Lab: Data Types and Variables

Problems for exercises and homework for the ["Technology Fundamentals" course @ SoftUni.](https://softuni.org)

You can check your solutions in [Judge.](https://judge.softuni.org)

I. Integer and Real Numbers

1. Convert Meters to Kilometers

You will be given an integer that will be distance in meters. Write a program that converts meters to kilometers formatted to the second decimal point.

Examples

Input	Output
1852	1.85
798	0.80

2. Pounds to Dollars

Write a program that converts British pounds to US dollars formatted to 3th decimal point.

1 British Pound = 1.31 Dollars

Examples

Input	Output
80	104.800
39	51.090

3. Exact Sum of Real Numbers

Write program to enter **n** numbers and calculate and print their **exact sum** (without rounding).

Examples

Input	Output
3 1000000000000000000 5 10	1000000000000000015
2 0.00000000003 333333333333.3	333333333333.30000000003

Hints

Use **`BigDecimal`** to not lose precision.

II. Data Types and Type Conversion

4. Town Info

You will be given 3 lines of input. On the first line you will be given the name of the town, on the second – the population and on the third the area. Use the correct data types and print the result in the following format:

"**Town {town name} has population of {population} and area {area} square km**".

Examples

Input	Output
Sofia 1286383 492	Town Sofia has population of 1286383 and area 492 square km.

5. Concat Names

Read two names and a delimiter. Print the names joined by the delimiter.

Examples

Input	Output
John Smith ->	`John->Smith`
Jan White <->	`Jan<->White`
Linda Terry =>	`Linda=>Terry`

6. Chars to String

Write a program that reads 3 lines of input. On each line you get a single character. Combine all the characters into one string and print it on the console.

Examples

Input	Output
a b c	abc
% 2 o	%2o

1 5 p	15p

7. Reversed Chars

Write a program that takes 3 lines of characters and prints them in reversed order with a space between them.

Examples

Input	Output
A B C	C B A
1 L &	& L 1

8. Lower or Upper

Write a program that prints whether a given character is upper-case or lower-case.

Examples

Input	Output
L	upper-case
f	lower-case

9. Centuries to Minutes

Write program to enter an integer number of **centuries** and convert it to **years**, **days**, **hours** and **minutes**.

Examples

Input	Output
1	1 centuries = 100 years = 36524 days = 876576 hours = 52594560 minutes
5	5 centuries = 500 years = 182621 days = 4382904 hours = 262974240 minutes

Hints

- Use appropriate data type to fit the result after each data conversion.
- Assume that a year has 365.2422 days at average (the Tropical year).

Solution

You might help yourself with the code below:

```java
Scanner scanner = new Scanner(System.in);

int centuries = Integer.parseInt(scanner.nextLine());
int years = centuries * 100;
int days = (int) (years * 365.2422);
int hours = days * 24;
int minutes = hours * 60;

System.out.printf("%d centuries = %d years = %d days = %d hours = %d minutes",
        centuries,
        years,
        days,
        hours,
        minutes
);
```

10. Special Numbers

A **number** is **special** when its **sum of digits is 5, 7 or 11**.

Write a program to read an integer **n** and for all numbers in the range **1…n** to print the number and if it is special or not (**`True`** / **`False`**).

Examples

Input	Output
15	1 -> False 2 -> False 3 -> False 4 -> False 5 -> True 6 -> False 7 -> True 8 -> False 9 -> False 10 -> False 11 -> False 12 -> False 13 -> False 14 -> True 15 -> False

Hints

To calculate the sum of digits of given number **num**, you might repeat the following: sum the last digit (**num % 10**) and remove it (**sum = sum / 10**) until **num** reaches **0**.

III. Variables

11. Refactor Volume of Pyramid

You are given a **working code** that finds the **volume of a pyramid**. However, you should consider that the variables exceed their optimum span and have improper naming. Also, search for variables that **have multiple purpose**.

Code

Sample Code

```
Scanner scanner = new Scanner(System.in);
double dul, sh, V = 0;
System.out.print("Length: ");
dul = Double.parseDouble(scanner.nextLine());
System.out.print("Width: ");
sh = Double.parseDouble(scanner.nextLine());
System.out.print("Height: ");
V = Double.parseDouble(scanner.nextLine());
V = (dul + sh + V) / 3;
System.out.printf("Pyramid Volume: %.2f", V);
```

Hints

- **Reduce the span** of the variables by declaring them in the moment they receive a value, not before
- Rename your variables to **represent their** real **purpose** (example: "dul" should become length, etc.)
- Search for variables that have multiple purpose. If you find any, **introduce a new variable**.

12. Refactor Special Numbers

You are given a **working code** that is a solution to **Problem 9. Special Numbers**. However, the variables are **improperly named, declared before** they are needed and some of them are used for multiple things. Without using your previous solution, **modify the code** so that it is **easy to read and understand**.

