# Homework: Operators Expressions and 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.

## Odd or Even Integers

Write an expression that checks if given integer is **odd or even**. Examples:

|  |  |
| --- | --- |
| **n** | **Odd?** |
| 3 | true |
| 2 | false |
| -2 | false |
| -1 | true |
| 0 | false |

## Gravitation on the Moon

The gravitational field of the Moon is approximately 17% of that on the Earth. Write a program that calculates the **weight of a man on the moon** by a given weight on the Earth. Examples:

|  |  |
| --- | --- |
| **weight** | **weight on the Moon** |
| 86 | 14.62 |
| 74.6 | 12.682 |
| 53.7 | 9.129 |

## Divide by 7 and 5

Write a Boolean expression that checks for given integer if it can be **divided** (without remainder) **by 7 and 5 in the same time**. Examples:

|  |  |
| --- | --- |
| **n** | **Divided by 7 and 5?** |
| 3 | false |
| 0 | false |
| 5 | false |
| 7 | false |
| 35 | true |
| 140 | true |

## Rectangles

Write an expression that calculates **rectangle’s perimeter** and **area** by given **width** and **height**. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **width** | **height** | **perimeter** | **area** |
| 3 | 4 | 14 | 12 |
| 2.5 | 3 | 11 | 7.5 |
| 5 | 5 | 20 | 25 |

## Third Digit is 7?

Write **an expression** that checks for given integer **if its third digit** from right-to-left **is 7**. Examples:

|  |  |
| --- | --- |
| **n** | **Third digit 7?** |
| 5 | false |
| **7**01 | true |
| 9**7**03 | true |
| **8**77 | false |
| 777**8**77 | false |
| 9999**7**99 | true |

## Four-Digit Number

Write a program that takes as input a **four-digit number** in format **abcd** (e.g. 2011) and performs the following:

* Calculates the sum of the digits (in our example 2+0+1+1 = 4).
* Prints on the console the number in reversed order: dcba (in our example 1102).
* Puts the last digit in the first position: dabc (in our example 1201).
* Exchanges the second and the third digits: acbd (in our example 2101).

The number has always exactly **4 digits** and cannot start with 0. Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **sum of digits** | **reversed** | **last digit in front** | **second and third digits exchanged** |
| 2011 | 4 | 1102 | 1201 | 2101 |
| 3333 | 12 | 3333 | 3333 | 3333 |
| 9876 | 30 | 6789 | 6987 | 9786 |

## Point in a Circle

Write **an expression** that checks if given point (**x**, **y**) is inside a **circle K**({**0**, **0**}, **2**). Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **y** | **inside** |  |
| 0 | 1 | true |
| -2 | 0 | true |
| -1 | 2 | false |
| 1.5 | -1 | true |
| -1.5 | -1.5 | false |
| 100 | -30 | false |
| 0 | 0 | true |
| 0.2 | -0.8 | true |
| 0.9 | -1.93 | false |
| 1 | 1.655 | true |

## Prime Number Check

Write an **expression** that checks if given positive integer number **n** (**n** ≤ 100) is [**prime**](https://en.wikipedia.org/wiki/Prime_number) (i.e. it is divisible without remainder only to itself and 1). Examples:

|  |  |
| --- | --- |
| **n** | **Prime?** |
| 1 | false |
| 2 | true |
| 3 | true |
| 4 | false |
| 9 | false |
| 97 | true |
| 51 | false |
| -3 | false |
| 0 | false |

## Trapezoids

Write an expression that calculates **trapezoid's area** by given sides **a** and **b** and height **h**. Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a** | **b** | **h** | **area** |  |
| 5 | 7 | 12 | 72 |
| 2 | 1 | 33 | 49.5 |
| 8.5 | 4.3 | 2.7 | 17.28 |
| 100 | 200 | 300 | 45000 |
| 0.222 | 0.333 | 0.555 | 0.1540125 |

