University Interscholastic League

Computer Science Competition

2014 District 1 Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.
2. All problems have a value of 60 points. Incorrect submissions may be reworked and resubmitted, but will receive a deduction of 5 points for each incorrect submission. Deductions are only included in the team score for problems that are ultimately solved correctly.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Table of Contents

1. Bank

Program Name: Bank.java   Input File: bank.dat

You are a text analyst working for a bank and are given a series of text lines to read and analyze. Management wants a list of all the numbers in each line of text but is not concerned about what they are or what they mean. As one form of analysis, they want you to provide them a sum of these numbers in each line of text.

Your job is to print each number in each line of text, in the order they appear, with single space separation, with their sum on the next line. A number is defined as any whole number. For example, $4.20 would be considered two separate numbers: 4 and 20.

Input

The first line will be a single integer N, the number of text lines to read.
The remaining lines are each text lines to be read. These lines can consist of spaces, alphanumeric characters, and punctuation. The numbers to read will be non-negative integers between 0 and 1073741824, and may contain preceding zeros that are not to be printed. Their sum will also be between 0 and 1073741824.

Output

For every text line there should be two lines of output: a list of numbers and a sum. Between the numbers in the list should be exactly one space. Should there be no numbers, the list line should simply be a dash, "`-`", and the sum 0.

Example Input File

```
6
Stanley worked for a company in a big building as employee number 427.
Employee 427's job was simple: he sat in room 427 and worked at his desk.
Stanley was happy.
That coffee will cost you $4.20.
Linux L1 3.5.0-45-generic #68-Ubuntu SMP Mon Dec 2 21:58:52 x86_64 GNU/Linux
2,014 CE
```

Example Output to Screen

```
427
427
427 427
854
-
0
4 20
24
1 3 5 0 45 68 2 21 58 52 2013 86 64 86 64 86 64
2718
2 14
16
```

2. Code

Program Name: Code.java   Input File: code.dat

In light of the recent NSA revelations regarding cyber security, you have decided to encode all of your Facebook messages to a friend. You first devise a simple replacement cypher of mapping every letter to a different letter, sending your friend a shuffled string of the 26 characters a - z, with `'a'` mapping to the first letter in the shuffled string, `'b'` to the second letter, and so on.

For example, if the shuffled string was `"zikhmpxjfsvqdaolywtecrbugn"`, and the text to encode was `"test"`, the newly encoded string would be `"emte"`, since `"t"` maps to `"e"` in the shuffled string, `"e"` maps to `"m"`, and `"s"` maps to `"t"`, as you can see in the diagram below.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
z	i	k	h	m	p	x	j	f	s	v	q	d	a	o	l	y	w	t	e	c	r	b	u	g	n

To strengthen the encoding process even more, you add an offset value to the process. For example, with an offset value of 1, the encoded letter `"a"` becomes `"b"`, `"b"` becomes `"c"`,…, `"y"` becomes `"z"`, and `"z"` becomes `"a"`. Therefore the encoded string, `"emte"` becomes `"fnuf"`, the final encoded string for `"test"`.

In a decoding example with `"dgr"`, subtract 1 from each character to get `"cfq"`, and in the above mapping, `"u"` maps to `"c"`, `"i"` maps to `"f"`, and `"l"` maps to `"q"`, which results in `"uil"` as the decoded message.

Your job is to write a program that both sends (encodes) and receives (decodes) messages according to the process described above. If a character in a string is not a lowercase letter, leave it as is.

Input

The first line will be a single integer N, the number of protocols to process. The first line for each protocol is the shuffled string of 26 characters, and the integer offset. The bounds on this offset are `-26 < offset < 26`. The next line is a single integer M, the number of messages to encrypt or decrypt. Each message is on a single line, with `'E'` or `D'` for encode or decode, followed by the message to process.

Output

For each protocol, output "Protocol N" on its own line, followed by the resulting string from each message, each on its own line, with a blank line after each protocol set.

