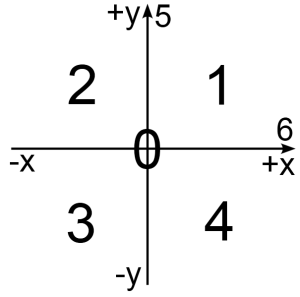
**Problem 1 – Cartesian Coordinate System**

You are given a two-dimensional Cartesian coordinate system and the two coordinates (**X** and **Y**) of a point in the coordinate system. If you don't know what Cartesian coordinate system is Google it with Bing. As you will find, the coordinate system is divided by 2 lines (see the picture bellow) which divide the plain in four parts. Each of these parts has a lot of points that are numbered between 1 and 4. There is one point where our lines are crossing. This point has the following coordinates: X=0 and Y=0. As a result this point is numbered 0. The points on the lines are also numbered with the numbers 5 and 6 (again see the picture below).

Your task is to write a program that finds the number of the location of the given point in the coordinate system.

**Input**

* Input data is read from the console.
* The number **X** stays at the first input line.
* The number **Y** stays at the second input line.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

* The output data must be printed on the console.
* On the only output line you must print an integer number between 0 and 6, depending on the location of the given point in the coordinate system.

**Constraints**

* The numbers **X** and **Y** are numbers between -2 000 000 000 001 337 and 2 000 000 000 001 337, inclusive.
* Time limit: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1  2 | 1 | -0033  -4 | 3 | -3000  9000 | 2 | 12345  -98786543 | 4 |

**Problem 2 – Tribonacci**

The Tribonacci sequence is a sequence in which every next element is made by the sum of the previous three elements from the sequence.



Write a computer program that finds the **N**th element of the Tribonacci sequence, if you are given the first three elements of the sequence and the number **N**. Mathematically said: with given T1, T2 and T3 – you must find Tn.

**Input**

* The input data should be read from the console.
* The values of the first three Tribonacci elements will be given on the first three input lines.
* The number **N** will be on the fourth line. This is the number of the consecutive element of the sequence that must be found by your program.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

* The output data should be printed on the console.
* At the only output line you must print the **N**th element of the given Tribonacci sequence.

**Constraints**

* The values of the first three elements of the sequence will be integers between -2 000 000 000 and 2 000 000 000.
* The number **N** will be a positive integer between 1 and 15 000, inclusive.
* Time limit: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1  1  1  4 | 3 | 2  3  4  10 | 335 | 5  -1  2  33 | 208691269 |

**Problem 3 – Sand Glass**

Once upon a time a powerful wizard was born. His name was Gwenogfryn and soon he became a great sorcerer. Kind-hearted he was. He would only use his magic to protect humans from the evil witches that would come at night. Gwenogfryn, however was a pacifist and did not want to fight or hurt the witches, so he came up with another solution. He would catch the witches and throw them into a sand-glass (the only prison a witch cannot escape from). Unfortunately, he is running out of sand-glasses. Help Gwenogfryn catch all witches by making your own sand-glasses.

**Input**

* The input data should be read from the console.
* You have an integer number **N** (always **odd number**) showing the height of the sand clock.
* 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.
* You should print the hourglass on the console. Each row can contain only the following characters: “.” (dot) and “\*” (asterisk). As shown in the example: the middle row must contain only one ‘\*’ and all other symbols must be “.”. Every next row (up or down from the middle one) must contain the same number of ‘\*’ as the previous one plus two. You should only use “.” to fill-in the rows, where necessary.