Code

Sample Code

```
Scanner scanner = new Scanner(System.in);
int kolkko = Integer.parseInt(scanner.nextLine());
int obshto = 0;
int takova = 0;
boolean toe = false;
for (int ch = 1; ch <= kolkko; ch++) {
    takova = ch;
    while (ch > 0) {
        obshto += ch % 10;
        ch = ch / 10;
    }
    toe = (obshto == 5) || (obshto == 7) || (obshto == 11);
    System.out.printf("%d -> %b%n", takova, toe);
    obshto = 0;
    ch = takova;
}
```

Hints

- Reduce the span of the variables by declaring them in the moment they receive a value, not before
- Rename your variables to represent their real purpose (example: "toe" should become isSpecialNum, etc.)
- Search for variables that have multiple purpose. If you find any, introduce a new variable

Exercise: Data Types and Variables

Problems for exercises and homework for the ["Technology Fundamentals" course @ SoftUni.](https://softuni.org)

You can check your solutions in [Judge.]()

1. Integer Operations

Read **four integer** numbers.
Add first to the second, divide (integer) the sum by the third number and multiply the result by the fourth number.
Print the result.

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
10 20 3 3	30

Input	Output
15 14 2 3	42

2. Sum Digits

You will be given a single **integer**. Your task is to find the **sum of its digits**.

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 with **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	1	All the persons fit inside in the elevator.

5		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 = 2).
- Another approach is to round up **n / p** to the nearest integer (ceiling), e.g. 17/3 = 5.67 → rounds up to 6.
- Sample code for the round-up calculation:

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

4. Sum of Chars

Write a program, which **sums the ASCII codes** of **n** characters.
Print 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**

Examples

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

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

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
69 79	E F G H I J K L M N O
97 104	a b c d e f g h
40 55	( ) * + , - . / 0 1 2 3 4 5 6 7

6. Triples of Latin Letters

Write a program to read an integer **n** and print all **triples** of the first **n small Latin letters**, ordered alphabetically:

Examples

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 ccc

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 other two characters
```

Concat all characters in a string and print it. You can use **`String.format()`**.

```
System.out.printf("%c%c%c%n", firstChar, secondChar, thirdChar);
```

7. Water Overflow

You have a **water tank** with capacity of **255 liters**.

On the next **n** lines, you will receive **liters of water**, which you have to **pour** in 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 **n** – the number of **lines**, which will **follow**
- On the next **n lines** – you receive **quantities** of water, which you have to **pour** in 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]**

Examples

Input	Output
5 20 100 100	Insufficient capacity! 240

Input	Output
1 1000	Insufficient capacity! 0

100 20				

Input	Output		Input	Output
7 10 20 30 10 5 10 20	105		**4** 250 10 20 40	Insufficient capacity! Insufficient capacity! Insufficient capacity! 250

8. Beer Kegs

Write a program, which calculates the volume of **n** beer kegs.
You will receive in total **3 * n** lines. **Each 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: **`π * r^2 * h`**.

At 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]**

Examples

Input	Output		Input	Output
3 Keg 1 10 10 Keg 2 20 20 Keg 3 10 30	Keg 2		**2** Smaller Keg 2.41 10 Bigger Keg 5.12 20	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 constant threat of giant sand worms. To make the work as efficient as possible, the Duke has tasked you with the creation of a management software.

Write 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 spice is expected in a day, abandon the source.

The mining crew **consumes** 26 spice **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 range [0 … 2 147 483 647]

Examples

Input	Output	Explanation
111	2 134	**Day 1** we extract 111 spice and at the end of the shift, the workers consume 26, leaving 85. The yield drops by 10 to 101. **Day 2** we extract 101 spice, 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 2 days.

10. *Poke Mon

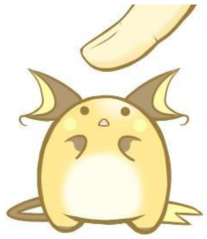
A Poke Mon is a special type of pokemon which likes to Poke others. But at the end of the day, the Poke Mon wants to keeps statistics, about how many pokes it has managed to make.

The Poke Mon pokes his target, and then proceeds to poke another target. The **distance** between his **targets reduces** his **poke power**.

You will be **given** the **poke power** the Poke Mon has, **N** – an **integer**.

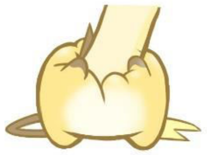
Then you will be **given** the **distance** between the **poke targets**, **M** – an **integer**.

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 Poke Mon 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**.

The Poke Mon becomes gradually more exhausted. **IF N becomes equal** to **EXACTLY 50 %** of its **original value**, you must **divide N** by **Y**, if it is **POSSIBLE**. **This DIVISION** is between **integers**.



If a division is **not possible**, you should **NOT** do it. Instead, you should continue **subtracting**.

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

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.

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, 2.000.000.000]**.
- The integer **M** will be in the **range [1, 1.000.000]**.
- The integer **Y** will be in the **range [0, 9]**.
- Allowed time / memory: **16 MB / 100ms**.

Examples

Input	Output	Comments
5 2 3	1 2	`N = 5, M = 2, Y = 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 5 2	2 1	`N = 10, M = 5, Y = 2.` `We start` **`subtracting M`** `from` **`N.`** **`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`**`).`

11. *Snowballs

Tony and Andi love playing in the snow and having snowball fights, but they always argue 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`

At 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 **range [0, 100]**.
- The **`snowballSnow`** is an **integer** in **range [0, 1000]**.
- The **`snowballTime`** is an **integer** in **range [1, 500]**.
- The **`snowballQuality`** is an **integer** in **range [0, 100]**.
- Allowed working **time** / **memory**: **100ms** / **16MB**.

Examples

Input	Output
2 10 2 3 5 5 5	`10 : 2 = 125 (3)`
3 10 5 7 16 4 2 20 2 2	`10 : 5 = 128 (7)`

More Exercise: Data Types and Variables

You can check your solutions in Judge.

1. Data Type Finder

You will receive an input until you receive "END". Find what **data type** is the input. Possible data types are:

- Integer
- Floating point
- Characters
- Boolean
- Strings

Print the result in the following format: "**{input}** is **{data type}** type"

Examples

Input	Output
5 2.5 true END	5 is integer type 2.5 is floating point type true is boolean type
a asd -5 END	a is character type asd is string type -5 is integer type

From Left to the Right

You will receive a number which represents how many lines we will get as an input. On the next N lines, you will receive a string with 2 numbers separated by a single space. You need to compare them. If the left number is greater than the right number, you need to print the sum of all digits in the left number, otherwise print the sum of all digits in the right number.

Examples

Input	Output
2 1000 2000 2000 1000	2 2
4 123456 2147483647 5000000 -500000 97766554 97766554 9999999999 8888888888	46 5 49 90

2. Floating Equality

Write a program that safely compares floating-point numbers (double) with precision eps = 0.000001. Note that we cannot directly compare two floating-point numbers **a** and **b** by a==b because of the nature of the floating-point arithmetic. Therefore, we assume two numbers are equal if they are more closely to each other than some fixed constant eps.

You will receive two lines, each containing a floating-point number. Your task is to compare the values of the two numbers.

Examples

Number a	Number b	Equal (with precision eps=0.000001)	Explanation
5.3	6.01	False	The difference of 0.71 is too big (> eps)
5.00000001	5.00000003	True	The difference 0.00000002 < eps
5.00000005	5.00000001	True	The difference 0.00000004 < eps
-0.0000007	0.00000007	True	The difference 0.00000077 < eps
-4.999999	-4.999998	False	Border case. The difference 0.0000001== eps. We consider the numbers are different.
4.999999	4.999998	False	Border case. The difference 0.0000001 == eps. We consider the numbers are different.

3. Refactoring: Prime Checker

You are given a program that checks if numbers in a given range [2...N] are prime. For each number is printed "{number} -> {true or false}". The code however, is not very well written. Your job is to modify it in a way that is easy to read and understand.

Code

Sample Code

