# Accumulating Parameters: A Technique in Functional Programming

We introduce an interesting programming technique often used in functional programming. The basic idea is to define a subsidiary function with an extra parameter which is used to accumulate the required result.

## **Example 1**

As the first example, consider the problem of reversing a list. Now we define a function rev(x,y) where x is the given list and y holds the intermediate and final reversed list.

(defun rev (x y)  
 (if (null x)  
 y  
 (rev (cdr x) (cons (car x) y))  
 )  
)

Now the function reverse(x) can be defined as:

(defun reverse0 (x)  
 (rev x nil)  
)

## **Example 2**

The following example shows how more than one result may be accumulated. Given a list L of integers, we want to generate a pair (A B) where A is the sum of integers in L and B the product.

(defun sumprod (L)  
 (sp L 0 1)   
)   
; we want a subsidiary function sp where the second and the third parameters  
; are accumulators starting from the identities for + and \*.  
  
  
(defun sp (L S P)  
 (if (null L)  
 (cons S (cons P nil))  
 (sp (cdr L) (+ (car L) S) (\* (car L) P))  
 )  
)

## **Example 3**

In general, one can always place the selected results, from a given list, to a resulting list by accumulation.

Example: Select from a list L the numbers greater than or equal to N E.g.

(select '(3 6 8 4) 4) ==> (6 8 4)

Standard definition:

(defun select (L N)   
 (if (null L)  
 nil  
 (if (>= (car L) N)  
 (cons (car L) (select (cdr L) N))  
 (select (cdr L) N)  
 )  
 )  
)

Use an accumulator:

(defun select0 (L N)   
 (xselect L N nil)  
)  
  
(defun xselect (L N AC) ; AC is an accumulator  
 (if (null L)  
 AC  
 (if (>= (car L) N)  
 (xselect (cdr L) N (cons (car L) AC))  
 (xselect (cdr L) N AC)  
 )  
 )  
)