COMP 348 Review

Prolog

- Find function of a program.
  Will give you a code, what does it do?
- Recursive function
- Recursive List
  Can give you a list and make it reverse && make its internal sublist reverse
- Cut-off, !. : two types in the exam: one with cond, one that doesn't affect
  Explain what's the benefit of it/its role?

Prolog Questions

1) Write down prolog that reverse list

```
rev( [], []).
rev( [H|T], L) :_ rev(T, RL), append(RL,[H], L).
```

***2) Write prolog program from list, it produces reverse list concatenated with original list**

Ex [1,2,3] -> [3,2,1,1,2,3]
[1,[2,4], 3] -> [3, [2,4], 1, 1, [2,4], 3]

```
rev+(L1,L2):- rev(L1, L3), append(L3, L1, L2).
```

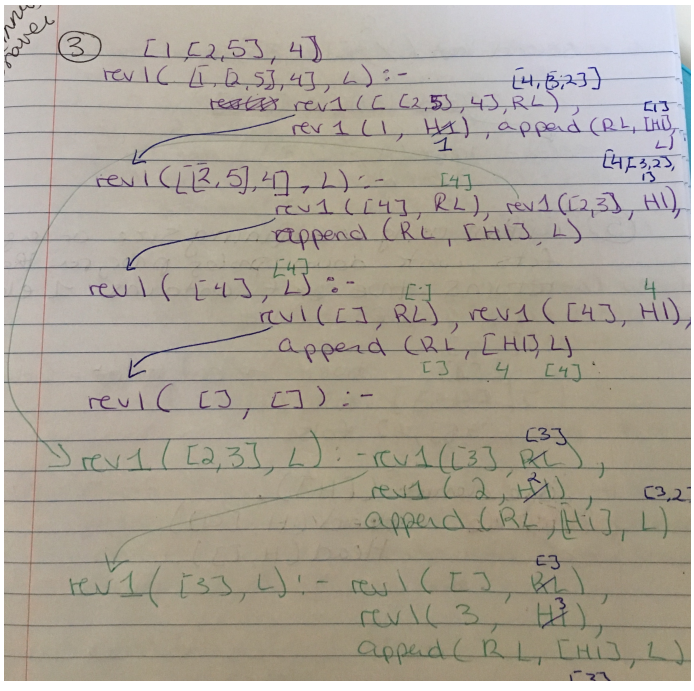
Need to know what does it do: explain and provide examples to support explanation

3) Write prolog program that reverse list but also its internal lists

Ex: [1,2,3] -> [3,2,1]
[1, 2, [3,8], 7] -> [7, [8,3], 2, 1]

```
rev1([], []).
rev1(X,X) :- X\==[_ , _].
rev1([H|T], L) :-
                rev1(T,RL),
                rev1(H,H1),
                append(RL, [H1],L).
```

tracing and backtracing--------->



4) Recursive prolog procedure 2nd-to-last to find the second to last element from a list.

ExL [1, 2, 3, 4, 5] returns 4

2nd-to-last(X, [X,_]).
3-last(X, [Y|Ys]): - 2nd-to-last(X, Ys).

5) without using or defining size or length function, write down prolog program that returns true if contains 1 element.

Ex: [1,2, 3, 8] false
[8] true
[[]] true
[[4, 5]] true
[[4,5], 4] false

oneelement([H]).

6) prolog program that returns true if a list is sorted. Suppose list ø contain lists.

Ex: [1,2,2,3, 8] true
[8, 4, 2] false
[7] true
[] true

isSorted([], []). *// empty*
isSorted[[X], [X]). *//same element*
isSorted([X, Y|L]):- X =>Y, isSorted([Y|L]).
//if head of list is bigger than head of tail

7) prolog procedure: series(N, Total) which sums up the series 1+ 1/2, + 1/3+1/4+...+1/(N-1).

sum([], 0).
sum([H|L], T) :- sum(L,T1); T is T1+H.

***8) Without using or defining member or functions, write recursive prolog procedure returns true only if elements of first list are contained in the second list in the same order**.

contains([], []) :-.
contains([], [_|_]) :-.
contains([X|T1], [X|T2]) :- contains (T1, T2).
contains([X|T1], [Y|T2]) :- X /==Y, contains ([X|T1], T2).

function of each statement; if we remove X /==Y what happens; the difference between two last statement, why we add last one.

