William P. Clements High School

Computer Science Invitational

Advanced Division

November 2013

Team Number:	School:

Problem Checklist

(Turn in at the end of contest)

Problem Number	Name	Tally of Incorrect Submissions (-5 each)	Solved (+60)	Score
1	Albi the [Ravenous] Dragon			
2	Barcodes			
3	Beach Time			
4	Coffee Break			
5	Exact Change			
6	Fifty Shades of Lavin			
7	Inventory Trouble			
8	RC Circuit			
9	Reverse Polish Notation			
10	Spiral of Life			
11	When Life Gives You Lemons			
12	Word Snake			
			Total:	

1. Albi the [Ravenous] Dragon

Program Name: Albi.java **Input File: albi.dat**

"In the Marmalade forest (forest), between the make believe trees, in a cottage cheese cottage! Lives Albi, Albi, Albi the [Rambunctious] Dragon... And so all of the villagers chased Albi the [revered] dragon, into a very cold and very scary cave. It was so dark and so scary there that Albi began to cry. Dragon tears, which as we all know turn into jellybeans!" – Flight of the Conchords

Input

The first line of input will contain a single integer `n` indicating the number of cases to follow. The first line of each dataset contains two integer `r` and `c`, representing the dimensions of the contour plot. The following `r` lines contain the contour plot with `r` rows and `c` columns of characters. Equi-elevation contour loops are represented by '`*`'. Each point within a closed loop has the same elevation. Inner loops will always have a greater elevation than their outer loops. Assume these loops will not overlap or intersect, and there will be at least one empty space between loops. Loops have right angles at all turns. The space between loops is filled by '`.`'.

Output

Albi the Dragon is always hiding on a mountain peak (a relative maximum height), where there must be an open space ("`.`"). Print out all possible locations in order pair `(r,c)` forms. Sort them in ascending `r` order; sort them by ascending `c` if `r` is the same. Separate output of different datasets with a blank line. The top-left corner of the contour plot is location `(0,0)`. At least one peak is guaranteed in each dataset.

Example Input File

```
2
7 7
*******
*.....*
*.***.*
*.*.*.*
*.***.*
*.....*
*******
11 10
.*********
**.......*
*..***.***
**.*.*.*..
.*.*.*.*..
.*.***.***
.*.......*
.***.....*
...*.....*
...*.....*
...*******
```

Example Output to Screen

```
(3,3)

(3,4)
(4,4)
```

2. Barcodes

Program Name: Barcodes.java **Input File: barcodes.dat**

Chris-co needs pictures of barcodes for his school project. He inexplicably forgot that search engines are convenient and effective tools for this assignment, opting instead for the most tiresome route of all: printing ASCII pictures of barcodes. Chris-co does not know how barcodes work, so he will use Java's Random class to generate the ASCII images. Chris-co is not experienced in the fine art of Java, but he has written an algorithm for making random barcodes in a mediocre fashion. Help him generate the barcodes to make his project marginally less terrible.

Note that the judges will provide a seed so that the numbers generated match the judges' output data.

1. Given a 64-bit integer seed, create a `java.util.Random` object.
2. Generate two numbers, the first between 4 and 10, inclusive, and the second between 1 and 2 times the first generated number, inclusive. These two numbers represent the number of rows and the number of columns the barcode will take up, respectively.
3. Generate integers between 0 and 1, inclusive. A zero will represent a white vertical bar. An one will represent a black vertical bar. Start printing vertical bars from the left side. Fill the whole bar code with bars until column limit is reached.
4. In the last row of the barcode, beginning from the first column, generate random single digits in every other column, until the row is filled.
5. Print the generated barcode.

Input

The first line will contain a number `n`, which represents the number of data sets to follow. Each of the next `n` lines will contain a 64-bit integer, which is the seed for each barcode.

Output

A barcode consists of two components – the bars and the numbers. The bars are vertical and can be either black or white. Black bars are represented by '`#`'. White bars are represented by a single space. Print an empty line after each barcode.

Example Input File

```
1
4010104051743530195
```

Example Output to Screen

```
#   ##   #
#   ##   #
#   ##   #
#   ##   #
#   ##   #
#   ##   #
#   ##   #
#   ##   #
#   ##   #
#   ##   #
0 1 5 6 6
```

Random numbers generated for this seed:

```
9 10 1 0 0 0 1 1 0 0 0 1 0 1 5 6 6
```

3. Beach Time

Program Name: Beach.java **Input File: beach.dat**

It's finally summertime! Whale-iam decides to go to the beach to get a tan. However, Texas weather is unpredictable. After years of battling the water cycle, Whale-iam decides to use his CS skills to beat Mother Earth, thereby winning his war against nature.

