# Homework: Loops

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.

## Numbers from 1 to N

Write a program that enters from the console a positive integer **n** and **prints all the numbers from 1 to n**, on a single line, separated by a space. Examples:

|  |  |
| --- | --- |
| **n** | **output** |
| 3 | 1 2 3 |
| 5 | 1 2 3 4 5 |

## Numbers Not Divisible by 3 and 7

Write a program that enters from the console a positive integer **n** and prints all the **numbers from 1 to n not divisible by 3 and 7**, on a single line, separated by a space. Examples:

|  |  |
| --- | --- |
| **n** | **output** |
| 3 | 1 2 |
| 10 | 1 2 4 5 8 10 |

## Min, Max, Sum and Average of N Numbers

Write a program that reads from the console a sequence of **n** integer numbers and returns the **minimal**, the **maximal** number, the sum and the average of all numbers (displayed with 2 digits after the decimal point). The **input** starts by the number **n** (alone in a line) followed by **n lines**, each holding an integer number. The **output** is like in the examples below. Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **input** | **output** |  | **input** | **output** |
| 3  2  5  1 | min = 1  max = 5  sum = 8  avg = 2.67 | 2  -1  4 | min = -1  max = 4  sum = 3  avg = 1.50 |

## Print a Deck of 52 Cards

Write a program that generates and prints **all possible cards from a** [**standard deck of 52 cards**](http://en.wikipedia.org/wiki/Standard_52-card_deck) (without the jokers). The cards should be printed using the classical notation (like 5♠, A♥, 9♣ and K♦). The card faces should start from 2 to A. Print each card face in its four possible suits: clubs, diamonds, hearts and spades. Use 2 nested for-loops and a switch-case statement.

|  |
| --- |
| **output** |
| 2♣ 2♦ 2♥ 2♠  3♣ 3♦ 3♥ 3♠  …  K♣ K♦ K♥ K♠  A♣ A♦ A♥ A♠ |

## Calculate 1 + 1!/X + 2!/X2 + … + N!/XN

Write a program that, for a given two integer numbers **n** and **x**, calculates the sum S = 1 + 1!/x + 2!/x2 + … + n!/xn. Use only one loop. Print the result with 5 digits after the decimal point.

|  |  |  |
| --- | --- | --- |
| **n** | **x** | **S** |
| 3 | 2 | 2.75000 |
| 4 | 3 | 2.07407 |
| 5 | -4 | 0.75781 |

## Calculate N! / K!

Write a program that calculates **n! / k!** for given **n** and **k** (1 < **k** < **n** < 100). Use only one loop. Examples:

|  |  |  |
| --- | --- | --- |
| **n** | **k** | **n! / k!** |
| 5 | 2 | 60 |
| 6 | 5 | 6 |
| 8 | 3 | 6720 |

## Calculate N! / (K! \* (N-K)!)

In combinatorics, the number of ways to choose **k** different members out of a group of **n** different elements (also known as the number of [**combinations**](http://en.wikipedia.org/wiki/Combination)) is calculated by the following formula:

 \binom nk = \frac{n!}{k!(n-k)!},

For example, there are 2598960 ways to withdraw 5 cards out of a standard deck of 52 cards. Your task is to write a program that calculates **n! / (k! \* (n-k)!)** for given **n** and **k** (1 < **k** < **n** < 100). Try to use only two loops. Examples:

|  |  |  |
| --- | --- | --- |
| **n** | **k** | **n! / (k! \* (n-k)!)** |
| 3 | 2 | 3 |
| 4 | 2 | 6 |
| 10 | 6 | 210 |
| 52 | 5 | 2598960 |

## Catalan Numbers

In combinatorics, the [Catalan numbers](http://en.wikipedia.org/wiki/Catalan_number) are calculated by the following formula:

C_n = \frac{1}{n+1}{2n\choose n} = \frac{(2n)!}{(n+1)!\,n!} = \prod\limits_{k=2}^{n}\frac{n+k}{k} \qquad\mbox{ for }n\ge 0.

