# 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**](https://judge.softuni.bg/Contests/82/Programming-Basics-Exam-26-April-2015-Evening)**.**

**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](http://svn.softuni.org/admin/svn/csharp-basics/Feb-2015/Programming-Basics-Exam-26-Apr-2015-Evening.zip) or check all [previous exams](https://softuni.bg/courses/programming-basics) (scroll down to the bottom of the page). You can also test your solutions in our automated [judge system](https://judge.softuni.bg/Contests/82/Programming-Basics-Exam-26-April-2015-Evening) to see if you pass all tests.

## \*\* Problem 1 – Compound interest

You really want a new TV, however you are a little short on money right now. You have a couple of options. You can **get a loan from a bank** or **get a loan from a friend**. Since you want to become the best programmer who ever lived, you decided to make a program to help you out.

Bank loans have **interest rate** and a **term (number of years** you have until you are required to pay the money back). Assume the bank loan will be for more than one year and the interest will be accrued on a yearly basis. Use this formula to calculate the amount of money you will have to pay back - '**FV = PV \* (1 + i)n**'. Where '**FV'** (future value) **is the money owed at the end of the period**. '**PV'** (present value) **is the money you want to withdraw today**, '**i'** **is the interest rate and 'n' is the term of the loan.** Your friend is a really nice dude and he will loan you the money, however he wants a percentage of the money in return.

You will be given the price of the TV, the term and yearly interest rate for the loan from the bank, and the percentage your friend will ask for.

Your task is to write a program that calculates **the best (cheapest) option to buy the TV**. If the options are the same, choose your friend’s offer - you are a nice guy after all. Check the example to get a better understanding of the task.

### Input

The input data should be read from the console. It consists of four input values, each at a separate line:

* The number **p** – **price** of **the TV**.
* The number **n** – number of **years you have until you must pay the bank back (term)**.
* The number **i** – the yearly interest rate for the bank’s loan.
* The number **f** – interest rate for your friend’s loan.

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.
* On the only output line you must print **the best loan price to the second digit after the decimal mark and the lender separated by a single space.**

### Constraints

* The number **y is** an integer in the range [0 ... 2 147 483 647].
* The numbers **p, i, f are** floating-point numbers in the range [0 … 7.9 x 1028].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2600  2  0.07  0.4 | 2976.74 Bank | 2600 leva is needed. Bank loan = 2600 \* (1 + 0.07) **2**= 2600 \* 1.07 **2** = 2600 \* 1.1449 = 2976.74. Friend loan = 2600 \* (1 + 0.4) = 2600 \* 1.4 = 3640. 2976.74 < 3640 |

## \*\* Problem 2 – Petar's Game

Petar likes to play with numbers. He is very good with **strings** too. One day he decided to invent a new game of **summing numbers.** He will get one number and will try to divide it by 5**, if the number can be divided without remainder** (for example 15 can be divided by 5 without remainder, but 17 divided by 5 is 3 with remainder 2) Petar will **add this number to the sum.** However **if the number cannot be divided without remainder, only the remainder will be added to the sum.** After he is done with the numbers**,** Petar likes to **replace some of the sum's digits with strings**. If the sum is **odd** he will replace **the last digit and all others that are the same as it** with a given string. If the sum is **even** **he will do the same, but with the first digit.** For example, if the sum is 2434, and the string is "a" - the result will be a434.

**You will be given a start number, an end number and a string.** You have to check all numbers between the start number and the end number (without the end number), do Petar's algorithm and finally replace the digits with the string as described above.

### Input

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

* The first line will hold the starting number;
* The second line will hold the end number;
* The third like will hold the replacement string

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.
* On the only output line you must print the result of the game.

