Exercises: Data Types and Variables

Problems for exercises and homework for the “Programming Fundamentals” course @ SoftUni.

You can check your solutions here: https://judge.softuni.bg/Contests/206/Data-Types-and-Variables-Exercises.

1. Practice Integer Numbers

Create a program that **assigns integer values** to **variables**. Be sure that each **value** is stored in the **correct variable type** (try to find the most suitable variable type in order to save memory). Finally, you need to **print** all variables to the console.

Examples

Input	Output
-100 128 -3540 64876 2147483648 -1141583228 -1223372036854775808	-100 128 -3540 64876 2147483648 -1141583228 -1223372036854775808

2. Practice Floating Point Numbers

Create a program that **assigns floating point values** to **variables**. Be sure that each **value** is stored in the **correct variable type** (try to find the most suitable variable type in order to save memory). Finally, you need to **print** all variables to the console.

Examples

Input	Output
3.141592653589793238 1.60217657 7.8184261974584555216535342341	3.141592653589793238 1.60217657 7.8184261974584555216535342341

3. Practice Characters and Strings

Create a program that **assigns character** and **string values** to **variables**. Be sure that each **value** is stored in the **correct variable**. Finally, you need to **print** all variables to the console.

Examples

Input	Output
Software University B y e I love programming	Software University B y e I love programming

4. Variable in Hexadecimal Format

Write a program that reads a number in **hexadecimal format** (**`0x##`**) convert it to **decimal format** and prints it.

Examples

Input	Output
`0xFE`	`254`

Input	Output
`0x37`	`55`

Input	Output
`0x10`	`16`

5. Boolean Variable

Write a program that reads a **string**, converts it to **Boolean** variable and **prints** "**Yes**" if the variable is **`true`** and "**No**" if the variable is **`false`**.

Examples

Input	Output
`True`	`Yes`
`False`	`No`

6. Strings and Objects

Declare two **string variables** and assign them with "**`Hello`**" and "**`World`**". Declare an **object variable** and assign it with the **concatenation** of the first two variables (mind adding an interval between). Declare a third **string** variable and initialize it with the value of the object variable (you should perform type **casting**).

Examples

Input	Output
`Hello` `World`	`Hello World`

7. Exchange Variable Values

Declare two integer variables **a** and **b** and assign them with 5 and 10 and after that **exchange their values** by using some programming logic. Print the variable values before and after the exchange, as shown below:

Examples

Input	Output

`5`
`10`
	`Before:`
	`a = 5`
	`b = 10`
	`After:`
	`a = 10`
	`b = 5`

8. Employee Data

A marketing company wants to keep record of its employees. Each record would have the following characteristics:

- First name
- Last name
- Age (0...100)
- Gender (`m` or `f`)
- Personal ID number (e.g. 8306112507)
- Unique employee number (27560000…27569999)

Declare the **variables** needed to keep the information for a single employee using appropriate primitive data types. Use descriptive names. **Print** the data at the console.

Examples

Input	Output
Amanda Jonson 27 f 8306112507 27563571	First name: Amanda Last name: Jonson Age: 27 Gender: f Personal ID: 8306112507 Unique Employee number: 27563571

9. Reverse Characters

Write a program to ask the user for **3 letters** and print them in **reversed order**.

Examples

Input	Output
A B C	CBA

Input	Output
x Y z	zYx

Input	Output
G g n	ngG

10. Centuries to Nanoseconds

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

Examples

Input	Output
1	1 centuries = 100 years = 36524 days = 876576 hours = 52594560 minutes = 3155673600 seconds = 3155673600000 milliseconds = 3155673600000000 microseconds = 3155673600000000000 nanoseconds
5	5 centuries = 500 years = 182621 days = 4382904 hours = 262974240 minutes = 15778454400 seconds = 15778454400000 milliseconds = 15778454400000000 microseconds = 15778454400000000000 nanoseconds

11. * Convert Speed Units

NB: You cannot solve this Problem with JAVA without using **additional logic** for the rounding.

Create a program to ask the user for a **distance (in meters)** and the time taken (as three numbers: hours, minutes, seconds), and **print the speed**, in meters per second, kilometers per hour and miles per hour.

Assume 1 mile = 1609 meters.

Input

- On first line, you receive – **distance in meters**
- On second – **hours**
- On third – **minutes**
- On fourth – **seconds**

Output

Every number in the output should be precise up to 6 digits after the floating point

- On first line – speed in **meters per second** (m/s)
- On second line – speed in **kilometers per hour** (km/h)
- On third line – speed in **miles per hour** (mp/h)

Examples

Input	Output
1000 1 1 0	0.2732241 0.9836066 0.6113155

Input	Output
10000 0 20 30	8.130081 29.26829 18.19036

Input	Output
200000 2 5 0	26.66667 96 59.66439

12. Rectangle Properties

Create a program to calculate rectangle's **perimeter**, **area** and **diagonal** by given its **width** and **height**.