Write a program to calculate the **nth Catalan number** by given **n** (1 < n < 100). Examples:

|  |  |
| --- | --- |
| **n** | **Catalan(n)** |
| 0 | 1 |
| 5 | 42 |
| 10 | 16796 |
| 15 | 9694845 |

## Matrix of Numbers

Write a program that reads from the console a positive integer number n (1 ≤ n ≤ 20) and **prints a matrix** like in the examples below. Use two nested loops. Examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | **matrix** |  | **n** | **matrix** |  | **n** | **matrix** |
| 2 | 1 2  2 3 | 3 | 1 2 3  2 3 4  3 4 5 | 4 | 1 2 3 4  2 3 4 5  3 4 5 6  4 5 6 7 |

## Odd and Even Product

You are given **n** integers (given in a single line, separated by a space). Write a program that checks whether the product of the odd elements is equal to the product of the even elements. Elements are counted from 1 to n, so the first element is odd, the second is even, etc. Examples:

|  |  |
| --- | --- |
| **numbers** | **result** |
| **2** 1 **1** 6 **3** | yes  product = 6 |
| **3** 10 **4** 6 **5** 1 | yes  product = 60 |
| **4** 3 **2** 5 **2** | no  odd\_product = 16  even\_product = 15 |

## Random Numbers in Given Range

Write a program that enters 3 integers n, min and max (min ≤ max) and prints n random numbers in the range [min...max]. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **min** | **max** | **random numbers** |
| 5 | 0 | 1 | 1 0 0 1 1 |
| 10 | 10 | 15 | 12 14 12 15 10 12 14 13 13 11 |

Note that the above output is just an example. Due to randomness, your program most probably will produce different results.

## \* Randomize the Numbers 1…N

Write a program that enters in integer n and prints the numbers 1, 2, …, n in random order. Examples:

|  |  |
| --- | --- |
| **n** | **randomized numbers 1…n** |
| 3 | 2 1 3 |
| 10 | 3 4 8 2 6 7 9 1 10 5 |

Note that the above output is just an example. Due to randomness, your program most probably will produce different results. You might need to use [arrays](http://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx).

## Binary to Decimal Number

Using loops write a program that converts a [binary integer](http://en.wikipedia.org/wiki/Binary_numeral_system) number to its decimal form. The input is entered as string. The output should be a variable of type long. Do not use the built-in .NET functionality. Examples:

|  |  |
| --- | --- |
| **binary** | **decimal** |
| 0 | 0 |
| 11 | 3 |
| 1010101010101011 | 43691 |
| 1110000110000101100101000000 | 236476736 |

## Decimal to Binary Number

Using loops write a program that converts an integer number to its [binary representation](http://en.wikipedia.org/wiki/Binary_numeral_system). The input is entered as long. The output should be a variable of type string. Do not use the built-in .NET functionality. Examples:

|  |  |
| --- | --- |
| **decimal** | **binary** |
| 0 | 0 |
| 3 | 11 |
| 43691 | 1010101010101011 |
| 236476736 | 1110000110000101100101000000 |

## Hexadecimal to Decimal Number

Using loops write a program that converts a [hexadecimal integer](http://en.wikipedia.org/wiki/Hexadecimal) number to its decimal form. The input is entered as string. The output should be a variable of type long. Do not use the built-in .NET functionality. Examples:

|  |  |
| --- | --- |
| **hexadecimal** | **decimal** |
| FE | 254 |
| 1AE3 | 6883 |
| 4ED528CBB4 | 338583669684 |

