# Lab: Intro and Basic Syntax

Problems for in-class lab for the [["C# Fundamentals" course @ SoftUni](https://softuni.bg/trainings/2363/csharp-fundamentals-may-2019)](https://softuni.bg/modules/57/tech-module-4-0)  
You can check your solutions in [Judge](https://judge.softuni.bg/Contests/1188)

## Student Information

You will be given 3 lines of input – student name, age and average grade. Your task is to print all the info about the student in the following format: "Name: {student name}, Age: {student age}, Grade: {student grade}".

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| John  15  5.40 | Name: John, Age: 15, Grade: 5.40 |
| Steve  16  2.50 | Name: Steve, Age: 16, Grade: 2.50 |
| Marry  12  6.00 | Name: Marry, Age: 12, Grade: 6.00 |

## Passed

Write a program, which takes as an input a **grade** and prints "**Passed!**" if the grade is **equal or more than 3.00**.

### Input

The **input** comes as a single floating-point number.

### Output

The **output** is either "**Passed!**" if the grade is **equal or more than 3.00**, otherwise you should print nothing.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5.32 | Passed! |  | 2.34 | *(no output)* |

### Solution

We need to take as an input a floating-point number from the console. We will use double.Parse() to convert **string** to **double**, which we receive from Console.ReadLine(). After that we compare the grade with **3.00** and prints the result **only** **if** the condition returns **true**.



## Passed or Failed

Modify the above program, so it will print "**Failed**!" if the grade is **lower than 3.00**.

### Input

The **input** comes as a single double number.

### Output

The **output** is either "**Passed**!" if the grade is **more than 2.99**, otherwise you should print "**Failed**!".

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5.32 | Passed! |  | 2.36 | Failed! |

### Solution

Again, we need to take **floating-point** number from the console. After that print in the **else** statement the appropriate message.



## Back in 30 Minutes

Every time Stamat tries to pay his bills he sees on the cash desk the sign: **"I will be back in 30 minutes"**. One day Stamat was sick of waiting and decided he needs a program, which **prints the time** after **30** **minutes**. That way he won’t have to wait on the desk and come at the appropriate time. He gave the assignment to you, so you have to do it.

### Input

The **input** will be on two lines. On the **first** **line**, you will receive the **hours** and on the **second** you will receive the **minutes**.

### Output

Print on the console the time after **30** minutes. The result should be in format **hh:mm**. The **hours** have **one or two** **numbers** and the **minutes** have always **two numbers (with leading zero)**.

### Constraints

* The **hours** will be between **0 and 23**.
* The **minutes** will be between **0 and 59**.

### Examples

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1  46 | 2:16 |  | 0  01 | 0:31 | 23  59 | 0:29 |  | 11  08 | 11:38 |  | 11  32 | 12:02 |

### Hints

* Add 30 minutes to the initial minutes, which you receive from the console. If the minutes are more than 59 – increase the hours with 1 and decrease the minutes with 60. The same way check if the hours are more than 23. When you print check for leading zero.

## Month Printer

Write a program, which takes an **integer** from the console and prints the corresponding **month**. If the number **is more than 12** or **less than 1** print "**Error!**".

### Input

You will receive a **single** **integer** on a **single line**.

### Output

If the number is within the boundaries print the corresponding month, otherwise print "**Error!**".

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 2 | February |  | 13 | Error! |

### Solution



## Foreign Languages

Write a program, which prints the language, that a given country speaks. You can receive only the following combinations: English **is spoken** in England and USA; Spanish **is spoken** in Spain, Argentina and Mexico; for the others**,** we should print "unknown".

### Input

You will receive a **single country name** on a **single line**.

### Output

**Print** the **language**, which the country **speaks**, or if it is **unknown** for your program, print **"**unknown**"**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| USA | English |  | Germany | unknown |

### Hint

Think how you can **merge** multiple cases, in order to **avoid** writing more code than you need to.

## Theatre Promotions

A theatre **is doing a ticket sale**, but they need a program **to** calculate the price of a single ticket. If the given age does not fit one of the categories**,** you should print "Error!". You can see the prices i**n** the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Day / Age** | **0 <= age <= 18** | **18 < age <= 64** | **64 < age <= 122** |
| **Weekday** | 12$ | 18$ | 12$ |
| **Weekend** | 15$ | 20$ | 15$ |
| **Holiday** | 5$ | 12$ | 10$ |

### Input

The input comes in **two lines**. On the **first** line, you will receive the **type of day**. On the **second** – the **age** of the person.

### Output

Print the price of the ticket according to the table, or "**Error!**" if the age is not in the table.

### Constraints

* The age will be in the interval **[-1000…1000]**.
* The type of day will **always be** **valid**.

### Examples

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| Weekday  42 | 18$ |  | Holiday  -12 | Error! | Holiday  15 | 5$ |  | Weekend  122 | 15$ |

## Divisible by 3

Write a program, which prints all the numbers from **1 to 100**, which are **divisible by 3**. You have to use a single for loop. The program should not receive input.

