# GENKA RUGINA

# Homework: Conditional Statements

This document defines homework assignments from the [“C# Basics“ Course @ Software University](http://softuni.bg/courses/csharp-basics/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code only) of all below described problems.

## Exchange If Greater

Write an **if**-statement that takes two integer variables a and b and **exchanges** their values if the first one is greater than the second one. As a result print the values a and b, separated by a space. Examples:

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **result** |
| 5 | 2 | 2 5 |
| 3 | 4 | 3 4 |
| 5.5 | 4.5 | 4.5 5.5 |

## Bonus Score

Write a program that applies bonus score to given score in the range [1…9] by the following rules:

* If the score is between 1 and 3, the program multiplies it by 10.
* If the score is between 4 and 6, the program multiplies it by 100.
* If the score is between 7 and 9, the program multiplies it by 1000.
* If the score is 0 or more than 9, the program prints “invalid score”.

Examples:

|  |  |
| --- | --- |
| **score** | **result** |
| 2 | 20 |
| 4 | 400 |
| 9 | 9000 |
| -1 | invalid score |
| 10 | invalid score |

## Check for a Play Card

Classical play cards use the following signs to designate the card face: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K and A. Write a program that enters a string and prints “yes” if it is a valid card sign or “no” otherwise. Examples:

|  |  |
| --- | --- |
| **character** | **Valid card sign?** |
| 5 | yes |
| 1 | no |
| Q | yes |
| q | no |
| P | no |
| 10 | yes |
| 500 | no |

## Multiplication Sign

Write a program that shows the sign (+, - or 0) of the product of three real numbers, without calculating it. Use a sequence of **if** operators. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **a** | **b** | **c** | **result** |
| 5 | 2 | 2 | + |
| -2 | -2 | 1 | + |
| -2 | 4 | 3 | - |
| 0 | -2.5 | 4 | 0 |
| -1 | -0.5 | -5.1 | - |

## The Biggest of 3 Numbers

Write a program that finds the **biggest of three numbers**. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **a** | **b** | **c** | **biggest** |
| **5** | 2 | 2 | 5 |
| -2 | -2 | **1** | 1 |
| -2 | **4** | 3 | 4 |
| 0 | -2.5 | **5** | 5 |
| **-0.1** | -0.5 | -1.1 | -0.1 |

## The Biggest of Five Numbers

Write a program that finds the **biggest of five numbers** by using only five if statements. Examples:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a** | **b** | **c** | **d** | **e** | **biggest** |
| **5** | 2 | 2 | 4 | 1 | 5 |
| -2 | -22 | **1** | 0 | 0 | 1 |
| -2 | **4** | 3 | 2 | 0 | 4 |
| 0 | -2.5 | 0 | **5** | **5** | 5 |
| -3 | -0.5 | -1.1 | -2 | **-0.1** | -0.1 |

## Sort 3 Numbers with Nested Ifs

Write a program that enters **3 real numbers** and prints them sorted in descending order. Use nested **if** statements. Don’t use arrays and the built-in sorting functionality. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **a** | **b** | **c** | **result** |
| 5 | 1 | 2 | 5 2 1 |
| -2 | -2 | 1 | 1 -2 -2 |
| -2 | 4 | 3 | 4 3 -2 |
| 0 | -2.5 | 5 | 5 0 -2.5 |
| -1.1 | -0.5 | -0.1 | -0.1 -0.5 -1.1 |
| 10 | 20 | 30 | 30 20 10 |
| 1 | 1 | 1 | 1 1 1 |

## Digit as Word

Write a program that asks for a **digit** (0-9), and depending on the input, **shows the digit as a word** (in English). Print “not a digit” in case of invalid inut. Use a **switch** statement. Examples:

|  |  |
| --- | --- |
| **d** | **result** |
| 2 | two |
| 1 | one |
| 0 | zero |
| 5 | five |
| -0.1 | not a digit |
| hi | not a digit |
| 9 | nine |
| 10 | not a digit |

