# Lab: Bitwise Operations

Problems for in-class lab for the “**Bitwise Exercises**” lesson from the “Programming Fundamentals” course at SoftUni. Submit your solutions in the automated Judge system: <https://judge.softuni.bg/Contests/2907>.

## Binary Digits Count

You are given a positive integer **number** and one binary digit b (0 or 1). Your task is to write a program that finds the **count of binary digits** b in given integer.

### Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Comments** |  | **Input** | **Output** | **Comments** |
| 20  0 | 3 | 20 -> 1**0**1**00**  We have **3 zeroes**. | 10  0 | 2 | 10 -> 1**0**1**0**  We have **2 zeroes**. |
| 15  1 | 4 | 15 -> **1111**  We have **4 ones**. | 22  1 | 3 | 22 -> **1**0**11**0  We have **3 ones**. |

### Hints

1. Declare **two** variables (**n** and **b**).
2. Read the user input from the console.
3. Convert the **n** into **binary representation** (you can use built-in method).
4. Count the **b** digit in the binary number.
5. Print the result on the console.

## Train

You will be given a count of **wagons** in a **train** **n**. On the next **n** lines you will receive how many people are going to get on each wagon. At the end print the entire train, and after it the sum of the people in the train, on the next line.

### Examples

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3  13  24  8 | 13 24 8  45 | 6  3  52  71  13  65  4 | 3 52 71 13 65 4  208 | 2  10  20 | 10 20  30 | 1  100 | 100  100 |

## Bit at Position 1

Write a program that prints the bit at **position 1** of given integer. Use the standard counting of bits: from right to left, starting from 0.

The **input** holds a **positive integer**.

The **output** should hold a single **digit** (0 or 1).

### Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Comments** |  | **Input** | **Output** | **Comments** |
| 2 | 1 | 000000**1**0 🡪 **1** | 13 | 0 | 00001101 🡪 0 |
| 51 | 1 | 001100**1**1 🡪 1 | 24 | 0 | 00011000 🡪 0 |

### Hints

1. Declare **two** variables (n and bitAtPosition1).
2. **Read** the user input from the console.
3. **Find** the **value** of the **bit at position 1** (position 1 is the second bit from right to left: [7, 6, 5, 4, 3, 2, **1**, 0]):
   1. **Shift** the number **n** times to the **right** (where **n** is the position, in this case it is **1**) by using the **>>** operator. In that way the bit we want to check will be at position **0**.
   2. **Find** the bit at **position 0**. Use **& 1** operator expression to extract the value of a bit. By using the following **formula** (**bitAtPosition1 & 1**) you **check** whether the bit at **position 0** is equal to **1** or **not**. If the bit is **equal** to **1** the **result** is **1** if the bit is **not** **equal** - the **result** is **0**.
   3. **Save** the result in **bitAtPosition1**.
4. **Print** the result on the console.

## P-th Bit

Write a program that prints the bit at position **p** of given integer. We use the standard counting: from right to left, starting from 0.

### Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Comments** |  | **Input** | **Output** | **Comments** |
| 2145  5 | 1 | 0000100001**1**00001 🡪 **1** | 111  8 | 0 | 0000000001101111 🡪 0 |
| 512  0 | 0 | 0000001000000000 🡪 0 | 255  7 | 1 | 0000000011111111 🡪 **1** |

### Hints

1. Declare **three** variables (**n**, **p** and **bitAtPositionP**).
2. **Read** the user input from the console.
3. **Find** the **value** of the **bit at position p**:
   1. **Shift** the number **p** times to the **right** (where **p** is the position) by using the **>>** operator. In that way the bit we want to check will be at position **0**;
   2. **Find** the bit at **position 0**. Use **& 1** operator expression to extract the value of a bit. By using the following **formula** (bitAtPositionP & 1) you **check** whether the bit at **position 0** is equal to **1** or **not**. If the bit is **equal** to **1** the **result** is **1** if the bit is **not** **equal** - the **result** is **0**;
   3. **Save** the result in **bitAtPosition1;**
4. **Print** the result on the console.

## Bit Destroyer

Write a program that sets the bit at **position** **p** to **0**. Print the resulting integer.

### Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Comments** |  | **Input** | **Output** | **Comments** |
| 1313  5 | 1281 | 010100100001 🡪 010100000001 | 111  6 | 47 | 000001101111 🡪 000000101111 |
| 231  2 | 227 | 000011100111 🡪 000011100011 | 111  4 | 111 | 000001101111 🡪 000001101111 |

### Hints

1. Declare **four** variables (n, p, mask and newNumber).
2. **Read** the user input from the console.
3. **Set** the **value** of the **bit at position p** to **0**:
   1. **Shift** the number **1**, **p** times to the **left** (where **p** is the position) by using the **<<** operator. In that way the bit we want to delete will be at position **p**. Save the resulting value in **mask**.
   2. **Invert** the **mask** (e.g. we move the number 1, 3 times and we get 00001000, after inverting we get 11110111).
   3. Use **& mask** operator expression to **set** the **value** of a number to **0**. By using the following **formulae** (n & mask) you **copy** **all** the **bits** of the **number** and you **set** the bit at **position** **p** to **0**.
   4. **Save** the result in **newNumber**.
4. **Print** the result on the console.

## \* Odd Times

You are given an **array of positive integers** in a single line, separated by a space (' '). All numbers occur even number of times except one number which occurs odd number of times. Find it, using only bitwise operations.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 2 3 1 3 | 3 |
| 5 7 2 7 5 2 5 | 5 |

### Hints

1. Read an array of integers.
2. Initialize a variable **result** with value **0**.
3. Iterate through all number in the array.
4. Use **XOR (^)** of **result** and **all numbers** in the **array**.
   1. **XOR** of **two elements** is **0** if **both elements** are **same** and **XOR** of a number **x** with **0** is **x**.
5. Print the **result**.

Think why the above algorithms is correct.

## \* Tri-bit Switch

Write a program that inverts the **3 bits** from position **p** to the left with their XOR opposites (e.g. **111** -> **000**, **101** -> **010**) in 32-bit number. Print the resulting integer on the console.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1234  7 | 1874 | 0000000000000000000001**001**1010010 🡪  0000000000000000000001**110**1010010 |
| 44444  4 | 44524 | 0000000000000000101011011**001**1100 🡪  0000000000000000101011011**110**1100 |

### Hints

1. **Shift** the number **7** (the number 7 has the bits 111 which we use to get 3 consecutive values), **p** times to the **left** (where **p** is the position) by using the **<<** operator. In that way the **3 bits** we want to **invert** will be at position **p**. Save the resulting value in **mask**.
2. Use **^ mask** operator expression to **invert** the **values** of the **three** **bits** starting from position **p**. By using the following **formulae** (n ^ mask) you **copy** all the **bits** of the **number**, and you **invert** the bits at position **p**, **p+1** and **p+2**.
3. Save the result in **result**.