Warm-up Assignment 2

CS 4500 Software Development

Due: Thursday, January 28th at 9pm

Submission: The submission process is as follows.

1. At the top-level of *your Khoury GitHub repository*, which we created for you, create a directory `A2` with the following:

   a) the file `traveller.md` – the artifact from Task 1
   b) the directory `src/` containing the source code for the program `a2` from Task 2

2. Create a new release on GitHub, named `a2` and add a ZIP file `a2-exec.zip` as the "binary" containing the `a2` executable and any auxiliary files required by the executable. Details on how to do that are here.

3. Download the *Source code (zip)* from the release and submit it via Handins.

Task 1

Your company wants hire a consultancy to develop a module. You were put in charge of writing a specification for the interface for a module, dubbed `Traveller`. This specification will be implemented by the newly hired programmers in your chosen language and ship back real soon.

The high-level purpose of the desired module is to provide the services of a route planner through a *network of towns* for a role-playing game. For our purposes, a *town network* is a simple graph specification together with a placement of in-game characters.

The module must support these operations:

1. the creation of a town network with named nodes;
2. the placement of a named character in a town; and
3. a query whether a specified character can reach a designated town *without* running into any other characters.

Formulate your specification in a mix of English and technical terms appropriate for your chosen language. For example, in Java you would use the term *package* and you might use types while a Python programmer would speak of *modules* and use informal data definitions. You might want to revise material on data definitions from Fundies I.

Write the specification using the Markdown format as the file `traveller.md`. When printed, it must fit on a single page. Specify the precise implementation language (name, version) in which you expect the product to be written in.

Pedagogy

The goal of Task 1 is to get a glimpse of the tasks that team leaders perform in software companies. This specification is quite small so that a single page should suffice to give precise instructions to developers.

Task 2

The company-wide poll on a data interchange format showed an overwhelming support for JSON Now it is your turn to explore the JSON capabilities of your chosen language.

Write a program, `a2`, that reads a series of well-formed JSON strings from `STDIN`. In addition to being well-formed JSON, the values are also valid elements of this set:

> A `NumJSON` is one of the following kinds of well-formed JSON values:
>
> - A `Number` or a `String`
> - An `Array` of `NumJSON`
> - An `Object` containing at least the key `"payload"` whose value is a `NumJSON`. Other keys might contains arbitrary `NumJSON` values, but these are irrelevant and to be ignored.
>
> No other well-formed JSON values are valid NumJSONs.

The program should also take a single command line argument, which is either `--sum` or `--product`. Once `STDIN` is closed, the program should, for each of the given `NumJSON` values, compute a sum or a product (depending on the given argument) of all numeric values contained in the NumJSON. The result for a `String` value should be the unit for the respective operation (0 for `--sum`, 1 for `--product`). In other words, they are ignored and don't change the result. For `Objects`, only consider the `"payload"` value. The output should be a *JSON array of objects* with the fields `"object"`, containing the original `NumJSON` value, and `"total"`, containing the total computed for that object.

Here is a sample single input:

```
12        [2, "foo",
4]  { "name" : "SwDev", "payload" :
  [12, 33]     ,
        "other" : { "payload" : [ 4, 7 ] } }
```

If invoked as `./a2 --sum`, the output should be:

```
[ { "object" : 12, "total" : 12 },
  { "object" : [2, "foo", 4], "total" : 6 },
  { "object" : { "name" : "SwDev",
                 "payload" : [12, 33],
                 "other" : { "payload" : [4, 7] } },
    "total" : 45 } ]
```

If invoked as `./a2 --product`, the output should be:

```
[ { "object" : 12, "total" : 12 },
  { "object" : [2, "foo", 4], "total" : 8 },
  { "object" : { "name" : "SwDev",
                 "payload" : [12, 33],
                 "other" : { "payload" : [4, 7] } },
    "total" : 396 } ]
```

Note that whitespace is insignificant, both on input and on output.

Pedagogy

The goal of Task 2 is to figure out

i) how your language processes command line arguments,

ii) whether it has a good library for reading and writing JSON, and

iii) how to deploy programs runnable in LINUX in your chosen language. See sources like this one for details.