## Play with Int, Double and String

Write a program that, depending on the user’s choice, inputs an **int**, **double** or **string** variable. If the variable is **int** or **double**, the program increases it by one. If the variable is a **string**, the program appends "**\***" at the end. Print the result at the console. Use **switch** statement. Example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **program** | **user** |  | **program** | **user** |
| Please choose a type: 1 --> int  2 --> double  3 --> string | 3 | Please choose a type: 1 --> int  2 --> double  3 --> string | 2 |
| Please enter a string: | hello | Please enter a double: | 1.5 |
| hello\* |  | 2.5 |  |

## \* Beer Time

A beer time is after 1:00 PM and before 3:00 AM. Write a program that **enters a time** in format “hh:mm tt” (an hour in range [01...12], a minute in range [00…59] and AM / PM designator) and prints “**beer time**” or “**non-beer time**” according to the definition above or “**invalid time**” if the time cannot be parsed. Note that you may need to learn how to parse dates and times. Examples:

|  |  |
| --- | --- |
| **time** | **result** |
| 1:00 PM | beer time |
| 4:30 PM | beer time |
| 10:57 PM | beer time |
| 8:30 AM | non-beer time |
| 02:59 AM | beer time |
| 03:00 AM | non-beer time |
| 03:26 AM | non-beer time |

## \* Number as Words

Write a program that **converts a number in the range [0…999] to words**, corresponding to the English pronunciation. Examples:

|  |  |
| --- | --- |
| **numbers** | **number as words** |
| 0 | Zero |
| 9 | Nine |
| 10 | Ten |
| 12 | Twelve |
| 19 | Nineteen |
| 25 | Twenty five |
| 98 | Ninety eight |
| 273 | Two hundred and seventy three |
| 400 | Four hundred |
| 501 | Five hundred and one |
| 617 | Six hundred and seventeen |
| 711 | Seven hundred and eleven |
| 999 | Nine hundred and ninety nine |

## \* Zero Subset

We are given 5 integer numbers. Write a program that finds all **subsets of these numbers whose sum is 0**. Assume that repeating the same subset several times is not a problem. Examples:

|  |  |
| --- | --- |
| **numbers** | **result** |
| 3 -2 1 1 8 | -2 + 1 + 1 = 0 |
| 3 1 -7 35 22 | no zero subset |
| 1 3 -4 -2 -1 | 1 + -1 = 0  1 + 3 + -4 = 0  3 + -2 + -1 = 0 |
| 1 1 1 -1 -1 | 1 + -1 = 0  1 + 1 + -1 + -1 = 0  1 + -1 + 1 + -1 = 0  … |
| 0 0 0 0 0 | 0 + 0 + 0 + 0 + 0 = 0 |

Hint: you may check for zero each of the 32 subsets with 32 if statements.

# Exam problems.\*\*

**All of the problems below are given from Variant 5 of C# Basics Practical Exam (12 April 2014 Morning). You can submit your solutions** [**HERE**](http://judge.softuni.bg/Contests/6/CSharp-Basics-Exam-12-April-2014-Morning)**.**

**You are not obligated** to submit any of them in your homework. We highly recommend you to try solving some or all of them so you can be well prepared for the upcoming exam. You need to learn how to use conditional statements, loops, arrays and other things (learn in internet how or read those chapters in the book “[Fundamentals of computer programming with C#](http://www.introprogramming.info/intro-csharp-book/read-online/)”). If you still find those problems too hard for solving it’s very useful to **check** and **understand** the solutions. You can download all solutions and tests for this variant [here](https://softuni.bg/downloads/svn/csharp-basics/Feb-2014/9.%20CSharp-Basics-Exam-April-2014-Variant-1.zip) or check all [previous exams](https://softuni.bg/trainings/coursesinstances/details/2) (scroll down to the bottom of the page). You can also test your solutions in our automated [judge system](http://judge.softuni.bg/Contests/2/CSharp-Basics-Exam-10-April-2014-Morning) to see if you pass all tests.

