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

## Variant 2

## Problem 1 – Bitwise Split

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]) and an integer number **k** in the range [1..50]. Consider the input sequence **b[]** as a sequence of bits. Your task is to split **b[]** into **k** sequences of bits **s[0]**, **s[1]**, …, **s[k-1]** so that the first (leftmost) bit of **b[]** is appended to **s[0]**, the second – to **s[1]**, …, bit **(k-1)** – to **s[k-1]**, bit **k** – to **s[0]**, etc. Print the obtained sequences of bits as sequences of decimal bytes. If the number of bits is not multiple of 8, the sequences should first be padded with zeroes at the right to a multiple of 8 bits. The result should consist of **k** lines printed on the console, containing the bytes in the sequences **s[0]**, **s[1]**, …, **s[k-1]** (after padding) separated by a space (see the sample).

**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 = 4**  **b[0] = 154**  **b[1] = 149**  **b[2] = 53**  **b[3] = 15**  **k = 3** | | Sample output  |  | | --- | | **s[0] = 231 32**  **s[1] = 74 96**  **s[2] = 49 64** | |

### Explanation

The input sequence of 4 bytes **b[]** is represented as a sequence of 32 bits as follows:

|  |
| --- |
| **10011010 10010101 00110101 00001111** |

It is split into k=3 sequences as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **s** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** | **2** | **0** | **1** |
| **b** | **1** | **0** | **0** | **1** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **1** |

Thus the output sequences of bits are as follows:

|  |
| --- |
| **s[0] = 11100111 001 // after padding we have 11100111 00100000 == 231 32**  **s[1] = 01001010 011 // after padding we have 01001010 01100000 == 74 96**  **s[2] = 00110001 01 // after padding we have 00110001 01000000 == 49 64** |

## Problem 2 – K-Subsequences

Write a program that reads from the console an integer number **n** (1 < **n** < 100 000), a sequence **a[]** of **n** integer numbers in the range [-1000..1000] and integer numbers **p** and **k** in the range [1..**n**]. Find **k** different subsequences **b0**, **b1**, …, **bk**, each consisting of **p** consecutive numbers from **a[]** with a maximal total sum of all their elements. The p-element subsequences are allowed to overlap.

The output should be printed on the console and should consist of several lines. The first line should contain the sum of the elements of those k subsequences **b0**, **b1**, …, **bk** with a maximal sum. If **k** subsequences of **p** elements do not exist in **a[]**, print “No”. The next **k** lines should contain the subsequences found in format **start..end**. If multiple solutions exist with the same maximal sum, print only one of them.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Sample input  |  | | --- | | **n = 10**  **a[0] = 8**  **a[1] = 12**  **a[2] = 5**  **a[3] = 4**  **a[4] = 8**  **a[5] = 9**  **a[6] = 1**  **a[7] = 4**  **a[8] = 3**  **a[9] = 4**  **p = 2**  **k = 3** | | Sample output  |  | | --- | | **54**  **0..1**  **1..2**  **4..5** | |

## Problem 3 – Tetris Matrix

We are given a square matrix of **n** x **n** cells, initially filled with dash symbols “**-**“. We have the following four Tetris-like figures:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a) | tetris-a | b) | tetris-b | c) | tetris-c | d) | tetris-d |

We start filling the matrix as follows: first we place figure a), next – figure b), next – figure c), next – figure d), next – figure a), etc. Each placement is done at empty position (over dashes only) at the smallest possible row and as close as possible to the start of this row. When the current figure cannot be placed anywhere in the matrix, it is skipped. When no figure can be placed anywhere in the matrix, the filling process is considered 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 of letters on the console.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Sample input  |  | | --- | | **n = 6** | | Sample output  |  | | --- | | **a - - b c c**  **a a b b a c**  **d d b – a a**  **d b b c c -**  **d d a – c b**  **d – a a b b** | |

## Problem 4 – “Hangman” Game

Your task is to write an interactive console-based implementation of the game “Hangman” in which the player tries to guess a secret word by guessing its letters sequentially. At the start of the game the computer generates a random secret word from a predefined set of 10 words: *computer*, *programmer*, *software*, *debugger*, *compiler*, *developer*, *algorithm*, *array*, *method* and *variable*. The generated word is initially shown as underscores (e.g. if the secret letter is *developer*, it is shown as \_ \_ \_ \_ \_ \_ \_ \_ \_). At each step the player enters a guess letter to the computer and the computer reveals all occurrences of this letter in the secret word while the other letters remain shown as underscores. For example, if the secret word is *developer* and the guess letter is *e*, after revealing this letter, the result will be as follows: *\_ e \_ e \_ \_ \_ e \_*. Initially the player has health of 5. In case of zero occurrences of the guessed letter, the health of the player is decreased. If the health reaches 0, the game is considered lost and a message “You lost!” should be shown. If the player guesses all the letters in the secret word, the game is considered won.

### Example game session

In this game session the computer’s secret random word is *developer*. The player’s input is shown in *italic*:

|  |
| --- |
| **Welcome to “Hangman” game. Please try to guess my secret word. Your health is 5.**  **The secret word is: \_ \_ \_ \_ \_ \_ \_ \_ \_**  **Enter your guess: *e***  **Good job! You revealed 3 letters. Your health is 5.**  **The secret word is: \_ e \_ e \_ \_ \_ e \_**  **Enter your guess: *a***  **Sorry! There are no unrevealed letters “a”. Your health is now 4.**  **The secret word is: \_ e \_ e \_ \_ \_ e \_**  **Enter your guess: *r***  **Good job! You revealed 1 letter. Your health is 4.**  **The secret word is: \_ e \_ e \_ \_ \_ e r**  **Enter your guess: *d***  **Good job! You revealed 1 letter. Your health is 4.**  **The secret word is: d e \_ e \_ \_ \_ e r**  **Enter your guess: *v***  **Good job! You revealed 1 letter. Your health is 4.**  **The secret word is: d e v e \_ \_ \_ e r**  **Enter your guess: *l***  **Good job! You revealed 1 letter. Your health is 4.**  **The secret word is: d e v e l \_ \_ e r**  **Enter your guess: *o***  **Good job! You revealed 1 letter. Your health is 4.**  **The secret word is: d e v e l o \_ e r**  **Enter your guess: *p***  **You won! Your health is 4.**  **The secret word is: d e v e l o p e r** |

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