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.
   It is guaranteed that 1 <= a <= b < c <= d <= N.
   Print "swap has been performed"

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 a (1-based) index, the position of the card containing the person with the specified 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.

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.

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 should develop your program in the subdirectory **ex1** and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. You are free to modify the skeleton file and add classes as you wish. Note that your **algorithm must only use linked list** to solve this problem. You are **not allowed** to use Arrays, ArrayList, HashMap, etc. for this lab. **Failure to comply will result in a score of 0.**
3. You are given a **fully-functional singly-linked list** in the skeleton file which you are free to modify. Alternatively, you are allowed to use Java's API. Please take note to **remove the provided linked list (and list node) class** in case you want to use the API instead.
4. Please be reminded that the marking scheme is:

   Input : 10%
   Output : 10%
   Correctness : 50%
   Programming Style : 30% (awarded if you score **at least 20% from the above**):
   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (modular programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

   Compilation Error : Deduction of **50% of the total marks obtained**.