## \* – Triangle

You are given a two-dimensional Cartesian coordinate system and three points A, B, C with coordinates: A(Ax, Ay), B(Bx, By), C(Cx, Cy). Write a program to check if these **three points can form a triangle**. Then calculate the area of this triangle. To find the distance between two points with the coordinates (x1, y1) and (x2, y2) use the formula:

You can use the inequalities of a triangle to check whether three segments **a**, **b** and **c** can form a triangle:

; ;

To calculate the area you can use Heron`s formula (a method for calculating the area of a triangle when you know the lengths of all three sides). Let **a**, **b**, **c** be the lengths of the sides of a triangle. Thus:

, where **p** is half the perimeter: .

### Input

The input data comes from the console. It consists of exactly 6 lines holding the coordinates of the three points: **Ax**-coordinate, **Ay**-coordinate, **Bx**-coordinate, **By**-coordinate, **Cx**-coordinate and **Cy**-coordinate. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output data should be printed on the console and must contain two lines.

* **First line:** If the given points can form a triangle you must print the message “**Yes**”, otherwise “**No**”.
* **Second line:** If the given points can form a triangle you must print the **area of the triangle** rounded to two decimal places (see the examples), otherwise you must print the **distance between point A and point B**. Use "**.**" as decimal separator.

### Constraints

* The coordinate **X** is integer in the range [-10000… 10000] inclusive.
* The coordinate **Y** is integer in the range [-10000… 10000] inclusive.
* Allowed work time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Comments** |  | **Input** | **Output** | **Comments** |
| 2  2  0  0  1  1 | No  2.83 |  |  | 2  3  0  -1  4  -2 | Yes  9.00 |  |

## \* – Pairs

You are given **2\*N elements** (even number of integer numbers). The first and the second element form a **pair**, the third and the fourth element form a pair as well, etc. Each pair has a **value**, calculated as the **sum** of its two elements. Your task is to write a program to check **whether all pairs have the same value** or print the **max difference** between two consecutive values.

### Input

You are given at the console **even number of integers**, all in a single line, separated by a space.

### Output

The output is single line, printed at the console.

* In case all pairs have the same value, print "**Yes, value=…**" and the value.
* Otherwise, print "**No, maxdiff=…**" and the maximal difference between two consecutive values, always a **positive integer**.

### Constraints

* All input values will be integers in the range [-1000…1000] inclusive.
* The count of elements is even number in the range [2…1000] inclusive.
* Allowed work time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 0 3 4 -1 | Yes, value=3 | values = {3, 3, 3} --> equal values |
| 1 2 2 2 | No, maxdiff=1 | values = {3, 4}, different values --> max difference = 4-3 = 1 |
| 1 1 3 1 2 2 0 0 | No, maxdiff=4 | values = {2, 4, 4, 0}, differences = {2, 0, 4}, max difference = 4 |
| 5 5 | Yes, value=10 | values = {10} --> single value --> equal values |
| -1 0 0 -1 | Yes, value=-1 | values = {-1, -1}, equal values |

## \* – House

Ivaylo decided he needs a new house. Since he is not a structural engineer yet, you have to help him construct the building from the ground zero.

The roof is a triangle. The walls are straight vertical lines. The base is a straight horizontal line. The roof, the walls and the base of the house it build of '**\***'. Everything else is filled with '.' (see the examples below to catch the idea).

You will be given an odd integer **N**, representing the width and the height of the house. The roof’s top starts from the center (**N+1)/2** and forms a triangle with base of width **N**. The roof’s height is (**N+1)/2**. The distance between the roofs’ end point and the walls of the building is **N/4**, rounded down to an integer number. See the examples below to understand better these formulas.

### Input

* Input data is read from the console.
* The number **N** stays alone at the first line.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output data must be printed on the console.
* You must print at the console a house of size **N** following the formulas above and the examples below.

