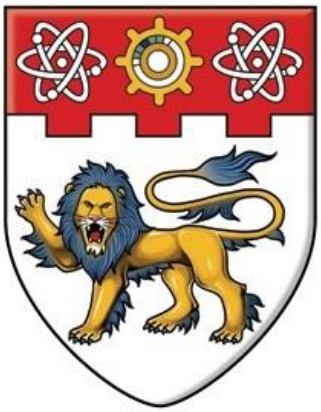

NANYANG TECHNOLOGICAL UNIVERSITY

School of Computer Science and Engineering

Artificial Intelligence Assignment 3 Report

Name (Matriculation):	Munirah Binte Mohamad (U1822418F)
Module Code:	CZ3005
Tut/Lab Group:	TSP2

Exercise 1 : The Smart Phone Rivalry (15 marks)

Sumsum, a competitor of appy, developed some nice smart phone technology called Galactica-s3, all of which was stolen by stevey, who is a boss. It is unethical for a boss to steal business from rival companies. A competitor of appy is a rival. Smart Phone technology is business.

1.1 Translate the natural language statements above describing the dealing within the Smart Phone industry in to First Order Logic (FOL)

company(sumsum).
company(appy).
smartPhoneTech (galacticaS3).
developed(galacticaS3, sumsum).
boss(stevey).
competitor(sumsum,appy).
steal(stevey, galacticaS3).
∀ Tech smartPhoneTech (Tech) => business(Tech)
∀ Comp competitor(Comp,appy) ∨ competitor(appy,Comp) => rival(Comp)
∀ X, Biz, CompA box(X) ^ steal(X,Biz) ^business(Biz) ^developed(Biz, CompA) ^rival(CompA) => unethical(X)

1.2 Write these FOL statements as Prolog clauses.

```
%Facts given
company(sumsum).
company(appy).
smartPhoneTech(galacticaS3).
developed(galacticaS3, sumsum).
boss(stevey).
competitor(sumsum,appy).
steal(stevey, galacticaS3).


%Rules
business(Tech):-smartPhoneTech(Tech).
rival(Comp):- competitor(Comp, appy); competitor(appy, Comp).


%Run Program
unethical(X):- boss(X), steal(X, Biz), business(Biz), developed(Biz,CompA), rival(CompA).
```

1.3 Using Prolog, prove that Stevey is unethical. Show a trace of your proof.

1.3.1 Trace and Result

```
?- unethical(stevey).
true .

?- trace, unethical(stevey).
   Call: (13) unethical(stevey) ? creep
   Call: (14) boss(stevey) ? creep
   Exit: (14) boss(stevey) ? creep
   Call: (14) steal(stevey, _1260) ? creep
   Exit: (14) steal(stevey, galacticaS3) ? creep
   Call: (14) business(galacticaS3) ? creep
   Call: (15) smartPhoneTech(galacticaS3) ? creep
   Exit: (15) smartPhoneTech(galacticaS3) ? creep
   Exit: (14) business(galacticaS3) ? creep
   Call: (14) developed(galacticaS3, _1524) ? creep
   Exit: (14) developed(galacticaS3, sumsum) ? creep
   Call: (14) rival(sumsum) ? creep
   Call: (15) competitor(sumsum, appy) ? creep
   Exit: (15) competitor(sumsum, appy) ? creep
   Exit: (14) rival(sumsum) ? creep
   Exit: (13) unethical(stevey) ? creep
true .
```

Exercise 2: The Royal Family (10 marks)

The old Royal succession rule states that the throne is passed down along the male line according to the order of birth before the consideration along the female line – similarly according to the order of birth. Queen Elizabeth, the monarch of United Kingdom, has four offsprings; namely:- prince Charles, princess ann, prince andrew and prince edward – listed in the order of birth.

2.1 Define their relations and rles in a Prolog rule base. Hence, define the old Royal succession rule. Using this old succession rule, determine the line of succession based on the information given. Do a trace to show your results.

%Facts given

```
offspring(prince, charles).
offspring(princess, ann).
offspring(prince, andrew).
offspring(prince, edward).

older(charles, ann).
older(ann, andrew).
older(andrew, edward).

male(A):- offspring(prince,A).
female(A):- offspring(princess,A).
```

```
%RULES

%How it is being Ordered, based on age
isOlder(X, Y):- older(X, Y).
isOlder(A, B):- older(A, X),isOlder(X, B).


%How it is being Ordered,  based on gender
inOrder(X, Y) :- offspring(prince, X), offspring(princess, Y).
inOrder(X, Y) :- offspring(prince, X), offspring(prince, Y), isOlder(X, Y).
inOrder(X, Y) :- offspring(princess, X), offspring(princess, Y), isOlder(X, Y).


%Insertion Sort on A
successors(X, Y) :- insertSort(X, [], Y).
insertSort([], OldList, OldList).
insertSort([H|T], OldList, Y) :- insertion(H, OldList, NewList), insertSort(T, NewList, Y).


% How it is inserted (Can change insertion factor (By age or gender) )
insertion(X, [], [X]).
insertion(X, [Y|T], [X, Y|T]) :- inOrder(X, Y).
insertion(X, [Y|T], [Y|NewT]) :- not(inOrder(X, Y)), insertion(X, T, NewT).


succession(SuccessionList):-
findall(Y,offspring(_,Y),OffspringList),successors(OffspringList,SuccessionList).
```