Examples

Input	Output
10 5	30 50 11.1803398874989

Input	Output
22.1 10.2	64.6 225.42 24.3402958075698

13. Vowel or Digit

Create a program to check if given symbol is **digit**, **vowel** or any **other symbol**.

Examples

Input	Output
a	vowel

Input	Output
9	digit

Input	Output
g	other

14. Integer to Hex and Binary

Create a program to convert a **decimal number** to **hexadecimal** and **binary** number and print it.

Examples

Input	Output
10	A 1010

Input	Output
420	1A4 110100100

Input	Output
256	100 100000000

15. Fast Prime Checker - Refactor

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

```
int ___Do___ = int.Parse(Console.ReadLine());
for (int DAVIDIM = 0; DAVIDIM <= ___Do___; DAVIDIM++)
{    bool TowaLIE = true;
         for (int delio = 2; delio <= Math.Sqrt(DAVIDIM); delio++)
    {
        if  (DAVIDIM % delio == 0)
        {
            TowaLIE = false;
            break;
        }
    }
    Console.WriteLine($"{DAVIDIM} -> {TowaLIE}");
}
```

Examples

Input	Output
5	2 -> True 3 -> True 4 -> False 5 -> True

16. * Comparing Floats

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 0.000001 == eps. We consider the numbers are different.
4.999999	4.999998	False	Border case. The difference 0.000001 == eps. We consider the numbers are different.

17. 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

18. * Different Integers Size

Given an input integer, you must **determine which primitive data types** are capable of properly storing that input.

Input

- You receive **N** – integer which can be arbitrarily large or small

Output

You must determine if the given primitives are capable of storing it. If yes, then print:

```
can fit in:
* dataType
```

If there is more than one appropriate data type, print each one on its own line and order them by size (**sbyte** < **byte** < **short** < **ushort** < **int** < **uint** < **long**).

If the number cannot be stored in one of the four aforementioned primitives, print the line:

```
{N} can't fit in any type
```

Examples

Input	Output

`-150`	`-150 can fit in:`
	`* short`
	`* int`
	`* long`

Input	Output

`150000`	`150000 can fit in:`
	`* int`
	`* uint`
	`* long`

Input	Output

`1500000000`	`1500000000 can fit in:`
	`* int`
	`* uint`
	`* long`

Input	Output

`213333333333333333333333333333333333`	`213333333333333333333333333333333333`
	`can't fit in any type`

19. * Thea the Photographer

This problem is from the Programming Fundamentals Retake Exam (11 September 2016).

Thea is a photographer. She takes pictures of people on special events. She is a good friend and you want to help her.

She wants to inform her clients when their pictures will be ready. Since the number of pictures is big and it requires time for editing (#nofilter, #allnatural) **every** single picture - you decide to write a program in order to help her.

Thea follows this pattern: first she takes **all** pictures. Then she goes through every single picture to **filter** these pictures that are considered "good". Then she needs to upload every single **filtered** picture to her cloud. She is considered ready when all **filtered** pictures are **uploaded** in her picture storage.

You will receive the **number** of pictures she had taken. Then the approximate **time** in **seconds** for every picture to be filtered. Then a **filter factor** – a **percentage** (integer number) of the **total photos** (rounded to the nearest **bigger integer** value e.g. 5.01 -> 6) that are good enough to be given to her clients (Photoshop may do miracles but Thea does not). Approximate **time** for every picture to be uploaded will be given again in **seconds**. Your task is: based on this input to display total time needed for her to be ready with the pictures in given below format.

Input

On the first line, you will receive an integer **N** – the number of pictures Thea had taken.

On the second line, you will receive an integer **FT** – the amount of time (filter time) in seconds that Thea will require to filter every single picture.

On the third line, you will receive an integer **FF** – the filter factor or the percentage of the total pictures that are considered “good” to be uploaded.

On the fourth line, you will receive an integer **UT** – the amount of time needed for every **filtered** picture to be uploaded to her storage.

The input will be in the described format, there is no need to check it explicitly.

Output

Print the amount of time Thea will need in order to have her pictures ready to be sent to her client in given format:

`d:HH:mm:ss`

d - days needed – starting from 0.

HH – hours needed – from 00 to 24.

`mm` – minutes needed – from 00 to 59.

`ss` – minutes needed – from 00 to 59.

Constraints

The number of total pictures Thea will have taken is range [0 … 1 000 000]

The seconds for both filtering and uploading will be in range [0 … 100 000]

The filter factor will be an integer number between [0 … 100].

Examples

Input	Output	Comments
1000 1 50 1	0:00:25:00	Total pictures = 1 000, 50% of them are useful -> Filtered pictures = 500 Total pictures * filter time = 1000 s Filtered pictures * upload time = 500 s Total time = 1500 s
5342 2 82 3	0:06:37:07	Total pictures = 5342 - 82% of them are useful -> 4380.44-> 4381 filtered.