DKC^3 2014 SHORT COMPUTING PROBLEMS

1. Largest Product in Grid (25 pts)

Imagine a grid composed of two digit numbers (leading zeros are acceptable). In the grid, find the largest product that can be found with four adjacent numbers in a row from the grid. The adjacent numbers could be up, down, left, right, or diagonally in the grid.

Input

Each test case will consist of the following. The first line of input will be the height, *M*, of the grid. The second line of input will be the width, *N*, of the grid. *M* and *N* will be greater than 3 and less than 21. The next *M* lines will be the composition of the grid. There will be 10 test cases with a blank line between each test case.

Output

For each test case output the largest product calculated from the 4 adjacent numbers in the grid.

Input File: **D:\DKC3\GridIn.txt**
Output File: **D:\DKC3\GridOut.txt**

Examples:

Input:
```
4
4
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16

5
6
08 87 22 97 38
49 49 99 40 17
81 49 31 73 55
52 70 95 23 53
22 31 16 71 04
24 47 32 60 99
```

Output:
```
43680
33323697
```

2. Ant (20 pts)

An ant moves on a regular grid of squares that are colored either black or white. The ant is always oriented in one of the cardinal directions (left, right, up or down) and moves from square to adjacent square according to the following rules:

1. If the ant is on a black square, it flips the color of the square to white, rotates 90 degrees counter-clockwise and moves forward one square.
2. If the ant is on a white square, it flips the color of the square to black, rotates 90 degrees clockwise and moves forward one square.

Starting with a grid that is entirely white, how many squares are black after *M* moves of the ant?

Input

Each test case will consist of a number M. This number represents the number of moves the ant will make. There will be 10 test cases with a blank line between each test case.

Output

For each test case out the number of black squares on the grid.

Input File: D:\DKC3\AntIn.txt
Output File: D:\DKC3\AntOut.txt

Examples:

Input:
```
3

10
```

Output:
```
3
6
```

3. Comfortable Distance (15 pts)

There are *N* seats in a row. *P*eople come one after another to fill the seats according to the following rules:

1. No person sits beside another.
2. The first person chooses any seat.
3. Each subsequent person chooses the seat furthest from anyone else already seated, as long as it does not violate rule 1. If there is more than one choice satisfying this condition, then the person chooses the leftmost choice.

Note that due to rule 1, some seats will surely be left unoccupied, and the maximum number of people that can be seated is less than *N* (for *N* > 1).

Input

Each test case will consist of an integer N representing the number of seats in a row with N > 1 and N < 200. There will be 10 test cases with a blank line between each test case.

Output

For each test case output the maximum number of people that can be seated following the 3 rules above.

Input File: D:\DKC3\ComfortIn.txt
Output File: D:\DKC3\ComfortOut.txt

Examples:

Input:
```
15

2
```

Output:
```
7
1
```

The diagram below represents the method used to calculate the maximum number of people that can be seated in a row of 15 by following the rules stated above.

1															⟶	1			4		6		3			5		7		2
	1														⟶		1			4			3			5		6		2
		1													⟶	6		1			4			3			5			2
			1												⟶	4			1		6			3			5			2
				1											⟶	4		5		1		6			3		7			2
					1										⟶	3		5			1		6		4		7			2
						1									⟶	3			5			1		6		4		7		2
							1								⟶	2			4		6		1			5		7		3
								1							⟶	2		6		4		7		1			5			3
									1						⟶	2		5		4		6			1		7			3
										1					⟶	2		5			3		6			1		7		4
											1				⟶	2		6			3			4			1			5
												1			⟶	2			4			3			5			1		6
													1		⟶	2			4			3			5		6		1	
														1	⟶	2			4		6		3			5		7		1

We see that if the first person chooses correctly, the 15 seats can seat up to 7 people. We can also see that the first person has 9 choices to maximize the number of people that may be seated.

4. Read Reduction (10 pts)

Write a program that reads a list of words, replaces the words by numbers indicating their positions within the list then prints the numeric representation of the list to the output file. The first word read (words being separated by spaces, and deleting all punctuation from the word), would be word one, with all following occurrences of that word replaced with the number 1. You will output the space and then the next word that doesn't match word number 1 will become word number two, with all following occurrences of that word being replaced with the number 2 etc… until the whole line of text has been read. Case is insensitive.