### Constraints

* Start number and end number will be integers in the range [0 … 18 446 744 073 709 551 615].
* The string will contain letters and numbers.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 10  14  a | a6 | 10 + 1 + 2 + 3 = 16. 16 is even 1 will be replaced. Result is a6 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| 10  99  as | asas2as | 234  3547  SadPanda | 12SadPanda969SadPanda |

## \*\* Problem 3 - Sunlight

Sandy is a little girl who spends her free time playing with her friends. Unfortunately, Sandy broke her wristwatch a week ago. Now she is trying a new trick using the sunlight to guess the time. But this trick is useful only when the sky isn’t cloudy. Your task is to help Sandy by writing a program which shows how bright the sun is at the moment.

You are given an integer number **N** (always **odd**), corresponding to the **width and height** of the sun and the **length** of the **horizontal and vertical** sunbeams. The **diagonal** sunbeams have length equal to **N – 1.**

### Input

The input data should be read from the console.

* On the only input line you will be given an integer **N** - the **size** of the sun.

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. Use the “**\***” (asterisk) to mark the sun and the sunbeams and “**.**” (dot) for the rest. Follow the examples below.

### Constraints

* **N** will always be a positive **odd** number in the range [**3** … **33].**
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3 | ....\*....  .\*..\*..\*.  ..\*.\*.\*..  ...\*\*\*...  \*\*\*\*\*\*\*\*\*  ...\*\*\*...  ..\*.\*.\*..  .\*..\*..\*.  ....\*.... |  | 5 | .......\*.......  .\*.....\*.....\*.  ..\*....\*....\*..  ...\*...\*...\*...  ....\*..\*..\*....  .....\*\*\*\*\*.....  .....\*\*\*\*\*.....  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .....\*\*\*\*\*.....  .....\*\*\*\*\*.....  ....\*..\*..\*....  ...\*...\*...\*...  ..\*....\*....\*..  .\*.....\*.....\*.  .......\*....... |

## \*\* Encrypt the Messages

You are working for a company which is very concerned about its information and communication. For this reason, they have invented an internal approach to communication between different departments – they are communicating to each other via **messages, which are reversed (written backwards) and then encrypted**. Your task is to write a program, which **encrypts** **all messages** in a specific communication, **prints them at the console as well as the total number of messages** that have been sent.

**At the beginning** of a communication, you will receive either the keyword “**START**” (upper case) or “**start**” (lower case), which indicates that you will **start receiving reversed and encrypted messages**. At the end of the communication, you will receive either the keyword “**END**” (upper case) or “**end**” (lower case), which indicates that the communication is over and you need to **show the encrypted messages’ content and total count**. Any **non-empty string** between the “start” and “end” keywords is considered a message. If **no messages have been sent** between the “**start**” and the “**end**” keywords, you should print on the console: **“No messages sent.”**

All messages are case-sensitive and consist of **letters**, **digits,** as well as **some special characters** – ‘’, ‘**,**’, ‘.’, ‘**?**’ and ‘**!**’. Letters **from A to M** are **converted** to letters **from N to Z** (A 🡪 N; B 🡪 O; … M 🡪 Z) and letters **from N to Z** are **converted** to letters **from A to M** (N 🡪 A; O 🡪 B; … Z 🡪 M). The **converted** letter should keep the **case** of the **original** letter. The **special characters** are converted in the following way: ‘’ (space) is converted to a **plus sign** (**‘ +’**), ‘**,**’ is converted to **‘%’**, ‘**.**’ is converted to **‘&’**, ‘**?**’ is converted to **‘#’** and ‘**!**’ is converted to **‘$’**. **Digits** (0-9) are **not converted** and stay the same.

