***PLF WRITTEN 8 IANUARIE 2020 EX1***

***A close up of a paper

Description automatically generated***

(DEFUN F(L1 L2)

(APPEND **(F (CAR L1) L2** )

(COND

((NULL L1) (CDR L2))

(T (LIST **(F (CAR L1) L2)** (CAR L2)))

)

)

)

(defun f(l1 l2)

(lambda (x)

(append x

(cond

((null l1) (cdr l2))

(t (list x (car l2)))

)

)

(f(car l1) l2)

)

)

;In the original function , (f (car l1) l2) is computed twice , once in append and once in list. This duplicates the effort. In my solution I introduce lamba(x) to encapsulate the logic , ensuring the recursive computation (f (car l1) l2) happens only once and can be reused.This lambda function takes as parameter x , representing the result of the recursive call (f (car l1) l2)

A close up of a paper

Description automatically generated

;A nonlinear list is given.Write a LISP function to remove all numeric atoms divisible with 3 from all

;levels.Use a MAP function.Write the mathematical model and the meaning of all parameters for

;each function used.

;Ex: a) (1,(2 A (3 A)) (6)) == >(1,(2 A (A))NIL)

;removeNumericDivisibleWith3 - a function which eliminates all numeric atoms divisible with 3

;mathematical model

;removeNumericDivisibleWith3(l1l2...ln) = removeNumericDivisibleWith3(l2l3...ln) , l1 is number and divisible with 3

; removeNumericDivisibleWith3(l1) U removeNumericDivisibleWith3(l2l3...ln) , l1 is list

; l1 U removeNumericDivisibleWith3(l2l3...ln) otherwise

(defun removeNumericDivisibleWith3(l)

(mapcar

(lambda(x)

(cond

((and (numberp x) (= 0 (mod x 3))) nil)

((listp x) (removeNumericDivisibleWith3 x))

(t x)

)

)l

)

)

(print (removeNumericDivisibleWith3 '(1 (2 A (3 A)) (6))))

A close up of a book

Description automatically generated

(defun replaceNodesFromOddLevels(l e level)

(mapcar

(lambda(x)

(cond

((listp x) (replaceNodesFromOddLevels x e (+ 1 level)))

((= (mod level 2) 1) e)

(t x)

)

)

l

)

)

(print (replaceNodesFromOddLevels '(a (b (g)) (c (d (e)) (f))) 'h 0))

A close up of a paper

Description automatically generated

(defun replaceAtoms(l level k)

(mapcar

(lambda (x)

(cond

(( listp x) (replaceAtoms x (+ level 1)))

((= level k) 0)

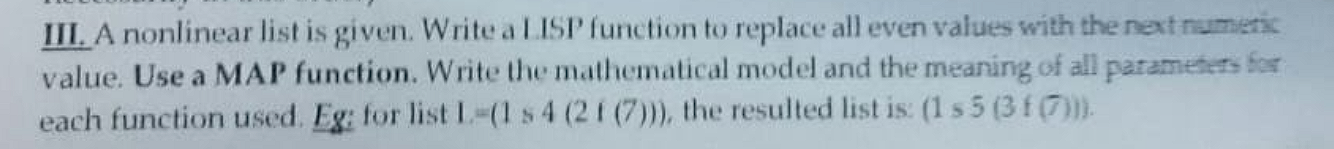
(t x)

)

)

)

)



(defun replaceEvenValuesWithNext(l)

(mapcar

(lambda (x)

(cond

((listp x) (replaceEvenValuesWithNext x))

((and (numberp x) (= 0 (mod x 2))) (+ 1 x))

(t x)

)

)

l

)

)

(print (replaceEvenValuesWithNext '(10 2 7 7 (a b 6))))

A text on a page

Description automatically generated

(defun substituteElementAtOddLevels(l e e1 level)

(mapcar

(lambda(x)

(cond

((listp x) (substituteElementAtOddLevels x e e1 (+ 1 level)))

((and (atom x) (equal x e) (= 1 (mod level 2))) e1)

(t x)

)

)

l

)

)

(print (substituteElementAtOddLevels '(1 d (2 d (d))) 'd 'f 1))

A close up of a paper

Description automatically generated

(defun multiplyNumberWithLevel(l level)

(mapcar

(lambda(x)

(cond

((listp x) (multiplyNumberWithLevel x (+ 1 level)))

((numberp x) (\* x level))

(t x)

)

)

l

))

(print (multiplyNumberWithLevel '(1(2)(3(4(5)))) 1))

(print (multiplyNumberWithLevel '(1 (2 (A (B))) (3 (4 (C (5))))) 1))

A close up of a paper

Description automatically generated

RESPONSE:

The function mapcon applies the function list to each element from the list L and then concatenates the results. The result of the evaluation is ((1) (2)) .Next we have the apply function which will take every element from the list ((1) (2)) and concatenates the two sublists using append resulting in (1 2).