## Point Inside a Circle & Outside of a Rectangle

Write an expression that checks for given point (x, y) if it is **within the circle K**({1, 1}, 1.5) and **out of the rectangle R**(top=**1**, left=**-1**, width=**6**, height=**2**). Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **y** | **inside K & outside of R** |  |
| 1 | 2 | yes |
| 2.5 | 2 | no |
| 0 | 1 | no |
| 2.5 | 1 | no |
| 2 | 0 | no |
| 4 | 0 | no |
| 2.5 | 1.5 | no |
| 2 | 1.5 | yes |
| 1 | 2.5 | yes |
| -100 | -100 | no |

## Bitwise: Extract Bit #3

Using bitwise operators, write an **expression** for finding the value of the bit #**3** of a given unsigned integer. The bits are counted from right to left, starting from bit #0. The result of the expression should be either **1 or 0**. Examples:

|  |  |  |
| --- | --- | --- |
| **n** | **binary representation** | **bit #3** |
| 5 | 00000000 0000**0**101 | 0 |
| 0 | 00000000 0000**0**000 | 0 |
| 15 | 00000000 0000**1**111 | 1 |
| 5343 | 00010100 1101**1**111 | 1 |
| 62241 | 11110011 0010**0**001 | 0 |

## Extract Bit from Integer

Write an expression that extracts from given integer **n** the value of given **bit at index** **p**. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **binary representation** | **p** | **bit @ p** |
| 5 | 00000000 00000**1**01 | 2 | 1 |
| 0 | 000000**0**0 00000000 | 9 | 0 |
| 15 | 00000000 000011**1**1 | 1 | 1 |
| 5343 | 00010100 **1**1011111 | 7 | 1 |
| 62241 | 1111**0**011 00100001 | 11 | 0 |

## Check a Bit at Given Position

Write a **Boolean expression** that returns if the **bit at position p** (counting from **0**, starting from the right) in given integer number **n** has value of **1**. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **binary representation of n** | **p** | **bit @ p == 1** |
| 5 | 00000000 00000**1**01 | 2 | true |
| 0 | 000000**0**0 00000000 | 9 | false |
| 15 | 00000000 000011**1**1 | 1 | true |
| 5343 | 00010100 **1**1011111 | 7 | true |
| 62241 | 1111**0**011 00100001 | 11 | false |

## Modify a Bit at Given Position

We are given an integer number **n**, a bit value **v** (v=0 or 1) and a position **p**. Write a **sequence of operators** (a few lines of C# code) that modifies **n** to hold the value **v** at the position **p** from the binary representation of **n** while preserving all other bits in **n**. Examples:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **n** | **binary representation of n** | **p** | **v** | **binary result** | **result** |
| 5 | 00000000 00000**1**01 | 2 | 0 | 00000000 00000**0**01 | 1 |
| 0 | 000000**0**0 00000000 | 9 | 1 | 000000**1**0 00000000 | 512 |
| 15 | 00000000 000011**1**1 | 1 | 1 | 00000000 000011**1**1 | 15 |
| 5343 | 00010100 **1**1011111 | 7 | 0 | 00010100 **0**1011111 | 5215 |
| 62241 | 1111**0**011 00100001 | 11 | 0 | 1111**0**011 00100001 | 62241 |

## \* Bits Exchange

Write a program that **exchanges bits** **3**, **4** and **5** with bits **24**, **25** and **26** of **given 32-bit unsigned integer**. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **binary representation of n** | **binary result** | **result** |
| 1140867093 | 01000**100** 00000000 01000000 00**010**101 | 01000**010** 00000000 01000000 00**100**101 | 1107312677 |
| 255406592 | 00001**111** 00111001 00110010 00**000**000 | 00001**000** 00111001 00110010 00**111**000 | 137966136 |
| 4294901775 | 11111**111** 11111111 00000000 00**001**111 | 11111**001** 11111111 00000000 00**111**111 | 4194238527 |
| 5351 | 00000**000** 00000000 00010100 11**100**111 | 00000**100** 00000000 00010100 11**000**111 | 67114183 |
| 2369124121 | 10001**101** 00110101 11110111 00**011**001 | 10001**011** 00110101 11110111 00**101**001 | 2335569705 |