Input

The first line of input contains an integer `n` indicating the number of cases. The first line of each case contains a String indicating what day of the week July 1st lies upon (`Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`, or `Sunday`). The next line contains one or more inclusive intervals of sunny dates (not the edible variety). The intervals will be in order and never overlap. July has 31 days.

Output

Whale-iam only wants to go to the beach if it's a sunny weekend. Your job is to find exactly what days he can go to the beach. Print the dates in July when Whale-iam can go to beach in ascending order. If there are none, print "`Bad Luck`".

Example Input File

```
2
Monday
1 4-11 17-19 23 25-27
Sunday
2-6 9-13 16-20 23-27 30-31
```

Example Output to Screen

```
6 7 27
Bad Luck
```

4. Coffee Break

Program Name: Coffee.java **Input File: none**

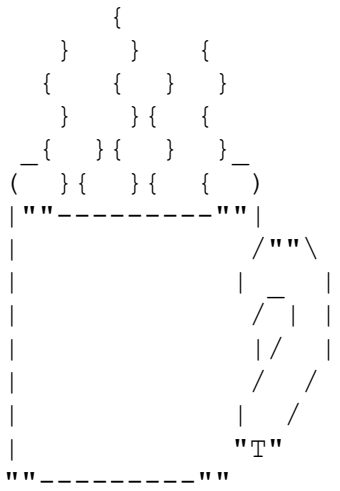
"Coffee is a language in itself." -- Jackie Chan

As a programmer, the best two things to consume during late night coding are cold pizza and hot coffee. We've already bought your pizza. Here is your coffee.

Input
None.

Output
Print the ASCII art of Java.

Output to Screen



5. Exact Change

Program Name: Exact.java **Input File: exact.dat**

Looking into your pocket, TKLau find a few miscellaneous positive and negative dollar bills. He remember that he must repay a sketchy guy named A-nan, but A-nan only accepts exact change. We sure hope TKLau has the right ones. Or else.

Input

There are an unknown number of lines of input. Each line has a series of integers. The first integer indicates the amount of money TKLau must hand over to A-nan. Following in the same line are N integers ($0 < N <= 25$), indicating the values of the bills he have found in his pocket. Each integer value is separated by a single space.

Output

Display "`Yay. TKLau has paid the guy.`" if TKLau is able to pay A-nan exactly. Display "`No! TKLau cannot pay exactly.`" if he cannot. Bills have either negative or positive values, but all values are non-zero.

Example Input File

```
5 -4 51 7 4 9
1 -8 -4 -3 6 9
9 1 1 1 1 1
-7 5 9 8 4 8 -14
8 1 2 3 -1 -2 -3 8
```

Example Output to Screen

```
Yay. TKLau has paid the guy.
Yay. TKLau has paid the guy.
No! TKLau cannot pay exactly.
No! TKLau cannot pay exactly.
Yay. TKLau has paid the guy.
```

6. Fifty Shades of Lavin

Program Name: Shades.java Input File: shades.dat

Lavin's girlfriend likes abstract paintings of rectangles filled with different characters. She will give Lavin the starting coordinate of a rectangle and its dimensions to paint. Lavin paints these rectangles on a starting square canvas of a given dimension. Assume that Lavin is painting in layers and that rectangles will overlap. Lavin begins painting from the top of the direction list. Assume that the upper left hand corner of the painting canvas is the coordinate (0,0). Width is in the horizontal direction while height is in the vertical direction.

Input

The first line represents the number of canvas to follow. The second line contains an integer, representing the side length of the square canvas, which begin with boards full of "X". The third line another integer, representing the number of rectangles Lavin needs to paint. Each subsequent line will have the coordinate of the rectangle's upper left hand corner (row,column), the height and width of the rectangle, and the rectangle character. Assume painted rectangles will not go outside the canvas' borders.

Output

Output the paintings that Lavin needs to create. Separate paintings with a blank line.

Example Input File

```
1
16
4
(2,1) 2 4 O
(4,2) 8 4 -
(0,0) 5 2 ?
(2,5) 4 6 /
```

Example Output to Screen

```
??XXXXXXXXXXXXXX
??XXXXXXXXXXXXXX
??OOO//////XXXXX
??OOO//////XXXXX
??---//////XXXXX
XX---//////XXXXX
XX----XXXXXXXXXX
XX----XXXXXXXXXX
XX----XXXXXXXXXX
XX----XXXXXXXXXX
XX----XXXXXXXXXX
XX----XXXXXXXXXX
XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX
```

7. Inventory Trouble

Program Name: Inventory.java **Input File: inventory.dat**

Justin Beaver is an employee at a big chain-store who is tasked with returns. He has been handed a barcode scanner, shoved behind a service desk, and told to deal with any and all customers. However, due to strange management policies, the receipts list the product's barcode in hexadecimal format. Being paid minimum wage, Justin puts in minimum effort and writes a program to automatically credit customers.

