Puzzle

Example task **Available memory: 256 MB. Maximum running time: 1 s.**

After some time John figured out a solution to a long forgotten puzzle. He wants to check if his idea is correct, so he asked you to provide a program for verification. The riddle goes something like this:
"There is given an area of $3 \times n$ size. Can you cover all of it with bricks of 1×2 size?" John thinks he knows when it's possible, but have some special requirements about your program.

The left lower corner of the area will be our $(0, 0)$ point. If the task is possible your program should give the integer k denoting the number of bricks needed for coverage. After that it should print exactly k lines, each containing 3 integers x, y, z. The first two numbers denotes coordinate (x, y) where the brick should be put. Last one defines, should it be put vertically or horizontally. If $z = 0$ the brick will be put horizontally (so it will cover fields (x, y) and $(x + 1, y)$). If $z = 1$ the brick will be put vertically (it will cover fields (x, y) and $(x, y + 1)$). Don't worry! If you know how many bricks John should put on the area, but you can't show him how to place them you can still get a reward! As you can see, there may exist more then one valid coverage. You can print **any valid one**. If the task is not possible, your program should print one line saying "Can't do that".

Input

In the first and only line of input there is a single integer n ($1 \leq n \leq 100\,000$) denoting the length of the rectangle.

Output

If coverage is possible, the first line of standard output should contain exactly one integer k. Each of next k lines should contain exactly 3 integers x, y, z. If $z = 0$ the brick will cover fields (x, y) and $(x + 1, y)$. If $z = 1$ the brick will cover fields (x, y) and $(x, y + 1)$. Remember that bricks **cannot** intersect!
If coverage is not possible, print only one line saying "Can't do that".

Example

For the input data:

```
1
```

the correct result is:

```
Can't do that
```

For the input data:

```
2
```

the correct result is:

```
3
0 0 0
0 1 1
1 1 1
```