DIT636 / DAT560 - System Testing Exercise

The airport connection check is a high-level function exposed by the API of a travel reservation system. It is intended to check the validity of a single connection between two flights in an itinerary.

```
validConnection(Flight flightA, Flight flightB)
               returns ValidityCode
```

For example, a traveler may intend to fly from Gothenburg to Los Angeles, but there is a connection through Frankfurt. Therefore, their itinerary is Gothenburg -> Frankfurt (Flight A) and Frankfurt -> Los Angeles (Flight B).

This service will ensure that the connection through Frankfurt is a valid one. For example, if the arrival airport of Flight A differs from the departure airport of Flight B, the connection is invalid. That is, if we pass in two flights, and Flight A arrives in Frankfurt, but Flight B departs from Munich, it is not a valid connection.

Likewise, if the departure time of Flight B is too close to the arrival time of Flight A, the connection is invalid. If Flight A arrives in Frankfurt at 8:00, and Flight B departs at 8:05, there is not sufficient time to complete the customs process and board the flight.

A `Flight` is a data structure consisting of:
- A unique identifying flight code (string, three characters followed by four numbers).
- The originating airport code (three character string).
- The scheduled departure time from the originating airport (in universal time).
- The destination airport code (three character string).
- The scheduled arrival time at the destination airport (in universal time).

There is also a **flight database**, where each record contains:
- Three-letter airport code (three character string).
- Airport country (string).
  - If the country is in the Schengen Area, this is indicated instead of the home country.
- Minimum domestic connection time (integer, minimum number of minutes that must be allowed for flight connections to be valid).
- Minimum international connection time (more time is required due to need to clear customs and meet regulations)

Note that connection times are calculated based on the originating airport and destination airport of Flight A. If the country of Flight A's originating and destination airports differ, then the international connection time is required. For example, if you flew from Los Angeles to Frankfurt (USA -> Schengen Area), then the traveler would need to clear customs before boarding flight B. This requires a longer connection time. If Flight A's originating airport is in a Schengen Area country, connections to all other Schengen Area countries are considered to be "domestic flights" and do not require completion of customs.

`ValidityCode` is an integer with value 0 for OK, 1 for invalid airport code, 2 for a connection that is too short, 3 for flights that do not connect (Flight A does not land in the same location that Flight B departs from), or 4 for any other errors (malformed input or any other unexpected errors).

Begin the process of designing test cases for the `validConnection` function.
- **Identify choices (aspects of execution that you control and that affect the outcome of executing the function) for each parameter of the function.**
   - **The parameters include FlightA, FlightB, and the flight database.**
- **For each choice, identify the representative values (the options for that choice).**
- **Apply ERROR, SINGLE, and IF constraints to those representative values, when applicable.**
   - **ERROR = This representative value for Choice A will trigger an error no matter what representative values are chosen for the other choices.**
   - **SINGLE = This representative value is an uncommon one that should lead to a normal outcome, but we want to make sure we try it once in case an issue occurs.**
   - **IF = This representative value can only be selected for Choice A if a particular value is set for Choice B.**

You do not need to complete the test specifications at this time, just identify choices, representative values, and constraints.

Hints:

Recall the lectures on system testing (Lectures 5-6).

The approximate process of creating system tests is to:
1. For an **independently testable feature**, identify the parameters.
   - These can be explicit (passed into the function) or implicit (configuration options or other environmental factors - such as databases - that influence the outcome of the function).
2. Each parameter can be manipulated in many ways through testing. For each parameter, you must identify **choices**.
   - Aspects of that input that you control, and that will have some impact on the outcome of testing this function.
   - For example, if an input is a data structure, the **choices** might include fields of that data structure that impact the outcome of the function. If that data structure is serialized from a file, then **choices** may also include the status of the file (whether that file exists, is corrupted, and so on).
3. For each choice, you must then identify the **representative values**. These are the options that can be selected for that choice.
   - If there are many possible options (e.g., any possible integer), you should partition the values into distinct groups (e.g., negative integers, 0, and positive integers).
4. Once representative values are chosen, you can form **test specifications** - abstract "recipes" for tests - by choosing one value for each choice.

5. You can transform test specifications into one or more concrete **test cases** by selecting concrete input values that map to each chosen representative value.

In this exercise, you have been asked to perform Steps 1-3 above - identify the parameters, split the parameters into choices, then identify representative values for each choice. You can constrain the number of test specifications by adding constraints (ERROR, SINGLE, IF) that note when certain representative values should be used in combination with values for other choices. **Note that you do not have to use all constraints or use constraints in all situations.**

This function has two explicit inputs - the two flights - and an implicit input - an airport database. A flight is a complex data structure containing several fields, each of those fields represents a controllable input category. Your choices should revolve around those fields.

Remember that the function's parameters may influence each other (testing this function requires considering both Flight A and B's field values as well as what is in the database), so the representative values must reflect how different choices and parameters can interact. IF-constraints are also a good way to indicate when representative values for two choices should be paired.

As an initial example:

FlightA

Choice: Originating Airport Code

- Valid airport
- Not in database **[error]**
- Not correctly formatted (not a three-letter string) **[error]**

FlightB

Choice: Originating Airport Code

- Valid airport, same as FlightA's Destination Airport Code
- Valid airport, different than FlightA's Destination Airport Code **[error]**
- Not in database **[error]**
- Not a correctly formatted airport **[error]**