## Decimal to Hexadecimal Number

Using loops write a program that converts an integer number to its [hexadecimal representation](http://en.wikipedia.org/wiki/Hexadecimal). The input is entered as long. The output should be a variable of type string. Do not use the built-in .NET functionality. Examples:

|  |  |
| --- | --- |
| **decimal** | **hexadecimal** |
| 254 | FE |
| 6883 | 1AE3 |
| 338583669684 | 4ED528CBB4 |

## \* Calculate GCD

https://www.youtube.com/watch?v=fwuj4yzoX1o

Write a program that calculates the [**greatest common divisor**](http://en.wikipedia.org/wiki/Greatest_common_divisor) (**GCD**) of given two integers a and b. Use the **Euclidean algorithm** (find it in Internet). Examples:

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **GCD(a, b)** |
| 3 | 2 | 1 |
| 60 | 40 | 20 |
| 5 | -15 | 5 |

## \* Trailing Zeroes in N!

<https://www.youtube.com/watch?v=SdXo65Inymg>

Write a program that calculates with how many zeroes the factorial of a given number n has at its end. Your program should work well for very big numbers, e.g. n=100000. Examples:

|  |  |  |
| --- | --- | --- |
| **n** | **trailing zeroes of n!** | **explaination** |
| 10 | 2 | 36288**00** |
| 20 | 4 | 243290200817664**0000** |
| 100000 | 24999 | think why |

## \*\* Spiral Matrix

Write a program that reads from the console a positive integer number n (1 ≤ n ≤ 20) and **prints a matrix** holding the numbers from 1 to n\*n in the form of **square spiral** like in the examples below. Examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | **matrix** |  | **n** | **matrix** |  | **n** | **matrix** |
| 2 | 1 2  4 3 | 3 | 1 2 3  8 9 4  7 6 5 | 4 | 1 2 3 4  12 13 14 5  11 16 15 6  10 9 8 7 |

# Exam problems.\*\*

**All of the problems below are given from Variant 6 of C# Basics Practical Exam (12 April 2014 Evening). You can submit your solutions** [**HERE**](http://judge.softuni.bg/Contests/7/CSharp-Basics-Exam-12-April-2014-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](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.

## \*\* – Exam Schedule

At SoftUni we have a new trainee Stamat, who is assigned to make **schedules for the entrance exams**. Since today is his first day at work he is a little bit nervous and he is not working very fast. Unfortunately, it seems that he will not have enough time to make the schedule for the next exam and there is no one else to do the job … except you of course. You will be given **exam starting time** in the standard 12-hour clock (**hours**, **minutes** and **part of the day**) and **exam duration** (**hours** and **minutes**). Your job is to write a program that calculates **at what time the exam ends**.

\* Note that the **standard 12-hours clock** uses the following arrangements of the hours of the day: 12AM (midnight), 1AM, 2AM, 3AM, 4AM, 5AM, 6AM, 7AM, 8AM, 9AM, 10AM, 11AM, 12PM (noon), 1PM, 2PM, 3PM, 4PM, 5PM, 6PM, 7PM, 8PM, 9PM, 10PM, 11PM, 12AM, 1AM, … (learn more at <http://en.wikipedia.org/wiki/12-hour_clock>).

### Input

The input data should be read from the console. The input data consists of exactly 5 lines:

* The first three lines are holding the exam start time: **hour**, **minutes** and **part of the day (AM or PM)**.
* The last two lines are holding two integer number: the exam **duration hours** and **minutes**.

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

### Output

You have to print the time the exam ends in format **HH:MM:PartOfDay**.

### Constraints

* The **starting hour** will be an integer number in the range [1…12] inclusive.
* The **starting minutes** will an integer number in the range [0…59] inclusive.
* The **part of the day** will consist of exactly two capital letters: **AM** or **PM**.
* The **duration hours** will be an integer number between [0…23] inclusive.
* The **duration minutes** will be an integer number between [0…59] inclusive.
* Allowed work time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 9  30  AM  6  00 | 03:30:PM | 2  0  PM  2  30 | 04:30:PM |  | 11  30  AM  2  0 | 01:30:PM |  | 11  59  PM  0  3 | 12:02:AM |

## \*\* – Odd / Even Elements

You are given **N numbers**. Calculate the **sum**, **min** and **max** of its **odd elements** and **sum**, **min** and **max** of its **even elements.** Consider that the first element is odd, the second is even, etc.

