Cards

The four donkey men is a famous magician group known to follow the art of magic taught by "The Ear", a mysterious magical society led by the (supposedly) late Lionel Shriek. He died while performing a trick that went wrong (or did he really die?).

Today, the four donkey men is going to have a magic show involving cards. They asked the audience to write their name and age on a card and hand it to them so that they can perform their trick. This trick involves a lot of swapping and shuffling the aforementioned deck of cards. They are able to:

1. Swap position of many cards at one go.
2. Get the details of a card at a specific position.
3. Know the position of a specific card.
4. "Shuffle" the deck of cards.
5. Know the cards in order from top to bottom.

As we all know, all magicians use tricks. There is, honestly, no such thing as real magic. They need a program that can perform all of the above queries fast enough. As the best programmer in the world, you are asked to make it for them. Good luck!

Input

The first line of the input consists of a single integer, **N (5 <= N <= 1000)** the number of cards. N lines follow. Each line contains a person's name and his/her age. All names are unique and consist of lowercase English letters only. Each age is a positive integer at most 99. The next line has a single integer, **Q (1 <= Q <= 500)**, the number of queries (or performance) during the show. It is then followed by **Q** lines, each containing a single query. The queries follow the following specification:

Query Type Input Format: `<QUERY_TYPE> <APPROPRIATE_PARAMETERS>`

1. `swap a b c d`
   Swap cards from (1-based) index a-b and c-d. You are required **to swap all cards** inside the range [a,b] and [c,d], both inclusive. It is guaranteed that 1 <= a <= b < c <= d <= N. Print "swap has been performed" after the swap occurred.

2. `details index`
   Print the details of the card at the specified index. Print the name and age of the person, separated by a single space.

3. `position name`
   Print the position (1-based) of the card containing the person with the name `name`.

4. `shuffle`
   Perform a "riffle shuffle". A riffle shuffle splits the deck into two (if there is an odd number of cards, the first pile contains 1 more card than the other). Then, cards from the first pile and the second pile will form one pile alternatively. If you have cards 1-2-3-4-5 originally, you will have 1-4-2-5-3 after this operation is performed. Print the string "shuffle has been performed" after the shuffle occurred.

5. `print`
   Print all the names from index 1 to N. There is no whitespace after the last name.

Output

Print the result of all queries, as described above. The last line of the output should contain a newline character.

Sample Input

```
5
one 1
two 2
three 3
four 4
five 5
6
swap 1 2 4 5
details 3
position four
print
shuffle
print
```

Sample Output

```
swap has been performed
three 3
1
four five three one two
shuffle has been performed
four one five two three
```

Explanation

We will use 1-2-3-4-5 as the card description for this explanation. The first operation swaps cards "one" and "two" with "four" and "five". We use 1-based indexing for everything in this problem. After the first swap, the deck is now 4-5-3-1-2 (as printed above). The shuffle query will split the deck into two piles. The first pile is 4-5-3 and the second pile is 1-2. It will then merge the piles into one starting from the first pile. Hence, the deck is now 4-1-5-2-3.

If we have 1-2-3-4-5-6-7-8-9 and the query is "swap 2 4 6 8" then the new ordering of cards would be like this: 1-6-7-8-5-2-3-4-9.

Skeleton

You are given the skeleton file **Cards.java**. Please make sure that you do not see an empty file when opening the file, otherwise you might be in the incorrect directory. The skeleton file contains a working tailed linked list implementation similar to the one in the "Eels and Escalators" take-home lab problem.

Notes

1. You **must use linked list** to solve this problem.
2. You are free to define your own linked list class (encouraged for practice and skeleton file given), but you are allowed to use Java's built-in linked list implementation if it is suitable for this problem.
3. You are free to (and should) modify the skeleton file and add more attributes or methods when necessary.