A1. Grading: (def) – 1p; (1) – 3p (4x0.75p); (2) – 3p; (3) – 3p.

The Prolog problems *will be solved in SWI Prolog. You will explain the code, give the reasoning, predicates specification including recursive formula, flow model, meaning of all variables and parameters.* ***The Lisp problems*** *will be solved in Common Lisp. You will explain the code, give the reasoning, functions specification, meaning of all variables and parameters, the formula for recursion. The MAP problem implies writing a main and an auxiliary function. For a penalty, this may be solved without using MAP functions.*

I.

I.1 Consider the following function definition in LISP

```
    (DEFUN Fct (F L)
        (COND
            ((NULL L) NIL)
            ((FUNCALL F (CAR L)) (CONS (FUNCALL F (CAR L)) (Fct F (CDR L))))
            (T NIL)  ((lambda (v) (cond ((v) (cons v (Fct F (CDR L))))) (
                                                          funcall F(car
        )
    )
```

Rewrite it in order to have only one recursive call (FUNCALL F (CAR L)). Do not create global variables. Do not write a new subalgorithm to achieve the same thing. Justify the answer.

I.2 Let L be a numerical list and consider the following PROLOG definition for the predicate **f(list, integer)**, with the flow model (i, o):

```
    f([], -1).
    ~~f([H|T],S) :- H>0, f(T,S1),S1<H,!,S is H.~~   aux(S1,S,H).
    f([H|T],S) :- f(T,S1), ~~S is S1.~~   aux(S1,S,H).
```

```
aux(S1, S, H) :-          aux(S1, S, _) :-
    H>0,                      S is S1.
    S1 < H,
    !,
    S is H.
```

Rewrite the predicate in order to have only one recursive call f(T,S1) in all clauses. Do not write a new predicate to achieve the same thing. Justify the answer.

I.3 The LISP function G is defined by (DEFUN G(L) (LIST (CAR L) (CAR L))). In order to rename the function G we execute (SETQ Q 'G) followed by (SETQ P Q). What is the result of evaluating the form (FUNCALL P '(A B C))? Justify the answer. (A A)

I.4 Consider the PROLOG predicate **f(list, integer)** with the flow model **(i, o)**.

```
    f([],0).                        these should be swapped
    f([H|T],S) :- f(T,S1), S1 is S-H.   → res = - Sum = -36.
```

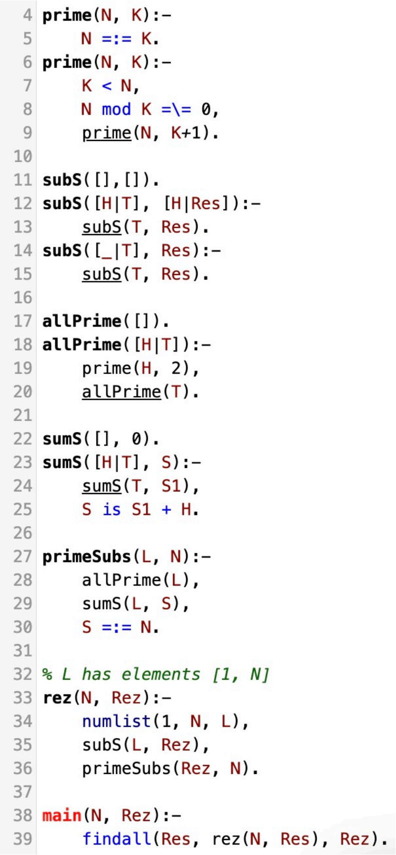
Give the result of the evaluation **f([1,2,3,4,5,6,7,8], S)**? Justify the answer.

II. For any given positive natural number N, generate all the sets of prime numbers that add up to N (including N, if applicable). Write the mathematical model, flow model and the meaning of all variables for each predicate used. Ex: for N = 18 => [[2,3,13], [2,5,11], [5,13], [7,11]].

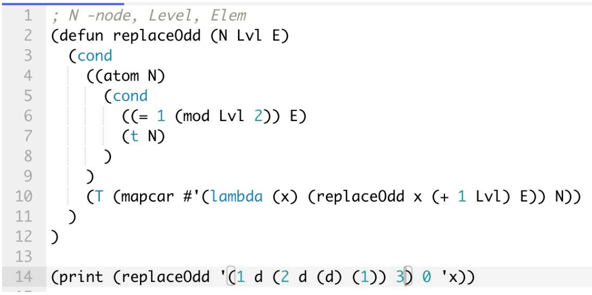
III. An n-ary tree is represented in LISP: (root subtree1 subtree2 ... ...). Write a function to replace all nodes from the odd levels in the tree with a given value **e**. The level of root is considered 0. **Use a MAP function.** Write the mathematical model and the meaning of all parameters for each function used. *Eg:* for tree (a (b (g)) (c (d (e)) (f))) şi e=h => (a (h (g)) (h (d (h)) (h)))

```
f([1,2,3,4,5,6,7,8], S)    [8], S
      S1 = S-H
f([8], S)
      f([], 0),
      S1 is S-H
      0
```

II.



```
prime(N, K):-
    N =:= K.
prime(N, K):-
    K < N,
    N mod K =\= 0,
    prime(N, K+1).

subS([],[]).
subS([H|T], [H|Res]):-
    subS(T, Res).
subS([_|T], Res):-
    subS(T, Res).

allPrime([]).
allPrime([H|T]):-
    prime(H, 2),
    allPrime(T).

sumS([], 0).
sumS([H|T], S):-
    sumS(T, S1),
    S is S1 + H.

primeSubs(L, N):-
    allPrime(L),
    sumS(L, S),
    S =:= N.

% L has elements [1, N]
rez(N, Rez):-
    numlist(1, N, L),
    subS(L, Rez),
    primeSubs(Rez, N).

main(N, Rez):-
    findall(Res, rez(N, Res), Rez).
```



```
; N -node, Level, Elem
(defun replaceOdd (N Lvl E)
  (cond
    ((atom N)
      (cond
        ((= 1 (mod Lvl 2)) E)
        (t N)
      )
    )
    (T (mapcar #'(lambda (x) (replaceOdd x (+ 1 Lvl) E)) N))
  )
)

(print (replaceOdd '(1 d (2 d (d) (1)) 3) 0 'x))
```