Homework #1 – Hints for solving problems 4 and 5.

Problem #3

Your function will have to return a result – which is a list of “buckets”. Each bucket will be a dotted pair whose CAR is an element from your original input list, and CDR is the counter value that represents the frequency of that element.

Bucket = (*element* **.** *counter*) 🡨 created with (CONS *element* *counter*)

Result = (*bucket1* *bucket2* ...) = ( (elem1 . counter1) (elem2 . counter2) ...)

If your main function is called, say, “***frequency*** (*l*)”, consider an auxiliary function, say, “***frequency1*** (*l result*)” of two arguments: your original ***l*** and a list (initially empty) where you would store the result and which you will pass along and update at each step of your recursion.

Now, when you call (frequency argument), the implementation of “frequency” will only go and call “frequency1” with the same argument l PLUS the empty list (initially the result is empty).

So, your main focus is now to implement “***frequency1*** (*l result*)”. You will have to look at the CAR of *l* and update the result accordingly, and then move on to the rest of the list (CDR) until *l* is empty. At every step, the result will be modified (depending on what the CAR is) and then passed to the next recursive step of “***frequency1***”.

Finally, your task is to implement a (third) function which takes as arguments an *element* (from the original list) and the *result* (list of buckets), and it updates this *result* in one of two ways:

1. If *element* is the CAR of a bucket in your *result* then, replace the bucket with a new one whose CAR is the same but whose CDR is incremented by one (i.e. *element* was already encountered at an earlier step of calling “***frequency1***”)
2. If *element* is not the CAR of any of the buckets in *result* then (first encounter) create a new cons cell (bucket) whose CAR is *element* and CDR is 1 (counter = 1) and append this bucket to the *result.* Then return the updated result.

So, your implementation “skeleton” may look something like this:

(defun frequency (l)

(frequency1 l ‘())

)

(defun frequency1 (l result)

(cond ((null l) l)

(t (frequency1 (cdr l) (*“call function below to update* ***result*** *with (****car l****)”*)))

)

)

(defun updateResult (element result)

... *(use CAAR and CDAR for testing if* element *is in a bucket and for accessing the counter)*

)

The last function (*updateResult*) when invoked, would work as shown in the examples below:

>(updateResult 3 ‘((1 . 3) (alpha . 4) (22 . 2)))

((1 . 3) (alpha . 4) (22 . 2) (3 . 1))

>(updateResult 3 ‘((2 . 5) (beta . 1) (3 . 6) (jokes . 5)))

((2 . 5) (beta . 1) (3 . 7) (jokes . 5))

***Note***: This is not a case where you want to use the actual BUCKET SORT. You may have symbols instead of numbers. Sorting is not part of this assignment.

Problem #4

To simplify things, you may use the “*flatten*” function covered in class and then use the implementation from Problem #3. (**3 extra points if you don’t use “flatten”**).