For example, you receive the following message – “**Secret message 1!**” and you start encrypting it. Convert the 1st character ‘**!**’ to ‘**$**’, then the 2nd character – ‘**1**’ stays the same, then covert the 3rd character – ‘’ to ‘+’, ‘**e**’ 🡪 ’**r**’, ‘**g**’ 🡪 ‘**t**’, ‘**a**’ 🡪 ‘**n**’, ‘**s**’ 🡪 ‘**f**’, ‘**s**’ 🡪 ‘**f**’, ‘**e**’ 🡪 ’**r**’ , ‘**m**’ 🡪 ’**z**’, ‘’ 🡪 ‘+’, ‘**t**’ 🡪 ‘**g**’, ‘**e**’ 🡪 ’**r**’ , ‘**r**’ 🡪 ’**e**’ , ‘**c**’ 🡪 ’**p**’ , ‘**e**’ 🡪 ’**r**’ , ‘**S**’ 🡪 ’**F**’. After encrypting all letters, the message is: “**Frperg+zrffntr+1$**” and when you reverse it, you get the final encrypted message: “**$1+rtnffrz+greprF**”

### Input

The input data should be read from the console. The input will contain a random number of lines. The line that holds the **keyword “START” or “start”** will always be before the line that holds the **keyword “END” or “end”**. 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.

* On the **first line** print the total number of messages that have been sent in format: “**Total number of messages: N**” – where N is the number of encrypted and sent messages.
* On the next N lines print the encrypted messages.
* If **no messages have been sent** between the “**start**” and the “**end**” keywords, you should **print on the console** only one line holding: “**No messages sent.**”

### Constraints

* The **number of messages** between the “**start**” and the “**end**” keywords will be between 0 and 100.
* The **length of each message** will be between 1 and 1000 symbols.
* Each unencrypted message will contain only Latin letters, digits and the special symbols described above.
* Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Comments** | **Output** |
| START  Hello!!!  END | We start conversion from the 1st character: ! 🡪 $, ! 🡪 $, ! 🡪 $, o 🡪 b, l 🡪 y, l 🡪 y, e 🡪 r, H 🡪 U and reverse the newly received string “Uryyb$$$” to the encrypted message “$$$byyrU” | Total number of messages: 1  $$$byyrU |

|  |  |
| --- | --- |
| **Input** | **Output** |
| START  abcdefg  meSSage1  end | Total number of messages: 2  tsrqpon  1rtnFFrz |

|  |  |
| --- | --- |
| **Input** | **Output** |
| start  END | No messages sent. |

|  |
| --- |
| **Input** |
| Normal communication message.  START  Please, try to encrypt the following message!  end |
| **Output** |
| Total number of messages: 1  $rtnffrz+tavjbyybs+rug+gclepar+bg+leg+%rfnryC |

## \*\* Game of Life

The game of life is a simple game in which the player sets the initial state of the board **(always with size 10x10)** and the computer calculates the next states of the board, by following four simple rules.

1. Any live cell with fewer than two live neighbours dies.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies due to overcrowding.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

A live cell is a cell which has value of ‘1’, dead cell is a cell which has value of ‘0’. By default all cells, except the ones specified in the input, are dead.

**Note that all calculations need to happen simultaneously! See the example.**

Your task is to **generate the next state of the board and the print it to the console**.

### Input

The input data should be read from the console. It consists of several input values, each at a separate line:

* Number of input coordinates (x, y) **n**: how many coordinates will be entered.
* Coordinates: **x** and **y** (**each at a separate line**), which **set the cell [x, y] to 1**.
* Rows are counted from top to bottom (0 to 9) and columns are counter from right to left (0 to 9).

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 calculate the next generation and then print the resulting board.

### Constraints

* **n** is an integer number in range [0 ... 2 147 483 647].
* **x** and **y** are integer numbers in range [0…9].
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  1  6  2  6  3  6 | 0000000000  0000000000  0011100000  0000000000  0000000000  0000000000  0000000000  0000000000  0000000000  0000000000 | **n** = 3 => we’ll receive 3 pairs **(x, y)**. We set the bits to 1 at these positions so the board looks like this:  0000000000  0001000000  00**0*1*0**00000  0001000000  0000000000  0000000000  0000000000  0000000000  0000000000  0000000000  The **shaded zeros** have **exactly 3** **live neighbours**, so in the next generation they will become ones. The **bolded one between them** has **2 live neighbours**, so it will **stay alive**. The **other ones** each have only **one live neighbour**, so they will **become zeros**. |