CMSC330 Spring 2014 Practice Problems 7

1. Given the following set of clauses:

```
eats(alf, cats).
eats(mary, cheese).
eats(mary, bread).
```

   List all answers generated for the following queries
   a. ?- eats(mary,cheese).
   b. ?- eats(mary,cats).
   c. ?- eats(mary,X).
   d. ?- eats(X,cats).
   e. ?- eats(X,alf).
   f. ?- eats(X,Y).

2. Given the following set of clauses:

```
travel(X) :- on_vacation(X), has_money(X).
on_vacation(mary).
on_vacation(peter).
has_money(peter).
```

   a. List all answers generated for ?- on_vacation(X).
   b. List all answers generated for ?- travel(X).
   c. Draw the Prolog search tree for travel(X).
   d. Draw the Prolog clause tree for travel(peter).

3. Given the following set of clauses:

```
foo([X], X).
foo ([_|T],X) :- foo (T,X).
```

   a. ?- foo([1],1).
   b. ?- foo([3],1).
   c. ?- foo([1,2,3],1).
   d. ?- foo([1,2,3],3).
   e. ?- foo([1,2,3],X).
   f. ?- foo([X,2,3],1).
   g. ?- foo([1,2,X],1).
   h. ?- foo([1,2|X],1).

4. Given a set of facts of form parent(name1,name2) where (name1 is the parent of name2):

    a. Define a predicate sibling(X,Y) which holds iff X and Y are siblings.
    b. Define a predicate cousin(X,Y) which holds iff X and Y are cousins.
    c. Define a predicate grandchild (X,Y) which holds iff X is a grandchild of Y.
    d. Define a predicate descendent(X,Y) which holds iff X is a descendent of Y.

5. Consider the following genealogical tree (and its graphical representation):

Genealogical Tree	Graphic Representation
parent(a,b). parent(a,c). parent(b,d). parent(b,e). parent(c,f).	a / \ b c / \ | d e f

List all answers generated by your definitions for the following queries:

    a. ?- sibling (X,Y).
    b. ?- cousin(X,Y).
    c. ?- grandchild (X,Y).
    d. ?- descendent(X,Y).

6. Given the following set of clauses:

```
jedi(luke).
jedi(yoda).
sith(vader).
sith(maul).
fight(X,Y) :- jedi(X), sith(Y).
fight(X,Y) :- sith(X), X\=Y, sith(Y).
fight(X,Y) :- jedi(X), !, jedi(Y).
```

List all answers generated for the following queries

    a. ?- fight(luke,yoda).
    b. ?- fight(luke,vader).
    c. ?- fight(vader,yoda).
    d. ?- fight(vader,maul).
    e. ?- fight(luke,X).
    f. ?- fight(vader,X).
    g. ?- fight(X,yoda).
    h. ?- fight(X,maul).
    i. ?- fight(X,Y).

7. Given the following set of clauses, what is the output for foo([3,1,2,0],R), if any?

Part	Code	Answer
A	foo([H|_], H). foo([_|T],X) :- foo(T,X).	
B	foo([_|T],X) :- foo(T,X). foo([H|_], H).	
C	foo([H|_], H) :- H > 1. foo([_|T],X) :- foo(T,X).	
D	foo([_|T],X) :- foo(T,X). foo([H|_], H) :- H > 1.	
E	foo([H|_], H) :- H > 1, !. foo([_|T],X) :- foo(T,X).	
F	foo([_|T],X) :- foo(T,X). foo([H|_], H) :- H > 1, !.	
G	foo([H|_], H). foo([_|T],X) :- X > 1, foo(T,X).	
H	foo([_|T],X) :- X > 1, foo(T,X). foo([H|_], H).	
I	foo([H|_], H). foo([_|T],X) :- foo(T,X), X > 1.	
J	foo([_|T],X) :- foo(T,X), X > 1. foo([H|_], H).	
K	foo([H|_], H). foo([_|T],X) :- foo(T,X), !, X > 1.	
L	foo([_|T],X) :- foo(T,X), !, X > 1. foo([H|_], H).	

8. Define a predicate reverse(L,K) which holds if and only if the list K is the reverse of the list L.

9. Define a predicate add_up_list(L,X) which, given a list of integers L, returns a list of integers in which each element is the sum of all the elements in L up to the same position. Example:

```
?- add_up_list([1,2,3,4],X).
X = [1,3,6,10].
```

10. Consider the following Prolog predicate definition

```
remove_at(X,[X|Xs],1,Xs).
remove_at(X,[Y|Xs],K,[Y|Ys]) :- K1 is K - 1, remove_at(X,Xs,K1,Ys).
```

It works for queries like

```
?- remove_at(X,[a,b,c,d],2,R).
X = b
R = [a,c,d].
```

However, it throws an exception for queries like

```
?- remove_at(c,[a,b,c,d],V,R).
ERROR: remove_at/4: Arguments are not sufficiently instantiated
```

Modify the predicate definition to make it work for the above query.

11. Write the prolog predicate flatten(L,R) that flattens a list of lists in L to a single list R.

The equivalent OCaml function is given by

```
let rec flatten l = match l with
[ ] | [[ ]] -> [ ]
| [ ]::t -> flatten t
| [h]::t -> h::flatten t
| ((h1::t1)::t) -> h1::flatten(t1::t);;
```