### Solution



## Sum of Odd Numbers

Write a program that prints the next **n** **odd numbers** (starting from 1) and on the **last row** prints the **sum of them**.

### Input

On the first line, you will receive a number – **n**. This number shows how many **odd numbers** you should print.

### Output

Print the next **n** odd numbers, starting from **1**, separated by **new lines**. On the last line, print the **sum** of these numbers.

### Constraints

* **n** will be in the interval **[1…100]**

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5 | 1  3  5  7  9  Sum: 25 |  | 3 | 1  3  5  Sum: 9 |

### Solution



## Multiplication Table

You will receive an **integer** as an input from the console. Print the **10 times table** for this integer. See the examples below for more information.

### Output

Print every row of the table in the following format:

{theInteger} X {times} = {product}

### Constraints

* The integer will be in the interval **[1…100]**

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5 | 5 X 1 = 5  5 X 2 = 10  5 X 3 = 15  5 X 4 = 20  5 X 5 = 25  5 X 6 = 30  5 X 7 = 35  5 X 8 = 40  5 X 9 = 45  5 X 10 = 50 |  | 2 | 2 X 1 = 2  2 X 2 = 4  2 X 3 = 6  2 X 4 = 8  2 X 5 = 10  2 X 6 = 12  2 X 7 = 14  2 X 8 = 16  2 X 9 = 18  2 X 10 = 20 |

## Multiplication Table 2.0

Rewrite you program so it can receive the **multiplier from the console**. Print the **table from the given multiplier to 10**. If the given multiplier is **more than 10** - print only one row with the **integer**, the given **multiplier** and the **product**. See the examples below for more information.

### Output

Print every row of the table in the following format:

{theInteger} X {times} = {product}

### Constraints

* The integer will be in the interval **[1…100]**

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 5  1 | 5 X 1 = 5  5 X 2 = 10  5 X 3 = 15  5 X 4 = 20  5 X 5 = 25  5 X 6 = 30  5 X 7 = 35  5 X 8 = 40  5 X 9 = 45  5 X 10 = 50 |  | 2  5 | 2 X 5 = 10  2 X 6 = 12  2 X 7 = 14  2 X 8 = 16  2 X 9 = 18  2 X 10 = 20 |  | 2  14 | 2 X 14 = 28 |

## Even Number

Take as an input an even number and **print its absolute value**. If the number is odd, print "Please write an even number." and continue reading numbers.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1  3  6 | Please write an even number.  Please write an even number.  The number is: 6 |  | -6 | The number is: 6 |

## Debug the Code: Holidays Between Two Dates

You are assigned to **find and fix the bugs** in an existing piece of code, using the Visual Studio **debugger**. You should trace the program execution to find the lines of code that produce incorrect or unexpected results.

You are given a program (existing **source code**) that aims to **count the non-working days between two dates** given in format day.month.year (e.g. between **1.05.2015** and **15.05.2015** there are **5** non-working days – Saturday and Sunday).

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1.05.2016  15.05.2016 | 5 | There are 5 non-working days (Saturday / Sunday) in this period: 1-May-2016, 7-May-2016, 8-May-2016, 14-May-2016, 15-May-2016 |
| 1.5.2016  2.5.2016 | 1 | Only 1 non-working day in the specified period: 1.05.2016 (Sunday) |
| 15.5.2020  10.5.2020 | 0 | The second date is before the first. No dates in the range. |
| 22.2.2020  1.3.2020 | 4 | Two Saturdays and Sundays:   * 22.02.2020 and 23.02.2020 * 29.02.2020 and 1.03.2020 |

You can **find the broken code** in the judge system: [Broken Code for Refactoring](http://softuni.bg/downloads/svn/soft-tech/Sep-2016/Programming-Fundamentals-Sep-2016/03.%20Programming-Fundamentals-Methods-Debugging-and-Troubleshooting-Code/03.Programming-Fundamentals-Methods-and-Debugging-Lab-Broken-Solutions.zip). It looks as follows:

|  |
| --- |
| HolidaysBetweenTwoDates.cs |
| using System;  using System.Globalization;  class HolidaysBetweenTwoDates  {  static void Main()  {  var startDate = DateTime.ParseExact(Console.ReadLine(),  "dd.m.yyyy", CultureInfo.InvariantCulture);  var endDate = DateTime.ParseExact(Console.ReadLine(),  "dd.m.yyyy", CultureInfo.InvariantCulture);  var holidaysCount = 0;  for (var date = startDate; date <= endDate; date.AddDays(1))  if (date.DayOfWeek == DayOfWeek.Saturday &&  date.DayOfWeek == DayOfWeek.Sunday) holidaysCount++;  Console.WriteLine(holidaysCount);  }  } |

### Hints

There are **4** **mistakes** in the code. You’ve got to **use the debugger** to find them and fix them. After you do that, submit your **fixed code in the judge contest**: <https://judge.softuni.bg/Contests/Practice/Index/304#8>.