More Labyrinth

CS 4500 Assignment C

Due: Tuesday, October 1, 11:59pm

Submission: You must deliver the following artifacts in a directory called `C` in your repository's master branch.

Task 1: An executable `C`

Task 2: A source file, `server-for-given-traversal.PP`
OR Markdown memo, `server-for-given-traversal.md`
OR both (see below)

Task 3: Source file: `client-for-your-traversal.PP`

Task 4: `micro-service.md`

Note: `PP` is a language-specific filename extension (e.g., `java` for Java, `py` for Python, ... you get the gist).

Any and all auxiliary files must be put into a sub-directory called `Other` within `C`.

Task 1

The purpose of this task is to explore the TCP facilities of your chosen language. Based on the survey of `format` files from Assignment B, Task 2, all of you seem to prefer JSON as the data exchange format, and we will limit ourselves to this format.

Develop a TCP-based server variant of program `B` from Task 2, Assignment B. A client connects to port 8000 and then sends a series of JSON values, following the **same restrictions as in Assignment B**. Once the TCP input stream is closed, the server sends the sorted list of values back on the TCP output stream. To keep things simple, sort the list in **ascending** order only.

Note: The server you produce in this task is just a wrapper around the solution for Task 2 of Assignment B. You should only improve the code as absolutely needed. If you did a good job of structuring your code and splitting it into subroutines, reusing it for this task should be easy.

Task 2

A Labyrinth specification `traversal.md` has been deposited in your repository in directory `C`. Your task is to provide an implementation of this 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 `client-for-your-traversal` (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 `server-for-given-traversal.md` (in addition to your source file), explaining how you interpreted the specification for your language and each choice you had to make.

What if we cannot implement the given specification?

If you receive a specification that you cannot implement, because you cannot understand it, 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 ten pages in the format of memo A.

If you cannot implement the specification, deliver your memo as `server-for-given-traversal.md`.

Task 3

Implement a client module that relies on **your own** specification of `Labyrinth` (`traversal.md` from Assignment B) to build and query labyrinths. Once the team responsible for implementing the server sends you the requested implementation back, you can link the two pieces together to obtain a complete, running program.

The client reads JSON values from STDIN and prints answers to STDOUT. Here are the *well-formed* JSON requests (commands) the client must deal with:

1. **Labyrinth creation**

   ```
   ["lab", {"from" : name:string, "to" : name:string} ...]
   ```

   ~~The command is valid if the names specified in the "from" fields of the objects are pairwise distinct.~~ The command is valid if the `name` pairs specified in the `"from"`-`"to"` fields are pairwise distinct, considered as *unordered*. That is, if `{"from" : "A", "to" : "B"}` appears, there is no other `{"from" : "A", "to" : "B"}` and no `{"from" : "B", "to" : "A"}`.

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

2. **Place-token request**

   ```
   ["add" , token:color-string, name:string]
   ```

   The command is valid if the `name` is a node of the given labyrinth.

3. **Move token query**

   ```
   ["move", token:color-string, name:string]
   ```

   The command is valid if a token of the specified `token` has been placed and the node refered to by `name` is a node in the specified labyrinth.

`color-string` is one of the following:

- `"white"`
- `"black"`
- `"red"`
- `"green"`
- `"blue"`

All invalid `add` and `move` JSON requests are discarded. If the program encounters a non-JSON text, it may shut down without further notice.

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

The goals of tasks 2 and 3 are:

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

Task 4

Your company has decided to build a labyrinth "micro service" so that it can be commercialized as `lab.com`. They put you in charge of adding a complete protocol specification for the server, that is, a description of how clients connect to the server (ordering and shape of messages).