

Warm-up Assignment 3

CS 4500 Software Development

Due: Tuesday, February 2nd at 9pm

Submission:

1. At the top-level of Khoury Github, create a directory A3 with the following:
 - a) a directory `src/` containing the source code for the program `a3` from Task 1
 - b) a directory `traveller-server/` containing the source code for Task 2
 - c) a directory `traveller-client/` containing the source code for Task 3
2. Create a new release on GitHub, tagged `a3` and add a ZIP file `a3-exec.zip` as the “binary” containing the `a3` 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. Note that this zip file will contain all files from your repository. This is fine.

Note: Even though we use lower-case letter for source files and directories, your implementation language might require you to capitalize, or use CamelCase. In that case, follow the rules of your implementation language.

In programming languages we speak of two kinds of servers and two kinds of clients. A *server module* is a component that implements an interface for the purpose of serving functionality to other modules; conversely, a *client module* consumes the services of a server module. In distributed (web) systems people speak of server and client when they refer to software components like server and client modules, except that the services are used across a network protocol (possibly on the same machine) and the components can be implemented in distinct programming languages.

Task 1

Since your eventual product will consist of a server and remote clients, your manager wishes to understand how well your chosen language deals with TCP connections.

Develop a TCP-based server variant of program a2 from [Warm-up Assignment 2](#). A client connects to port 8000 and then sends a series of NumJSON values. Once the server reads the non-JSON sequence END, it computes the *sum* of the numeric values (i.e., the behavior of `./a2 --sum`), and sends the resulting JSON array back to the client over the TCP socket. Then it closes the socket and exits.

Ideally, you should be able to reuse code from Warm-up 2 *without* modification.

Pedagogy

The goals of Task 1 are

- i. to get an idea of TCP-based processing (including ports) and
- ii. to practice practical code re-use. Clearly this server is just a wrapper around the solution of Task 2 of [Warm-up 2](#). Improve only as absolutely needed.

Task 2

You find yourself in charge of implementing another team's specification for the Traveller module. You will find the specification deposited to your Github repository later today. Implement the specification in the language given in the specification.

If the given specification does not articulate what is to be computed in certain situations, you may implement whatever is convenient.

If the specification requests capabilities that are unnecessary to implement `traveller-client` (see below), you do not have to implement them.

The specification you receive may be requesting an implementation in a language different than your chosen language. If you cannot implement it in the requested language, implement it in your chosen language. As an experienced programmer, even if the details of the specification language are foreign to you, you should be able to adapt it to your setting. If you take this route, supply an additional memo in `traveller-server/implementation.md` (in addition to your source file), explaining how you interpreted the specification for your language and each choice you had to make.

Name the main module `traveller-server.<ext>` where `<ext>` is the extension appropriate for the implementation language.

Exception: What if I receive a specification that I cannot implement??

If you cannot understand the specification, you write a memo that diagnoses the problems of the specification (ambiguity, under-specification, over-specification, etc). The memo must list examples of each problem. For each problem also propose a solution on how the specifier could have done better. You may write up to 5 pages in the format of the memo from [Warm-up 1](#).

If you raise an exception, deliver `implementation.md` in the `traveller-server/` directory.

Task 3

Implement a *client module* (= module that uses the services of another module) that relies on your *own* specification (`traveller.md`) to build and query town networks. Once the outsourced implementation of Traveller is complete, you should be able to link the two pieces together and obtain a complete program.

The client specification is as follows:

`traveller-client` reads JSON values from STDIN and prints answers to STDOUT. Below are the *well-formed JSON* values `traveller-client` must deal with. We use **String** to indicate the type of the actual JSON value in that position.

1. Creating a road network of towns

```
{ "command" : "roads",  
  "params" : [ {"from" : String, "to" : String }, ... ] }
```

The command must be used once and as the first one; if it is invalid, the client shuts down.

2. Placing a character in a town

```
{ "command" : "place",  
  "params" : { "character" : String, "town" : String } }
```

The command is valid if town is a node of the given town network graph.

3. Query if a character can move to another town

```
{ "command" : "passage-safe?",  
  "params" : { "character" : String, "town" : String } }
```

The command is valid if a character specified by character has been placed and the town called name is a node of the specified town network.

All invalid "place" and "passage-safe?" JSON values are discarded. If the program encounters a non-JSON text, it may shut down without further warning.

You must make a decision whether (or how much) this client module can check the validity of well-formed JSON expressions *without* re-implementing the town-network functionality. You must rely on our own specification to make this decision.

Pedagogy

The goals of tasks 1 and 2 are

- i. to code against “foreign” specifications of code and
- ii. to show you the difficulties of writing good specifications for a software component.