2.1.1 Trace and Result

```
?- succession(SuccessionList).
SuccessionList = [charles, andrew, edward, ann]
```

```
?- trace, succession(SuccessionList).
   Call: (13) succession(_688) ? creep
^  Call: (14) findall(_1152, offspring(_1150, _1152), _1212) ? creep
   Call: (19) offspring(_1150, _1152) ? creep
   Exit: (19) offspring(prince, charles) ? creep
   Redo: (19) offspring(_1150, _1152) ? creep
   Exit: (19) offspring(princess, ann) ? creep
   Redo: (19) offspring(_1150, _1152) ? creep
   Exit: (19) offspring(prince, andrew) ? creep
   Redo: (19) offspring(_1150, _1152) ? creep
   Exit: (19) offspring(prince, edward) ? creep
^  Exit: (14) findall(_1152, user:offspring(_1150, _1152), [charles, ann, andrew, edward]) ? creep
   Call: (14) successors([charles, ann, andrew, edward], _688) ? creep
   Call: (15) insertSort([charles, ann, andrew, edward], [], _688) ? creep
   Call: (16) insertion(charles, [], _1798) ? creep
   Exit: (16) insertion(charles, [], [charles]) ? creep
   Call: (16) insertSort([ann, andrew, edward], [charles], _688) ? creep
   Call: (17) insertion(ann, [charles], _1936) ? creep
   Call: (18) in_order(ann, charles) ? creep
   Call: (19) offspring(prince, ann) ? creep
   Fail: (19) offspring(prince, ann) ? creep
   Redo: (18) in_order(ann, charles) ? creep
   Call: (19) offspring(prince, ann) ? creep
   Fail: (19) offspring(prince, ann) ? creep
   Redo: (18) in_order(ann, charles) ? creep
   Call: (19) offspring(princess, ann) ? creep
   Exit: (19) offspring(princess, ann) ? creep
   Call: (19) offspring(princess, charles) ? creep
   Fail: (19) offspring(princess, charles) ? creep
   Fail: (18) in_order(ann, charles) ? creep
   Redo: (17) insertion(ann, [charles], _2520) ? creep
^  Call: (18) not(in_order(ann, charles)) ? creep
   Call: (19) in_order(ann, charles) ? creep
   Call: (20) offspring(prince, ann) ? creep
   Fail: (20) offspring(prince, ann) ? creep
   Redo: (19) in_order(ann, charles) ? creep
   Call: (20) offspring(prince, ann) ? creep
   Fail: (20) offspring(prince, ann) ? creep
   Redo: (19) in_order(ann, charles) ? creep
   Call: (20) offspring(princess, ann) ? creep
   Exit: (20) offspring(princess, ann) ? creep
   Call: (20) offspring(princess, charles) ? creep
   Fail: (20) offspring(princess, charles) ? creep
   Fail: (19) in_order(ann, charles) ? creep
^  Exit: (18) not(user:in_order(ann, charles)) ? creep
   Call: (18) insertion(ann, [], _2510) ? creep
   Exit: (18) insertion(ann, [], [ann]) ? creep
   Exit: (17) insertion(ann, [charles], [charles, ann]) ? creep
   Call: (17) insertSort([andrew, edward], [charles, ann], _688) ? creep
   Call: (18) insertion(andrew, [charles, ann], _3380) ? creep
   Call: (19) in_order(andrew, charles) ? creep
   Call: (20) offspring(prince, andrew) ? creep
   Exit: (20) offspring(prince, andrew) ? creep
   Call: (20) offspring(princess, charles) ? creep
   Fail: (20) offspring(princess, charles) ? creep
   Redo: (19) in_order(andrew, charles) ? creep
   Call: (20) offspring(prince, andrew) ? creep
   Exit: (20) offspring(prince, andrew) ? creep
```

```
Call: (20) offspring(prince, charles) ? creep
Exit: (20) offspring(prince, charles) ? creep
Call: (20) isOlder(andrew, charles) ? creep
Call: (21) older(andrew, charles) ? creep
Fail: (21) older(andrew, charles) ? creep
Redo: (20) isOlder(andrew, charles) ? creep
Call: (21) older(andrew, _4050) ? creep
Exit: (21) older(andrew, edward) ? creep
Call: (21) isOlder(edward, charles) ? creep
Call: (22) older(edward, charles) ? creep
Fail: (22) older(edward, charles) ? creep
Redo: (21) isOlder(edward, charles) ? creep