### Constraints

* **N** will be an **odd** number between **5** and **49**.
* Time limit: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 5 | ..\*..  .\*.\*.  \*\*\*\*\*  .\*.\*.  .\*\*\*. |  | 7 | ...\*...  ..\*.\*..  .\*...\*.  \*\*\*\*\*\*\*  .\*...\*.  .\*...\*.  .\*\*\*\*\*. | 9 | ....\*....  ...\*.\*...  ..\*...\*..  .\*.....\*.  \*\*\*\*\*\*\*\*\*  ..\*...\*..  ..\*...\*..  ..\*...\*..  ..\*\*\*\*\*.. |

## \*\* – Magic Dates

Consider we are given a **date** in format dd-mm-yyyy, e.g. 17-03-2007. We calculate the **weight of this date** by joining together all its digits, multiplying each digit by each of the other digits and finally sum all obtained products. In our case we will have 8-digits: 17032007, so the weight is 1\*7 + 1\*0 + 1\*3 + 1\*2 + 1\*0 + 1\*0 + 1\*7 + 7\*0 + 7\*3 + 7\*2 + 7\*0 + 7\*0 + 7\*7 + 0\*3 + 0\*2 + 0\*0 + 0\*0 + 0\*7 + 3\*2 + 3\*0 + 3\*0 + 3\*7 + 2\*0 + 2\*0 + 2\*7 + 0\*0 + 0\*7 + 0\*7 = 144.

Your task is to write a program that finds all **magic dates**: **dates between two fixed years matching given magic weight**. The dates should be printed in increasing order in format dd-mm-yyyy. We use the traditional calendar (years have 12 months, each having 28, 29, 30 or 31 days, etc.). Years start from 1 January and end in 31 December.

### Input

The input data should be read from the console. It consists of 3 lines:

* The first line holds an integer number – **start year**.
* The second line holds an integer number – **end year**.
* The third line holds an integer number – **magic weight**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output should be printed on the console as a sequence of dates in **format dd-mm-yyyy** in **alphabetical order**. Each string should stay on a separate line. In case no magic dates exist, print “**No**”.

Beware that the **regional settings** at your computer and at the judge's computer may be different!

### Constraints

* The **start** and **end year** are **integers** in the range [1900-2100].
* The **magic weight** is an integer number in range [1…1000].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 2007  2007  144 | 17-03-2007  13-07-2007  31-07-2007 | 2003  2004  1500 | No | 2012  2014  80 | 09-01-2013  17-01-2013  23-03-2013  11-07-2013  01-09-2013  10-09-2013  09-10-2013  17-10-2013  07-11-2013  24-11-2013  14-12-2013  23-11-2014  13-12-2014  31-12-2014 | 2011  2012  14 | 01-01-2011  10-01-2011  01-10-2011  10-10-2011 |

## \*\* – Bit Killer

You are given a **sequence of bytes**. Consider each byte as sequence of exactly 8 bits. You are given also a number **step**. Write a program to remove (kill) all the bits at positions: **1**, **1 + step**, **1 + 2\*step**, ... Print the output as a sequence of bytes. In case the last byte have less than 8 bits, add trailing zeroes at its right end. Bits in each byte are counted from the leftmost to the rightmost. Bits are numbered starting from 0.

### Input

* The input data should be read from the console.
* The number **n** stays at the first line.
* The number **step** stays at the second line.
* At each of the next **n** lines **n** bytes are given, each at a separate line.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output should be printed on the console. Print the output bytes, each at a separate line.

### Constraints

* The number **n** will be an **integer** number in the range [1…100].
* The number **step** will be an **integer** number in the range [1…20].
* The **n numbers** will be integers in the range [0…255].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  11  109  87 | 90  188 | We have the following input sequence of 16 bits (2 bytes):  0**1**101101 0101**0**111. We kill the bits 1 and 12 (step=11). Obtained sequence: 01011010 101111. Padding: 2 zeroes at the end. Result: **01011010 10111100**. |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  2  45  87  250 | 97  240 | We have the following input sequence of 24 bits (3 bytes):  0**0**1**0**1**1**0**1** 0**1**0**1**0**1**1**1** 1**1**1**1**1**0**1**0**. We kill bits 1, 3, …, 23 (step=2). Obtain the sequence: 01100001 1111. We pad it with 4 zeroes at the end to obtain 2 full bytes. Result: **01100001 11110000**. |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  2  45  87 | 97 | We have the following input sequence of 16 bits (2 bytes):  0**0**1**0**1**1**0**1** 0**1**0**1**0**1**1**1**. We kill bits 1, 3, 5, …, 15 (step=2). Obtained sequence: 01100001 (8 bits). No padding is needed. Result: **01100001**. |