Input

The first line of the input will contain a number `n` indicating the number of customers that day. In each dataset, the first line consists of two integers. The first integer `a` represents the number of unique items purchased, and the second integer `b` represents the number of products (and their individual barcodes) being returned. Each line of the receipt will contain the name of a purchased product, the quantity purchased, the barcode number in hexadecimal, and the price as a double. Each barcode is represented by '`|`' and '`-`', where a line represents a 1 and each space a 0 in binary.

Output

If a customer tries to return a product that they never bought, print "`Unknown barcode <number>, `" with `<number>` replaced with the hex number of the barcode. Hexadecimal numbers should have at least three digits after their "0x" header. If a customer returns too many of something, print "`Too many <name>, `" where `<name>` is the name of the product. Print each error clause only when necessary. Print each erroneous item only once in an error clause (if necessary). If multiple `<number>` or `<name>` need to be printed in the same error statement, separate each with a space and list them in the order that they occur. If there are no errors, print "`All returns correct, `". Do not use plurals in outputs. A customer can't get refunded for items not on his or her receipt. All returns should be followed by "`Credit customer $X.XX `" with properly formatted dollar amounts.

Example Input File

```
2
2 3
LCD_TV 2 0x12A $1000.00
Hat 2 0x33E $20.00
|--|-|-|-
|--|-|-|-
||--|||||-
1 6
Polo_Shirt 3 0x2AB $15.00
|-|-|-|-||
|-|-|-|-||
|-|-|-|-||
|-|-|-|-||
|-----|-|-||
----------
```

Example Output to Screen

```
All returns correct, Credit customer $2,020.00
Unknown barcode 0x82B 0x000, Too many Polo_Shirt, Credit customer $45.00
```

8. RC Circuit

Program Name: RC.java **Input File: rc.dat**

Aiken, one of Clements' many resident physicists, is building an RC circuit, a complete circuit with resistors and capacitors. RC circuits can be used to filter a signal by blocking certain frequencies and passing others.

He begins by building a first order RC circuit composed of only one resistor and one capacitor. Voltage drop across the capacitor at time t follows a simple exponential decay formula:

$$V(t) = V_0 e^{-t/(RC)}$$

where V_0 is the initial voltage across the capacitor in volts, e is Euler's number, t is the independent variable time in seconds, R is the resistance of the resistor in ohms, and C is the capacitance of the capacitor in farads. Help Aiken write a program to display the voltage drop across the capacitor of his first order RC circuit.

Input

The first line of input will contain a single integer `n` that indicates the number of simple RC circuits to follow. Each circuit will contain four non-negative floating point values `v r c t`, representing the initial voltage, V_0, across the capacitor, the resistance of the resistor, the capacitance of the capacitor, and time, respectively. Both `r` and `c` will be nonzero.

Output

Print out the voltage drop across the capacitor at time t for given circuits. Round result to two decimal places and append the unit, volt (V), after.

Note

It is possible for `t` to be zero or infinity. If `t` is infinity, it will be represented by the string literal `"INFINITY"`.

Example Input File

```
4
1.5 10 1 0.5
10 1.0E3 5.0E-3 2
3 1.0E6 1.0E-9 0
9 1000 0.000001 INFINITY
```

Example Output to Screen

```
1.43V
6.70V
3.00V
0.00V
```

9. Reverse Polish Notation

Program Name: RPN.java **Input File: rpn.dat**

"Reverse Polish notation (RPN) is a mathematical notation in which every operator follows all of its operands." - Wikipedia

RChome walks into a bar. "Ouch," he says. He then proceeds to his calculus class. His teacher gives him a set of number sense problems. Seeing the first question, RChome immediately takes out his HP-33 calculator. Unfortunately, the calculator can only do operations in reverse polish notation. Your job from three weeks ago was to teach RChome how to use the HP-33. Solve the problems given to you in RPN.

Input

There are an unknown number of lines of input. Each line has a series of integers and operators. The operators will include `+ - * / %` and `?`, where `/` is integer division, `%` is modulus operation, and `?` is the square operation (x^2).

Output

On each line, print the final result of each dataset as an integer.

Note

For example, operation "`5 + 4`" will be represented as "`5 4 +`" in RPN. As another example, "`(5+4)*8`" will be represented as "`5 4 + 8 *`." Similarly, "`3*(1+2)`" will be represented as "`3 1 2 + *`." Finally, "`(1+2)`^{2}" will be represented as "`1 2 + ?`".

