# C++ Programming: Judge Assignment 3 (JA3)

The following tasks should be submitted to the SoftUni Judge system, which will be open starting Saturday, 2 May 2017, 10:00 (in the morning) and will close on Sunday, 14 May 2017, 23:59. Submit your solutions here: <https://judge.softuni.bg/Contests/547/Judge-Assignment-3-JA3-Algorithms-STL-Data-Structures>

Solutions for each task will be submitted in the form of compressed archive (.zip) files, containing .h and .cpp files.

Please be mindful of the strict input and output requirements for each task, as well as any additional requirements on running time, used memory, etc., as the tasks are evaluated automatically and not following the requirements strictly may result in your program’s output being evaluated as incorrect, even if the program’s logic is mostly correct.

You can use C++03 and C++11 features in your code.

Unless explicitly stated, any integer input fits into int and any floating-point input can be stored in double.

NOTE: this assignment is focused on data structures and algorithms. Some tasks require trivial algorithms, but others require efficient usage of data structures and finding optimum algorithms to fit into the time and memory constraints. If you are having difficulties with thinking up an efficient algorithm or data structure – check the STL documentation on running time of various data structures, look up similar problems or sub-problems of the task you’re solving on the Internet, try to find standard algorithms which solve those sub-problems efficiently and combine them – you could also ask for hints or guidance in the forum (<https://softuni.bg/forum/categories/42/cplusplus-programming>), as long as you don’t expect someone to solve the tasks for you.

NOTE: the tasks here are NOT ordered by difficulty level.

## Task 2 – Rox (JA3-Task-2-Rox)

You are a fictional astronaut on the fictional planet Rox (*well, the planet is fictional at least, you shouldn’t be if you’re reading this*). There are a lot of rocks on the planet Rox, and there are a lot of microorganisms on those rocks. Since those microorganisms are the native life form on Rox, let’s call them… well, Roxs doesn’t sound very well… so… Roxettes!

There’s an interesting thing about Roxettes – they live in groups, each group is isolated from the others. Roxettes have DNA (like Earth organisms do) and they multiply by dividing into two organisms with the same DNA. Each group has one unique Roxette – the leader, which never multiplies – and multiple other duplicate Roxettes.

That means the **DNA of the leader is unique among the group**, and for **each other DNA in the group there are 2, 4, 6, 8, etc. organisms with the same DNA** (it is also possible that the unique DNA is the only DNA in the group). That is, for each group you have **1 unique DNA**, and all **other DNAs are encountered an even number of times** ("even" means divisible by 2).

You have a device which scans a group of Roxettes and gives you a large sequence of characters, representing all the Roxettes in the group. The DNA of each Roxette in the sequence is a hexadecimal number represented by exactly 5 characters (00001 to fffff) from the scan of the device.

Unfortunately, the sequence of characters representing the DNAs is very, very large, and NASA’s budget has been bit low lately, so the device has very little memory (you can’t hope to have enough memory to keep info on all the DNAs in the group when working with larger groups).

Write a program which can find the unique DNA in a group of Roxettes, where each DNA except the unique one is encountered an even number of times, using as little memory as possible.

### Input

A single line containing sequence of hexadecimal characters (0-9, a-f), of which each 5 consecutive characters represent the DNA of a single Roxette. So characters 0, 1, 2, 3 and 4 represent the DNA of the first Roxette, characters 5, 6, 7, 8, 9 represent the DNA of the second Roxette and so on.

There are no spaces or other delimiters and the sequence ends with '.' (dot), i.e. char(46).

### Output

A single line containing exactly 5 hexadecimal characters, representing the unique Roxette DNA.

### Restrictions

The total number of DNAs in the input will be less than or equal to 4194240 (so that gives us a maximum of 4194240\*5 = 20971200 symbols in the input, plus one '\0' and end of line symbols '\r' and '\n', since the Judge system works on Windows, for a total of a maximum of 20971203 symbols in the input). DNAs will be hexadecimal numbers from 1 to 1048575 (0001 to fffff).

The total running time of your program should be no more than 2.5s

The total memory allowed for use by your program is 2MB, however keep in mind that (as of the time writing this) an empty C++ program running on Judge will consume about 1.77MB, without any variables in the program. Overall, you should try to keep memory used for variables, arrays, etc., to less than 100KB to be sure you will fit into the memory constraint.

In 50% of the test cases, the total number of DNAs will be less than or equal to 5000 and their numerical values will be between 1 and 10000 (00001 to 02710 in hexadecimal).

### Example I/O

|  |  |
| --- | --- |
| Example Input | Expected Output |
| abab1bca98abab1. | bca98 |
| 44444111112222244444222224444444444. | 11111 |
| 01aefabcdeabcde01aef01aefb00b501aef. | b00b5 |