A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((NULL L) 0)

((> (F (CAR L)) 2) (+ (CAR L) (F (CDR L))))

(T (F (CAR L)))

)

)

Rewrite the definition in order to avoid the double recursive call (F (CAR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer.

(DEFUN F(L)

(

( LAMBDA (V)

(COND

((NULL L) 0)

((> V 2) (+ (CAR L) (F (CDR L))))

(T V)

)

)

(F (CAR L))

)

)

A. The following function definition in LISP is given

(DEFUN F(N)

(COND

((= N 1) 1)

(> (F (- N 1)) 2) (- N 2))

(> (F (- N 1)) 1) (F (- N 1)))

(T (- (F (- N 1)) 1))

)

)

Rewrite the definition in order to avoid the repeated call (F (- N 1)). Do NOT redefine the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(N)

(

( LAMBDA (V)

(COND

((= N 1) 1)

(> V 2) (- N 2))

(> V 1) V)

(T (- V 1))

)

) (F (- N 1))

)

)

**Pentru a evita apelul recursive, se va utiliza o functie lambda care e evaluate ca valoarea returnata de functie**

**The solution to avoid the duplicate call (f l) is to use an anonymous function locally, which can be called with the actual parameter (f l). That is why we will use lambda expressions.**

A. Let L be a list of numbers and given the following PROLOG predicate definition f(list, integer),

with the flow model (i, o):

f([], 0).

f([H|T],S):-f(T,S1),S1<H,!,S is H.

f([\_|T],S):-f(T,S1),S is S1.

Rewrite the definition in order to avoid the recursive call f(T,S) in both clauses. Do NOT redefine

the predicate. Justify your answer

f([], 0).

f([H|T],S):-

f(T,S1),

aux(H,S1,S)

Aux(H,X,H):- X<H, !.

Aux(\_,X,X).

We define an auxiliary predicate which has the same parameters as function f plus a parameter which represents the value of the predicate (T,S1).

A. The following function definition in LISP is given

(DEFUN F(G L)

(COND

((NULL L) NIL)

(> (FUNCALL G L) 0) (CONS (FUNCALL G L) (F (CDR L))))

(T (FUNCALL G L))

)

)

Rewrite the definition in order to avoid the repeated call (FUNCALL G L). Do NOT redefine the

function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(G L)

( (LAMBDA (V)

(COND

((NULL L) NIL)

(> V 0) (CONS V (F (CDR L))))

(T V)

)

) (FUNCALL G L)

)

)

A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((NULL L) NIL)

((LISTP (CAR L)) (APPEND (F (CAR L)) (F (CDR L)) (CAR (F (CAR L)))))

(T (LIST(CAR L)))

)

)

Rewrite the definition in order to avoid the double recursive call (F (CAR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answe

(DEFUN F(L)

(

(LAMBDA (V)

(COND

((NULL L) NIL)

((LISTP (CAR L)) (APPEND V (F (CDR L)) (CAR V)))

(T (LIST(CAR L)))

)

) (F (CAR L))

)

)

A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((ATOM L) -1)

((> (F (CAR L)) 0) (+ (CAR L) (F (CAR L)) (F (CDR L))))

(T (F (CDR L)))

)

)

Rewrite the definition in order to avoid the double recursive call (F (CAR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(L)

(

(LAMBDA (V)

(COND

((ATOM L) -1)

((> V 0) (+ (CAR L) V (F (CDR L))))

(T (F (CDR L)))

)

) (F (CAR L))

)

)

A. Let L be a list of numbers and given the following PROLOG predicate definition with flow

model (i, o):

f([],0).

f([H|T],S):-f(T,S1),S1>=2,!,S is S1+H.

f([\_|T],S):-f(T,S1),S is S1+1.

Rewrite the definition in order to avoid the recursive call f(T,S) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff([],0).

ff([H|T],S):-

ff(T,S1),

aux([H|T],S1,S).

aux([H|\_],S1,S):-

S1>=2,

!,

S is S1+H.

aux([\_|\_],S1,S):-

S is S1+1.

A. Let L be a list of numbers and given the following PROLOG predicate definition f(list,

integer), with the flow model (i, o):

f([], 0).

f([H|T],S):-f(T,S1),H<S1,!,S is H+S1.

f([\_|T],S):-f(T,S1), S is S1+2.

Rewrite the definition in order to avoid the recursive call f(T,S) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff([], 0).

ff([H|T],S):-

ff(T,S1),

aux([H|T], S1,S).

aux([H|\_],S1,S):-

H<S1,

!,

S is H+S1.

aux([\_|\_],S1,S):-

S is S1+2.

