The task is to develop a program to help architects in Sim city game.

Assume lots arranged in a single row along a road. The lots are of a square shape, their sizes are identical. Each lot may be assigned one of: R - residential, C - commercial, I - industrial, or P - park. There is a total of `n` lots to assign. The task is to compute how many different options there exists. To make the task a bit more complicated, there are some rules that apply. There is a modulus `m` that limits:

- there may be at most `m` successive lots without a park,
- two commercial lots must be separated by at least `m` lots of different kind,
- the industrial lots cannot be allocated separately, i.e. at least two industrial lots must be adjacent.

The input of the program is the modulus `m` followed by a list o computation requests. Each request is passed on a single line. There are two types of computation requests:

- `list n prefix` - list all possible combinations that may be assigned to the `n` lots given the restrictions above. The value `prefix` is optional. If used, it lists the lots that precede the `n` lots to assign. If prefix is specified, it restricts the generated assignments as the whole sequence of lots (the existing assignment from the prefix and the newly assigned `n` lots) must obey the above restrictions. The prefix parameter is not used in the mandatory tests, it is only used in bonus tests,
- `count n prefix` - this command is very similar to the list command above except it does not list the possible assignments. Instead, it just prints how many distinct assignments exists. The meaning of `n` and `prefix` is the same as above. And again, `prefix` is only used in the bonus tests.

The sequence of requests ends when EOF is detected.

The output of the program is either the number of existing assignments (`count` type request), or the list of possible assignments (`list` type request). Each possible combination is to be printed on a separate line, enclosed in square braces. The `prefix` goes first (if the prefix is used), followed by the `n` lots with their assignments. The assignment is denoted by upper case letters (P, I, R, C). The output list is followed by the number of combinations found. The order of the lines in the list is not specified, the testing environment fixes the order before it starts the comparison.

The program must detect invalid input. If the input is invalid, the program must display an error message and terminate. The following is considered an invalid input:

- the modulus is invalid (not an integer, less than 1, or greater than 10),
- the request is invalid (only `list` and `count` is valid),
- the length `n` is invalid (not an integer, less than 1),
- the `prefix` is invalid (only uppercase letters I, P, R, and C are allowed in the prefix). This only applies to the bonus tests.

The program is tested in a limited environment. Both time and memory is limited. The limits are set such that a correct implementation of a naive algorithm passes all tests. There is quite a lot of extra memory available, the extra memory may be used to implement some advanced algorithm to speed up the computation (required for the bonus tests). The time complexity of the basic solution is exponential in `n`. The bonus tests require either an optimized exponential solution (bonus #1) or a solution based on the dynamic programming paradigm (the second bonus). Moreover, the prefix parameter must be handled correctly in the bonus tests.

Sample program run (short version, full version in the attached archive):

```
Modulus:
3
Searches:
list 1
[P]
[R]
[C]
=> 3
list 2
[PP]
[PR]
[PC]
[II]
[RP]
[RR]
[RC]
[CP]
[CR]
=> 9
list 4
[PPPP]
[PPPR]
[PPPC]
[PPII]
[PPRP]

...

[CPRR]
[CIIP]
[CRPP]
[CRPR]
[CRRP]
=> 60
count 5
=> 159
count 12
=> 135946
```

```
Modulus:
8
```

```
Searches:
list 4
[PPPP]
[PPPR]
[PPPC]
[PPII]
[PPRP]

...

[CRPP]
[CRPR]
[CRII]
[CRRP]
[CRRR]
=> 79
count 9
=> 12907
list abc
Invalid input.
```

```
Modulus:
38
Invalid input.
```

Advice:

- The sample runs above list both the output of your program (bold face text) and user input (regular text). The bold/regular formatting is included here, in the problem statement page, to increase readability of the listing. Your program must output the text without any additional markup.
- Bonus tests do test higher values of `n` and do test mainly `count` type requests (the printing of the list is very time consuming).
- The number of combinations found may be very high in the bonus tests. Standard `int` type is fine for all tests except bonus tests. Type `long long int` is required to pass the bonus tests.
- The bonus tests to use the `prefix` parameter. The prefix limits the possible following lot types, thus decreases the number of possible combinations. Some samples are provided in the attached archive, the examples are stored under the directory named bonus. Note that the combination of lots in the prefix may violate the lot usage rules. The result is empty (0 combinations) in such a case.