## \*\* Bit Exchange (Advanced)

Write a program that **exchanges bits** **{p, p+1, …, p+k-1}** with bits **{q, q+1, …, q+k-1}** of a given 32-bit unsigned integer. The first and the second sequence of bits may **not overlap**. Examples:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **n** | **p** | **q** | **k** | **binary representation of n** | **binary result** | **result** |
| 1140867093 | 3 | 24 | 3 | 01000**100** 00000000 01000000 00**010**101 | 01000**010** 00000000 01000000 00**100**101 | 1107312677 |
| 4294901775 | 24 | 3 | 3 | 11111**111** 11111111 00000000 00**001**111 | 11111**001** 11111111 00000000 00**111**111 | 4194238527 |
| 2369124121 | 2 | 22 | 10 | **10001101 00**110101 1111**0111 000110**01 | **01110001 10**110101 1111**1000 110100**01 | 1907751121 |
| 987654321 | 2 | 8 | 11 | - | - | overlapping |
| 123456789 | 26 | 0 | 7 | - | - | out of range |
| 33333333333 | -1 | 0 | 33 | - | - | out of range |

# Exam problems.\*\*

All of the problems below are given from the previous C# Basics exams. **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.

## \*\*– Volleyball !!!!!!!

This problem is from Variant 2 of C# Basics exam from 10-04-2014 Evening. You can test your solution [here](http://judge.softuni.bg/Contests/Practice/Index/3#0) .

Vladi loves a lot to play volleyball. However, he is a programmer now and he is very busy. Now he is able to **play only in the holidays** and in the **weekends**. Vladi plays in **2/3 of the holidays** and each **Saturday**, but **not every weekend** – only when he is **not at work** and only when he is **not going to his hometown**. Vladi goes at his hometown **h** weekends in the year. The other weekends are considered “**normal**”. Vladi is **not at work in 3/4 of the normal weekends**. When Vladi is at his hometown, he always plays volleyball with his old friends once, at **Sunday**. In addition, if the year is **leap**, Vladi plays volleyball **15% more times** additionally. We assume the year has **exactly 48 weekends** suitable for volleyball.

Your task is to write a program that calculates **how many times Vladi plays volleyball** (rounded down to the nearest integer number).

### Input

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

* The string “**leap**” for leap year or “**normal**” for year that is not leap.
* The number **p** – number of **holidays** in the year (which are not Saturday or Sunday).
* The number **h** – number of weekends that Vladi spends in his **hometown**.

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 an integer representing how many times Vladi plays volleyball for a year.

### Constraints

* The numbers **p** is in range [0...300] and **h** is in range [0…48].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| leap  5  2 | 45 | 48 weekends total in the year, split into:   * 2 hometown weekends 🡪 2 Sundays 🡪 **2** plays * 46 normal weekends 🡪 46 \* 3 / 4 🡪 **34.5** plays   5 holidays 🡪 5 \* 2/3 🡪 **3.33** plays  Leap year 🡪 additional 15% \* 39.83 🡪 **5.97** plays  Total plays = **45.8** plays 🡪 **45** (rounded down) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| normal  3  2 | 38 | leap  2  3 | 43 | normal  11  6 | 44 | leap  0  1 | 41 | normal  6  13 | 43 |

## \*\* – Odd / Even Sum

This problem is from Variant 2 of C# Basics exam from 10-04-2014 Evening. You can test your solution [here](http://judge.softuni.bg/Contests/Practice/Index/3#1) .

You are given a number **n** and **2\*n** numbers. Write a program to check whether **the sum of the odd numbers is equal to the sum of the even n numbers**. The first number is considered odd, the next even, the next odd again, etc. Print as result “Yes” or “No”. In case of **yes**, print also the sum. In case of **no**, print also the difference between the odd and the even sums.

### Input

The input data should be read from the console.

* The first line holds an integer **n** – the count of numbers.
* Each of the next **2\*n** lines holds exactly one number.

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

### Output

* The output must be printed on the console.
* Print “**Yes, sum=*S***” where ***S*** is the sum of the odd **n** numbers in case of the sum of the odd **n** numbers is equal to the sum of the even **n** numbers.
* Otherwise print “**No, diff=*D***” where ***D*** is the difference between the sum of the odd **n** numbers and the sum of the even **n** numbers. ***D*** should always be a **positive number**.

