Birthday party

Input Files: `party*.in` **100 points**
Output Files: `party*.out`

John's birthday is approaching slowly, and as every year, John is going to organize a great garden party. He wants all his friends to come, but (sadly) he knows, that it is almost impossible. For example Susie left Steve last week, and it will be almost impossible to make both of them come. John spent most of the last week visiting his friends and asking them to come. He got some promises, but even more requests. ('If you invite me, you just have to invite my boyfriend!' exclaimed Veronica. 'If you invite the Burdiliak twins, don't expect me or Joseph to come!' stated Peter.) Suddenly, John realized, that it will be quite hard just to satisfy all the requests he got.

Task

You are given a description of the requests John got from his friends. Your task is to find a group of people such that if John invites the people in this group (and nobody else) to his party, all the requests he got will be satisfied. The requests are described in the following way:

- `name` is a request. This request is satisfied if and only if John invites `name`.
- `-name` is a request. This request is satisfied if and only if John doesn't invite `name`. (In both cases, `name` is a string of at most 20 lowercase letters without spaces.)
- If `R1`, ..., `Rk` are requests, then `(R1 & ... & Rk)` is a request. This request is satisfied if and only if all requests `R1`, ..., `Rk` are satisfied.
- If `R1`, ..., `Rk` are requests, then `(R1 | ... | Rk)` is a request. This request is satisfied if and only if at least one of the requests `R1`, ..., `Rk` is satisfied.
- If `R1`, `R2` are requests, then `(R1 => R2)` is a request. This request is **not** satisfied if and only if `R1` is satisfied and `R2` is not satisfied.

Input

You can find ten input files, called `party1.in` to `party10.in` on the web page. Each of the inputs is worth 10 points.

On the first line of the input file is the number of John's friends F, next F lines contain their names, one per line. On the next line is the number of requests N. Each of the following N lines contains one request.

Output

For each file `partyX.in` you have to produce the corresponding output file `partyX.out`, containing one correct solution. The first line of the output file will be the number K of people John should invite. The following K lines should contain their names, one per line. You may assume that each of the input files has a (not necessarily unique) solution. If there are more possible solutions, you may output any of them.

Submits

Submit the files `party*.out` using the web interface in the same way you submit your programs for the other tasks.

Example

Input	Output
`3`	`2`
`veronica`	`steve`
`steve`	`dick`
`dick`	
`3`	
`(veronica => dick)`	
`(steve => -veronica)`	
`(steve & dick)`	