Assignment-3

Name : Chandrabhushan Reddy
Roll No. : 200101027

Ans 1)
Knowledge:

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

list_append(X, L, L) :- member(X, L), !.
list_append(X, L, [X|L]).
```

Explanation:
For list_append there are 2 statements. First one has an element X and a list L as arguments.
It checks whether X is a member of L or not. It does so by calling other compound term "member".
This member has an element X and a list as arguments. If the list has X as its head then it just cuts it.
Else it will call member again with the same element X and the tail of previous list's as its arguments.
So at last we get to know if the element X is a member of the list or not. Is it is a member then it cuts of in the middle (The point where it finds that X is a member).
Else we get to the fact "list_append(X, L, [X|L])." So now L is [X|L] i.e., the list L is getting appended with the element X.

Queries:
Query-1: ?- list_append(a, [a,b,c,d,e], L).
Output: L = [a, b, c, d, e].
Walkthrough:
member(a,[a,b,c,d,e]) % Cuts it here since head of the list is a.
So L = [a,b,c,d,e].

```
[trace]  ?- list_append(a, [a,b,c,d,e], L).
   Call: (10) list_append(a, [a, b, c, d, e], _17384) ? creep
   Call: (11) member(a, [a, b, c, d, e]) ? creep
   Exit: (11) member(a, [a, b, c, d, e]) ? creep
   Exit: (10) list_append(a, [a, b, c, d, e], [a, b, c, d, e]) ? creep
L = [a, b, c, d, e].
```

Query-2: ?- list_append(k, [a,b,c,d,e], L).
Output: L = [k, a, b, c, d, e].
Walkthrough:
member(k,[a,b,c,d,e]) % k is not a. So it calls again
member(k,[b,c,d,e]) % k is not b. So it calls again
member(k,[c,d,e]) % k is not c. So it calls again
member(k,[d,e]) % k is not d. So it calls again
member(k,[e]) % k is not e. So it calls again
member(k,[]) % Fails. So it goes to next list_append
list_append(X,L,[X,L]).
So L = [k,a,b,c,d,e].

```
[trace]  ?- list_append(k, [a,b,c,d,e], L).
   Call: (10) list_append(k, [a, b, c, d, e], _23066) ? creep
   Call: (11) member(k, [a, b, c, d, e]) ? creep
   Call: (12) member(k, [b, c, d, e]) ? creep
   Call: (13) member(k, [c, d, e]) ? creep
   Call: (14) member(k, [d, e]) ? creep
   Call: (15) member(k, [e]) ? creep
   Call: (16) member(k, []) ? creep
   Fail: (16) member(k, []) ? creep
   Fail: (15) member(k, [e]) ? creep
   Fail: (14) member(k, [d, e]) ? creep
   Fail: (13) member(k, [c, d, e]) ? creep
   Fail: (12) member(k, [b, c, d, e]) ? creep
   Fail: (11) member(k, [a, b, c, d, e]) ? creep
   Redo: (10) list_append(k, [a, b, c, d, e], _23066) ? creep
   Exit: (10) list_append(k, [a, b, c, d, e], [k, a, b, c, d, e]) ? creep
L = [k, a, b, c, d, e].
```

Ans 2)
Knowledge:
likes(mary,food).
likes(mary,wine).
likes(john,wine).
likes(john,mary).

Explanation:
All the above statements are facts.
English translations of these facts are as follows:
likes(mary,food). % mary likes food
likes(mary,wine). % mary likes wine
likes(john,wine). % john likes wine
likes(john,mary). % john likes mary

Queries:
Query-1 ?- likes(mary,food).
Output: true.
Explanation:
English translation of the query is "mary likes food".
The query is already a fact in the knowledge base. So it returns true.

Query-2 ?- likes(john,wine).
Output: true.
Explanation:
English translation of the query is "john likes wine".
The query is already a fact in the knowledge base. So it returns true.

Query-3 ?- likes(john,food).
Output: false.
Explanation:
English translation of the query is "john likes food".
The query is not a fact in the knowledge base. So it returns false.

Ans 3)
Knowledge:
eats(lion,goat).
eats(lion,deer).
eats(tiger,lamb).
eats(tiger,deer).

common(X,Y,Z):-eats(X,Z),eats(Y,Z).

Explanation:
The first 4 lines of the knowledge are facts.
The next line "common(X,Y,Z):-eats(X,Z),eats(Y,Z)." does a dfs search on Z for a given set of values of X and Y
such that eats(X,Z) is true and eats(Y,Z) is true. It returns the first value of Z which satisfies these both.

Query:?- common(lion,tiger,Z).
Output: Z = deer.
Explanation:
It does a dfs search on Z such that eats(lion,Z) and eats(tiger,Z) are true.
common(lion,tiger,Z):
eats(lion,goat) --> true eats(tiger,goat) --> false
eats(lion,deer) --> true eats(tiger,deer) --> true
So Z = deer.