**Constraints**

* The number **N** will be a positive integer number between 3 and 101, inclusive.
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3 | \*\*\*  .\*.  \*\*\* | 5 | \*\*\*\*\*  .\*\*\*.  ..\*..  .\*\*\*.  \*\*\*\*\* | 7 | \*\*\*\*\*\*\*  .\*\*\*\*\*.  ..\*\*\*..  ...\*...  ..\*\*\*..  .\*\*\*\*\*.  \*\*\*\*\*\*\* |

**Problem 4 – Bulls and Cows**

All we love the “Bulls and Cows” game (<http://en.wikipedia.org/wiki/Bulls_and_cows>). Given a 4-digit **secret number** and a 4-digit **guess number** we say that we have **b** **bulls** and **c** **cows** when we have **b** matching digits on their right positions (bulls) and **c** matching digits on different positions (cows). Examples are given below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Secret number | 1 | 4 | 8 | 1 | Bulls = 1  Cows = 2 |  | Secret number | 2 | 2 | 4 | 9 | Bulls = 0  Cows = 3 |
| Guess number | 8 | 8 | 1 | 1 | Guess number | 9 | 9 | 2 | 4 |

Given the secret number and the number of bulls and cows your task is to write a program to **find all matching guess numbers** in increasing order.

**Input**

* The input data should be read from the console.
* It will consist of exactly 3 lines. At the first line there **the secret number** will be given. At the second line the number of bulls **b** will be given. At the third line the number of cows **c** will be given.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

* The output data should be printed on the console
* It should consist of a single line holding all matched guess numbers, given in increasing order, separated by single space.

**Constraints**

* The **secret number** will always consist of exactly 4 digits, each in the range [1…9].
* The numbers **b** and **c** will be in the range [0…9].
* Time limit: 0.15 seconds.
* Allowed memory: 4 MB.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 2228  2  1 | 1222 2122 2212 2232 2242 2252 2262 2272 2281 2283 2284 2285 2286 2287 2289 2292 2322 2422 2522 2622 2722 2821 2823 2824 2825 2826 2827 2829 2922 3222 4222 5222 6222 7222 8221 8223 8224 8225 8226 8227 8229 9222 | 1234  3  0 | 1134 1214 1224 1231 1232 1233 1235 1236 1237 1238 1239 1244 1254 1264 1274 1284 1294 1334 1434 1534 1634 1734 1834 1934 2234 3234 4234 5234 6234 7234 8234 9234 | 1234  3  1 | No |

**Problem 5 – We All Love Bits!**

One of the things the programmers love the most is bitwise operations. The "bitwise guy" is a synonym for a programmer that loves bits more than everything else in programming. Mitko is a "bitwise guy". He invented a new bitwise algorithm. The algorithm takes one positive integer number **P**, makes magic with it and returns a new positive integer number. He also defined a new number **P̃** which represents the number **P** in binary numeral system with inverted bits. All zeros in **P** are ones in **P̃** and all ones in **P** are zeros in **P̃**. For example if we have P = 9 (which is 1001in binary numeral system) its inverted number P̃ will be equal to 6 (which is 110 in binary numeral system). But that’s not all! He invented another number **P̈**, which represents reversed number **P** in binary numeral system. For example if we have P = 11 (which is 1011 in binary numeral system) its reversed number P̈ is equal to 13 (which is 1101 in binary numeral system). The Mitko's magical algorithm takes a number **P** and transforms it to a new number **Pnew** using the following bitwise transformation: **Pnew = (P ^ P̃) & P̈**.

Your task is to write a program that transforms a sequence of **N** positive integer numbers using Mitko's algorithm.

**Input**

* The input data should be read from the console.
* At the first input line there will be one positive integer – the number **N**.
* At each of the next **N** lines there will be one positive integer – the consequent number that must be converted using Mitko's algorithm.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

* The output data should be printed on the console.
* The output must consist of **N** lines, containing the transformed numbers for each number from the input.

**Constraints**

* The number **N** will be positive integer number between 1 and 20 000, inclusive.
* Each of the **N** numbers will be positive integer numbers between 1 and 2 147 483 647, inclusive.
* Time limit: 0.20 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1  2 | 1 | 2  19  248 | 25  31 | 4  6732654  255  36372344  60000 | 3894963  255  8125777  1623 |