

CS306 Spring 2009
Homework Assignment #4
Due: Monday, March 23rd, 2009 at 4pm

Question 1. Complex Data Structures (50 points)

Part A. Read the Perl Data Structures Cookbook, which is part of the perldoc and can be accessed via the command “perldoc perldsc”. A few small sections (e.g. Debugging and Database Ties) may not make much sense, but the majority of this document is accessible to you based on what we have learned thus far.

Part B. Describe why it is especially important to “use strict;” when coding with references. Provide an example of broken code (meaningfully different than the one in the perldoc) that illustrates how one could get into trouble if they forget to “use strict” when working with references.

Part C. The document discusses four specific types of complex data structures (e.g. array of arrays, array of hashes, hash of arrays, and hash of hashes). Describe a real world example for each of these four types of complex data constructs. For example, an array-of-arrays would be a good choice to model a tic-tac-toe board. (You must think of a different example for an array-of-arrays).

Part D. Describe a real world example that would require a three-level data structure (other than one we discussed in class or which appears in the book or elsewhere in this assignment).

Program 1. Flight Tracker (100 points)

Your task is to write a program to track flight information. The pieces of information about a flight that are of interest to you are the airline, the flight number, its origin, its departure time, its destination, its arrival time, and the distance of the flight.

You want to design your data structure so that the most common types of lookups are efficient. The most common type of lookup is by airline and flight number (e.g. “Print the details of DL 323” where DL is Delta's code and 323 is the flight number). It would be nice if you could access this data directly, like this:

```
my $flight_ref = $flights{'DL'}->{323};
```

rather than having to do a loop to search for the flight that you want. You should design your data structure appropriately. (The code above tells you that the structure is a hash of hashes).

At the same time, you also want to be able to answer questions like “list the flights to ATL” or “print the schedule of BHM departures ordered by departure time”.

Your program has four parts.

1. Read the data in from a data file, if the data file exists.
2. Present the user with a menu of options to view/query the data in a variety of ways
3. Allow the user to add or delete a flight

4. Save the data structure to disk

The combination of #1 and #4 implies that the data lives on from one invocation of the program to the next – a common pattern in real world programming, and our first experience with “data persistence”.

Part 1. Read a Data File

Your program should allow for the name of a data file as its first parameter in @ARGV. This parameter is optional. If it is not provided, the program should start with an empty data structure. If it is provided but the file cannot be accessed, the user should be warned of this fact, and the program should continue with an empty data structure.

The format of this file is as follows:

```
DL 123 BHM 0820 ATL 1040 150
AA 3302 BHM 0855 DCA 1205 622
US 7720 NAS 0922 MEM 0955 144
etc....
```

The fields are:

```
airline flightnum origin departure destination arrival miles
```

Your program should successfully parse this format and create a hash of hashes as discussed before.

Part 2. Query the Data

After parsing the input file (if it exists and its name was provided as the first parameter on the command line), your program should enter a loop which presents the user with a series of choices about querying the data. The menu is as follows.

Please make a selection:

1. Lookup flight by airline code and flight number.
2. List all departures for a origin city (sorted by departure time)
3. List all arrivals at a destination city (sorted by arrival time)
4. List all flights for a particular airline (sorted by departure time)
5. List all of the known airline codes (sorted alphabetically)
6. Quit

Your choice:

Each choice leads to a different routine to handle the user request. Each choice will require gathering more information from the user, depending on their choice you will need to ask one or more followup questions to service their request.

The printout of a single flight should look something like this:

```
DL Flight # 123
  BHM to ATL
  Departs at 0820
  Arrives at 1040
  Distance: 150 miles
```

Note that depending on the user's choice, 0,1 or more than 1 flights might be in the answer list, and that list may need to be sorted by departure or arrival time, for instance.

After answering the user's question, keep presenting the menu until the user quits.

Part 3. Addition and Deletion of Flights

Add two new menu choices, Add a Flight and Delete a Flight.

To add a flight, you need to gather all of the pertinent information from the user and insert the data at the appropriate spot in the data structure.

To delete a flight, you simply need its airline code and flight number, which uniquely identifies it. You may be interested in the delete() function here.

Part 4. Save to a File

Change the Quit menu option to Save and Quit.

This task uses the provided input file name (if given on the command line) or a default of flightdata.txt if no filename was provided. It should then traverse the entire data structure and write to this file in the format specified above.

The net result is that I should be able to call the program like:

```
perl flighttrack.pl flightdata.txt
# add, delete some flights
perl flighttrack.pl flightdata.txt
# this second invocation sees all of the changes that were made during the first
invocation.
```

BONUS (25 pts). Allow for a flight to have intermediate destinations. This means a flight record might look like:

```
DL 123 BHM 0820 ATL 1040 MIA 1225 472
```

You will have to change you parse your data fields and also the way you store your arrival time and destination fields (they now need to be arrays). You will also need to change the way that you print out a flight, to something like:

```
DL Flight # 123
  BHM-ATL-MIA
  Departs at 0820
  Arrives at ATL 1040
  Arrives at MIA 1225
  Distance: 472 miles
```

Finally, you will need to change the way that you ask the questions to allow the user to enter new flights.

We are going to ignore the idea that intermediate destinations (ATL in the above case) could also be considered origins in their own right. For the sake of this program, we will make the assertion that a flight only has one origin (assume in this bizarro world people can only get off at ATL, not get on).

You should allow for ANY NUMBER of intermediate stops.