Your good friend Ivancho has become a university student and, like all university students, keeps all of his notes in (physical) folders. Each folder contains a certain amount of papers and he has N piles, on which he piles up his folders. When he needs to take a folder from a given pile, he always takes the top folder from it. At the end of the day Ivancho wants to know how many papers there are in the top folder of each of the piles.

**Input**

The first row of the file piles.in contains an integer **N** – the number of piles on Ivanchos desk, and an integer **M** – the number of operations which he will perform (putting a folder on top of a pile or taking a folder from a pile).

**M** lines follow – the operation, which Ivancho performs, given in the following way:

<operation> <pile\_number> <number\_of\_papers\_in\_folder>

Where:

* <operation> = **“ADD”** or **“REMOVE”**
* **0** ≤ <pile\_number> ≤ **N-1**
* **1** ≤ <number\_of\_papers\_in\_folder> ≤ **216-1** which is given only when the operation is **“ADD”**

When performing operation **“ADD”** at the top of the <pile\_number>’th pile is placed a folder containing <number\_of\_papers\_in\_folder> papers.

When performing operation **“REMOVE”** the folder at the top of the <pile\_number>’th pile is removed.

 It is guaranteed that operation **“REMOVE”** is always performed on a nonempty pile.

**Output**

In the output file piles.out print N space separated integers – the number of papers in the top folder of pile number 0, 1, 2, …, N-1. If a pile has no folders, print 0.

**Constraints**

3 ≤ *N* ≤ 103

5 ≤ *M* ≤ 104

**Time limit: 0.5 sec**

**Memory limit: 256 MB**

**Example test**

|  |  |
| --- | --- |
| **Input (piles.in)** | **Output (piles.out)** |
| 3 8ADD 0 2ADD 0 8ADD 2 15REMOVE 0REMOVE 0REMOVE 2ADD 0 9ADD 2 6 | 9 0 6 |
| 4 6ADD 0 4ADD 0 15ADD 0 2REMOVE 0ADD 3 14ADD 2 2 | 15 0 2 14 |