Input

There will be 10 test cases. Each test case will consist of a list of words on 1 line.

Output

There will be 1 line of output for each test case.

Input File: D:\DKC3\ReductionIn.txt
Output File: D:\DKC3\ReductionOut.txt

Examples:

Input:

```
Fuzzy Wuzzy was a bear. Fuzzy Wuzzy had no hair. Fuzzy Wuzzy wasn't fuzzy was he?
I have been testy, and I have been testy, and I have been testy.
```

Output:

```
1 2 3 4 5 1 2 6 7 8 1 2 9 1 3 10
1 2 3 4 5 1 2 3 4 5 1 2 3 4
```

5. Circular Reasoning (10 pts)

The number, 197, is called a circular prime because all rotations of the digits: 197, 971, and 719, are themselves prime. There are thirteen such primes below 100: 2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, and 97.

Input

Each test case will provide the min and max number of a range.

Output

Output the number of circular primes inclusively between the two numbers.

Input File: D:\DKC3\CircularIn.txt
Output File: D:\DKC3\CircularOut.txt

Examples:

Input:

```
10 3333
200 222222
```

Output:

```
24
28
```

6. Rummikub (30 pts)

In the game of Rummikub® you get tiles numbered from 1-13 of four different colors (red, blue, orange, and black). The purpose of the game is to make groups or runs from your 14 tiles. Groups are 3 or 4 of a kind of different color tile, and runs are three or more consecutive tiles of the same color. The point value of a group or run is the sum of point values of the tiles. The object is to play all of your tiles. This is like the card game gin or gin rummy. What a great program to write!

The input will have a color character (A, B, C and D) followed by a 1-2 digit number (1-13). So, a group would look like A5, B5, D5; a run would look like A5, A6, A7.

For this program, consider only the following objectives:

- determine whether or not your hand can make groups or runs
- determine the maximum point value of each combination

For example, the following 14 tiles:

A5 B5 C5 A6 A7 B13 A12 C1 C9 D1 D2 D3 D7 D12

can have the following groups: A5 B5 D5 (15 points) or the following runs: A5 A6 A7 (18 points) and D1 D2 D3 (6 points). So the maximum point value for a group or run is 18 points.

Input

The first line consists of the number of data sets. Each data set is on one line and contains 14 strings (a character followed by a 1 or 2 digit integer). Assume the tiles will be in random order.

Output

For each data set, print out either "NO GROUPS OR RUNS" or a sorted list of the group or run with the highest point score. Sort groups alphabetically and runs numerically. This sorted list should have spaces and have the points as the last item on the output line.

Input File: D:\DKC3\RummikubIn.txt
Output File: D:\DKC3\RummikubOut.txt

Examples:

Input:

```
3
A5 B5 C5 A6 A7 B13 A12 C1 C9 D1 D2 D3 D7 D12
A1 A2 B1 B2 C5 C7 C9 C11 B6 B8 B10 B12 D3 D4
D13 C9 C13 A1 A3 A4 A5 A9 A13 B1 B2 B11 B12 B13
```

Output:

```
A5 A6 A7 18
NO GROUPS OR RUNS
A13 B13 C13 D13 52
```

7. Clean Up (20 pts)

You work for the IT department of a major league baseball team. Baseball has an almost infinite number of stats—what a great job for a computer program!

You will write a program to determine who will bat 4th in the batting order. This is called the "clean-up" position and some managers use the person with the best batting average.

You will be given the roster (with their hits and at bats). The clean-up batter will be fourth and have the highest batting average.

You will also be given a series of hits and at bats which may change the best current batting average. Here are three possibilities:

1. The clean-up batter still has the greatest average (no change!)
2. One of the first 3 batters now has the greatest—switch positions with the clean-up hitter.
3. One of the last 5 batters now has the greatest—insert the newest batter at the fourth position and the 4th becomes the 5th, the 5th becomes the 6th… until the old batting position is reached.

Input

There are 10 test cases with a blank line between each test case. For each test case, the first 9 lines consist of the current roster (last name only) and their past hits and at bats. The next 9 lines consist of the hits and at bats for the next series of games (in the same order as the roster).

Output

Show the new roster with the highest batting average in the clean-up position. Separate output with a blank line.