Exercise 2

Exercise 2 (Prolog) (5P)

Write down a Prolog procedure having as input a list and returning true if the first element of the list is equal to the last element or if the list has only one element. The procedure should return false in the following cases: 1) the input is not a list; and 2) the input list is empty.

```
fela([X]):-!.  (1P)
fela([X,X]):-!.  (1P)
fela([X,Y|T]):- fela([X|T]).  (3P)
```

Exercice 3

Exercise 3 (Prolog) (4P)

Write down a Prolog program (we call it square) that for each element of the list, it replaces it by its square. For example, if we ask Prolog: ?- square([2, 8], L). The answer would be L= [4, 64].
You can assume that the list elements are always numbers and the list does not contain other lists. If the list is empty, the program should return false.

```
square([X],[X1]):- X1 is X * X, !. (1P)
square([X|T], [X1|T1]):- X1 is X * X, square(T, T1). (3P)
```

Exercise 5

Exercise 5 (Prolog) (3P)

Write down a Prolog program **nullist** that takes as input a list of numbers and succeeds if the list contains only 0s.

Examples

```
?- nullist([0,0,0,0]).
Yes
?- nullist([0,1,0,0]).
No
?- nullist([]).
No
?- nullist([3,2,5]).
No
```

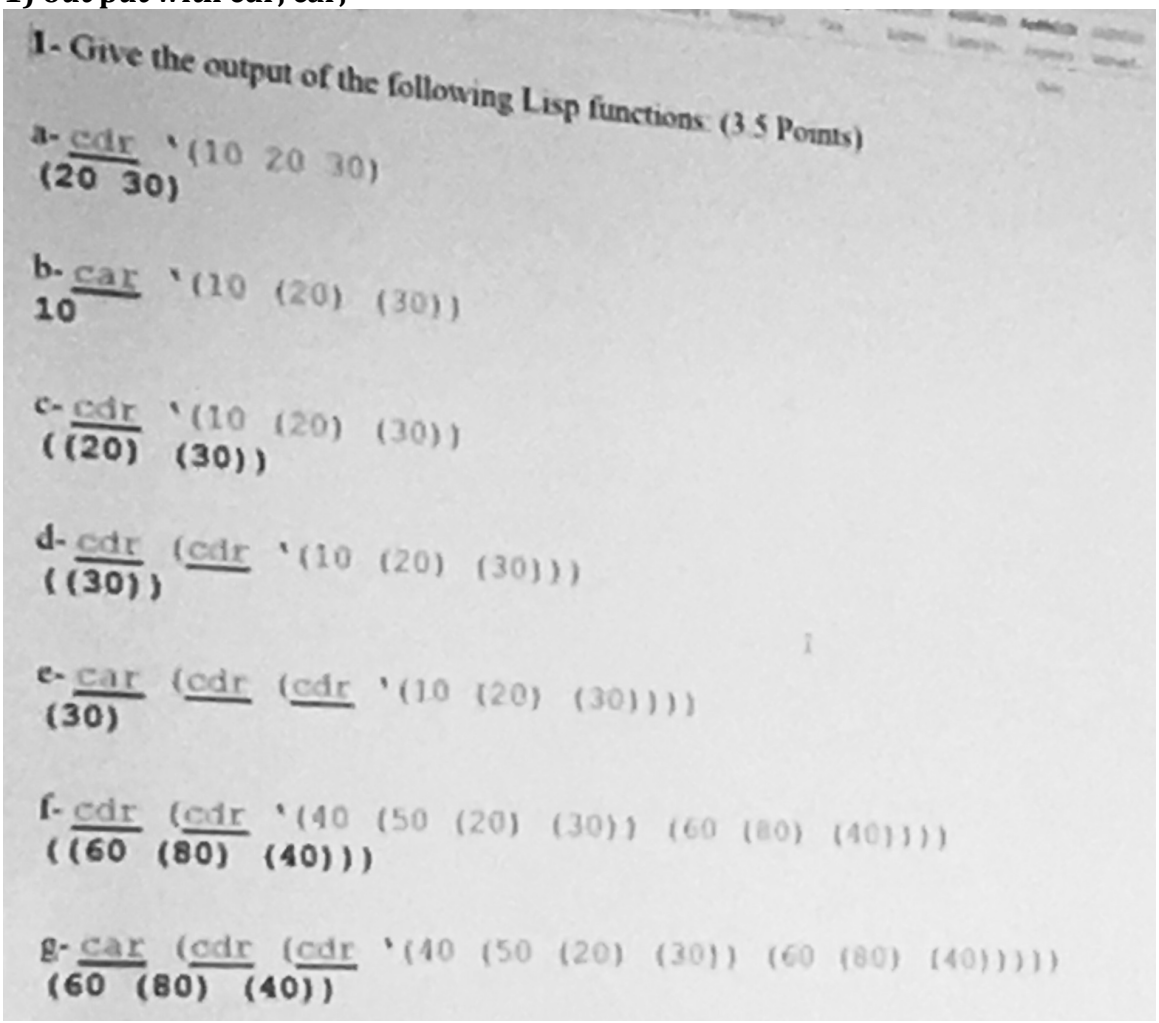
```
nulllist([0]). Or nullist([0]):-!.   1 point
nullist([0|T]) :- nullist(T).        2 points
```

