# Homework: Combinatorial Algorithms

This document defines the **homework assignments** for the ["Algortihms" course @ Software University](https://softuni.bg/trainings/1907/algorithms-march-2018#lesson-8055). Submit your solutions to the automated [Judge](https://judge.softuni.bg/Contests/695/Combinatorial-Algoritms) system.

## Iterative Permutations without Repetitions

Write a **non-recursive algorithm** to generate all permutations without repetition. There shouldn’t be any recursive calls in your program (only loops). You may use the judge system to check whether your solution is correct.

Hint: <http://www.quickperm.org/>

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| A B C | A B C  A C B  B A C  B C A  C B A  C A B |

## Iterative Combinations without Repetition

Write an **iterative** program to generate all combinations (without repetition) of **k** elements from a set of **n** elements. Remember, in combinations, the order of elements doesn’t matter – (1 2) and (2 1) are considered the same combination. You are not allowed to use recursion. Search the Internet for a suitable algorithm.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| A B C  2 | A B  A C  B C |

## Iterative Permutations with Repetition

### Write a program to generate all permutations with repetition of a given multi-set. Ensure your program efficiently avoids duplicated permutations. Test it with { 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 }.

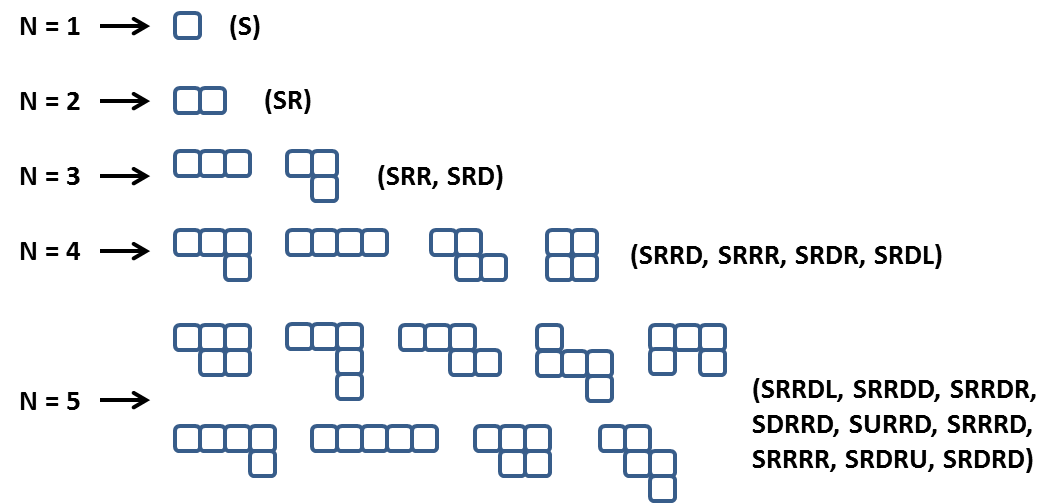
### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| A B B | A B B  B A B  B B A |

## \* Snakes

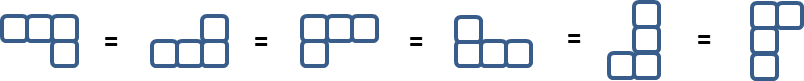
A **snake** is a sequence of several square blocks, attached one after another. A snake starts with a block at some position and continues with another block to the left, right, up or down, then again with another block to the left, right, up or down, etc. A snake of size **N** consists of a sequence of **N** blocks and is not allowed to cross itself.

You are given a number **N** and you should find all possible snakes of **N** blocks, represented as sequences of moves denoted as: **S** (start), **L** (move left), **R** (move right), **U** (move up) and **D** (move down). Examples (for N = 1, 2, 3, 4, and 5):

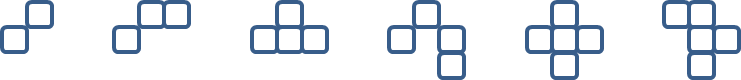


Note: some figures could look visually the same but represent different snakes, e.g. **SRRDL** and **SRDRU**.

Some snakes (sequences of blocks) are the same and should be printed only once. If after a number of rotations and/or flips two snakes are equal they are considered the same and should be printed only once. For example the snakes **SRRD**, **SRRU**, **SLLD, SLLU**, **SRUU** and **SUUR** are the same:



Not all forms consisting of N blocks are snakes of size N. Examples of non-snake forms:



**Note: When generating the snakes, there may be different correct answers. When testing your solution, priority should be as follows: R -> D -> L -> U. The visual example above for n = 5 does NOT follow this priority.**

### Input

* The input should be read from the console.
* It will contain an integer number **N** in the range [1 ... 15].
* 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. It should consist of a variable number of lines:
* Each line should hold a snake represented as a sequence of moves.
* On the last line, print the number of snakes in format: **"Snakes count = {0}"**.

### Constraints

* Allowed working time for your program: 10 seconds. Allowed memory: 512 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Sample Output** | **Comments** |
| 2 | SR  Snakes count = 1 | Note that **SU**, **SL** and **SD** are also correct outputs. However, SR takes precedence because R has priority over all other directions. |
| 4 | SRRR  SRRD  SRDR  SRDL  Snakes count = 4 | Note that there are many other correct outputs for N = 4, but this is the expected output according to the priority of directions (right, down, left, up). |

## \* Cubes

You are given 12 sticks of the same length, each colored with a color in the range [1 … 4]. Write a program to find the number of different cubes that can be built using these sticks. Note that two cubes are equal if a sequence of rotations exists that transforms the first cube to the second. For example, the first two cubes below are equal (after two rotations) but are different from the third cube:

cubes

Print on the console the number of different cubes that can be obtained using the sticks provided.

### Input

* The input data should be read from the console. It will consist of a single line that contains 12 numbers in the range [1 … 4] separated by spaces.
* 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. It should consist of 1 line:
* On the only output line print a single integer number, representing the number of cubes that can be created using the provided sticks.

### Constraints

* Allowed working time for your program: 0.75 seconds. Allowed memory: 128 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 2 2 2 2 2 2 2 2 2 2 | 1 |
| 1 1 2 2 2 3 3 3 3 3 3 3 | 340 |