```
Scanner chetec = new Scanner(System.in);

int ___Do___ = Integer.parseInt(chetec.nextLine());
for (int takoa = 2; takoa <= ___Do___; takoa++) {
    boolean takovalie = true;
    for (int cepitel = 2; cepitel < takoa; cepitel++) {
        if (takoa % cepitel == 0) {
            takovalie = false;
            break;
        }
    }
    System.out.printf("%d -> %b%n", takoa, takovalie);
}
```

Examples

Input	Output
5	2 -> true 3 -> true 4 -> false 5 -> true

4. Decrypting Messages

You will receive a **key** (**integer**) and **n** characters afterward. Add the key to each of the characters and append them to **message**. At the end print the message, which you decrypted.

Input

- On the **first line**, you will receive the **key**
- On the **second line**, you will receive **n** – the number of **lines**, which will **follow**
- On the next **n lines** – you will receive **lower** and **uppercase** characters from the **Latin** alphabet

Output

Print the **decrypted message**.

Constraints

- The **key** will be in the interval **[0…20]**
- **n** will be in the interval **[1…20]**
- The **characters** will always be **upper** or **lower**-case letters from the **English alphabet**
- You will receive **one letter** per **line**

Examples

Input	Output
3 **7** P l c q R k f	SoftUni

Input	Output
1 **7** C d b q x o s	Decrypt

5. Balanced Brackets

You will receive **n** lines. On **those lines**, you will receive **one** of the following:

- Opening bracket – “**(**“,
- Closing bracket – “**)**” or
- **Random string**

Your task is to find out if the **brackets** are **balanced**. That means after every **closing** bracket should follow an **opening** one. Nested parentheses are **not valid**, and if **two consecutive opening brackets** exist, the expression should be marked as **unbalanced**.

Input

- On the **first line**, you will receive **n** – the number of lines, which will follow
- On the next **n** lines, you will receive "**(**", "**)**" or **another** string

Output

You have to print "**BALANCED**", if the parentheses are balanced and "**UNBALANCED**" otherwise.

Constraints

- **n** will be in the interval **[1…20]**
- The length of the stings will be between **[1…100]** characters

Examples

Input	Output
8 (5 + 10) * 2 + (5) -12	BALANCED

Input	Output
6 12 *) 10 + 2 - (5 + 10)	UNBALANCED