# Fundamentals of C# Programming – Intermediate Exam – March 2010

## Variant 3

## Problem 1 – Bitwise XOR K-Encryption

Write a program that reads from the console an integer number **n** (0<**n**<1000), a sequence **b[]** of **n** bytes (integer numbers in the range [0..255]), an integer number **k** in the range [1..7] and a sequence of **k** bits (string of binary digits ‘0’ or ‘1’) called **cipher**. Consider the input sequence **b[]** as a sequence of bits and the following encryption scheme for the numbers in **b[]**: first the sequence is padded with **z** zeroes on the left to a multiple of **k** bits; later it is split into regions of **k** bits; and last each region of **k** bits is encrypted by applying “bitwise exclusive or” with the **cipher**. Finally the last **z** bits are stripped to obtain a multiple of 8 bits. Your task is to calculate and print the obtained encrypted sequence of bytes on the console, on a single line, using a space as separator.

**Note:** You are not allowed to convert the input into a sequence of bits (e.g. as string, as list or as array of bits or as other data structure holding the bits). You should process the input using bitwise operations.

The input data will be correct and it is not required to check it explicitly.

|  |  |  |  |
| --- | --- | --- | --- |
| Sample input  |  | | --- | | **n = 3**  **b[0] = 172**  **b[1] = 169**  **b[2] = 89**  **k = 5**  **cipher = 01101** | | Sample output  |  | | --- | | **61 14 122** | |

### Explanation

The input sequence of 3 bytes **b[]** is represented as a sequence of 24 bits as follows:

|  |
| --- |
| **10101100 10101001 01011001** |

After padding the sequence to 25 bits (25 is the closest multiple of 5) by inserting z=1 zeroes at its start and splitting it into regions of k=5 bits, it takes the following form:

|  |
| --- |
| **01010 11001 01010 01010 11001** |

After performing XOR of the sequence’s regions with the **cipher** we obtain:

|  |
| --- |
| **01010 11001 01010 01010 11001** // the input sequence split into 5-bits regions  **XOR**  **01101 01101 01101 01101 01101** // the cipher repeated 5 times  **=**  **00111 10100 00111 00111 10100** // the result sequence of 5-bits regions |

After stripping the last z=1 bits of the result, it can naturally be split to 8-bit sequences (3 bytes):

|  |
| --- |
| **00111101 00001110 01111010** // the result without the last z bits as 8-bit regions  **= 61 14 122** // the result sequence as decimal bytes |

## Problem 2 – Increasing Subsets

Write a program that reads from the console an integer number **n** (1 < **n** < 20 000) and a sequence **a[]** of **n** integer numbers in the range [0..1000]. Find how many ordered subsets of at least two increasing numbers exist in the sequence **a[]**.

The output should consist of a single number and should be printed on the console.

The input data will be correct and it is not required to check it explicitly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sample input  |  | | --- | | **n = 5**  **a[0] = 1**  **a[1] = 3**  **a[2] = 2**  **a[3] = 4**  **a[4] = 8** | | Sample output  |  | | --- | | **18** |  The ordered subsets of increasing numbers  |  | | --- | | **{1,3} {1,2} {1,4} {1,8} {3,4} {3,8} {2,4} {2,8} {4,8} {1,3,4} {1,3,8} {1,2,4} {1,2,8} {1,4,8} {3,4,8} {2,4,8} {1,3,4,8} {1,2,4,8}** | |

## Problem 3 – Horse Matrix

We are given a square matrix of **n** x **n** cells. We need to fill it by walking in the way horses walk in the chess. At the figure below it is shown where a horse located at the central cell (indicated with “**o**”) can go with a single move – the 8 cells indicated with “\*”.

horse-movesWe start filling the matrix with numbers from 1 to **n2** from its top left corner. At each step we go to this an empty horse-neighbor cell, which is located at the smallest row possible and as close as possible to the start of this row. If no such empty cell is available, we restart filling from an empty cell at the smallest possible row and as close as possible to the start of this row. When no empty cell is left in the matrix, the filling is finished.

Your task is to write a program that reads from the console an integer number **n** (1 ≤ **n** ≤ 100) and displays the filled matrix on the console.

The input data will be correct and it is not required to check it explicitly.

|  |  |  |  |
| --- | --- | --- | --- |
| Sample input  |  | | --- | | **n = 6** | | Sample output  |  | | --- | | **1 8 5 12 3 10**  **6 13 2 9 18 23**  **27 16 7 4 11 29**  **14 21 26 17 24 19**  **32 28 15 20 30 33**  **22 34 31 25 35 36** | |

## Problem 4 – “Escape from Labyrinth” Game

Your task is to write an interactive console-based game in which the player should escape from a labyrinth of size 7 x 7 cells. Each cell is either free (‘-‘) or occupied (‘X’). The labyrinth should consist of randomly generated free and occupied cells and the player’s position is initially in its center. The player is shown as ‘\*’. In the randomly generated labyrinth at least one exit should always be reachable by a sequence of moves in the standard 4 directions: left, right, up, down. At each turn the player enters a single letter – the direction to follow (L - left, R - right, U - up, D - down). As a response the computer either moves the player position to the specified empty cell or indicates that the cell is occupied and the requested move is invalid. If the player’s position is at some of the labyrinth’s walls, the game is considered finished.

### Example game session

The player’s input is shown in *italic*:

|  |
| --- |
| **Welcome to “Labirinth” game. Please try to escape.**  **- - X X X X -**  **- X - - - - X**  **- X - X X - X**  **- X - \* X - X**  **- X - X - - -**  **- X - - - X X**  **X - X - - X X**  **Enter your move (L=left, R-right, U=up, D=down): *L***  **- - X X X X -**  **- X - - - - X**  **- X - X X - X**  **- X \* - X - X**  **- X - X - - -**  **- X - - - X X**  **X - X - - X X**  **Enter your move (L=left, R-right, U=up, D=down): *L***  **Invalid move!**  **Enter your move (L=left, R-right, U=up, D=down): *D***  **- - X X X X -**  **- X - - - - X**  **- X - X X - X**  **- X - - X - X**  **- X \* X - - -**  **- X - - - X X**  **X - X - - X X**  **Enter your move (L=left, R-right, U=up, D=down): *D***  **- - X X X X -**  **- X - - - - X**  **- X - X X - X**  **- X - - X - X**  **- X - X - - -**  **- X \* - - X X**  **X - X - - X X**  **Enter your move (L=left, R-right, U=up, D=down): *R***  **- - X X X X -**  **- X - - - - X**  **- X - X X - X**  **- X - - X - X**  **- X - X - - -**  **- X - \* - X X**  **X - X - - X X**  **Enter your move (L=left, R-right, U=up, D=down): *D***  **Congratulations! You escaped.** |

Some players could try to cheat by entering illegal moves, so be cautious.