Homework 7

This homework is designed to give you experience with files and command line arguments, so read each problem carefully to determine where your inputs are coming from (usually a file or the command line) and where your outputs are going to (either a file or stdout).

1. Write a Perl file named **exp.pl** that contains a function that computes the value of e^x by using the formula:

   $e^x = 1 + (x/1!) + (x^2/2!) + (x^3/3!) + ...$

   The function should take x and the number of terms to use in the computation. If n is the number of terms then the last term used in the computation should be $x^n/n!$. The function's signature should be:

   e(x, n)

   The file should take x and n as command line arguments and print the result to stdout. For example:

   ```
   bvz% perl exp.pl 1 10
   2.71828180114638
   ```

2. Repeat problem 4 from Homework 6, except that now you I want you to 1) ignore case (e.g., Robin and robin are the same), 2) strip leading or trailing punctuation (e.g., robin, should be robin), 3) create a separate function to sort and print the results, and 4) provide command line argument(s) for the file name(s). As part of your solution, you must write a function that sorts/prints the words and their frequencies. The function should take a reference to your hash table as its sole argument. For this assignment, I also want you to sort the output by decreasing frequency of the words. If two words have the same frequency count, then you should print the words in alphabetical order. For example:

   ```
   robin: 20
   jay: 15
   cardinal: 12
   mockingbird: 12
   wren: 12
   hummingbird: 10
   ```

   The user may provide multiple files to your script, in which case you should keep combined frequency counts for the words from all of the files. You should still write your results to stdout. An example invocation might be:

   perl word_count.pl birds.txt wildlife.txt fox.txt

   Here are a number of additional tips/notes:
   a. You can use the lc (lower case) function to convert a word to all lowercase.

b. You may assume that a word starts with an alphanumeric character (alphabetic character, digit from 0-9, or _) and ends with the first non-alphanumeric character that is encountered. For example:
   #include, ➔ include
   &!Brad!&3 ➔ Brad
   $dict{$word}\n"; ➔ dict
   3 ➔ 3
   { ➔ discarded because it does not contain an embedded word
   for(i=0;i<10;i++) ➔ for
c. Remember that \W is the Perl character code for anything that is not alphanumeric. A word may start with 0 or more non alphanumeric characters, then 1 or more alphanumeric characters, and finally may end with 0 or more non alphanumeric characters.
d. It is okay to create the hash table that keeps the frequency counts and to populate this hash table in the main body of your perl file, but you must call your print function to print/sort the words in the hash table.

3. Write a script named **times.pl** that reads a file of data that contains race results of the form h:mm:ss, where h is hours, mm is minutes, and ss is seconds and perform the following two actions:
   a. Calculate and print the average of all the times in the file. The printf statement that will force a leading 0 into the minutes or seconds position if the average minutes or seconds is a single digit is:
      printf("average time = %d:%02d:%02d\n")
      The 0 in %02d says to pad the field with leading 0's if there are not enough digits to fill the field.
   b. Convert each string from the form "h:mm:ss" to the form "h hours, mm minutes, and ss seconds". Use a substitution pattern to do the replacement inline and write out the converted file.

   Hours is a single digit while minutes and seconds must be 2 digits. You may assume that the times have been correctly entered and that they are embedded between <time> and </time> tags. An example input file is:

   > Jim Colatis won Tuesday's Knoxville half marathon in a blistering pace of <time> 0:56:45 </time>. He was followed to the line by long time nemesis Mickey Mouse in a time of <time>0:58:49</time>. Mary Strauss finished first among the women with a time of <time>1:02:39</time>. She outsprinted Becky Smith and Joan Hare to the line, with Becky and Joan finishing in a time of <time>1:02:41</time> and <time> 1:02:45</time> respectively.

   An example invocation of the script might be:

   > perl times.pl KnoxHalfResults updatedResults

   Your input will come from the first command line argument, KnoxHalfResults, and your output should be written to the second command line argument, updatedResults. Notice that the first line of updatedResults contains the average time.

4. Write a Perl program named **center.pl** that searches for all header tags of the form <h1>, <h2>, or <h3> that start at the beginning of a line in an html file. Place the tags and around all such headers. The headers may span more than one line in the file. For example, your program should replace:

    <h1> An Introduction to Perl
    and Python 101 </h1>.

    with:

    <h1> An Introduction to Perl
    and Python 101 </h1> .

    Your program should read its input from stdin and write its output to stdout. In practice your program will redirect stdin and stdout to different files. For example:

    perl center.pl < report.html > centeredReport.html

5. This problem is going to give you practice with Perl's object-oriented programming facilities. You are going to create two classes named BoxShape and TextShape. To do so, create two files, one named **BoxShape.pm** and one named **TextShape.pm**. Here are the specs for the two files:

    a. TextShape is a subclass of BoxShape
    b. Boxshape has instance variables for its left, top, width, and height. It provides the following methods:
        i. a constructor called **new** that takes left, top, width, and height as parameters and initializes the instance variables to these values.
        ii. a **print** method that takes no parameters and that prints the string "BoxShape(left, top, width, height)" with left, top, width, and height replaced with the appropriate values.
        iii. a **containsPt** method that takes two integers, x and y, as parameters, and determines whether the point (x,y) lies within the BoxShape. It should return 1 if the point lies within the BoxShape, and undef otherwise. A good pseudo-code algorithm for determining whether a point lies within a bounding box is:

            (x > left) && (x < right) && (y > top) && (y < bottom)

        where right and bottom are easily computable from (left,width) and (top, height) respectively. For those of you familiar with graphical interfaces the reason for this code is that the y-axis starts from the upper left corner of a window and goes down.
        iv. a **getBounds** method that takes no parameters and that returns the bounding box of the BoxShape as a list of 4 elements. The bounding box is the smallest rectangle that encloses an object, and hence is simply a list consisting of left, top, width, and height. Return the bounding box as a list of four values, not as a reference to a list. I want to be able to call the function as follows:
        ($left, $top, $width, $height) = $myBox->getBounds();
        v. a **setBounds** method that takes four parameters—left, top, width, and height—and sets the corresponding instance variables to these parmeters. It has no return value.

c. TextShape modifies BoxShape in the following manner
    i. it adds instance variables called *font* and *text*. For this problem both of these instance variables should be string-valued, but you needn't error check them.
    ii. its **new** constructor takes 4 parameters—left, top, font, text. It calculates the width as the length of the text multiplied by 5 and the height as 20 (a real object would use the font to determine the actual width and height of the string).
    iii. its **print** method prints the string "TextShape(left, top, width, height, font, text)" with left, top, etc. replaced with the values of their instance variables.
    iv. it adds two new methods called **setText** and **getText.** setText takes a string as its parameter and sets the *text* instance variable to this string. It also updates the width to be 5 times the length of this string. getText takes no parameters and returns the current value of *text*. You do not have to write accessor methods for the font field, although you would in a real system.
    v. the remaining methods are inherited from BoxShape.

No error checking is required by your code. You do not have to handle 0-argument constructors.