode and mediode are inventions of the authors for the purposes of this problem and are not actual terms.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of sets to analyze.

Each of the following  $n$  lines will consist of a comma-separated list of integers enclosed in curly braces, as shown in the introduction. Each list will be sorted in ascending order.

### Output

For each set of integers, print a single line of the mean, median, mode, meanode, and mediode values, each separated by a single space. Print the mode values in set notation (comma-separated list of integers enclosed in curly braces). Truncate any portions of values beyond two decimal values. (e.g., print 4.16 for a value of 4.16666666).

### Example Input File

```
4
{1, 1, 4, 9, 9}
{1, 1, 1, 4, 9, 9}
{1}
{9, 9, 9, 9, 9, 9, 9}
```

### Example Output To Screen

```
4.8 4 {1, 9} 5 5
4.16 2.5 {1} 1 1
1 1 {1} 1 1
9 9 {9} 9 9
```

---

## 9. Chilly Rock

**Program Name:** rock.java

**Input File:** rock.in

It may seem simple to create a yummy ice cream dessert, but you have to know the proper actions and perform them in a valid sequence to get customers to pay.

Here is a list of possible actions that can be used when creating and selling a dessert:

Action	Description
cone	get a cone (holds up to 2 scoops)
waffle	get a waffle cone (holds up to 4 scoops)
scup	get a small cup (holds up to 2 scoops)
cup	get a regular cup (holds up to 3 scoops)
bcup	get a big cup (holds up to 4 scoops)
dip	dip a cone in liquid chocolate
scoop	add a scoop of ice cream
sprinkles	add sprinkles
wc	add whipped cream
cherry	add a cherry on top
cash	get cash for your dessert creation

And here is a list of rules for how the actions must be performed:

The set of actions { cone, waffle, scup, cup, bcup } will be known collectively as starting actions.

The set of actions { sprinkles, wc, cherry } will be known collectively as topping actions.

The first action must be a starting action, and only the first action can be a starting action.

The dip action is only valid immediately following a waffle action or a cone action.

The scoop action must occur at least once.

The scoop action cannot occur more times than the capacity stated for the starting action.

All scoop actions must occur before any topping actions.

The wc action cannot occur more than once.

The cherry action can only occur immediately after a wc action.

The last action must be a cash action, and only the last action can be a cash action.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of desserts to evaluate. Each of the following lines will contain a list of actions that may or may not be valid for creating and selling a dessert.

All actions in the input will come from the list given in the introduction.

### Output

For each list of actions, determine whether or not they are valid for creating and selling a dessert. If they are valid, print the message, "You created a dessert with  $x$  scoop(s)" where  $x$  is the total number of scoops in the dessert. If the list of actions is not valid, print the message, "Your prospects for promotion are looking grim".

### Example Input File

3

```
cone dip scoop wc cherry sprinkles cash
bcup scoop scoop scoop scoop cash
cone dip scoop cherry cash
```

### Example Output To Screen

```
You created a dessert with 1 scoop(s)
You created a dessert with 4 scoop(s)
Your prospects for promotion are looking grim
```

---

## 10. Super Shipping

**Program Name:** shipping.java

**Input File:** shipping.in

As the owner of a widget manufacturing company, you manufacture widgets, process orders, and make shipments. Each day begins at 4:00 AM and ends at 8:00 PM (two shifts). Due to an overzealous janitorial staff, unfilled orders and unshipped widgets from previous days are lost, but at the beginning of each new day a fresh set of orders arrive.

The widget assembly line produces widgets at a rate of one every five minutes, and widgets are shipped by the case (9 widgets each). Partial cases cannot be shipped, so in the event that an order would result in an incomplete case, the order is increased just enough to result in a full case. For example, if an order is made for 11 widgets, two full cases of widgets must be shipped to complete it.

Write a program that calculates what time of day shipments are made to which customers. Process the orders in the sequence that they are given (i.e., no order can be filled unless all orders before it are filled).

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of days of operations to calculate. For each day, the first line of input will contain a single integer,  $m$ , indicating the number of orders for that day. The following  $m$  lines will each contain an order consisting of the name of the ordering city (one word) followed by the number of widgets they need.

Note, no city will have more than one order in any given day.

### Output

For each day, first print a line, "Day #X" where X is 1 for the first day, 2 for the second, etc. Then, for each order in the input, determine if it can be filled. If so, display the statement, "Ship Y cases to CITY at TIME." where Y is the minimum number of cases necessary to satisfy the order and TIME is the time of day the last case is ready to ship. If the order cannot be filled, display the statement, "CITY does not get a shipment today."

### Example Input File

```
2
4
Houston 15
Dallas 20
Austin 30
Tulsa 185
2
Tulsa 185
Boston 1
```

### Example Output To Screen

```
Day #1
Ship 2 cases to Houston at 5:30AM.
Ship 3 cases to Dallas at 7:45AM.
Ship 4 cases to Austin at 10:45AM.
Tulsa does not get a shipment today.
Day #2
Ship 21 cases to Tulsa at 7:45PM.
Boston does not get a shipment today.
```

---

# 11. Global Warming

**Program Name:** warming.java

**Input File:** warming.in

Recent reports on global warming are projecting a future rise in sea levels. Write a program that will take a topological map and update it to show the effects of a particular sea level rise.

## Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of maps that need to be processed. For each map, the input will contain:

1. A single line with integers  $r$ ,  $c$ , and  $m$ , where  $r$  and  $c$  are the dimensions of the map and  $m$  is the amount of rise in sea level to simulate.
2. A topological map with  $r$  rows and  $c$  columns. Each square unit of the map will be one of:
  - ocean – represented by a period ('.').
  - lake – represented by a lower case letter  $w$ .
  - land – represented by an integer in the range from 1 to 9 (inclusive) which indicates the height of the land above sea level.

## Output

For each map, reduce the elevation of all land areas by the sea level rise, replacing any land that would then be at sea level (height 0) or below as an ocean or lake, as appropriate.

The difference between an ocean and a lake is that the lake does not touch the border of the map. Note that it is possible that global warming will cause an ocean to connect to a lake or for a lake to touch the edge of the map. In that case, the lake becomes part of the ocean and should then be represented appropriately (with a period). These types of connections can only happen horizontally or vertically on the map (not diagonally).

## Example Input File

```
2
5 5 1
.....
.999.
.919.
.999.
..1..
8 10 3
552111111.
6522www21.
6555w521.
6626ww521.
72w276521.
662666521.
555555521.
22222221.
```

## Example Output To Screen

```
.....
.888.
.8w8.
.888.
.....
22.....
32.....
32222.2...
33w3..2...
4www432...
33w3332...
2222222...
.....
```

---

## 12. What Did You Say?

**Program Name:** what.java

**Input File:** what.in

Write a program that will read in a given piece of text from an input file and write that same text back to standard output.

### Input

The first line of the input file will contain a single integer,  $n$ , indicating the number of lines of text to read.

### Output

Output each of the lines of text.

### Example Input File

3

This is sample input.

It is also sample output.

Here is a random string: l;hasf098771234\_\_!@#\$^

### Example Output To Screen

This is sample input.

It is also sample output.

Here is a random string: l;hasf098771234\_\_!@#\$^