B1. 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 F (L)   ((lambda (v)
        (COND
            ((NULL L) NIL)
            ((LISTP (CAR L)) (APPEND (F (CAR L)) (F (CDR L)) (CAR (F (CAR L)))))
            (T (LIST(CAR L)))
        )) (F(car L)))
    )
```

$\overset{v}{\overbrace{\text{(F (CAR L))}}}$ $\overset{v}{\overbrace{\text{(F (CAR L))}}}$

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

I.2 Consider the following PROLOG definition for the predicate **f(integer, integer)**, with the flow model (i, o):

```
    f(0, 0) :- !.
    f(I,Y) :- J is I-1, f(J,V), V>1, !, K is I-2, Y is K.
    f(I,Y) :- J is I-1, f(J,V), Y is V+1.
```

$\overbrace{\text{f(J,V), V>1, !, K is I-2,}}^{\text{aux(V,K,Y,I).}}$

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

I.3 The LISP function F is defined by

```
    (DEFUN F (X &OPTIONAL Y)
        (COND
            ((NULL Y) (CDR X))
            (T (CONS (CAR X) Y))
        )
    )
```

append((2) (3 5 6)) $\rightarrow$ (2 3 5 6)

What is the result of evaluating the form (APPEND (F '(1 2)) (F '(3 4) '(5 6)))? Justify the answer.

I.4 Consider the PROLOG predicate **p(integer)** with the flow model **(i)**

```
    p(100).
    p(N) :- write(N), N1 is N-1, p(N1).
```

Give the effect of the following goal: **p(0).** Justify the answer.

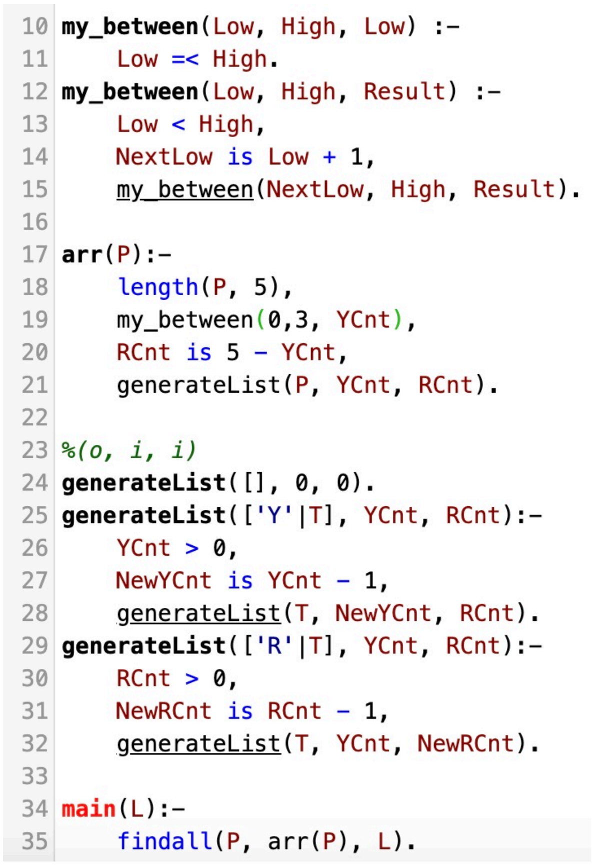
II. Chairs must be arranged for a show. There are red chairs and yellow chairs. One row contains 5 chairs. Find all the possible arrangements of chairs on a row, knowing that there can be at most 3 yellow chairs on a row. Write the mathematical model, flow model and the meaning of all variables for each predicate used. Some elements in the solution: ['Y', 'R', 'R', 'R', 'R'], ['R', 'Y', 'Y', 'R', 'R'], ['Y', 'Y', 'Y', 'R', 'R'], ['R', 'Y', 'Y', 'R', 'Y'], etc.

III. Write a LISP function to substitute an element **e** with another element **e1** at any odd level from a nonlinear list (The superficial level is considered 1). **Use a MAP function.** Write the mathematical model and the meaning of all parameters for each function used. Eg: for list (1 d (2 d (d))), **e**=d and **e1**=f the resulted list is (1 f (2 d (f))).

p(0) out 0-1-2-3 ....

never ending because we have to end clause to match for negative nums. this would happen for any p(N), N < 100.

ii.

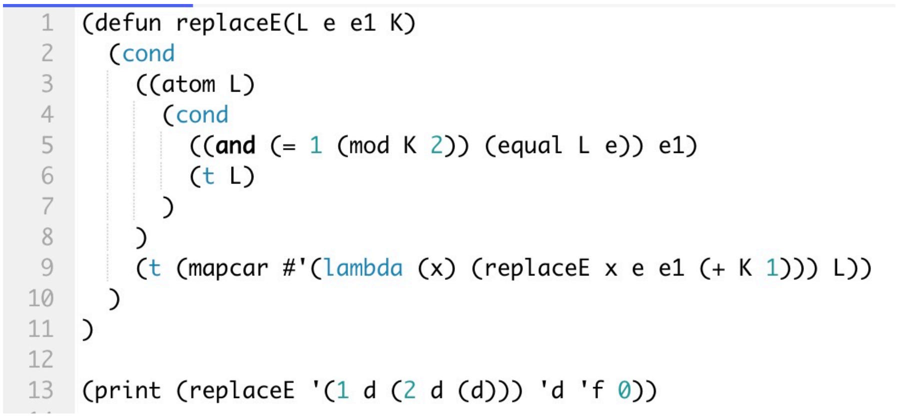


```prolog
my_between(Low, High, Low) :-
    Low =< High.
my_between(Low, High, Result) :-
    Low < High,
    NextLow is Low + 1,
    my_between(NextLow, High, Result).

arr(P):-
    length(P, 5),
    my_between(0,3, YCnt),
    RCnt is 5 - YCnt,
    generateList(P, YCnt, RCnt).

%(o, i, i)
generateList([], 0, 0).
generateList(['Y'|T], YCnt, RCnt):-
    YCnt > 0,
    NewYCnt is YCnt - 1,
    generateList(T, NewYCnt, RCnt).
generateList(['R'|T], YCnt, RCnt):-
    RCnt > 0,
    NewRCnt is RCnt - 1,
    generateList(T, YCnt, NewRCnt).

main(L):-
    findall(P, arr(P), L).
```



```lisp
(defun replaceE(L e e1 K)
  (cond
    ((atom L)
      (cond
        ((and (= 1 (mod K 2)) (equal L e)) e1)
        (t L)
      )
    )
    (t (mapcar #'(lambda (x) (replaceE x e e1 (+ K 1))) L))
  )
)

(print (replaceE '(1 d (2 d (d))) 'd 'f 0))
```