```
[trace]  ?-
|     common(lion,tiger,Z).
   Call: (10) common(lion, tiger, _68362) ? creep
   Call: (11) eats(lion, _68362) ? creep
   Exit: (11) eats(lion, goat) ? creep
   Call: (11) eats(tiger, goat) ? creep
   Fail: (11) eats(tiger, goat) ? creep
   Redo: (11) eats(lion, _68362) ? creep
   Exit: (11) eats(lion, deer) ? creep
   Call: (11) eats(tiger, deer) ? creep
   Exit: (11) eats(tiger, deer) ? creep
   Exit: (10) common(lion, tiger, deer) ? creep
Z = deer.
```

Ans 4)
Knowledge:
male(homer).
male(bart).
male(abe).

```
male(luke).
female(marge).
female(lisa).
female(maggie).
female(mona).
female(jane).
parent(homer, bart).
parent(homer, lisa).
parent(homer, maggie).
parent(marge, bart).
parent(marge, lisa).
parent(marge, maggie).
parent(abe, homer).
parent(mona, homer).
parent(luke, mona).
parent(jane, abe).
mother(X, Y) :- parent(X, Y), female(X).
father(X, Y) :- parent(X, Y), male(X).
son(X, Y) :- parent(Y, X), male(X).
daughter(X, Y) :- parent(Y, X), female(X).
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```

Explanation:

The first 9 lines are facts which states homer, bart, abe, luke are males and marge, lisa, maggie, mona, jane are females.

The next 10 lines are facts which states homer is parent of bart, lisa, maggie and marge is parent of bart, lisa, maggie
and abe is parent of homer and mona is parent of homer and luke is parent of mona and jane is parent of abe.

The next statement tells that X is mother of Y if X is parent of Y and X is a female.

The next statement tells that X is father of Y if X is parent of Y and X is a male.

The next statement tells that X is son of Y if Y is parent of X and X is a male.

The next statement tells that X is daughter of Y if Y is a parent of X and X is a female.
The last statement tells that X is grandparent of Y if X is parent of some Z and that Z is a parent of Y.

Queries:
Query-1: mother(X,maggie).
Output: X = marge.
Explanation:
It does a dfs search on X such that parent(X, maggie) and female(X) are true.
It returns the first value of X which satisfies both of them i.e., X = marge.

```
[trace]  ?- mother(X,maggie).
   Call: (10) mother(_58198, maggie) ? creep
   Call: (11) parent(_58198, maggie) ? creep
   Exit: (11) parent(homer, maggie) ? creep
   Call: (11) female(homer) ? creep
   Fail: (11) female(homer) ? creep
   Redo: (11) parent(_58198, maggie) ? creep
   Exit: (11) parent(marge, maggie) ? creep
   Call: (11) female(marge) ? creep
   Exit: (11) female(marge) ? creep
   Exit: (10) mother(marge, maggie) ? creep
X = marge.
```

Query-2: son(X,mona).
Output: X = homer.
Explanation:
It does a dfs search on X such that parent(mona, X) and male(X) are true.
It returns the first value of X which satisfies both of them i.e., X = homer.

```
[trace]  ?- son(X,mona).
   Call: (10) son(_51270, mona) ? creep
   Call: (11) parent(mona, _51270) ? creep
   Exit: (11) parent(mona, homer) ? creep
   Call: (11) male(homer) ? creep
   Exit: (11) male(homer) ? creep
   Exit: (10) son(homer, mona) ? creep
X = homer.
```

Query-3: grandparent(luke,Y).
Output: Y = homer.
Explanation:
It does a dfs search on Z and Y such that parent(luke, Z) and parent(Z, Y) are true.
It returns the first value of Y which satisfies both of them i.e., Y = homer.

```
[trace]  ?- grandparent(luke,Y).
   Call: (10) grandparent(luke, _44332) ? creep
   Call: (11) parent(luke, _45630) ? creep
   Exit: (11) parent(luke, mona) ? creep
   Call: (11) parent(mona, _44332) ? creep
   Exit: (11) parent(mona, homer) ? creep
   Exit: (10) grandparent(luke, homer) ? creep
Y = homer.
```

Query-4: grandparent(jane, Y).
Output: Y = homer.
Explanation:
It does a dfs search on Z and Y such that parent(jane, Z) and parent(Z, Y) are true.
It returns the first value of Y which satisfies both of them i.e., Y = homer.

```
[trace]  ?- grandparent(jane, Y).
   Call: (10) grandparent(jane, _37390) ? creep
   Call: (11) parent(jane, _38690) ? creep
   Exit: (11) parent(jane, abe) ? creep
   Call: (11) parent(abe, _37390) ? creep
   Exit: (11) parent(abe, homer) ? creep
   Exit: (10) grandparent(jane, homer) ? creep
Y = homer.
```