### Input

The input data should be read from the console. It will consists of exactly one line.

* At the first line **N numbers** will be given, separated one from another by a single **space**.

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

### Output

You have to print the output in a single line at the console in the following format:

* **OddSum=…, OddMin=…, OddMax=…, EvenSum=…, EvenMin=…, EvenMax=…**

Print the numbers in the output without any unneeded trailing zeroes (i.e. print 1.5 instead of 1.50; 1 instead of 1.00). In case the sum, the minimal or the maximal element cannot be calculated (due to missing data), print "**No**".

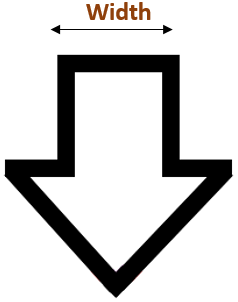
### Constraints

* All numbers in the input will be in the range [-1 000 000 … 1 000 000], with no more than 10 digits (total, before and after the decimal point). The decimal separator in the non-integer numbers will be '**.**' and the numbers will have up to 2 digits after the decimal separator.
* The count N of the numbers in the input is in the range [0 … 1000].
* All numbers in the output should be formatted **without unneded trailing zeroes**.
* Allowed work time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 3 5 4 2 1 | OddSum=9, OddMin=2, OddMax=5, EvenSum=8, EvenMin=1, EvenMax=4 |
| 3 -2 8 11 -3 | OddSum=8, OddMin=-3, OddMax=8, EvenSum=9, EvenMin=-2, EvenMax=11 |
| 1 | OddSum=1, OddMin=1, OddMax=1, EvenSum=No, EvenMin=No, EvenMax=No |
| 1.5 -2.5 | OddSum=1.5, OddMin=1.5, OddMax=1.5, EvenSum=-2.5, EvenMin=-2.5, EvenMax=-2.5 |
| 1.5 1.75 1.5 1.75 | OddSum=3, OddMin=1.5, OddMax=1.5, EvenSum=3.5, EvenMin=1.75, EvenMax=1.75 |

## \*\* – Arrow

SoftUni has opened a new training center in Kaspichan, but the people there did not know how to find it. Your task is to **print a vertical arrow**, which will be used to indicate the path to the new building in the city. This will help thousands of people to become software engineers. Please help them!

### Input

The input data should be read from the console.

* On the only line will hold and integer number **N** (always **odd** number), indicating the **width** of the arrow.

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 “**#**” (number sign) to mark the arrow and “**.**” (dot) for the rest. Follow the examples below.

### Constraints

* **N** will always be a positive **odd** number between **3** and **79** inclusive.
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 5 | ..#####..  ..#...#..  ..#...#..  ..#...#..  ###...###  .#.....#.  ..#...#..  ...#.#...  ....#.... | 9 | ....#########....  ....#.......#....  ....#.......#....  ....#.......#....  ....#.......#....  ....#.......#....  ....#.......#....  ....#.......#....  #####.......#####  .#.............#.  ..#...........#..  ...#.........#...  ....#.......#....  .....#.....#.....  ......#...#......  .......#.#.......  ........#........ |  | 3 | .###.  .#.#.  ##.##  .#.#.  ..#.. |

## \*\* – Five Special Letters

We are given two numbers: **start** and **end**. Write a program to **generate all sequences of 5 letters**, each from the set { '**a**', '**b**', '**c**', '**d**', '**e**' }, such that the weight of these 5 letters is a number in the range [**start** … **end**] inclusively. Print them in alphabetical order, in a single line, separated by a space.