A. Given the following PROLOG predicate definition f(integer, integer), with the flow model (i,

o):

f(100, 0):-!.

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

f(I,Y):-J is I+1, f(J,V), Y is V+1.

Rewrite the definition in order to avoid the recursive call f(J,V) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff(100, 0):-!.

ff(I,Y):-

J is I+1,

ff(J,V),

aux(I,V,Y).

aux(I,V,Y):-

V>2,

!,

K is I-2,

Y is K+V-1.

aux(\_, V, Y):-

Y is V+1.

A. Let L be a list of numbers and given the following PROLOG predicate definition f(list, integer),

with the flow model (i, o):

f([], 0).

f([H|T],S):-f(T,S1),S1<H,!,S is H.

f([\_|T],S):-f(T,S1),S is S1.

Rewrite the definition in order to avoid the recursive call f(T,S) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff([], 0).

ff([H|T],S):-

ff(T,S1),

aux([H|T],S1,S).

aux([H|\_],S1,S):-

S1<H,

!,

S is H.

aux([\_|\_],S1,S):-

S is S1.

A.

(DEFUN F(L)

(COND

((NULL L) 0)

((> (CAR L) 0)

(COND

((> (CAR L) (F (CDR L))) (CAR L))

(T (F (CDR L)))

)

(T (F (CDR L)))

)

)------------------------------

(DEFUN F(L)

( (LAMBDA (V)

(COND

((NULL L) 0)

((> (CAR L) 0)

(COND

((> (CAR L) V) (CAR L))

(T V)

)

(T V)

)

) (F (CDR L))

)

)

A. Let G be LISP function and given the following definition

(DEFUN F(L)

(COND

((NULL L) 0)

(> (G L) 2) (+(G L) (F (CDR L))))

(T (G L))

)

)

Rewrite the definition in order to avoid the repeated call (G L). Do NOT redefine the function. Do

NOT use SET, SETQ, SETF.Justify your answer

(DEFUN F(L)

( (LAMBDA (V)

(COND

((NULL L) 0)

(> V 2) (+V (F (CDR L))))

(T V)

)

) (G L)

)

)

A. Let L be a list of numbers and given the following PROLOG predicate definition with flow

model (i, o):

f([],0).

f([H|T],S):-f(T,S1),S1>=2,!,S is S1+H.

f([\_|T],S):-f(T,S1),S is S1+1.

Rewrite the definition in order to avoid the recursive call f(T,S) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff([],0).

ff([H|T],S):-

ff(T,S1),

aux([H|T],S1,S).

aux([H|\_],S1,S):-

S1>=2,

!,

S is S1+H.

aux([\_|\_], S1,S):-

S is S1+1.

A. Given the following PROLOG predicate definition f(integer, integer), with the flow model (i,

o):

f(100, 0):-!.

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

f(I,Y):-J is I+1, f(J,V), Y is V+1.

Rewrite the definition in order to avoid the recursive call f(J,V) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff(100, 0):-!.

ff(I,Y):-

J is I+1,

ff(J,V),

aux(I,V,Y).

aux(I,V,Y):-

V>2,

!,

K is I-2,

Y is K+V-1.

aux(\_,V,Y):-

Y is V+1.

A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((NULL L) NIL)

(> (F (CAR L)) 0) (CONS (F (CAR L)) (F (CDR L))))

(T (F (CAR L)))

)

)

Rewrite the definition in order to avoid the repeated recursive call (F (CAR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(L)

( (LAMBDA (V)

(COND

((NULL L) NIL)

(> V 0) (CONS V (F (CDR L))))

(T V)

)

) (F (CAR L))

)

)

A. Given the following PROLOG predicate definition 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.

Rewrite the definition in order to avoid the recursive call f(J,V) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff(0, 0):-!.

ff(I,Y):-

J is I-1,

ff(J,V),

aux(I,V,Y).

aux(I,V,Y):-

V>1,

!,

K is I-2,

Y is K.

aux(\_,V,Y):-

Y is V+1.

