Problem

Given a deeply nested JSON object, write code to flatten out the JSON and split it up into individual JSON objects with one level of nesting. The child objects generated from the JSON should be such that the original JSON can be reconstructed.

e.g. The JSON listed below has three levels of nesting. Once your code runs and processes it, it should generate multiple objects with a single level of nesting. In this case, four different types of child objects would be generated. The sample output is listed on the next page.

Input

```
{
    "restaurants": [{
        "id": "58b868503c6f4d322fa8f552",
        "version": ‘asdjasd’,
        "address": {
            "building": "1007",
            "coord": "[-73.856077, 40.848447]",
            "street": "Morris Park Ave",
            "zipcode": "10462"
        },
        "borough": "Bronx",
        "cuisine": "Bakery",
        "grades": [{
            "date": "2014-03-03T00:00:00.000Z",
            "grade": "A",
            "score": {
              "x": 1,
              "y": 2
            }
        }, {
            "date": "2011-11-23T00:00:00.000Z",
            "grade": "A",
            "score": {
              "x": 11,
              "y": 22
            }
        }],
        "name": "Morris Park Bake Shop"
    }]
}
```

Output

restaurant (Top-level object)

```
[
   {
       "id": "58b868503c6f4d322fa8f552",
       "borough": "Bronx",
       "cuisine": "Bakery",
       "name": "Morris Park Bake Shop"
   }
]
```

restaurant_address

```
[
  {
       "id": "58b868503c6f4d322fa8f552",
       "building": "1007",
       "coord": "[-73.856077, 40.848447]",
       "street": "Morris Park Ave",
       "zipcode": "10462"
  }
]
```

restaurant_grade

```
[
  {
         "id": "58b868503c6f4d322fa8f552",
         "__index": "0",
         "date": "2014-03-03T00:00:00.000Z",
         "grade": "A"
  },
  {
         "id": "58b868503c6f4d322fa8f552",
         "__index": "1",
         "date": "2011-11-23T00:00:00.000Z",
         "grade": "A"
  },
]
```

restaurant_grade_score

```
[
  {
          "id": "58b868503c6f4d322fa8f552",
          "__index": "0",
          "x": "1",
          "y": "2"
  },
  {
          "id": "58b868503c6f4d322fa8f552",
          "__index": "1",
          "x": "11",
          "y": "22"
  },
]
```

Note that the top-level fields which are primitive data types form one object and each nested JSON or array becomes a new object. This transformation is recursively done until all the nesting is eliminated.

The naming convention used for the child object in our output is as follows:

{parent_name}_{child_attribute_name}

The utility of all this data wrangling is that once the flattened JSONs are generated, we can extract, transform and load data that resides in an object store and load it into a relational database.

Deliverables

1. You should come up with a solution that takes a file with a parent JSON as input and generates files containing the individual child JSONs after flattening the parent JSON. In the example above, you would be given a file called `restaurants.json` containing an array of nested restaurant objects and your solution would output the following 4 files:
    a. `restaurant.json` - contains an array of restaurants
    b. `restaurant_address.json` - contains an array of restaurant addresses
    c. `restaurant_grade.json` - contains an array of restaurant grades
    d. `restaurant_grade_score.json` - contains an array of restaurant grade scores
2. To simplify things, you can assume the following:
    a. The maximum JSON nesting depth is 5.
    b. Strings only use UTF-8 chars
3. To help with your testing, you can use the three JSON files which can be downloaded from the following s3 location. https://s3-us-west-2.amazonaws.com/datacoral.codex.

4. Email a working version of your solution to solved@datacoral.co
5. We may validate your code with other test inputs
6. For bonus points,
    a. You can add unit tests for your code.
    b. Your code should take in the flattened JSONs that you generated as input and be able to reconstruct the original JSON.

FAQ

1. What language can I use?
   We like nodejs and python but feel free to use any language that you are comfortable with.
2. What libraries can I use?
   You can use any standard library that is available.