Exercise: Data Types and Variables

Problems for exercise and homework for the "C# Fundamentals" course @ SoftUni You can check your solutions in Judge

1. Integer Operations

Create a program that receives four integer numbers.

You should perform the following operations:

- Add first to the second.
- **Divide** (integer) the result of the first operation by the third number.
- **Multiply** the result of the second operation by the fourth number.

Print the result after the last operation.

Constraints

- First number will be in the range [-2,147,483,648...2,147,483,647].
- Second number will be in the range [-2,147,483,648...2,147,483,647].
- Third number will be in the range [-2,147,483,648...2,147,483,647].
- Fourth number will be in the range [-2,147,483,648...2,147,483,647].

Examples

Input	Output	Input	Output
10	30	15	42
20		14	
3		2	
3		3	

2. Sum Digits

Create a program that receives a single integer. Your task is to find the sum of its digits.

For example: $12345 \rightarrow 1 + 2 + 3 + 4 + 5 = 15$

Examples

Input	Output
245678	32
97561	28
543	12

3. Elevator

Calculate how many courses will be needed to elevate n persons by using an elevator of the capacity of p persons. The input holds two lines: the number of people n and the capacity p of the elevator.

















Examples

Input	Output	Comments
17 3	6	5 courses * 3 people + 1 course * 2 persons
4 5	1	All the persons fit inside in the elevator. Only one course is needed.
10 5	2	2 courses * 5 people

Hints

- You should divide n by p. This gives you the number of full courses (e.g. 17/3 = 5).
- If **n** does not divide **p** without a remainder, you will need one additional partially full course (e.g. **17** % **3** =
- Another approach is to round up \mathbf{n} / \mathbf{p} to the nearest integer (ceiling), e.g. $\mathbf{17} / \mathbf{3} = \mathbf{5.67} \rightarrow \mathbf{rounds}$ up to $\mathbf{6}$.
- Sample code for the round-up calculation:

int courses = (int)Math.Ceiling((double)n / p);

4. Sum of Chars

Create a program, which sums the ASCII codes of **n** characters and prints the **sum** on the console.

Input

- On the **first line**, you will receive **n** the number of **lines**, which will **follow**.
- On the next **n lines** you will receive letters from the **Latin** alphabet.

Output

Print the total sum in the following format:

"The sum equals: {totalSum}"

Constraints

- n will be in the interval [1...20].
- The characters will always be either upper or lower-case letters from the English alphabet.
- You will always receive one letter per line.

Input	Output			
5	The	sum	equals:	399
Α				
b				
b C d E				
d				
E				

Input			Output	
12	The	sum	equals:	1263
S				
0				
f				
t				
U				
n				
i				
R				
u				



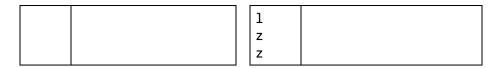












5. Print Part of the ASCII Table

Find online more information about ASCII (American Standard Code for Information Interchange) and write a program that prints part of the ASCII table of characters at the console. On the first line of input, you will receive the char index you should start with, and on the second line - the index of the last character you should print.

Examples

Input	Output
60 65	< = > ? @ A
35 49	# \$ % & ' () * + , / 0 1

6. Triples of Latin Letters

Create a program that receives an integer n and print all triples of the first n small Latin letters, ordered alphabetically:

Input	Output
3	aaa
	aab
	aac
	aba
	abb
	abc
	aca
	acb
	acc
	baa
	bab
	bac
	bba
	bbb
	bbc
	bca bcb
	bcc
	caa
	cab
	cac
	cba
	cbb
	cbc
	cca
	ccb
	ссс











Hints

Perform 3 nested loops from 0 to n-1.

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
        }
}
```

For each iteration generate new letters