A. The following function definition in LISP is given

(DEFUN F(N)

(COND

((= N 0) 0)

(> (F (- N 1)) 1) (- N 2))

(T (+ (F (- N 1)) 1))

))

Rewrite the definition in order to avoid the double recursive call (F (- N 1)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(N)

( (LAMBDA (V)

(COND

((= N 0) 0)

(> V 1) (- N 2))

(T (+ V 1))

)

) (F (- N 1))

)

)

A. The following function definition in LISP is given

(DEFUN Fct(F L)

(COND

((NULL L) NIL)

((FUNCALL F (CAR L)) (CONS (FUNCALL F (CAR L)) (Fct F (CDR L))))

(T NIL)

)

)

Rewrite the definition in order to avoid the double recursive call (FUNCALL F (CAR L)). Do NOT

redefine the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN Fct(F L)

( (LAMBDA (V)

(COND

((NULL L) NIL)

(V) (CONS V (Fct F (CDR L))))

(T NIL)

)

) (FUNCALL F (CAR L))))

A. Let L be a list of numbers and given the following PROLOG predicate definition 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.

f([\_|T],S):-f(T,S1), S is S1.

Rewrite the definition in order to avoid the recursive call f(T,S) in both clauses. Do NOT redefine

the predicate. Justify your answer

ff([], -1).

ff([H|T],S):-

H>0,

ff(T,S1),

aux([H|T],S1,S).

aux([H|\_],S1,S):-

S1<H,

!,

S is H.

aux([\_|\_],S1,S):-

S is S1.

A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((NULL L) 0)

(> (F (CDR L)) 2) (+ (F (CDR L)) (CAR L)))

(T (+ (F (CDR L)) 1))

)

)

Rewrite the definition in order to avoid the repeated recursive call (F (CDR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(L)

( (LAMBDA (V)

(COND

((NULL L) 0)

(> V 2) (+ V (CAR L)))

(T (+ V 1))

)

) (F (CDR L))

)

)

A. Given the following PROLOG predicate definition f(integer, integer), with the flow model (i,

o):

f(0, -1):-!.

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

f(I,Y):-J is I-1, f(J,V), Y is V+I.

Rewrite the definition in order to avoid the recursive call f(J,V) in both clauses. Do NOT redefine

the predicate. Justify your answer.

ff(0, -1):-!.

ff(I,Y):-

J is I-1,

ff(J,V),

aux(J,V,Y).

aux(J,V,Y):-

V>0,

!,

K is J,

Y is K+V.

aux(J,V,Y):-

Y is V+J+1.

A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((NULL L) NIL)

(> (F (CAR L)) 0) (CONS (F (CAR L)) (F (CDR L))))

(T (F (CAR L)))

)

)

Rewrite the definition in order to avoid the repeated recursive call (F (CAR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(L)

( (LAMBDA (V)

(COND

((NULL L) NIL)

(> V 0) (CONS V (F (CDR L))))

(T V)

)

) (F (CAR L))

)

)

A. Given the following PROLOG predicate definition f(list, integer), with the flow model (i, o):

f([], -1):-!.

f([\_|T], Rez):- f(T,S), S<1, !, Y is S+2.

f([H|T], Rez):- f(T,S), S<0, !, Y is S+H.

f([\_|T], Rez):- f(T,S), Y is S.

Rewrite the definition in order to avoid the recursive call f(T,S) in all clauses. Do NOT redefine

the predicate. Justify your answer

f([], -1):-!.

f([\_|T], Y):-

f(T,S),

S<1,

!,

Y is S+2.

f([H|T], Y):-

f(T,S),

S<0,

!,

Y is S+H.

f([\_|T], Y):-

f(T,S),

Y is S.

%--------------------

f([],-1).

f([H|T],Y):-

f(T,S),

aux(S,Y,H).

aux(S,Y,\_):-

S<1, !, Y is S+2.

aux(S,Y,H):-

S<0, !, Y is S+H.

aux(S,Y,\_):-

Y is S.

A. Given the following PROLOG predicate definition f(integer, integer), with the flow model (i,

o):

f(20, -1):-!.

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

f(I,Y):-J is I+1, f(J,V), Y is V-1.

Rewrite the definition in order to avoid the recursive call f(J,V) in both clauses. Do NOT redefine

the predicate. Justify your answer

f(20, -1):-!.

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

f(I,Y):-J is I+1, f(J,V), Y is V-1.

%----------------

ff(20, -1):-!.

ff(I,Y):-J is I+1, ff(J,V), aux(J,V,Y).

aux(J,V,Y):-V>0, !, K is J, Y is K.

aux(\_,V,Y):- Y is V-1.

A. Given the following PROLOG predicate definition f(integer, integer), with the flow model (i,

o):

f(100, 1):-!.

f(K,X):-K1 is K+1, f(K1,Y), Y>1, !, K2 is K1-1, X is K2+Y.

f(K,X):-K1 is K+1, f(K1,Y), Y>0.5, !, X is Y.

f(K,X):-K1 is K+1, f(K1,Y), X is Y-K1.

