ASSIGNMENT 3

(200101025 - Pramodh Billa)

Question 1

Query 1:

list_append(a,[a,b,c,d,e],L)
Calls list_member(a,[a,b,c,d,e]). % which cuts here because a is the head of the list
>> 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], _18956) ? creep
   Call: (11) list_member(a, [a, b, c, d, e]) ? creep
   Exit: (11) list_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)
Calls the list_member(k,[a,b,c,d,e]) % checks whether the head is same as the k and is not a so it check further

Calls list_member(k,[b,c,d,e])
Calls list_member(k,[c,d,e])
Calls list_member(k,[d,e])
Calls list_member(k,[e])
Calls list_member(k,[])
And at last it exits like this list_append(k,[a,b,c,d,e],[k,a,b,c,d,e])
>> 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], _24388) ? creep
   Call: (11) list_member(k, [a, b, c, d, e]) ? creep
   Call: (12) list_member(k, [b, c, d, e]) ? creep
   Call: (13) list_member(k, [c, d, e]) ? creep
   Call: (14) list_member(k, [d, e]) ? creep
   Call: (15) list_member(k, [e]) ? creep
   Call: (16) list_member(k, []) ? creep
   Fail: (16) list_member(k, []) ? creep
   Fail: (15) list_member(k, [e]) ? creep
   Fail: (14) list_member(k, [d, e]) ? creep
   Fail: (13) list_member(k, [c, d, e]) ? creep
   Fail: (12) list_member(k, [b, c, d, e]) ? creep
   Fail: (11) list_member(k, [a, b, c, d, e]) ? creep
   Redo: (10) list_append(k, [a, b, c, d, e], _24388) ? 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].
```

Question 2

Query 1:

likes(mary,food) >> true % questions if mary likes food and it is true from fact

Query 2:

likes(john,wine) >> true % questions if john likes wine and it is true from fact

Query 3:

likes(john,food) >> false % question if john likes food , there is no such fact exist, so false

Question 3

Query :
common (lion,tiger,Z) >> Z = deer

common(lion,tiger,Z) : eats(lion,goat) , eats(tiger,goat). >> false
eats(lion,deer) , eats (tiger,deer) >> true
>> Z = deer

```
[trace]  ?- common(lion,tiger,Z).
   Call: (10) common(lion, tiger, _798) ? creep
   Call: (11) eats(lion, _798) ? creep
   Exit: (11) eats(lion, goat) ? creep
   Call: (11) eats(tiger, goat) ? creep
   Fail: (11) eats(tiger, goat) ? creep
   Redo: (11) eats(lion, _798) ? 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.
```

Question 4:

Query 1:

mother(X,maggie) >> X = marge
 mother(X,Y) :- parent(X,Y),female(X). Checks for parent(X,maggie) it is true for homer and marge but female(x) it is true for marge only so our answer is marge

```
[trace]  ?-
|    mother(X,maggie).
   Call: (10) mother(_10978, maggie) ? creep
   Call: (11) parent(_10978, maggie) ? creep
   Exit: (11) parent(homer, maggie) ? creep
   Call: (11) female(homer) ? creep
   Fail: (11) female(homer) ? creep
   Redo: (11) parent(_10978, 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) >> X = homer

son(X,Y) :- parent(Y,X),male(X) Checks for parent(mona,X) it is true for homer and male(x) it is true for homer

```
[trace]  ?-
|    son(X,mona).
   Call: (10) son(_21140, mona) ? creep
   Call: (11) parent(mona, _21140) ? 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) >> Y = homer

grandparent(X,Y) :- parent(X,Z),parent(Z,Y) Checks the parent(luke,Z) it is true for mona And parent(mona,Y) it is true for homer

```
[trace]  ?-
|
|    grandparent(luke,Y).
   Call: (10) grandparent(luke, _28070) ? creep
   Call: (11) parent(luke, _29366) ? creep
   Exit: (11) parent(luke, mona) ? creep
   Call: (11) parent(mona, _28070) ? creep
   Exit: (11) parent(mona, homer) ? creep
   Exit: (10) grandparent(luke, homer) ? creep
Y = homer.
```

Query 4:

grandparent(jane,Y) >> Y = homer

grandparent(X,Y) :- parent(X,Z),parent(Z,Y)
Checks the parent(jane,Z) it is true for abe
And parent(abe,Y) it is true for homer

```
[trace]  ?- grandparent(jane,Y).
   Call: (10) grandparent(jane, _796) ? creep
   Call: (11) parent(jane, _2094) ? creep
   Exit: (11) parent(jane, abe) ? creep
   Call: (11) parent(abe, _796) ? creep
   Exit: (11) parent(abe, homer) ? creep
   Exit: (10) grandparent(jane, homer) ? creep
Y = homer.
```