Input File: C:\DKC3\CleanIn.txt
Output File: C:\DKC3\CleanOut.txt

Examples:

Input:

```
Uecker 15 150
Ruth 50 150
Canseco 45 150
Beltre 55 150
Cruz 44 150
Kinsler 51 150
Andrus 40 150
Moreland 35 150
Darvish 10 150
0 10
5 10
0 10
0 10
5 10
9 10
5 10
```

Output:

```
Uecker
Ruth
Canseco
Kinsler
Beltre
Cruz
Andrus
Moreland
Darvish

Johnson
Harold
Cruz
Nelson
Bonds
Aaron
```

```
5 10
1 10

Johnson 10 100
Harold 20 80
Nelson 30 100
Cruz 31 100
Bonds 12 100
Aaron 20 120
Sosa 31 110
Ruth 15 80
Ryan 5 100
5 10
2 12
10 10
8 10
0 20
3 15
2 9
3 16
10 10
```

```
Sosa
Ruth
Ryan
```

8. Bob's Hexabetical Sort (5 pts)

Bob has been tasked with an interesting sorting request from his marketing team. They would like an algorithm that sorts a series of numbers alphabetically by their hexadecimal representation.

Input

There are 10 test cases. Each test case will consist of a series of numbers between 0 and 1,000,000.

Output

The numbers from the test case sorted by their hexadecimal representation.

Input File: C:\DKC3\BobIn.txt
Output File: C:\DKC3\BobOut.txt

Examples:

Input

```
1 10 11 150 170
10 11 12 13 14 15 170 187 204 221 238 255
```

Output

```
1 150 10 170 11
10 170 11 187 12 204 13 221 14 238 15 255
```

9. Bob has a Lisp (15 pts)

Bob is trying to reinvent his own blend of Lisp. The problem is, he keeps getting confused on nesting his parenthetheth. He would like a program that would tell him the index of the firtht bad parenthethith in hith line of code (1 ith the firtht pothition) or "OK" if the parenthetheth are properly nethted. The line doeth not need to thtart with a parenthethith to be correct.

Input

There are 10 test cases.

Output

The index of the first bad parenthesis.

Input File: C:\DKC3\LispIn.txt
Output File: C:\DKC3\LispOut.txt

Examples:

Input

```
def bob(in)((()la de da(x);)()
()()()()()()()
```

Output

```
12
OK
```

10. Digital Printout (25 pts)

Write a program that reads in a digital number printout and outputs the numbers represented. Each test case will have 3 lines of input. There will be between 1 and 10 digits presented, so between 3 and 30 characters per line.

Input

There are 10 test cases. Each test case will be separated by an asterisk.

Output

Each line of output will be a list of the numbers represented by the 10 test cases.

Input File: C:\DKC3\DigitalIn.txt
Output File: C:\DKC3\DigitalOut.txt

Examples:

Input

```
 _     _  _     _ 
|_ |_| _| _|  || |
 _|  | _||_   ||_|
*

 _  _  _ 
| ||_||_|
|_||_| _|
*
```

Output

```
543210
089
```

11. Roman Numeral Multiplication (15 pts)

Write a program that will read in a list of Roman Numerals and print the product of the numerals in Roman numeral format to the output file. Each input line will contain between 2 and 8 Roman numerals separated by a space. The following symbols are used: I for 1, V for 5, X for 10, L for 50, C for 100, D for 500, and M for 1000. Numbers are formed by writing symbols from left to right as a sum, each time using the symbol for the largest possible value. The symbols M, C, X or I may be used at most three times in succession. Only if this rule would be violated, you can use the following rule. When a single I immediately precedes a V or X, it is subtracted. When a single X immediately precedes and L or C, it is subtracted. When a single C immediately precedes a D or M, it is subtracted. For simplicity, we will assume that the product will never be greater than 3,999.

Input

There will be one test case per line. There are 10 test cases.

Output

Print one line of output for each line of input.

Input File: C:\DKC3\RomanIn.txt
Output File: C:\DKC3\RomanOut.txt

Examples:

Input

```
II IX V
L III V II
```

Output

```
XC
MD
```

12. Lapindromes (5 pts)

A lapindrome is a string which when split down the middle, gives two halves that have the same characters and same frequency of those characters. For uneven strings, remove the middle character then split the string.

Input

There will be one test case per line with a total of 10 test cases.

