# Yelp Code Test

## Problem description

The year is 2113. Human colonies are thriving on Mars and the Moon, but the settlers frequently complain about the poor selection of freeze-dried foods available to them. Seeing a great business opportunity, Pete decided to start a galactic pizza delivery service.

Despite its astronomical delivery times, "Pete's Interplanetary Pizza" has received largely positive feedback on Yelp ("Connecting People with Great Galactic Businesses"). However, some customers have been complaining that Pete double-charged them for their pizza.

Suspecting a technical issue, Pete has hired you to re-write his company's ordering software, which was originally written by the delivery boy.

You will provide Pete with code to generate an `order-engine` executable. Your program will listen to `stdin` for order updates (sent from the kitchen and delivery ships), act on those updates appropriately, and then (when an end-of-file is received) output a small summary of the work it did to `stdout`. Your code may be written in any language (but Python is preferred).

## Order updates

Updates are newline-separated (`\n`) JSON objects, required to have at least these three fields:

- `orderId`: a signed integer,
- `updateId`: a signed integer,
- `status`: one of 'NEW', 'COOKING', 'DELIVERING', 'DELIVERED', 'REFUNDED', or 'CANCELED'

An update may also include additional fields, such as:

- `amount`: a non-negative integer

Orders with a NEW status will contain an `amount` field for the amount to be charged (in US dollars). NEW orders without an `amount` field should be ignored. An update may include additional fields, but these extra fields can be ignored.

A few example updates are below:

```
{"orderId": 100, "status": "NEW", "updateId": 287, "amount": 20}
{"orderId": 100, "status": "COOKING", "updateId": 289, "cookTime": 7}
```

```
{"orderId": 100, "status": "COOKING", "updateId": 289, "cookTime": 7}
{"orderId": 101, "status": "NEW", "updateId": 289, "amount": 13}
{"orderId": 100, "status": "DELIVERING", "updateId": 294}
{"orderId": 102, "status": "NEW", "updateId": 291, "amount": 17}
{"orderId": 101, "status": "CANCELED", "updateId": 290}
```

Due to the difficult nature of interstellar communication, updates may occasionally be malformed or invalid. If an update doesn't contain the three required fields, the fields are not the required type, or if it doesn't parse correctly, then it should be ignored.

Additionally, an update may be re-sent if the sender suspects you didn't receive it (reception is terrible in deep space, after all). Your program should only recognize an update once. Updates are uniquely identified by (`orderId`, `updateId`); that is, an update should not be applied to an order if you already applied an update to that order with the same `updateId`. Valid `updateId`s (for a given `orderId`) are provided in an increasing order.

## Business rules

- All valid orders will begin in the NEW state.
- A NEW order can become COOKING.
- A COOKING order can become DELIVERING.
- A DELIVERING order can become DELIVERED.
- An order can become REFUNDED only if it is DELIVERED. (Pete offers an aggressive '300-day delivery or you eat free' promotion.)
- An order can become CANCELED at any time, except if it is DELIVERED or REFUNDED.
- Updates not following these rules should be ignored.

## Summary

When your program gets an EOF, it should output a summary similar to this one (generated from the sample input above) and then exit:

```
New: 1
Cooking: 0
Delivering: 1
Delivered: 0
Canceled: 1
Refunded: 0
Total amount charged: $20
```

Orders only proceed from 'NEW' if Pete is able to successfully charge the customer, so the 'Total amount charged' that you display should be the sum of amounts for orders not in a NEW, REFUNDED, or CANCELED state.

This summary is all your program should write to `stdout`. Writing warnings/errors to `stderr` is fine.

## Other requirements

Your code should be written as if it were something you wanted others to read. It should be:

- delivered as a single archive (zip or tarball)
- unit tested
- documented, preferably in code, or in a README
- clear and readable
- easily extensible, in case Pete needs to add new steps to the ordering process
- able to handle a reasonably large volume of orders

We do not expect you to write a JSON parser as part of this project. Grab a parser from json.org if your language doesn't include one.