Homework: C Bit Manipulation

This document defines the homework assignments from the "C Programming" Course @ Software University. Please submit as homework a single **zip** / **rar** / **7z** archive holding the solutions (source code) of all below described problems.

Problem 1. First Bit

Write a program that prints the bit at **position 1** of a number.

n	Result
2	1
51	1
13	0
24	0

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 00000101	0
0	00000000 00000000	0
15	00000000 00001111	1
5343	00010100 11011111	1
62241	11110011 00100001	0

Problem 2. 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 00000101	2	1
0	00000000 00000000	9	0
15	00000000 00001111	1	1
5343	00010100 11011111	7	1
62241	11110011 00100001	11	0

Problem 3. 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 00000101	2	true
0	00000000 00000000	9	false

15	00000000 00001**1**1	1	true
5343	00010100 **1**1011111	7	true
62241	1111**0**011 00100001	11	false

Problem 4. Bit Destroyer

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

n	p	Result
1313	5	1281
231	2	227
111	6	47

Problem 5. 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 00001**1**1	1	1	00000000 00001**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

Problem 6. 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

Problem 7. Bits 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	**01110001 10**110101	1907751121

				11110111 00011001	11111000 11010001	
987654321	2	8	11	-	-	overlapping
123456789	26	0	7	-	-	out of range
33333333333	-1	0	33	-	-	out of range

Problem 8. ** Bits Up

This problem is from Variant 2 of C# Basics exam from 10-04-2014 Evening. You can test your solution here .

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): 01101101 01010111 We invert the bits 1 and 12 (step=11). We get: 01101101 01011111

Input	Output	Comments
3 5 45	111 87 254	We have the following sequence of 24 bits (3 bytes): 00101101 01010111 11111010 We invert the bits 1, 6, 11, 16 and 21 (step=5). We get:

87 250		01101111 01010111 11111110

Problem 9. ** Bit Sifting

This problem is from Variant 3 of C# Basics exam from 11-04-2014 Morning. You can test your solution here .

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.

Example

For 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
5849386444081894693 3	4

Input	Output
9180456054344844080 0	35

Input	Output
5019588773529942006 1	17

1817781288526917737860165243605839754851827709899390606	

5295337384025297044	