Mail Data Format

Each mail is stored in a file that contains information about sender, date, message-ID, subject, receiver and the content. Note that the Message-ID is an unique positive integer for each mail and the subject may be empty. Also, there is no space in the names of sender and receiver. The file is of the following format:

From: Dora
Date: 13 November 2011 at 13:35
Message-ID: 3
Subject: Bakurans
To: Austin
Content:
In Star Wars, the Bakurans were humans inhabiting the planet [Bakura]. Bakurans developed a distinct culture due to their isolated location and their lack of contact with any nonhuman beings besides the Kurtzen natives. The Bakurans descended from the Bakuran Mining Corporation colonists who arrived at the planet around 150 BBY. They soon broke away from the corporate control and created their own senate and elected a Prime Minister. The leader of the 150 BBY Bakuran Mining Corporation expedition, Deredith Arden's descendants were the leaders of the Bakurian government. In 4 ABY the Bakurans joined force with the Alliance to Restore the Republic to ward off the attacking Ssi-Ruuk, a saurian species from the Unknown Regions of the galaxy, who planned to conquer the planet.

Note that for the purpose of tokenizing in the field of "Content" or "Subject", please replace any character other than alphabets and digits to a space. For example, "[Bakura]" is same as "Bakura" and "NTU-CSIE" is same as "NTU CSIE".

Input Format

You program will read a from standard input and process a series of operations one-by-one. Your program will have to support four kinds of operations, *add*, *remove*, *longest*, and *query*, with details given as follows:

- `add <file>`: add a mail into database, where `<file>` is the absolute path of the file that stores a mail
- `remove <id>`: remove mail from database, where `<id>` is the unique positive Message-ID of a mail
- `longest`: return the Message-ID of mail whose content contains the largest number of alpha-numeric characters (excluding spaces)
- `query [-f"<from>"] [-t"<to>"] [-d<date>~<date>] <expression>`:
  return all the Message-ID of mails that fits the query description, where

```
<from>       := <string>
<to>         := <string>
<date>       := {YYYYMMDDhhmm} or {blank}
<expression> := <keyword> or (<expression>) or !<expression>
                  or <expression>|<expression> or <expression>&<expression>
<keyword>    := <string>
<string>     := {any alpha-numeric string without whitespace or punctuation}
```

  Specifically, for each query return those mails that has the same sender and recipient (if specified), with date falling in range inclusive of start and end date (again if specified), and whose contained keywords fits the expression. `<key1>` means the returned mail should contain `key1` as a complete token (i.e. "dsa" in "po**dsa**fe" is not a complete token), while `!<key2>` means the returned mail should not contain `key2`, likewise `<key1>|<key2>` means the returned mail should contain either `key1` or `key2` ... etc. As for precedence, parentheses are of the highest precedence, followed by `!`, and then `&`, and then `|`.

For all keywords and sender/recipient, see the query terms as **case-insensitive**. `<from>` and `<to>` will have a maximum length of **50**, and `<keyword>` will have a maximum length of **20**. `<expression>` will consist of **at least 1** and **at most 20** keywords. You can assume there will not be intentionally redundant not (`!`) or parenthesis in the query expression, so the maximum length of an expression is also bounded.

Note the formats above are strictly followed in test data. So you may safely assume for example, you will not stumble upon random white space in the middle of a query expression.

Some examples of valid queries are listed below:

```
query -t"dsa14spring" -d201404252230~201405141300 final|project
query -f"EddardStark" -d~201003181220 winter&vacation&is&coming
query -f"Shopgirl" -t"NY152" -d199812180000~ lessthan21characters
query JAVA&COFFEE&!(PROGRAM|PROGRAMMING)
query (nested&(parenthesis|(is&!good)))
```

Output Format

For each action, please output exactly one line to the stdout.

- `add <file>`: If the mail to be added already exists in the database, simply output "-"; otherwise output a single integer N, indicating the number of mails in the database after the mail is added. You can assume the path to the targeted file is always valid.
- `remove <id>`: If the mail to be removed is not present in the database, simply output "-"; otherwise output a single integer N, indicating the number of mails in the database after the mail is removed.
- `longest`: If there is no mail in the database, simply output "-"; otherwise output the Message-ID with longest content, and the length (defined as the number of alpha-numeric characters), separated by a single space. If multiple emails are of the same longest length, output the one with the *smallest* Message-ID.
- `query [-f"<from>"] [-t"<to>"] [-d<date>~<date>] <expression>`:
  Output **in ascending order** a list of Message-ID's corresponding to all mails matching the query description. Separate the Message-ID's by a single space. If there is no match, simply output "-".

Sample Input

```
longest
add ./mail1
add ./mail1
query termincommon
query terminmail3only
add ./mail2_len1126
query termincommon
query terminmail2only
query -d~123401010000 datetooearly
remove 1
remove 3
add ./mail3_len1126
longest
```

Sample Output

```
-
1
-
1
-
2
1 2
2
-
1
-
2
2 1126
```

Submission Rule

Please upload your program as a single ZIP compressed file to our competition website:

(to be opened later)

The ZIP file should contain the following items:

- You program, including any package that you use.
- The `Makefile`.

The website will use the command "make" and "make run" to test your program. In order to let the system judge your program correctly, please write your own Makefile that direct the input from a file, namely `input`, and direct the output to a file, namely `output`.

Warning : Any kind of cheating and server attack attempts can lead to heavy penalty.

Competition Evaluation

We will feed a very long sequence of actions to your program and give you a fixed amount of time. The performance in the competition is measured by how many successive correct answers you can get from the first action in this fixed amount of time.

In order to prevent you from "querying" or "overfitting" the order of the action sequence, we'll secretly run your program by another similar but different sequence of actions. You final score will not only depend on the score board, but also on performance on the hidden data.