Ferry

This is a problem called ferry. It is very simple.

Evan is a ferry captain. He is going to retire soon. In one of his last duties as a ferry captain, he will operate a ferry which helps cars cross from one side of a river to the other. There are two sides of the river: the left side and the right side. Cars have lined up in both sides of the river, waiting for the ferry to carry them across.

To save time and money, Evan wants to <u>minimise the number that the ferry has to cross the river</u>. He wants to carry as many cars as possible without overloading the ferry in each trip. The ferry can only carry up to **W** units of weight per trip. In each trip, the ferry will load as many cars as possible, carry those cars to the other side of the river, and unload them. Then the ferry will bring the cars from that side of the river, and so on, until there are no more cars left in each side. This strategy is <u>optimal</u>.

Initially, the ferry is at the left bank of the river. To begin his day, Evan wants to know how many times does the ferry need to cross the river in order to carry all the cars across. Your job is to help Evan find this magic number. Good luck and do your best!

Input

The first line contains two space-separated integers **W** (100 <= **W** <= 10^4) and **N** (**0** <= **N** <= 10^5), the capacity of the ferry (in units of weight) and the number of cars at <u>both</u> banks of the river combined.

N lines follow. Each of the **N** lines contain the weight and location of the cars ("left" or "right"). All the cars will be given upfront and all have arrived before the ferry starts its first trip. All cars have no weight larger than **W**, meaning that all cars can eventually be transported.

The cars are <u>listed in order of arrival</u> and they will queue up to enter the ferry. Those who arrive first will be **<u>given priority to enter</u>**. This means that if car A comes before car B in the input, and both are on the same bank initially, car B will only be allowed into the ferry if and only if car A had already entered the ferry (either on the same trip or not).

Output

Print the <u>minimum</u> number of times the ferry needs to cross the river. Your <u>output must contain a newline character.</u>

Sample Input 1	Sample Output 1

`2000 4`
`380 left`
`720 left`
`1340 right`
`1040 left`
	`3`

Sample Input 2	Sample Output 2

`2000 4`
`380 right`
`720 right`
`1340 right`
`1040 right`
	`6`

Explanation

Sample Input 1: **3 times**

1. In the first trip, the ferry carries car 1 and 2 (with weight 380 and 720) to the right bank. It cannot carry the last car on the left bank as it will overload the ferry.
2. In the second trip, the ferry carries the only car located at the right bank (with weight 1340).
3. In the last trip, the ferry carries the last remaining car on the left bank to the right bank.

Sample Input 2: **6 times**

1. The first trip requires the ferry to go from left to right without carrying any cars, since there are no cars on the left side of the river.
2. In the second trip, the ferry will carry the first two cars. Note that the ferry can actually accommodate the last car without overloading, but it could not do so because it needs to prioritise the third car over the fourth one. The next four trips are repetitions of the first two.

Skeleton

You are given the skeleton file `Ferry.java`. You should see a non-empty file when opening it, otherwise you might be in the wrong directory. We also provide you with a simple implementation of Stack and Queue inside the skeleton file. **DO NOT modify these two classes.** You **must** use them as it is.

```java
/**
 * Name         :
 * Matric No.   :
 * PLab Acct.   :
 */

import java.util.*;

public class Ferry {

    private void run() {
        // implement your "main" method here...
    }

    public static void main(String[] args) {
        Ferry trip = new Ferry();
        trip.run();
    }
}
```

Stack<E>
+ push(E) : void
+ pop() : E
+ peek() : E
+ size() : int
+ isEmpty() : boolean

Queue<E>
+ enqueue(E) : void
+ dequeue() : E
+ peek() : E
+ size() : int
+ isEmpty() : boolean

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. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm. You must use the given Stack and/or Queue in your algorithm to solve this problem. You are not allowed to use arrays, ArrayList, LinkedList, HashMap, etc. for this problem for any purpose. Failure to comply will result in 0 marks being given for the sit-in lab.
3. You are not allowed to violate the stack and/or queue properties, such as by traversing the stack and/or queue. **Use the given Stack and/or Queue and you should be fine and not fined.**
4. You are free to define your own classes if you want to.
5. 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.