Input

The provided input will be a CSV file in which each line contains exactly four columns: direction, protocol, ports, and IP address:

direction	Either "inbound" or "outbound", corresponding to whether the traffic is entering or leaving the machine.
protocol	Either "tcp" or "udp", all lowercase – we will just implement two of the most common protocols.
port	Either (a) an integer in the range [1, 65535] or (b) a port range, containing two integers in the range [1, 65535] separated by a dash (no spaces). Port ranges are inclusive, i.e. the port range "80-85" contains ports 80 and 85. Given a port range, you may assume that the range is well-formed i.e. the start of the range is strictly less than the end.
IP address	Either (a) an IPv4 address in dotted notation, consisting of 4 octets, each an integer in the range [0, 255], separated by periods or (b) an IP range containing two IPv4 addresses, separated by a dash (no spaces). Like port ranges, IP ranges are inclusive. Given an IP range, you may assume that the range is well-formed i.e. when viewed as a number, the starting address is strictly less than the ending address.

For example, the following are all valid inputs:

```
inbound,tcp,80,192.168.1.2
outbound,tcp,10000-20000,192.168.10.11
inbound,udp,53,192.168.1.1-192.168.2.5
outbound,udp,1000-2000,52.12.48.92
```

You may assume that the input file contains only valid, well-formed entries.

What to implement

Given a set of firewall rules, a network packet will be accepted by the firewall if and only if the direction, protocol, port, and IP address match at least one of the input rules. If a rule contains a port range, it will match all packets whose port falls within the range. If a rule contains an IP address range, it will match all packets whose IP address falls within the range.

Your job is to implement a Firewall class, whose interface contains two items:

- A constructor, taking a single string argument, which is a file path to a CSV file whose contents are as described above, e.g, in Ruby, this would be `Firewall.new(“/path/to/fw.csv”)`.
    - Note that you do not need to define a static ‘new’ method – simply use the constructor syntax in the language that you chose.
    - Remember that you may assume that all content in the input file is valid.

- A function, `accept_packet`, that takes exactly four arguments and returns a boolean: true, if there exists a rule in the file that this object was initialized with that allows traffic with these particular properties, and false otherwise.
    - direction (string): “inbound” or “outbound”
    - protocol (string): exactly one of “tcp” or “udp”, all lowercase
    - port (integer) – an integer in the range [1, 65535]
    - ip_address (string): a single well-formed IPv4 address.

You may assume that all arguments passed into the `accept_packet` function will be well-formed according to the spec above.

Feel free to define any additional functions or classes you might want; the only thing we ask is that you don’t change the interface described above.

In your implementation, you should consider efficiency and tradeoffs in both space and time complexity. There may be a massive number of rules (use 500K entries as a baseline), and real-world firewalls must be able to store this in a reasonably compact form while introducing only negligible latency to incoming and outgoing network traffic. Keep in mind that IP address and port ranges may be very wide – your code should be able to handle the cases of “all IPs” (0.0.0.0-255.255.255.255) and “all ports” (1-65535) specified via ranges.

Some examples

These examples are in Ruby, but should translate with minor changes to most languages. The lines that are not prefaced by “>” represent the output in a typical REPL; your `accept_packet` function should return the value, not print it.

The examples below and the example CSV input on the previous page are just examples! Successfully passing these example test cases in a reasonable amount of time doesn’t necessarily guarantee that your code is correct or efficient. You should write additional tests to convince yourself of the correctness and performance of your code.

```
> fw = Firewall.new("/path/to/fw.csv")
> fw.accept_packet("inbound", "tcp", 80, "192.168.1.2") # matches first rule
```

```
true
> fw.accept_packet("inbound", "udp", 53, "192.168.2.1") # matches third rule
true
> fw.accept_packet("outbound", "tcp", 10234, "192.168.10.11") # matches second rule
true
> fw.accept_packet("inbound", "tcp", 81, "192.168.1.2")
false
> fw.accept_packet("inbound", "udp", 24, "52.12.48.92")
false
```

Evaluation guidelines

We will be evaluating your code in three main areas:

1. Functionality
    a. Does the code work correctly?
    b. Are there any valid inputs for which the code returns incorrect results or breaks?
    c. Did you test your code, exercising common cases and edge cases? If you have test files or scripts, please include them in your repository. Otherwise, please include a description of how you tested your code in your submission.

2. Code clarity and cleanliness
    a. Is the code well-structured and does it make use of object-oriented principles where appropriate?
    b. Is the logic well encapsulated? Is common logic shared in functions with reasonable interfaces?
    c. Is it easy to understand what the code is doing? Are the names of variables and functions descriptive? Does the code avoid overly complex or esoteric syntax?
    d. Are particularly tricky areas of the code well-commented to guide the reader?

3. Performance
    a. There are no “right or wrong” answers when it comes to this section. Making tradeoffs between space and time complexity is a core component of this coding assignment.
    b. We are interested in seeing that you thought about performance instead of simply settling for the naïve solution. Even if you do not get around to implementing an optimal solution, we are interested in ideas that you thought about or areas of the code that you’ve identified as candidates for optimization.
    c. If you are selected for a subsequent round of interviews, you and your interviewer may discuss performance-related tradeoffs at length, so please be ready to talk about the decisions that you made.
    d. In general, we expect the code to work “reasonably quickly” (i.e. not appear unresponsive) for large datasets of 500K – 1M items, after the dataset has been loaded (i.e. after the constructor has successfully returned).