```
char firstChar = (char)('a' + i);
// TODO: Find the other two characters
```

7. Water Overflow

You have a water tank with a capacity of 255 liters. On the next n lines, you will receive liters of water, which you have to pour into your tank. If the capacity is not enough, print "Insufficient capacity!" and continue reading the next line. On the last line, print the liters in the tank.

Input

The **input** will be on two lines:

- On the **first line**, you will receive \mathbf{n} the number of **lines**, which will **follow**.
- On the next n lines, you will receive quantities of water, which you have to pour into the tank.

Output

Every time you do not have **enough capacity** in the tank to pour the given liters, **print**:

Insufficient capacity!

On the last line, **print** only the **liters** in the **tank**.

Constraints

- **n** will be in the interval [1...20]
- liters will be in the interval [1...1000]

Input	Output		
5	<pre>Insufficient capacity!</pre>		
20	240		
100			
100			
<mark>100</mark>			
20			

Input	Output
1 1000	<pre>Insufficient capacity! 0</pre>

















Input		Output
7	105	
10		
20		
30		
10		
5		
10		
20		

Input	Output
4	Insufficient capacity!
250 10	<pre>Insufficient capacity! Insufficient capacity!</pre>
20	250
<mark>40</mark>	

8. Beer Kegs

Create a program, which calculates the volume of n beer kegs. You will receive in total 3 * n lines. Every three lines will hold information for a single keg. First up is the model of the keg, after that is the radius of the keg, and lastly is the **height** of the keg.

Calculate the volume using the following formula: $\pi * r^2 * h$.

In the end, print the model of the biggest keg.

Input

You will receive **3** * **n** lines. Each group of lines will be on a new line:

- First model string
- Second -radius floating-point number
- Third **height integer** number

Output

Print the model of the biggest keg.

Constraints

- n will be in the interval [1...10].
- The radius will be a floating-point number in the interval [1...3.402823E+38].
- The height will be an integer in the interval [1...2147483647].

Input	Output
3	Keg 2
Keg 1	
10	
10	
Keg 2	
20	
20	
Keg 3	
10	
30	
30	

Input	Output		
Smaller Keg 2.41 10 Bigger Keg 5.12	Bigger Keg		











9. *Spice Must Flow

Spice is Love, Spice is Life. And most importantly, Spice must flow. It must be extracted from the scorching sands of Arrakis, under the constant threat of giant sandworms. To make the work as efficient as possible, the Duke has tasked you with the creation of management software.

Create a program that calculates the total amount of spice that can be extracted from a source. The source has a starting yield, which indicates how much spice can be mined on the first day. After it has been mined for a day, the yield drops by 10, meaning on the second day it'll produce 10 less spice than on the first, on the third day 10 less than on the second, and so on (see examples). A source is considered profitable only while its yield is at least 100 when less than 100 spices are expected in a day, abandon the source.

The mining crew consumes 26 spices every day at the end of their shift and an additional 26 after the mine has been exhausted. Note that the workers cannot consume more spice than there is in storage.

When the operation is complete, print on the console, on two separate lines, how many days the mine has operated and the total amount of spice extracted.

Input

You will receive a **number**, representing the **starting yield** of the source.

Output

Print on the console, on two separate lines, how many days the mine has operated and the total amount of spice extracted.

Constraints

The starting yield will be a positive **integer** within the range [0...2147483647].

Examples

Input	Output	Explanation
111	2 134	On day 1 we extract 111 spices and at the end of the shift, the workers consume 26, leaving 85. The yield drops by 10 to 101.
		On day 2 we extract 101 spices, the workers consume 26, leaving 75. The total is 160 and the yield has dropped to 91.
		Since the expected yield is less than 100, we abandon the source. The workers take another 26, leaving 134. The mine has operated for 2 days.
450	36 8938	

*Pokemon 10.

A Pokemon is a special type of pokemon which likes to Poke others. But at the end of the day, the Pokemon wants to keep statistics, about how many pokes it has managed to make.

The Pokemon pokes his target and then proceeds to poke another target. The distance between its targets reduces his poke power.

You will be given the poke power the Pokemon has, N – an integer.

Then you will be given the distance between the poke targets, M – an integer.













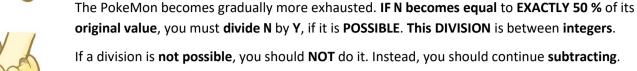






Then you will be given the exhaustionFactor Y – an integer. Your task is to start subtracting M from N until N becomes less than M, i.e. the Pokemon does not have enough power to reach the next target.

Every time you subtract M from N that means you've reached a target and poked it successfully. **COUNT** how many targets you've poked – you'll need that count.



When N becomes less than M, you must take what has remained of N and the count of targets you've poked, and print them as output.

After dividing, you should continue subtracting from N, until it becomes less than M.

NOTE: When you are calculating percentages, you should be PRECISE at maximum.

Example: 505 is **NOT EXACTLY 50** % from **1000**, its **50.5** %.

Input

- The input consists of 3 lines.
- On the **first line**, you will receive **N** an **integer**.
- On the **second line**, you will receive **M** an **integer**.
- On the **third line**, you will receive **Y** an **integer**.

Output

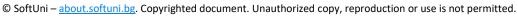
- The output consists of **2 lines**.
- On the first line, print what has remained of N, after subtracting from it.
- On the **second line**, print the **count** of **targets**, you've managed to poke.

Constrains

- The integer N will be in the range [1...2000000000].
- The integer **M** will be in the range [1...1000000].
- The integer Y will be in the range [0...9].
- Allowed time / memory: 16 MB / 100ms.

Input	Output	Comments
5	1 2	N = 5, M = 2, Y = 3.
3	_	We start subtracting M from N .
		N – M = 3. 1 target poked.
		N – M = 1 . 2 targets poked.
		N < M.
		We print what has remained of N, which is 1.
		We print the count of targets , which is 2 .
10	2	N = 10, M = 5, Y = 2.
5	1	We start subtracting M from N.





















2	N - M = 5. (N is still not less than M, they are equal).	
	N became EXACTLY 50 % of its original value.	
	5 is 50 % from 10 . So we divide N by Y .	
	N/Y = 5/2 = 2. (INTEGER DIVISION).	

*Snowballs 11.

Tony and Andi love playing in the snow and having snowball fights, but they always argue about which makes the best snowballs. They have decided to involve you in their fray by making you write a program, which calculates snowball data and outputs the best snowball value.

You will receive N – an **integer**, the **number** of **snowballs** being made by Tony and Andi.

For each snowball you will receive 3 input lines:

- On the first line, you will get the snowballSnow an integer.
- On the **second line** you will get the snowballTime an **integer**.
- On the **third line**, you will get the snowballQuality an **integer**.

For each snowball you must calculate its snowballValue by the following formula:

(snowballSnow / snowballTime) ^ snowballQuality

In the end, you must print the **highest** calculated **snowballValue**.

Input

- On the first input line, you will receive N the number of snowballs.
- On the **next N * 3 input lines**, you will be receiving **data** about **snowballs**.

Output

- As output, you must print the **highest** calculated **snowballValue**, by the formula, **specified above**.
- The output format is:

{snowballSnow} : {snowballTime} = {snowballValue} ({snowballQuality})

Constraints

- The number of snowballs (N) will be an integer in the range [0...100].
- The snowballSnow is an integer in the range [0...1000].
- The snowballTime is an integer in the range [1...500].
- The snowballQuality is an integer in the range [0...100].
- Allowed working time/memory: 100ms / 16MB.

Input	Output			
2	10 : 2 = 125 (3)			
10				
2				
3				
5				
5				











5						
3	10	:	5	=	128	(7)
10						
5						
7						
16						
4						
2						
20						
2						
2						













