

CS306 – Introduction to Perl  
Spring 2007  
Homework Assignment #3  
Due: **Friday, March 30th, 12:00pm**

### **Guidelines for Submitting Homework**

All homework should be submitted in the form of a zip file which contains one program for each question. Your zipfile should be named *blazerid-hw3.zip* and should contain one text file called *answers.txt* (plain text format only, please – not Word) for the written answers, and then files like *hw3p1.pl*, *hw3p2.pl*, etc... for the programming assignments. **Make sure your name also appears in all source code files!** Email this zip file to [cs306@cis.uab.edu](mailto:cs306@cis.uab.edu). When emailing this file email a copy to yourself so you know the email went through ok.

**Question 1. (30 pts)** Write a regular expression that matches email addresses. This will be somewhat simplified; one that matches all valid email addresses is famously difficult. However, it's much easier to write one that will match these addresses:

[user@cis.uab.edu](mailto:user@cis.uab.edu)  
[user@uab.edu](mailto:user@uab.edu)  
[user.name@gmail.com](mailto:user.name@gmail.com)  
[user\\_name@someplace.else.com](mailto:user_name@someplace.else.com)  
[user\\_name.123@foo.org](mailto:user_name.123@foo.org)  
[user@hoover.al.us](mailto:user@hoover.al.us)

Essentially, the front part of the address can contain any number of letters, numbers, \_ and periods, and the domain part can contain one or more sets of letters, numbers and periods, and then a variety of 2- and 3-letter suffixes. Make sure your pattern matches all of those above, but does NOT match the following:

[user@cis](mailto:user@cis)  
[user.cis.uab.edu](mailto:user.cis.uab.edu)  
[user@foo@cis.uab.edu](mailto:user@foo@cis.uab.edu)  
[user!name@foo.com](mailto:user!name@foo.com)  
[user@cis.uab.comm](mailto:user@cis.uab.comm)  
[user@some.place.lco](mailto:user@some.place.lco)

You'll need to write a small helper program to test.

**Question 2. (10 pts)** Why do we need references in Perl? List two advantages.

**Question 3. (20 pts)** Create an anonymous hash and a reference to it, and then dereference using each of the three styles of dereferencing.

**Question 4. (20 pts)** Explain reference counting and how Perl does garbage collection.

**Program 1 (120 pts). PerlAir Is Growing.** PerlAir has been wildly successful and is now acquiring three more planes! In addition, your CEO has asked you to create software to allow for the scheduling of flight crew to the planes in addition to the passengers. Finally, the flight schedule department has requested that you allow them to use your system to plan out their flight routes. And finally, the airline is hiring a second shift of reservation agents, so you need to design your software so that the data can persist from one shift to the next.

Luckily, your fearless instructor has provided you with all of the tools you need to meet this challenge! This program will test your use of file management, references, multilevel data structures, subroutines and regular expressions. This will also be the first assignment that will be very sensitive to your design choices – it's big enough that a bad design choice could make the program very difficult to write and debug, so proceed carefully and think about the big picture.

Here is the required functionality:

1. Data Representation – The airline now has 4 planes. The planes are named Vulcan, Sloss, Cahaba and Idol. They are identical models: 4 rows, with two seats in each. The names of the planes will be the top-level keys of the %perlair hash structure.
2. For each plane, you need a hash entry that is keyed by the name of the plane (Vulcan, Sloss, etc...) that stores the following information:
  1. Origin (value is a scalar)
  2. Destination (value is a scalar)
  3. DepartureDate (value is a scalar; format described below)
  4. DepartureTime (value is a scalar; format described below)
  5. Pax (short for passengers) (value is reference to hash of seats – format as HW2)
  6. Crew (value is reference to array of crew member names)
3. Data Persistence. When the program exits, it needs to write its current data to a file called perlair.txt. The format that it writes in is your choosing, but I strongly recommend the [Data::Dumper](#) method discussed in class. When the program starts, it needs to check for the existence of that file, and if it exists, it must read it and populate the %perlair hash with the data. Otherwise, it should initialize the hash with no data.
4. The following operations need to be supported:
  1. Create Flight – This asks for a plane name, an origin, a destination, and a departure time. It then does a check to make sure that the plane is not already in use for a different flight. If it isn't, this information is used to set up the new flight.
  2. Assign Crew – This asks first for a plane name, and then a list of crew member names, one per line. It populates the anonymous array referenced by the Crew value of that plane's hash.
  3. Cancel Flight – This asks for a plane name, and empties out all of the data fields of that plane's hash.
  4. Clear Crew – This asks for a plane name, and clears out the crew data.
  5. Make Reservation – Unlike last time, this does not ask for a specific seat. Instead, this takes a passenger's name, their origin and their destination airport

codes, and their date of departure. If a flight is available on that route, and if there is a seat available, assign it to the passenger. Otherwise, inform them that the reservation cannot be made.

6. Clear Reservation – this takes a passenger name, an origin and a destination, and if they have a reservation on that flight, this cancels it.
  7. Search for Reservation – This takes a **regular expression** and searches all flights for passenger names that contain that pattern. So, “ran” should find all of the Fran's and the Brandon's that have reservations in the system.
  8. Print Manifest – This takes a plane name and prints out all of the data associated with that plane. Origin, Destination, Depart Time, Crew and Passengers, with their seat assignments.
  9. Print All Manifests – Does a complete data printout of the details of all planes. (This is a place for code reuse from #8.)
  10. Exit – This creates/updates the save file and exits the program.
5. Case Insensitivity. This must work throughout – on the plane names, the passenger names, crew names, origin and destinations.
  6. Data Validation. This is the other area of the program that will use regular expressions. Use regular expressions to validate that the origin and destinations are 3-letter airport codes, that the departure date is in the form “3/7/07” or “10/14/07”, that the departure time is in the form “3:45 PM” or “11:11 AM”, that crew and passenger names only contain letters.
  7. Subroutines. This program will be highly modular. You may have two dozen subroutines. For instance, each type of data validation should be a subroutine returning true or false.

**BONUS (10 points).** Some routes are very popular. Allow for multiple planes to serve the same route on the same day. If the first plane is full, the passenger should be booked on the second plane, and so on. This should be generalized (I would just search all planes until the right route and an empty seat is found).