Example Input File

```
2
zikhmpxjfsvqdaolywtecrbugn 1
2
E UIL Computer Science rocks
D dgr 2014 xplwu
xyikbnmpvaqouctdjszerwlhfg -2
3
E sup bro!
E abcdefg
D 3947
```

Example Output to Screen

```
Protocol 1
UIL Cpemdfnx Slgnbln xplwu
uil 2014 rocks

Protocol 2
xpb wqr!
vwgizlk
3947
```

3. DNA

Program Name: DNA.java Input File: dna.dat

You are taking an intro biology class, and your first homework is to check whether two equal length DNA strands are "good". In other words, a pair of good strings only consists of the letters A, T, G and C, and for each position in the strand, an A in one of the strings must matched to a T in the other string, or a G matched to a C. This seems pretty easy at first, but as you get going on your homework, you realize the teacher has given you some pretty long strings! Since you would rather not spend forever on the homework, you decide to write a program that analyzes these DNA string pairs for you.

Input

The first line will be a single integer T, the number of problems on your homework. It will be followed by T pairs of lines. Each line will contain a single string representing a DNA strand.

Output

For each problem, output "GOOD" if the string pairs match properly as described above, or "BAD" if there is at least one non-matching pair of characters.

Example Input File

```
3
ATGC
TACG
ATGC
CGTA
AGQ
TCF
```

Example Output to Screen

```
GOOD
BAD
BAD
```

Explanation:

In the first case, the character pairs at all 4 index positions are valid, so the answer is good.
In the second case, the first pair, A-C is not a valid matching, so the answer is bad.
In the third case, the last pair, Q-F, is not valid, so the answer is bad.

4. Exponentiation

Program Name: Exponentiation.java   Input File: exponentiation.dat

Consider a number system that works with ordered pairs of decimal values, e.g. N = (a, b), where N is an ordered pair and a and b are decimal values. The following rules apply for arithmetic in this system:

- Addition: (a, b) + (c, d) = (a+c, b+d)
- Subtraction: (a, b) - (c, d) = (a-c, b-d)
- Multiplication: (a, b) * (c, d) = (a*c - b*d, a*d + b*c)
- Exponentiation simply repeats multiplication.

Given a single instance of N in the above system and an integer m, write a program that prints out the value N^M, i.e., N * N * N… M times.

Input
The first line has an integer `T,` with `T` data sets to follow.
Each data set contains two decimal values a, and b, representing the ordered pair (a, b), followed by an integer M representing the exponent.

Output
For each test case, output the answer as an ordered pair, bracketed as shown, without spaces.

Constraints
```
0 <= T <= 10
1 <= M <= 20
-10 <= a <= 10
-10 <= b <= 10
```

Example Input File
```
3
5.0 1.0 2
4.5 -7.5 3
-1.0 10 5
```

Example Output to Screen
```
(24.0,10.0)
(-668.25,-33.75)
(-49001.0,90050.0)
```

5. Fillings

Program Name: Fillings.java   Input File: fillings.dat

You are trying to edit your high school yearbook, and there are `N` slots where you can have pictures, with `M` pictures available. How many unique ways are there of filling these N slots with M pictures? Every slot should be filled with one of the M pictures, with no duplicates, of course. The number of pictures, M, will always be equal to or greater than the number of slots available, N.

For example, if you have 2 slots, A and B, and 3 pictures, P1, P2, and P3, there are six different ways these three pictures can be arranged in these 2 slots. Slots A and B could contain P1 and P2, or the pictures could be reversed, with P2 in the first slot and P1 in the second. The same possibilities apply to P1 and P3, or P2 and P3.

Input
The first line has the number of test cases, T, with T lines to follow, one for each test case.
Each test case line contains two integers N, M.

Output
For each test case, output the number of ways as demonstrated above, one per line.

Constraints
```
0 <= T <= 10
1 <= M <= 10
1 <= N <= 10
N <= M
```

Example Input File
```
3
2 3
2 10
4 9
```

Example Output to Screen
```
6
90
3024
```

6. JSON

Program Name: Json.java   Input File: json.dat

You are writing a program that needs to interact with javascript programs and find that you need to share objects in the JSON format. You will be given a list of instructions then be requested to print certain objects in the JSON format. JSON objects are instantiated at first by simply stating an object and assigning a value to an associated key, like this: "object.key = value".

For example: `cake.ingredient = sugar` and `cake.bake = oven` instantiates the `cake` object, and assigns two mappings: `ingredient = sugar`, and `bake = oven`.