### Constraints

* The number **n** is integer in range [0...500].
* All other numbers are integers in range [-500 000 ... 500 000].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| **4**  3  **4**  -1  **-1**  2  **1**  1  **1** | Yes, sum=5 | **3**  1  **2**  3  **1**  2  **2** | No, diff=1 | **2**  1  **0**  1  **0** | No, diff=2 |

## \*\* – The Explorer

This problem is from Variant 3 of C# Basics exam from 11-04-2014 Morning. You can test your solution [here](http://judge.softuni.bg/Contests/Practice/Index/4#2) .

Bai Vylcho is very an enthusiastic explorer. His passion are the diamonds, he just adores them. Today he is going on an expedition to collect all kind of diamonds, no matter small or large. Help your friend to find all the diamonds in the biggest known cave "The Console Cave". At the only input line you will be given the width of the diamond. The char that forms the outline of the diamonds is '**\***' and the surrounding parts are made of '**-**' (see the examples). Your task is to **print a diamond** of given size **n**.

### Input

Input data should be read from the console.

* The only input line will hold the width of the diamond – **n**.

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.

* The output lines should hold the diamond.

### Constraints

* The number **n is positive odd integer between 3 and 59**, inclusive.
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

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

## \*\* – Bits Up

This problem is from Variant 2 of C# Basics exam from 10-04-2014 Evening. You can test your solution [here](http://judge.softuni.bg/Contests/Practice/Index/3#4) .

You are given a **sequence of bytes**. Consider each byte as sequences of exactly 8 bits. You are given also a number **step**. Write a program to set to 1 the bits at positions: **1**, **1 + step**, **1 + 2\*step**, ... Print the output as a sequence of bytes.

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 exactly **n** bytes, each at a separate line and in range [0..255], obtained by applying the bit inversions over the input sequence.

### 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 | 109  95 | We have the following sequence of 16 bits (2 bytes):  0**1**101101 0101**0**111  We invert the bits 1 and 12 (step=11). We get:  0**1**101101 0101**1**111 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  5  45  87  250 | 111  87  254 | We have the following sequence of 24 bits (3 bytes):  0**0**1011**0**1 010**1**0111 **1**1111**0**10  We invert the bits 1, 6, 11, 16 and 21 (step=5). We get:  0**1**1011**1**1 010**1**0111 **1**1111**1**10 |

## \*\* – Bit Sifting

This problem is from Variant 3 of C# Basics exam from 11-04-2014 Morning. You can test your solution [here](http://judge.softuni.bg/Contests/Practice/Index/4#4) .

In this problem we'll be sifting bits through sieves (sift = пресявам, sieve = сито).

You will be given an integer, representing the **bits to sieve**, and several more numbers, representing the **sieves the bits will fall through**. Your task is to follow the bits as they fall down, and determine what comes out of the other end.

|  |  |
| --- | --- |
| ExampleFor this example, imagine we are working with 8-bit integers (the actual problem uses 64-bit ones). Let the initial bits be given as 165 (10100101 in binary), and the sieves be 138 (10001010), 84 (01010100) and 154 (10011010). The 1 bits from the initial number fall through the 0 bits of the sieves and stop if they reach a 1 bit; if they make it to the end, they become a part of the final number.In this case, the final number is 33 (00100001), which has two 1 bits in its binary form – the answer is 2. | 10100101↓ ↓ ↓ ↓10001010↓ ↓ ↓01010100↓ ↓10011010↓ ↓ 00100001 |

### Input

The input data should be read from the console.

* On the first line of input, you will read an integer representing the bits to sieve.
* On the second line of input, you will read an integer N representing the number of sieves.
* On the next N lines of input, you will read N integers representing the sieves.

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

### Output

The output must be printed on the console.

On the single line of the output you must print **the count of "1" bits** in the final result.

### Constraints

* All numbers in the input will be between 0 and 18,446,744,073,709,551,615.
* The count of sieves N is in range [0…100].
* Allowed work time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 584938644408189469  3  1817781288526917737  8601652436058397548  51827709899390606 | 4 | 918045605434484408  0 | 35 | 5019588773529942006  1  5295337384025297044 | 17 |