DIT635 - Functional Testing Exercise

First, if you have not finished the activity from Lecture 7 (Functional Testing), do so!

The airport connection check is part of a travel reservation system. It is intended to check the validity of a single connection between two flights in an itinerary. For instance, if the arrival airport of Flight A differs from the departure airport of Flight B, the connection is invalid. Likewise, if the departure time of Flight B is too close to the arrival time of Flight A, the connection is invalid.

```
validConnection(Flight arrivingFlight, Flight departingFlight)
                returns ValidityCode
```

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 (in universal time).
- The destination airport code (three character string).
- The scheduled arrival time (in universal time).

There is also a flight database, where each record contains:

- Three-letter airport code (three character string).
- Airport country (string).
- Minimum domestic connection time (integer, minimum number of minutes that must be allowed for flight connections).
- Minimum international connection time (more time is required due to need to clear customs and meet regulations)

`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 (arrivingFlight does not land in the same location as departingFlight), or 4 for any other errors (malformed input or any other unexpected errors).

In order to design requirements-based test cases, perform category-partition testing using this specification for the `validConnection` function.

1. **Identify the parameters that can be controlled through testing.**
2. **Identify testing categories (controllable items) for each parameter.**
3. **Identify representative input values for each category.**

Hints:

Recall the lecture on functional testing. The approximate process of taking the requirements and writing tests is to:

1. Refine the requirements so that they are testable.
2. As a requirement is just a property of the software, you usually can't directly "test the requirement." Instead, you need to use a function of the software and show that the requirement holds during that execution. So, the next step is to identify the independently testable features of the software.
3. For each independently testable feature, you need to identify the parameters. These can be explicit (passed into the function) or implicit (environmental factors that influence the outcome of the function.
4. Each parameter can be manipulated in many ways through testing. For each parameter, you must identify categories of input that can be chosen. These are things that you can control when you test, whether explicit variable values or factors under your control. For example, if an input is a data structure, the categories might include each field of that data structure that could influence the outcome. If that data structure is serialized from a file, then you can also control factors like whether that file exists, is corrupted, and so on.
5. You cannot exhaustively test a function, there are too many possible parameters. So, instead, you partition the input domain into representative regions (types) of input. For each category, identify representative input values for that category. If you try inputs from each of these regions, you are more likely to trigger a fault than through random testing alone. We discussed some methods of performing this partitioning in class.
6. Once you have the inputs partitioned, you can form abstract test cases for which you can transform into actual test cases by coming up with concrete input values from the identified partitions.

In this exercise, you have been given a testable feature, so you have been asked to perform the activity from Steps #3 - 5 above - identify the parameters, split the parameters into controllable input categories, then partition the categories into representative values.

This function has two explicit inputs - an arriving flight and a departing flight - and an implicit input - an airport connection database. A flight is a complex data structure containing several fields, each of those fields represents a controllable input category.

Remember that the function's parameters may influence each other (testing this function requires considering both the arriving and departing flight's field values as well as what is in the database), so the representative values must reflect how multiple categories and variables can interact.