Problem I. Interactive Memory Management

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

This is interactive problem.

In this problem you must implement the primitive memory manager that can allocate and free blocks of memory. Memory is represented as an array $M[1..n]$ of n cells. *Block* of size s consists of s consecutive cells of memory. Different blocks are not allowed to share the same cell.

Your memory manager must be able to allocate blocks of size 1, 2 and 3, free previously allocated blocks, and it can move previously allocated blocks.

There are two types of requests: allocate memory block and free memory block. All requests are numbered consequently starting from 1.

Before allocating a new block and after freeing a block you are allowed to make up to two 2 moves of currently allocated blocks.

Allocate request is specified as one integer from the set $\{1, 2, 3\}$ — size of the block to allocate.

Free request is specified as one negative integer. An integer $-k$ means that the block allocated at request k must be freed.

There are three types of operations that your memory manager can do.

- *allocate*: "`A` x" allocate requested block starting from cell number x.
- *move*: "`M` x y" move block starting from position x to start from position y.
- *done*: "`D`" report that moving blocks after performing free request is completed.

You have to implement memory manager that would perform all operations, given that at any moment the total size of allocated blocks is at most $n - 1$.

Interaction Protocol

First your program must read n — size of memory array ($4 \le n \le 100\,000$). After that one or more requests follow.

Each request is specified as an integer. Positive integer from 1 to 3 specifies *allocate* request, negative integer specifies *free* request. Zero specifies end of input, it must not be processed, after reading zero your program must exit.

After reading *allocate* request your program can perform up to 2 *move* operations, followed by *allocate* operation.

After reading *free* request your program must immediately free cells occupied by the corresponding block (it must not be reported), then it can perform up to 2 *move* operations, and then perform *done* operation.

All performed operations must be correct.

For each *allocate* when allocating block of size s its argument x must be index of the cell such that cells $x, \ldots, x + s - 1$ are free.

For each *move* its first argument x must be the first cell of some currently allocated block. If such block has size s, the second argument y must be index of the cell such that cells $y, \ldots, y + s - 1$ are free. In particular, source and destination locations of the block being moved may not share a common cell.

All requests are correct: no block is freed more than once, at any moment the total size of all allocated blocks is at most $n - 1$, each *free* request refers to the number of some *allocate* request.

It is guaranteed that there is strategy that correctly fulfils all requests.

There are at most 30 000 requests.

Examples

standard input	standard output
7 3 2 -1 1 3 0	 A 1 A 4 D A 1 M 1 6 A 1