The **weight of a single letter** is calculated as follows: weight('**a**') = **5**; weight('**b**') = **-12**; weight('**c**') = **47**; weight('**d**') = **7**; weight('**e**') = **-32**. The **weight of a sequence** of letters c1c2…cn is the calculated by first removing all repeating letters (from right to left) and then calculate the formula:

weight(c1c2…cn) = 1\*weight(c1) + 2\*weight(c2) + … + n\*weight(cn)

For example, the weight of "**bcddc**" is calculated as follows: First we remove the repeating letters and we get "**bcd**". Then we apply the formula: 1\*weight('**b**') + 2\*weight('**c**') + 3\*weight('**d**') = 1\*(-12) + 2\*47 + 3\*7 = 103. Another example: weight("cadea") = weight("cade") = 1\*47 + 2\*5 + 3\*7 - 4\*32 = -50.

### Input

The input data should be read from the console. It will consist of 2 lines:

* The number **start** stays at the first line.
* The number **end** stays at the second 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 as a sequence of strings in **alphabetical order**. Each string should be separated than the next string by a single space. In case no 5-letter strings exist with a weight in the specified range, print “**No**”.

### Constraints

* The numbers **start** and **end** will be an **integers** in the range [-10000…10000].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 40  42 | bcead bdcea | weight("bcead") = 41  weight("bdcea") = 40 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| -1  1 | bcdea cebda eaaad eaada eaadd eaade eaaed eadaa eadad eadae eadda eaddd eadde eadea eaded eadee eaead eaeda eaedd eaede eaeed eeaad eeada eeadd eeade eeaed eeea | 200  300 | baadc babdc badac badbc badca badcb badcc badcd baddc bbadc bbdac bdaac bdabc bdaca bdacb bdacc bdacd bdadc bdbac bddac beadc bedac eabdc ebadc ebdac edbac | 300  400 | No |

## \*\* – Bit Roller

Nakov enjoys playing with bits very much. Yesterday he invented a new game. He takes a 19-bit number and rolls it on the right (moves its most right bit at the left most position). He tried this several times and he found it is too easy. Then he invented a more complex game: freeze a certain bit as a pillar and roll right all other bits several times. Now he is happy but he wants to automate this process.

Help Nakov to write a program that **rolls r times a 19-bit number n with a frozen bit at position f**.

Example: we have the number **n** = **2521**, which is **0000000100111011001** in binary (as a 19-bit number). We **freeze** the bit at position **f** = **8** (we count the positions from the right, starting from 0). We roll out the number **r** = **4** times. We obtain the result **295245** as follows:

* 2521(10) = 0000000100**1**11011001 🡪 1000000010**1**01101100 (roll right with frozen position 8)
* 1000000010**1**01101100 🡪 0100000001**1**00110110 (roll right with frozen position 8)
* 0100000001**1**00110110 🡪 0010000000**1**10011011 (roll right with frozen position 8)
* 0010000000**1**10011011 🡪 1001000000**1**01001101 = 295245(10) (roll right with frozen position 8)

### Input

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

* The number **n** stays at the first line.
* The number **f** stays at the second line.
* The number **r** stays at the third line.

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

### Output

Print the obtained result after rolling **r** times **n** with a frozen bit at position **f** at the console (as decimal number).

### Constraints

* The number **n** will be a 19-bit unsigned **integer** (in the range [0…524287]).
* The number **f** will be integer in the range [0…18].
* The number **r** will be integer in the range [0…100].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2521  8  4 | 295245 | 2521(10) = 0000000100**1**11011001 🡪 1000000010**1**01101100 🡪 0100000001**1**00110110 🡪 0010000000**1**10011011 🡪 1001000000**1**01001101 = 295245(10) |
| 480678  18  2 | 447849 | 480679(10) = **1**110101010110100110 🡪 **1**011010101011010011 🡪 **1**101101010101101001 = 447849(10) |