Rewrite the definition in order to avoid the recursive call f(J,V) in all clauses. Do NOT redefine

the predicate. Justify your answer.

f(100, 1):-!.

f(K,X):-K1 is K+1, f(K1,Y), Y>1, !, K2 is K1-1, X is K2+Y.

f(K,X):-K1 is K+1, f(K1,Y), Y>0.5, !, X is Y.

f(K,X):-K1 is K+1, f(K1,Y), X is Y-K1.

%--------------------------------

ff(100, 1):-!.

ff(K,X):-K1 is K+1, ff(K1,Y), aux(K1,Y,X).

aux(K1,Y,X):-

Y>1, !, K2 is K1-1, X is K2+Y.

aux(\_,Y,X):-

Y>0.5, !, X is Y.

aux(K1,Y,X):-

X is Y-K1.

A. Let L be a list of numbers and given the following PROLOG predicate definition with flow

model (i, o):

f([],-1).

f([H|T],S):-f(T,S1), S1<1, S is S1-H, !.

f([\_|T],S):-f(T,S).

Rewrite the definition in order to avoid the recursive call f(T,S) in both clauses. Do NOT redefine

the predicate. Justify your answer

f([],-1).

f([H|T],S):-f(T,S1), S1<1, S is S1-H, !.

f([\_|T],S):-f(T,S).

%------------

ff([],-1).

ff([H|T],S):-ff(T,S1), aux([H|T],S1,S).

aux([H|\_],S1,S):-S1<1, S is S1-H, !.

A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((NULL L) 0)

((> (F (CAR L)) 1) (F (CDR L)))

(T (+ (F (CAR L)) (F (CDR L))))

)

)

Rewrite the definition in order to avoid the double recursive call (F (CAR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer

(DEFUN F(L)

( (LAMBDA (V)

(COND

((NULL L) 0)

((> V 1) (F (CDR L)))

(T (+ V (F (CDR L))))

)

) (F (CAR L))

)

)

A. The following function definition in LISP is given

(DEFUN F(L)

(COND

((NULL L) 0)

(> (F (CAR L)) 2) (+ (F (CDR L)) (F(CAR L))))

(T (+ (F (CAR L)) 1))

)

)

Rewrite the definition in order to avoid the repeated recursive call (F (CAR L)). Do NOT redefine

the function. Do NOT use SET, SETQ, SETF. Justify your answer.

(DEFUN F(L)

( (LAMBDA (V)

(COND

((NULL L) 0)

(> V 2) (+ (F (CDR L)) V))

(T (+ V 1))

)

) (F (CAR L))

)

)

**A.** The following function definition in LISP is given

(DEFUN F(L1 L2)

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

(COND

((NULL L1) (CDR L2))

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

)

)

)

Rewrite the definition in order to avoid the double recursive call **(F (CAR L1) L2)**. Do NOT

redefine the function. Do NOT use SET, SETQ, SETF. Justify your answer.0

(DEFUN F(L1 L2)

( (LAMBDA (V)

(APPEND V

(COND

((NULL L1) (CDR L2))

(T (LIST V (CAR L2)))

)

)

) (F (CAR L1) L2)

)

)

**A.** Given the following PROLOG predicate definition **f(integer, integer),** with the flow model (i,

o):

f(1, 1):-!.

f(K,X):-K1 is K-1, **f(K1,Y)**, Y>1, !, K2 is K1-1, X is K2.

f(K,X):-K1 is K-1, **f(K1,Y)**, Y>0.5, !, X is Y.

f(K,X):-K1 is K-1, **f(K1,Y)**, X is Y-1.

Rewrite the definition in order to avoid the recursive call **f(J,V)** in all clauses. Do NOT redefine

the predicate. Justify your answer.

f(1, 1):-!.

f(K,X):-K1 is K-1, **f(K1,Y)**, Y>1, !, K2 is K1-1, X is K2.

f(K,X):-K1 is K-1, **f(K1,Y)**, Y>0.5, !, X is Y.

f(K,X):-K1 is K-1, **f(K1,Y)**, X is Y-1.

f(1, 1):- !

f(K, X):-

K1 is K-1

f(K1, Y),

aux(X, Y, K1)

aux(X, Y, K1):

Y>1, !, K2 is K1-1, X is K2

aux(X, Y, \_):

Y>0.5, !, X is Y.

aux(X, Y, \_):

X is Y-1.