Lisp

- Might be simple recursive question
- Car, cdr, cons
- Sublist
- How to concatenate
- how to trace the code (functionality)
- support by examples any statement on functionality
- ! Nor awill play with it (practise at home, see assignment)
- recursive list (base case ([]) =[], fact, recursive call)

Lisp Questions

1) out put with cdr, car, ‘



rule in these kind of issue: image that a list is an array and () = une place in the array

Another question

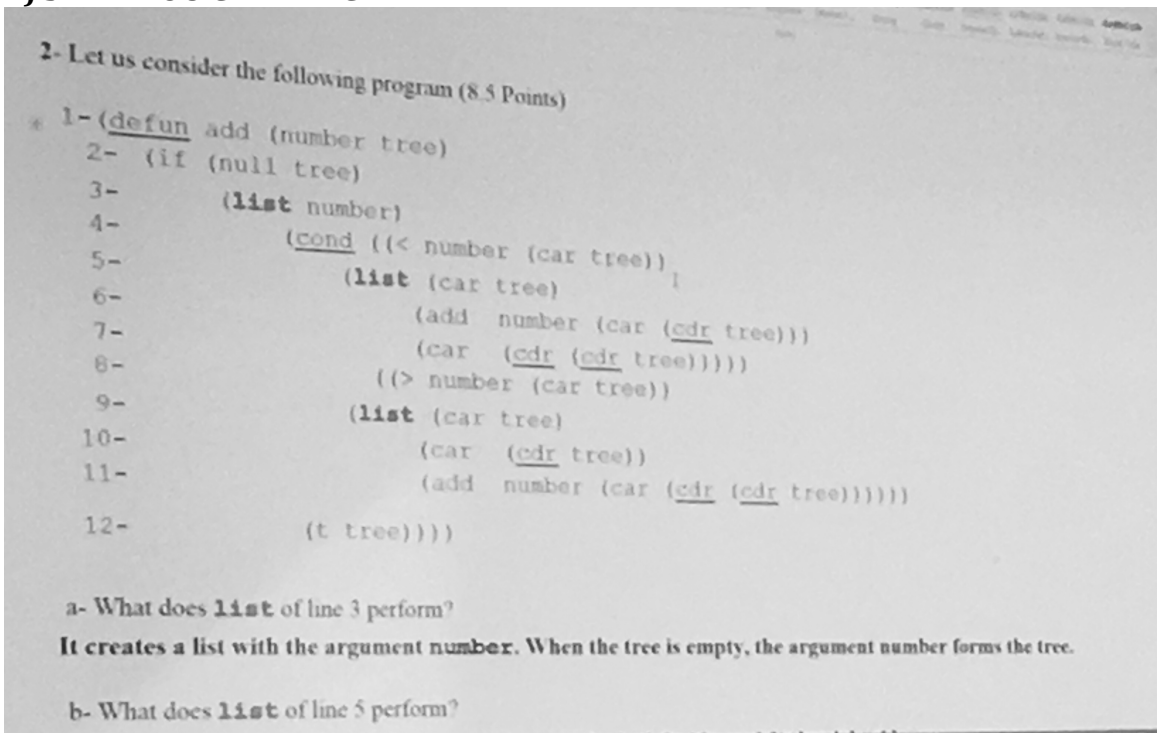
(print ‘ (a b c))	(A B C)	
(print ‘ (‘a ’b ’c))	(‘A ‘B ‘C)	‘ is taken as a character
(print ‘(‘() ‘() ‘()))	(‘NIL ‘NIL ‘NIL)	
(print ‘(() () ()))	(NIL NIL NIL)	
(print (1 2 3))	Error	
(print ‘((a b) ‘c)	((A B) ‘C)	
(print (cons 1 ()))	(1)	
(print (cons ‘1 ()))	(1)	
(print (cons a ()))	Error	Need ‘ because a is a string
(print (list ‘a ‘b ‘c))	(A B C)	
Print (list a b c))	Error	

In exam, multiple choice, review Lisp tuto 1

Ex: cons ‘1 ‘(2 3) what is the equivalent code for it?

