

CS306 – Introduction to Perl
Fall 2006
Homework Assignment #1
Due: Monday, October 2nd, 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 *hw1p1*, *hw1p2*, *hw1p3*, 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.**

Homework #1 (200 total points)

Question 1 (40 points). In your own words, explain and illustrate the concept of context. Be sure to construct code samples which illustrate the same data used in different contexts.

Notes on programs: While writing these programs below, remember to comment thoroughly, including your name near the top as well as comments throughout explaining what you are doing in your program.

Also, remember to both turn on warnings and use strict; in all of your programs!

```
#!/usr/bin/perl -w  
use strict;
```

The /usr/bin/perl part may change depending on your platform; it's the -w that is important, as it is what enables warnings.

Read each problem description carefully, and be sure to address everything that is asked for in the problem.

A NOTE ON DEBUGGING: WHEN YOU GET STUCK, PRINT OUT YOUR VARIABLE VALUES AT VARIOUS POINTS IN YOUR PROGRAM. THIS IS OFTEN THE EASIEST WAY TO FIND WHERE THE CODE IS BROKEN.

Program 1: Areas (60 points)

Write a program to calculate the area of the following common geometric shapes: a circle, a square, a rectangle and a right triangle. First, ask the user for the type of shape. Use three letter abbreviations like cir or tri. Depending on the user's response, you now must ask for one or more lengths (e.g. the radius for a circle, or a height and width for a rectangle). Finally, calculate the area and print the results to the user.

Your program interaction might look something like this:

```
# perl areas.pl
What shape's area are we calculating? (squ, rec, cir or tri)  cir
What is the radius of the circle?  2
The area is 12.56636.
#
```

Some helpful hints:

1. Remember to chomp user input before using it in a comparison!
2. The function `lc($string)` returns the lowercased version of `$string`. Helpful for forcing user input to lowercase before comparing with the allowed strings. 'CIR', 'Cir' and 'cir' can all be valid input if you use `lc()` wisely.

Program 2: Exchange Rate (100 points)

Write a program that converts between currencies. Keep two arrays: one is a list of countries; the other is the number of US dollars 1 unit of that country's currency translates into.

For example, as of 9/10/06, one US dollar gets you 0.54 pounds in England. So, if index 0 of the countries array was the string 'ENG', index 0 of the rate array would be the number 0.54. Store at least five countries' worth of data in these two arrays. xe.com is a helpful website for current rates.

Now, allow the user to enter in a target country and the amount of that country's currency they have, and report back how many US dollars they can get in exchange. Allow the user to perform as many conversions as he wants, providing a way for him to quit. Here's what a sample program interaction might look like:

```

# perl hw1p2.pl
Which country's currency do you have? [ENG EUR JPN CAN MEX or QUIT] eng
How much of that currency do you have? 4
You can exchange that for 7.41 US Dollars.
Which country's currency do you have? [ENG EUR JPN CAN MEX or QUIT] MEX
How much of that currency do you have? 11023
You can exchange that for 999.37 US Dollars.
Which country's currency do you have? [ENG EUR JPN CAN MEX or QUIT] quit
Goodbye!
#

```

This program is a little trickier than it looks. One difficulty lies in the fact that you have two arrays that must be manually “lined up” by your code. The user enters something like “MEX”. However, you don't know which index of the arrays this corresponds to, and must calculate this by examining the user's input against each value of the array.

Some tips:

- Try not to do unnecessary calculations. Break out of loops once you have accomplished the goal of the loop, for instance.
- When testing arrays, always test the edge conditions - that is, the first and last elements. You can often find “off by one” errors this way.
- Always try to make user input case-insensitive. The functions `uc()` and `lc()` help.
- Infinite loops can sometimes be helpful. `while (1) { }` Just make sure something in that will cause the code to break out of the loop at some point!
- Remember that your goal is to take something like 'MEX', figure out which index this corresponds to, and then examine the value at that index of the rates array to figure out how to calculate the exchange to US dollars.
- Be careful with your math and which rates you store. If someone has 70 British pounds and we know that 1 British pound converts to 0.54 US dollars, then $70 / 0.54$ will tell you how many US Dollars you can get for 70 British pounds.
- `printf()` is helpful to round to two decimal places so that the US Dollar values look normal.
- Remember to `chomp` user input.

BONUS! (20 points) Allow the user to also specify which currency he wants to convert to, instead of assuming US Dollars. However, do not add any additional arrays or elements to the existing array. You have all the data you need to do this already. Make sure to allow for USA to be one of the target currencies now.