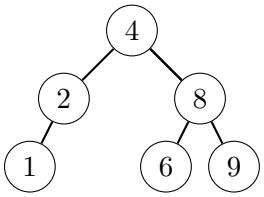
Tao of Trees (search trees)

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Do you know what a binary search tree is?



We are not asking for anything complicated here. Just vanilla binary search trees. Every `Node` contains a key `k` (an integer) and two `Node` pointers `left` and `right` to its two children. (To make things easier, we will assume that all keys are unique). The binary search tree is a `Node*` (called the root; 4 in the above example) and all its descendants. No cycles are allowed. For every node in the tree, we require that all its left descendants have smaller keys and all its right descendants have larger keys.

If you are not comfortable with pointers, maybe now is the time to learn how to use them? Or if you really don't want to, then manage yourself use the memory using a `vector<Node>` and indexes instead of pointers. (This technique can sometimes be more efficient than using `new` and `delete`.)

We ask you to implement the following operations:

- `PRINT`: Print the tree recursively: empty string if empty, `k` if it is a childless node of key k, and `k(L,R)` otherwise, where `L` and `R` are its left and right children.
- `ADD` k: Create a new node with key k and add it into a leaf position. If the node with key k already exists, do nothing.
- `DEL` k: Find and remove the node N with key k. If the node with key k does not exist, do nothing. To remove a node from a tree, proceed exactly as follows. If N has no children, simply delete the node. If N has a single child (left or right), then the child will take N's place in N's parent. If N has two children L and R, things get harder: find the rightmost descendant Y of the left child L of N (this is also called the "inorder predecessor of N"). Then remove that node Y from the tree L (since Y is the rightmost descendant of N we know that Y has no right child, and we already know how to delete such nodes). Finally, connect L and R to this node Y, and finally let Y take N's place.

If you have never done this, it might all sound a bit complicated, especially the deletion code. But it is just a few lines of recursive functions. Give it a try!

Input

Starting with an empty tree, execute a sequence of orders, as described. Keys k will be integers from 0 to 10^5.

Remember that our trees won't have repeated keys. Orders that add a key k that already exists or delete a key that does not exist should simply be ignored.

Output

For each command `PRINT`, output a line with the contents of the tree at that time.

Examples

standard input	standard output
ADD 4 PRINT ADD 2 ADD 1 PRINT ADD 8 ADD 6 ADD 6 ADD 9 PRINT DEL 1 PRINT DEL 2 DEL 8 PRINT DEL 3 PRINT	4 4(2(1,),) 4(2(1,),8(6,9)) 4(2,8(6,9)) 4(,6(,9)) 4(,6(,9))
ADD 1 PRINT DEL 1 PRINT ADD 10 ADD 11 ADD 12 ADD 13 ADD 14 ADD 5 ADD 6 ADD 7 PRINT DEL 10 PRINT	1 10(5(,6(,7)),11(,12(,13(,14)))) 7(5(,6),11(,12(,13(,14))))