You can then print these objects with the command `print(<object>)`. The output format for printing is `{key : value, key : value… key : value}` where curly braces are at the beginning and end, a comma and space after all values but the last one, with spaces on each side of each colon. They are printed in the order they were assigned to the object, and for this exercise no key will be updated with a new value. All printed objects are guaranteed to have at least one attribute assigned before a print statement.

For example: `print(cake)` will print "`{ingredient : sugar, bake : oven}`"

Input

The first line will contain a value T with T command lines to follow. Each command line will be either an assignment statement or a print command. The word "print" is reserved and will not be used as an identifier for any object, key, or value.

Output

Each time a print command is encountered, print out the object.

Constraints

`1 <= T < 20`

Example Input File

```
6
cake.ingredient = sugar
print(cake)
cake.bake = oven
print(cake)
dog.treat = bone
print(dog)
```

Example Output to Screen

```
{ingredient : sugar}
{ingredient : sugar, bake : oven}
{treat : bone}
```

7. M&M's

Program Name: MandMs.java   Input File: mandms.dat

My kid brother collected a lot of M&M's on Halloween, and being a little guy, he was playing with them before he ate them. He showed me his game -- he would spread a packet of M&M's out on the table, and either eat exactly one, or would arrange them in a rectangle of width > 1 (only if that was possible) and eat all rows but one in a single mouthful. Even if a rectangular arrangement were possible, he might still decide to eat only one. He would the repeat this with the remaining M&M's.

He asked me whether I could tell him how many mouthfuls it would take him, at a minimum, to finish off a packet, if he counted the M&M's he started out with. I could not tell him, but perhaps you can.

Input

The first line has the number of test cases, `T`. `T` lines follow.
Each test case contains a single integer `M`, the number of M&M's in the packet.

Output

The output should have a single integer for each test case, the fewest mouthfuls he would take to finish the packet.

Constraints

```
T <= 100000
1 <= M <= 1000000000
```

Example Input File

```
5
1
2
3
5
100
```

Example Output to Screen

```
1
2
3
4
3
```

8. Rabbits

Program Name: Rabbits.java Input File: rabbits.dat

In the year 1202, the mathematician Fibonacci decided to investigate how rabbits breed under ideal circumstances.

Assuming that a pair of rabbits never dies, and produces a new pair of rabbits every month (except the month they are born), he wanted to answer how many pairs of rabbits there were after N months. These restrictions led him to come up with the recursive formula $f(n) = f(n-1) + f(n-2)$.

Now, imagine he decides to change his model, so that each pair of rabbits doesn't produce a new pair until 2 months after they are born. Figure out the new formula and write a program that gives the number of pairs of rabbits in this model after N months. The answer is guaranteed to fit within a long.

Input

The first line will be a single integer T, the number of test cases.
Each test case consists of a single integer on its own line, N, or the number of months to plug into the formula.
$0 \leq N \leq 115$

Output

For each test case, output a single integer on its own line that is the number of rabbits after N months according to Fibonacci's new model.

Example Input File

```
5
1
2
3
5
14
```

Example Output to Screen

```
1
1
2
4
129
```

Explanation:
Assume at the start, there is one new pair of rabbits.
At 1 month they are not mature enough to conceive, so they remain as the only pair.
At 2 months they reach reproductive age, but have not yet produced children.
At 3 months their first pair of rabbits are born, so now there are two pairs of rabbits.
At 5 months, there are four pairs of rabbits: the original pair, the first pair of offspring, now two months old, the second pair of offspring, age 1 month, and the newly born pair.

9. Right Hand

Program Name: RightHand.java   Input File: righthand.dat

Most everyone knows about the right-hand rule as a maze escape strategy, or at least they should. In case you don't, it goes like this: if you hold your right hand against the wall to your right at all times, and move through the maze, you shall eventually find your way out – assuming that the maze isn't an infinite trap with no way out.

We've designed a few mazes, but don't necessarily know if they're solvable or not. We suppose it's best to figure out if someone can get out using the right-hand rule after putting them in, so it's your job to let us know!

You will always begin at the top left corner, which is not a wall, and attempt to find your way to the only exit located in the bottom right corner, using the right hand rule. Except for the edges of the bottom right corner exit location, all outside edges of the matrix are walls.