Example Input File

```
5 4 +
5 3 4 + 1 - *
1 2 + ?
5 6 * 1 ? 2 + /
```

Example Output to Screen

```
9
30
9
10
```

10. Spiral of Life

Program Name: Spiral.java **Input File: spiral.dat**

Ammonites are extinct marine animals that roamed the ocean until the Cretaceous era. They look similar to today's land snails, having similar shells. Their helical shell fossils serve as perfect geological time-indicators, as the size of an ammonite's shell grows as it ages. Smurfette's job is to assume that she knows everything about the ammonite and to display its shell's shape given its age.

Input

The first line of input will contain a single integer `n` that indicates the number of ammonite shells to follow. Each shell contains a single integer `m` (`m` >= 1), indicating the ammonite's age.

Output

Assume the shells are spiral shaped and one extra year means one extra complete spiral. A complete spiral means it contains characters on all 4 sides. Spirals start in the top-left corner, and then are drawn in a clockwise direction. Print the shape of each shell, and print a blank line between shells.

Example Input File

```
3
1
2
3
```

Example Output to Screen

```
****
   *
*  *
****

********
       *
****** *
*    * *
* *  * *
* **** *
*      *
********

************
            *
********** *
*         * *
* ****** * *
* *    * * *
* * *  * * *
* * **** * *
* *      * *
* ******** *
*          *
************
```

11. When Life Gives You Lemons

Program Name: Lemons.java **Input File: lemons.dat**

"When life gives you lemons, don't make lemonade - get mad! Demand to see life's manager. Make life rue the day it thought it could give Cave Johnson lemons. Do you know who I am? I'm the man who's gonna burn your house down! With the lemons. I'm going to get my engineers to invent a combustible lemon that burns your house down!" – Cave Johnson, Portal 2

Alright you, now back to testing. See if you can solve these test chambers -- WITH LEMONS! Best luck to you testing subject #42!

Input

The file contains an indefinite number of lemons. Each lemon is described by a line of input. Each line contains, in order, a string (name), a floating-point number (acidity), an integer (size), a boolean (isDefective), and another string (attributes). Different attributes of a lemon will be separated by a ",".

Output

Sort the lemons. First, all defective lemons will be sorted after all good lemons, separating the lemons into two kinds. The rest of the procedure is the same for both kinds. Now independently sort the two kinds of lemons by their acidity in ascending order. Then sort them by size in descending order. If the above two are the same, sort by name in String's natural order. Lastly, for each lemon, sort its attributes in String's natural order, and separate them with ",". When printing, please follow this format: `<name> <acidity rounded to 2 decimal places> <size>: <attributes>`. First print sorted, non-defective lemons, with one on each line. Then print sorted defective lemons, with one on each line. In the final output, replace all "_" with a space.

Example Input File

```
The_Beast 1.0001 100 false explosive,Extremely_Dangerous,Citrus
The_Kid 7.199 50 true not_even_good,the_beast_wanna-be
Cave_Johnson 7.199 50 true true_man_indeed,beast
Bic_Boi 0.2 250 true able_to_fly
Chometheus 5.5 200 false really_none,None
```

Example Output to Screen

```
The Beast 1.00 100: Citrus,Extremely Dangerous,explosive
Chometheus 5.50 200: None,really none
Bic Boi 0.20 250: able to fly
Cave Johnson 7.20 50: beast,true man indeed
The Kid 7.20 50: not even good,the beast wanna-be
```

12. Word Snake

Program Name: Word.java **Input File: word.dat**

El Presidente has never played a word search puzzle before. Frustrated by the complexity of this game, he gives an executive order to you: Make a program that will find all the words in the word search puzzle. A word bank is given for your convenience.

Input

The first line of input will contain a single integer `n` that indicates the number of word search puzzles to follow. The following 10 lines of each dataset will contain a 10x10 character matrix full of lowercase letters. The 11th line of each dataset will have one integer `m`, indicating the number of words in the word bank to follow. The next `m` lines will contain the words to search for.

Output

Print all the words you can find in the order they are provided in the word bank. Words can go horizontally, vertically, or both by making right angle turns — like those in the classic Nokia game, Snake. Letters cannot be reused in the construction of a single word. If no words are found, print "`No Word Found`". Print a blank line between outputs.

Example Input File

```
2
antidisest
tnemhsilba
aooooooooo
roohellooo
iooooooowoo
aooooooooo
nooooooroo
iooooooloo
soooooodoo
mooooooooo
5
zoo
hang
antidisestablishmentarianism
aunt
helloworld
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
3
zooooooooo
helloworld
compsci
```

Example Output to Screen

```
antidisestablishmentarianism
helloworld

No Word Found
```