A. ‘(1 2 3)
B. ‘(1 (2 3))
C. list ‘1 ‘(2 3)
D. None of them

2) SEE IF YOU CAN TRACE IT



2- Let us consider the following program (8.5 Points)

```
1- (defun add (number tree)
2-   (if (null tree)
3-       (list number)
4-           (cond ((< number (car tree))
5-                  (list (car tree)
6-                        (add  number (car (cdr tree)))
7-                        (car  (cdr (cdr tree)))))
8-                 ((> number (car tree))
9-                  (list (car tree)
10-                       (car  (cdr tree))
11-                       (add  number (car (cdr (cdr tree))))))
12-                (t tree))))
```

a- What does **list** of line 3 perform?

It creates a list with the argument number. When the tree is empty, the argument number forms the tree.

b- What does **list** of line 5 perform?

a- What does **list** of line 3 perform?

It creates a list with the argument number. When the tree is empty, the argument number forms the tree.

b- What does **list** of line 5 perform?

It creates a list having three elements, 1) the root, 2) the left side, and 3) the right side.

c- Why we do not have **list** in line 6 before car?

Because (car (cdr tree)) returns a list (a sub-list), which is the left side of the tree, so we don't need to convert this part to a list.

d- What does this program do? (Explain the details)

This program adds a number to a tree, which is a list having three components: 1) root, 2) left side, and 3) right side. The tree is represented as follows:
(Root Left subtree Right subtree). By adding an element, the function shecks that the elements of the Left subtree are less than the Root and the Root is less than the elements of the Right subtree. If the element we want to add already exists, the same tree is returned.

e- Illustrate with an example how this function can be executed in LispWorks (i.e. how it should be called). Give also the output of your call. In your example the second argument (tree) should have at least 5 elements.

```
Add 90 '(50 (40 (30) (45)) (60 (55) (65)))
Output: (50 (40 (30) (45)) (60 (55) NIL (90))))
```

This program adds a number to a tree, which is a list having three components: 1) root, 2) left side, and 3) right side. The tree is represented as follows:
(Root Left subtree Right subtree). By adding an element, the function shecks that the elements of the Left subtree are less than the Root and the Root is less than the elements of the Right subtree. If the element we want to add already exists, the same tree is returned.

e- Illustrate with an example how this function can be executed in LispWorks (i.e. how it should be called). Give also the output of your call. In your example the second argument (tree) should have at least 5 elements.

```
Add 90 '(50 (40 (30) (45)) (60 (55) (65)))
Output: (50 (40 (30) (45)) (60 (55) NIL (90))))
```

We have: (car tree) = 50 90 >
50

So we execute:

```
 9-(list (car tree)
10-        (car (cdr tree))
11-        (add number (car (cdr (cdr tree)))))
```

```
(car tree) = 50
(car (cdr tree)) = (40 (30) (45))
(car (cdr (cdr tree))) = (60 (55) (65))
```

→ (50 (40 (30) (45)) (add 90 '(60 (55) (65))))

```
 9-(list (car tree)
10-          (car (cdr tree))
11-          (add number (car (cdr (cdr tree))))))
(car tree) = 60
(car (cdr tree)) = (55)
(car (cdr (cdr tree))) = (65)
→ (50 (40 (30) (45)) (60 (55) (add 90 '(65))))
We have: (car tree) = 65 90 >
65
```

So we execute:

```
 9-(list (car tree)
10-          (car (cdr tree))
11-          (add number (car (cdr (cdr tree))))))

(car tree) = 65
(car (cdr tree)) = Nil
(car (cdr (cdr tree))) = Nil

→ (50 (40 (30) (45))(60 (55) NIL (add 90 Nil))))
(50 (40 (30) (45))(60 (55) NIL (90))))
```

Question 4)

4- Write down a Lisp function that reverses a list but also the internal lists.

Examples

(1 2 3) → (3 2 1)

(1 2 (3 8) 7) → (7 (8 3) 2 1)

```
(defun reverse3 (lst)
(cond ((null lst) '())
          ((listp (car lst))
             (append (reverse3 (cdr lst)) (list (reverse3 (car lst)))))
             (t (append (reverse3 (cdr lst)) (list (car lst))))))
```

Question 5)

5-Write a **recursive Lisp** function last1 that returns the last element of a list

```
(defun last1 (lst)
  (cond ((equal nil lst) nil)
        ((equal (cdr lst) nil) (car lst))
        (t (last1 (cdr lst))))
)
```

Question 6)

probablement pas a lexamen

6- Write a **recursive Lisp** function notlast that takes as argument a list and returns the same list but without the last element

Examples

(1 2 3 4) → (1 2 3)

(7) → ()

()→ ()

```
(defun notlast (lst)
  (cond ((equal nil lst) nil)
        ((equal (cdr lst) nil) nil)
        (t (cons (car lst) (notlast (cdr lst)))))
)
```

7- Write down a Lisp function that takes as argument a list and returns another list that contains the same elements but without repetition. (4 Points)