Input

The first integer T will represent how many mazes there are to follow. For each maze, there will be a line containing a single number `n`, which represents both the number of rows and columns of the subsequent square matrix. The next n lines each contain a row of length n. Each character in each row is either a `'.'` or a `'#'`. All `'#'`'s are interior walls, you are only allowed to walk on `'.'`s, and can only move up, down, left, or right – no diagonal movement is possible – the tiny diagonal gap is way too narrow for you to fit through.

Output

For each test case print out `YES` if you discover the maze to be "escape"-able, or `NO` if there is no way out.

Example Input File

```
3
3
...
.##
.#.
5
.###.
...#.
.###.
...##
.#...
10
.....#..##
#.#..###.#
#...#.#...
.#.##.##..
....#..##.
..#....#..
.####.#.#.
#.#.#.##..
.........#
.#.....#..
```

Example Output to Screen

```
NO
YES
YES
```

10. Stealing Gold

Program Name: Stealing.java   Input File: stealing.dat

A thief has broken into a bank where he found `N` gold bars in the vault, and wants to steal all of them before the police arrive. However he can only carry at most `M` gold bars at a time since they are so heavy. It takes `t` seconds to go between his car and the location of the gold bars in the bank. What is the least amount of time in which he can make off with all the gold bars?

Input

The first line has an integer value `T` with `T` test cases to follow, one line for each test case.
Each test case has three integers separated by spaces -- `N, M, T`.

Output

For each test case, print out a line with the minimum number of seconds he needs to escape with all of the gold, starting from the time he picks up the first `M` ounces.

Constraints

```
1 <= T <= 10
1 <= t <= 100
1 <= N <= 100
1 <= M <= 100
```

Example Input File

```
3
3 2 10
35 68 42
25 70 1
```

Example Output to Screen

```
30
42
1
```

11. Tale of a Tail

Program Name: Tale.java   Input File: none

In Lewis Carroll's "Alice's Adventures in Wonderland", there is a mouse who tells his long and sad tale to Alice. Alice misunderstands him and thinks he is talking about his 'tail', and imagines the mouse's story to be shaped like a mouse's tail. Please print out the start of the mouse's story, "This is a very long and unhappy tale.", exactly as in the sample output.

Input
None.

Output
Output the words as shown in the Example Output.

Example Output to Screen
```
This is a
 very long
  and un-
 happy
tale
  .
```

12. Teams

Program Name: Teams.java   Input File: teams.dat

Adults have been called to participate in the creation of computer science UIL exams. They believe that they need to reassert their superiority in the field of computer science because ***"kids these days just don't know what it was like working in assembly and command line programming before there were high level languages and fancy IDE's."***

To decide whether the older generation or newer generation is better, Texas is having a state wide coding competition between two teams: the ***young whippersnappers*** (the youngest half of contestants) and the ***old geezers***. To be fair (not sure about that, really), the young team will have a one person advantage when there is an odd number of contestants. The problem is that not all contestants can start at the beginning of the meet, so the teams will be rearranged every time a new person arrives so that the youngest half of contestants are all on the young team, and the oldest on the older team, with someone on one team possibly moving to the other team, depending on the age of who joins the meet. Each person has an age and a rating of his/her coding ability, and teams are rated by taking the sum of the coding abilities of their members. Once a person joins the contest, he or she remains for the duration of the meet.

For example, say the first team member that arrives is age 11 and has an experience rating of 98. She joins the younger team and since there is no one on the older team yet, the rating difference between the teams is simply 98. The next person to join the meet is age 16, with a rating of 149, and is automatically on the older team, for now anyway. The new rating difference is 149 – 98, which is 51. The next person to join is age 31, rated at 116. Clearly he is an "old geezer", for the moment, and since there are now an odd number of contestants, the 16 year-old switches to the young team, making the rating difference 131 (98+149-116). This process continues until all contestants have joined.

It can be difficult to keep track of the teams, so you have been enlisted to automate this with a computer program. Contestants will arrive one at a time, and you will print out the rating difference between the two teams each time. After each contestant arrives, rearrange the teams accordingly, and output the difference between the new ratings.

Input

The first line contains the number of contestants N. The following N lines describe the contestants in order as they come to compete. Each person is described by two integers, his or her age A and rating R. Each age will be unique.

Output

Every time a new contestant joins, reorganize the teams as described above, calculate the new team ratings, and output the difference between team ratings.

Example Input File

```
5
11 98
16 149
31 116
12 40
53 14
```

Example Output to Screen

```
98
51
131
127
157
```