You should **NOT** write the code using C++ and then translate. The goal is to be “Perlish” in your approach meaning you should use constructs and concepts specific to Perl. Code should be fairly easy to read and commented where it is not. Avoid using $\_ too often. Use hashes (associative arrays). Ask questions if you don’t understand the spec. For each problem write a file named problem-*i*.pl. Submit a zip file hw.zip containing all the pl files.

1. Write code that scans an array of months and replaces the month with the number of days in that month (ignore leap year). For example,

january

Should become

31

Each element in the array will be exactly 1 word, one of the 12 months in any combination of upper or lower case, and in any order.

In order to be able to easily handle upper and lower case words, I would suggest using one or both of these functions:

uc takes a scalar as a parameter and returns a scalar value with all letters uppercased. It does NOT change its parameter. For example,

$x= “aBc”;

$upper=uc $x; #$upper is “ABC” $x is still “aBc”;

Note you can have uc as part of a larger expression $y = fun(uc $x . $another);

This passes an upper cased form of $x concatenated with $another as the parameter to fun.

lc works in an analogous way but returns a lower case version of its parameter.

1. For each of the following, write **ONE** line of Perl code:
2. remove the element at index 3 from the array @blah
3. move the first element after the end @blah (so the number of items is unchanged,

(A, B, C) would become (B, C, A).

1. change the 3rd element (index 2) to be “hello”
2. remove the 3rd element and insert at that location the 3 elements: “new”, “again”, 4.7
3. remove the last element of @blah w/o using pop
4. add “hello”, “goodbye” to the beginning of @blah
5. insert @arr after index 7 in @blah
6. insert @arr after index 7 in @blah and remove all remaining elements from @blah
7. (2 points b/c you need more than 1 line) add 1 to every item in @arr

Clearly indicate in problem-2.pl which line of code answers which part (in case you get something wrong I don’t want to be confused). You may have additional code in the file as well (that you used during testing) just make sure I can find the answers. If you can’t get 1 line, do 2 lines for partial credit.

1. Write a subfunction named search that takes a value and a list as parameters and returns 1 or 0 (i.e. true or false) based on if the first parameter is found within the list. You must use a foreach loop in this problem. Search should work correctly for any number of parameters >= 2 (i.e. the list can be assumed to have at least 1 item).
2. Write a subfunction findMinIndices that takes as a parameter a list of numbers and returns a list of all the indices where the minimum value was found. You are NOT required to use a foreach loop in this problem. Your code should only iterate through the list 1 time. Examples:

searchLocation(3, 5, 7, 3, 3, 9) would return (0, 3, 4) b/c the 3 is the minimum value and it is found at indices 0, 3, and 4 within the list.

1. Given 2 arrays of strings named @people1 and @people2, write code that prints all the duplicate names between the 2 arrays. If there are no duplicates then print “No duplicates\n”. Your code should iterate through each array only 1 time (i.e. your code is O(n) where n = the length of the arrays). Treat John and john as different.

Next, your code should print the name that occurs the most often in total in the 2 arrays and the name that occurs least frequently. If there is a tie then all words that are tied should be printed. You may not use sort for this problem. The order of the words for a tie is not specified. The output would look like:

The most frequent name(s) are: Bob John occurring 4 times.

The least frequent name(s) are: Zachary occurring 1 times.

(Don’t worry about plurality or grammar.)