

CS306 – Introduction to Perl  
Fall 2006  
Homework Assignment #2  
Due: October 30<sup>th</sup>, 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 *lastname-hw1.zip* and should contain one text file called *answers.txt* for the written answers (if any), and then files like *hw2p1*, *hw2p2*, 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.

### **Reading Prerequisite**

You will need to read Chapter 8, Files and Data, in order to complete this homework assignment. This chapter will not be discussed in class.

**Question 1 (10 pts)**. Give at least two advantages of using subroutines.

**Question 2 (10 pts)**. Why might you want to design your subroutines so that they take a hash as its arguments instead of a plain list of scalars?

**Question 3 (10 pts)**. The following code does not work as expected. What is the problem?

```
my @arr1 = (1,2,3);  
my @arr2 = (4,5,6);  
foo (@arr1, @arr2);  
sub foo {  
    my (@a1, @a2) = @_;  
    ...  
}
```

**Question 4 (10 pts)**. What does “word boundary” mean in the context of regular expressions? Can you think of a difference between a word in Perl regular expressions and a word in English?

**Question 5 (10 pts)**. What is greedy matching? Give an original example where using greedy matching and nongreedy matching on the same input gives a different result.

**Question 6 (10 pts)**. What is the advantage of the /x modifier? Give an example of its usage.

**Question 7 (10 pts)**. What are the advantages of using the <> operator over using STDIN or a filehandle?

**Program 1 (65 pts)**. Airline Reservation System. You find yourself the proud new owner of a fledgling airline, PerlAir. PerlAir owns only one plane. This small plane has only 4 rows, with two seats in each row (hey, at least everyone gets an aisle AND a window!) Your task is to write a small program to handle reservations for this plane.

The good news is that you operate on a small island where the only piece of information you need about your passengers is their first name. The process of making a reservation is simply assigning “bob” to seat “1b” or “lisa” to “3a”. No flight numbers (one flight is all that is planned for now) or credit card information (they'll pay in seashells later on) is required.

Using hashes and subroutines, write a simple, menu-driven program that allows the user to perform the following actions:

1. Make a reservation – this asks for the name of the passenger and the seat they want to reserve.
2. Cancel a reservation – this asks for either the name or the seat, and cancels that reservation.
3. Display seats – this shows all of the seats of the plane, showing which are occupied and which are available.
4. Name lookup – takes a name, returns the seat they have reserved, if any.
5. Seat lookup – takes a seat, returns the status of that seat.
6. Save and Exit – this saves the current state of the reservations to the file “reservations.txt” and exits. You can define whatever storage format you want.

This data can be easily modeled using only one hash.

Additionally, you have to meet the following requirements:

1. When the program starts, it should check for the presence of the data save file

- (from requirement #6 above) and if it exists, it should initialize the contents of the hash(es) with that information.
2. It must be case-insensitive throughout.
  3. Each of the six actions above should be modeled as a subroutine, plus there might be other places where subroutines are helpful (like reading the user's choice from the menu – you may want to `chomp()` the choice and convert to lowercase before returning it to the main part of the code).
  4. Make sure all operations are valid. e.g. don't allow a reserved seat to be reserved again, or an empty seat to be cancelled.

#### Helpful hints:

1. Save file format – I suggest saving the file with one hash entry per line, separated by a space. An entry might look like “1a joe” or “2b empty”.
2. Menu-driven interface. Think of this program structure:

```
my %seats;
# initialize data by checking if data file exists
if (...) { # check for data file existence in if
condition
  %seats = readDataFile();
}

# start main program loop
my $choice;
do {
  # show user a menu
  printMenu();
  # get user's menu choice
  $choice = getMenuChoice();
  # react to user's choice by calling appropriate subroutines
  ....
} until ($choice eq 'q');
```

Note that the advantage of this is that even after the user chooses to (q)uit, you will have an opportunity inside the do loop to do something in response to the quit (i.e. save the data to a file) before exiting the loop.

3. Passing hashes to and from subroutines – Remember, hashes get flattened out into lists before being passed. So a call like `foo(%hash)` will become `foo(name1, value1, name2, value2, name3, value3, ...)`. This is fine as long as inside the subroutine, you do this:

```
sub foo {
  my %hash = @_;
  ...
  return %hash;
}
```

and similarly, on the return, this works fine so long as you then do:

```
%hash = foo(%hash);
```

It's not pretty, but until we learn references, this will do just fine.

**Program 2 (65 pts)**. Email Parsing Program. Email messages consist of two parts, the header and the body. These two parts are separated by a blank line. Here is a (simplified) sample email:

```
Received: from localhost (localhost.localdomain [127.0.0.1])
by crier.cis.uab.edu (Postfix) with ESMTMP id 632C983E155 for
<fran@cis.uab.edu>; Mon, 9 Oct 2006 11:01:01 -0500 (CDT)
Received: from crier.cis.uab.edu ([127.0.0.1]) by localhost
(crier.cis.uab.edu [127.0.0.1]) (amavisd-new, port 10024) with
ESMTMP id 08890-07 for <fran@cis.uab.edu>; Mon, 9 Oct 2006
11:00:52 -0500 (CDT)
Received: from [138.26.64.214] (unknown [138.26.64.214]) by
crier.cis.uab.edu (Postfix) with ESMTMP id 9182983E3A1 for
<fran@cis.uab.edu>; Mon, 9 Oct 2006 11:00:52 -0500 (CDT)
Date: Mon, 09 Oct 2006 11:00:59 -0500
From: Fran Fabrizio <fran@cis.uab.edu>
To: fran@cis.uab.edu
Subject: Pizza Party Friday
X-Spam-Score: -4.336
```

Please come to the party, it will be in the break room on Friday at lunch time!

Thanks,  
Fran

Note that each line of the header begins with a keyword followed by a :, such as "Date:". Some lines are long, so they've wrapped around in this document (e.g. Received:), but in the actual email file, they will be one long line.

Your task is to write a program that reads a file containing an email message, and using

regular expressions, extracts out details of that message into a hash. Assume that one file = one email message, simplifying processing.

I suggest using the diamond operator to model your program, like this:

```
use Data::Dumper;  
my %msg;  
while (my $line = <>) {  
    # process each line through series of regular expressions  
    # storing interesting pieces in hash.  
}  
print Dumper(\%msg);
```

This way you do not need to know the filename of the email message. Your program can simply be invoked as “perl myprogram.pl emailfile.txt” and it will do the right thing.

Store the following information:

1. The sender's name.
2. The sender's email address.
3. The recipient's email address.
4. The subject of the email.
5. The body of the email.
6. The number of mail servers it passed through (1 Received: header = 1 server).  
Just track the count of total number of servers the message passed through.
7. The spam score for the message (note that it -may- be negative).
8. The time of day the message was sent (the “11:00:59” part of the Date: line)
9. The day of week it was sent (e.g. “Mon”)
10. The date it was sent (e.g. “09 Oct 2006”)

Sometimes multiple objectives can be met with a single regular expression. At the end of the program, just use [Data::Dumper](#) to print the contents of your %msg hash to the screen to demonstrate that you correctly parsed the email. I will provide sample email files for download, and will test your program with different sample files.