

CS306 – Introduction to Perl
Spring 2007
Homework Assignment #2
Due: **Monday, March 5th, 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-hw2.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 *hw2p1.pl*, *hw2p2.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*. When emailing this file email a copy to yourself so you know the email went through ok.

Question 1 (24 pts). Context. In what context are the following operations being used?

- | | |
|-----------------------------------|--|
| 1a. $\$foo = something;$ | 1g. <code>if (something) { ... }</code> |
| 1b. $(\$foo, \$bar) = something;$ | 1h. <code>while (something) { ... }</code> |
| 1c. $(\$foo) = something;$ | 1i. <code>foreach \$foo (something) { ... }</code> |
| 1d. $@foo = something;$ | 1j. <code>reverse something;</code> |
| 1e. $\$foo[3] = something;$ | 1k. <code>\\$foo = reverse something;</code> |
| 1f. $123 + something;$ | 1l. <code>\\$foo = @array;</code> |

Question 2 (16 pts.) Some functions and operators behave differently depending upon in which context they are called. `<STDIN>` is one such operator. Please give two code examples, one of `<STDIN>` being used in scalar context and one of `<STDIN>` being used in list context. Explain how the behavior changes.

Question 3 (15 points). Explain what a hash is in Perl, and provide two examples of how it is similar to an array, and two examples how is it different from an array. Finally, explain the types of data or situations for which you would use a hash instead of an array.

Question 4 (10 pts). Give at least two advantages of using subroutines.

Question 5 (10 pts). Give at least two advantages of designing your subroutines so that they take a hash as its arguments instead of a plain list of scalars. Provide your own

example of a subroutine that uses these advantages.

Question 6 (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 7 (15 pts). Describe the difference between a package variable and a lexical variable. Be sure to include discussion of how to create both types of variables, the scope of both types of variables, and the advantages and disadvantages of both types of variables.

Program 1 (30 pts). Hooray for Hashes. This program is an exercise in the benefits of hashes. You will first implement the solution with arrays, and then implement it again with hashes. You should do this in two separate files, so this homework will have file names hw2p1a.pl and hw2p1b.pl.

1a. Start your program with the usual shebang line and use strict, and then include these two statements near the top of your program:

```
my @teams = qw/UAB Alabama Auburn Florida Georgia Tennessee/;
my @mascots = ("Blazers","Crimson Tide","Tigers","Gators","Bulldogs","Volunteers");
```

Your job now is to write one or more loops to iterate over these data structures and produce the following output:

```
The UAB Blazers are my favorite team.
The Alabama Crimson Tide are my favorite team.
The Auburn Tigers are my favorite team.
etc...
```

1b. Now, produce the same output as you did in 1a, but use only one hash instead of two arrays. Hard-code your single hash just like we hard-coded the two arrays at the top of the program.

Program 2 (70 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 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. Exit – this exits the program.

This data can be easily modeled using only one hash. Assume that each time you start your program, the plane is empty.

Additionally, you have to meet the following requirements:

1. It must be case-insensitive throughout.
2. 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).
3. Make sure all operations are valid. Don't allow a reserved seat to be reserved again, or an empty seat to be canceled. Assume it is ok for one person to reserve multiple seats.

Helpful hints:

1. Menu-driven interface. Think of this program structure:

```
my %seats;

# 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');
```

2. Passing hashes to and from subroutines – Like arrays, 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.

BONUS (10 points). Do not allow people to reserve more than one seat. Caveat: You must do this by reusing a subroutine that you already coded for another part of the problem (Action #4) – do not duplicate the code for determining if a person already has a reservation.