A close up of a paper

Description automatically generated

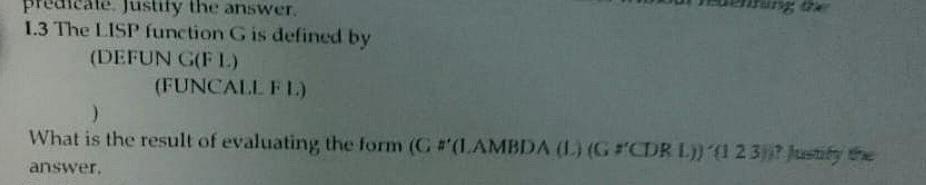
SOLUTION

The result of evaluating the form (FUNCALL P’ (A B C)) will be (A A). By calling funcall p we are invoking the function g, which is stored in p , having as parameter the list (A B C).The execution of the function will return a list containing the first element twice , in our case A. Symbol p will contain the symbol g but not the function itself because in the first setq we have the ‘g which will set q to correspond to the symbol g not the function.

A close up of a paper

Description automatically generated

The result of the evaluation will be (1 2 3 4 5 7). The function F takes as parameters x and y , y containing a flexible number of inputs , all saved in a list. The first call of the function will return (1 2) because Y is null since (1 2 ) will match x and there is no rest.For the second call x will contain (3 4) and and y = ((5 6) (7 8)). This call will return (3 4 5 7) because of the function append between x(1 2) and mapcar #’ car y which will take the first element of every element in Y.



The result of the form will be (2 3).In the form we are calling the function g with the function lambda and the list (1 2 3).Lambda function will invoke the function g with the parameters cdr as the function and the list l.In our first call of g we call the function lambda with the initial list as parameter. This calls again the function g but with cdr as function parameter. When funcall is called cdr l is executed and this will return (2 3).

A close up of a paper

Description automatically generated

(defun Fct(F L)

(lambda(x)

(cond

((null l) nil)

((x) (cons x (Fct F (cdr l))))

(t nil)

)

(funcall f (car l))

)

)

;The solution for avoiding the double call (funcall f (car l)) is defining a function lambda to encapsulate the logic and make sure the (funcall f (car l) is not computed twice during the execution and the result can be reusable.This lambda function takes as parameter an x representing the result of the call (funcall f (car l)).

A close up of a paper

Description automatically generated

(defun F(L)

(lambda(x)

(cond

((null l) 0)

((> x 2) (+ (car l) (f (cdr l)))

(t x)

)

(f (car l)))

):To avoid the double recursive function call (F (CAR L)) I use a lambda function which encapsulates the logic and avoids redundant recursive calls. This lambda function defines the recursion , ensuring that intermediate results are computed once and reused , preventing double computations of the same results. This approach optimizes ensuring that the recursion depth is controller and efficient.

A close up of a paper

Description automatically generated

(defun f(l)

(lambda(x)

(cond

((atom l) -1)

((> x 0) (+ (car l) x (f (cdr l))))

(t (f (cdr l)))

)

(f (car l))

)

)

JUSTIFICATION:

-For avoiding the double recursive call (f(car l)) I am using an lambda function which encapsulates the logic and avoids redundant recursive calls. This lambda function defines recursion , ensuring that intermediate results are compiuted only once and can be reused. This approach optimizes eusuring that the recursion depth is efficient and controller. A close up of a paper

Description automatically generated

(defun f(l)

(lambda(x)

(cond

((null l) nil)

((listp (car l)) (append x (f (cdr l)) (car x)))

(t (list (car l)))

)

(f (car l))

)

)

JUSTIFICATION:  
-For avoiding the double recursive call (f (car l)) we are using a lambda function to encapsulate the logic and avoid redundant recursive calls. The lambda function takes as parameter the result of (f(car l)) , ensuring that the computation is performed only once and making the result reusable along the code. This approach is more efficient because it maintains the depth of the recursion controlled and also keeps the value of the result (f(car l)) making it reusable along all code.