Output

Answer YES if the string is a lapindrome, otherwise answer NO. Following each answer, after dropping the middle character for uneven strings, print each character in the string alphabetically followed by its frequency. There should be a space between the YES/NO and each of the character/frequency pairs.

Input File: C:\DKC3\LapinIn.txt
Output File: C:\DKC3\LapinOut.txt

Examples:

Input

```
badbda
idknedi
```

Output

```
YES a2 b2 d2
NO d2 e1 i2 k1
```

13. Reverse Directions (10 pts)

A set of directions consists of several instructions. The first instruction is of the form "Begin on XXX", indicating the street that the route begins on. Each subsequent instruction is of the form "Left on XXX" or "Right on XXX", indicating a turn onto the specified road. When reversing directions, all left turns become right turns and vice versa, and the order of roads and turns is reversed. Reverse all directions for the given input leaving an empty line between each set of directions.

Input

There are 10 test cases. Each test case consists of 1 to 10 lines of input and each test case is separated by a blank line.

Output

List of instructions to return to the original street. Separate each test case with a blank line.

Input File: C:\DKC3\ReverseIn.txt
Output File: C:\DKC3\ReverseOut.txt

Examples:

Input

```
BEGIN on 28th St NW
LEFT on Hannah Ave NW
RIGHT on Paul Bunyan Dr NW
LEFT on HWY 2
RIGHT on HWY 371

BEGIN on 200th St
RIGHT on HWY 64
RIGHT on Broadway St W
LEFT on HWY 64
LEFT on HWY 200
RIGHT on HWY 71
LEFT on HWY 2
RIGHT on Paul Bunyan Dr NW
LEFT on Hannah Ave NW
RIGHT on 28th St NW
```

Output

```
BEGIN on HWY 371
LEFT on HWY 2
RIGHT on Paul Bunyan Dr NW
LEFT on Hannah Ave NW
RIGHT on 28th St NW
```

BEGIN on 28th St NW
LEFT on Hannah Ave NW
RIGHT on Paul Bunyan Dr NW
LEFT on HWY 2
RIGHT on HWY 71
LEFT on HWY 200
RIGHT on HWY 64
RIGHT on Broadway St W
LEFT on HWY 64
LEFT on 200th St

14. Sequences (25 pts)

Write a program that will read in a list of integers and print the next number in the sequence to the output file.

Notes:

- An arithmetic sequence is a list of integers with the property that the difference between consecutive integers is constant. Example 2, 4, 6, 8, 10… or 10, 8, 6, 4, 2…
- A geometric sequence is a list of integers with the property that the ratio between consecutive integers is constant. Example 2, 4, 8, 16, 32… or 32, 16, 8, 4, 2…
- A Lucas sequence is a list of integers with the property that each number, after the second, is the sum of the previous two integers. Example 2, 4, 6, 10, 16, 26…

Input

There are 10 test cases. Each input line will consist of four to six positive integers separated by a space. Each integer will be no longer than nine digits.

Output

List the next number in the sequence. If the input line doesn't represent one of the sequences listed above, print out the word "Unknown".

Input File: C:\DKC3\SequenceIn.txt
Output File: C:\DKC3\SequenceOut.txt

Examples:

Input:
```
2 4 6 8
16 8 4 2
```

Output:
```
10
1
```

15. Letter Replacement **(20 pts)**

Write a program that will read in a list of words and decipher them based on a control paragraph.

Input

The first five lines of the input file will be a full paragraph including punctuation. You should ignore the punctuation. There will be a blank line after the paragraph. Then there will be 10 test cases. Each test case is 1 line and contains a word from the previous paragraph. However, one of the letters has been substituted with an incorrect letter.

Output

For each of the 10 test case words, replace the incorrect letter with the correct letter and output each correct word on its own line.

Input File: **C:\DKC3\LetterIn.txt**
Output File: **C:\DKC3\LetterOut.txt**

Examples:

Input:

```
The first five lines of the input file will be a full paragraph including punctuation.  You should ignore the
punctuation. There will be a blank line after the paragraph. Then there will be 10 test cases.  Each test
case is 1 line and contains a word from the previous paragraph.  However, one of the letters has been
substituted with an incorrect letter.  Take the 10 test case words and replace the incorrect letter with the
correct letter and output each correct word on its own line.

lxnes
eacz
```

Output:

```
lines
each
```