Call: (22) older(edward, _4314) ? creep
Fail: (22) older(edward, _4358) ? creep
Fail: (21) isOlder(edward, charles) ? creep
Fail: (20) isOlder(andrew, charles) ? creep
Redo: (19) in_order(andrew, charles) ? creep
Call: (20) offspring(princess, andrew) ? creep
Fail: (20) offspring(princess, andrew) ? creep
Fail: (19) in_order(andrew, charles) ? creep
Redo: (18) insertion(andrew, [charles, ann], _4668) ? creep
Call: (19) not(in_order(andrew, charles)) ? creep
Call: (20) in_order(andrew, charles) ? creep
Call: (21) offspring(prince, andrew) ? creep
Exit: (21) offspring(prince, andrew) ? creep
Call: (21) offspring(princess, charles) ? creep
Fail: (21) offspring(princess, charles) ? creep
Redo: (20) in_order(andrew, charles) ? creep
Call: (21) offspring(prince, andrew) ? creep
Exit: (21) offspring(prince, andrew) ? creep
Call: (21) offspring(prince, charles) ? creep
Exit: (21) offspring(prince, charles) ? creep
Call: (21) isOlder(andrew, charles) ? creep
Call: (22) older(andrew, charles) ? creep
Fail: (22) older(andrew, charles) ? creep
Redo: (21) isOlder(andrew, charles) ? creep
Call: (22) older(andrew, _5388) ? creep
Exit: (22) older(andrew, edward) ? creep
Call: (22) isOlder(edward, charles) ? creep
Call: (23) older(edward, charles) ? creep
Fail: (23) older(edward, charles) ? creep
Redo: (22) isOlder(edward, charles) ? creep
Call: (23) older(edward, _5652) ? creep
Fail: (23) older(edward, _5696) ? creep
Fail: (22) isOlder(edward, charles) ? creep
Fail: (21) isOlder(andrew, charles) ? creep
Redo: (20) in_order(andrew, charles) ? creep
Call: (21) offspring(princess, andrew) ? creep
Fail: (21) offspring(princess, andrew) ? creep
Fail: (20) in_order(andrew, charles) ? creep
Exit: (19) not(user:in_order(andrew, charles)) ? creep
Call: (19) insertion(andrew, [ann], _4658) ? creep
Call: (20) in_order(andrew, ann) ? creep
Call: (21) offspring(prince, andrew) ? creep
Exit: (21) offspring(prince, andrew) ? creep
Call: (21) offspring(princess, ann) ? creep
Exit: (21) offspring(princess, ann) ? creep
Exit: (20) in_order(andrew, ann) ? creep
```

```
Exit: (19) insertion(andrew, [ann], [andrew, ann]) ? creep
Exit: (18) insertion(andrew, [charles, ann], [charles, andrew, ann]) ? creep
Call: (18) insertSort([edward], [charles, andrew, ann], _688) ? creep
Call: (19) insertion(edward, [charles, andrew, ann], _6502) ? creep
Call: (20) in_order(edward, charles) ? creep
Call: (21) offspring(prince, edward) ? creep
Exit: (21) offspring(prince, edward) ? creep
Call: (21) offspring(princess, charles) ? creep
Fail: (21) offspring(princess, charles) ? creep
Redo: (20) in_order(edward, charles) ? creep
Call: (21) offspring(prince, edward) ? creep
Exit: (21) offspring(prince, edward) ? creep
Call: (21) offspring(prince, charles) ? creep
Exit: (21) offspring(prince, charles) ? creep
Call: (21) isOlder(edward, charles) ? creep
Call: (22) older(edward, charles) ? creep
Fail: (22) older(edward, charles) ? creep
Redo: (21) isOlder(edward, charles) ? creep
Call: (22) older(edward, _7172) ? creep
Fail: (22) older(edward, _7216) ? creep
Fail: (21) isOlder(edward, charles) ? creep
Redo: (20) in_order(edward, charles) ? creep
Call: (21) offspring(princess, edward) ? creep
Fail: (21) offspring(princess, edward) ? creep
Fail: (20) in_order(edward, charles) ? creep
Redo: (19) insertion(edward, [charles, andrew, ann], _7482) ? creep
Call: (20) not(in_order(edward, charles)) ? creep
Call: (21) in_order(edward, charles) ? creep
Call: (22) offspring(prince, edward) ? creep
Exit: (22) offspring(prince, edward) ? creep
Call: (22) offspring(princess, charles) ? creep
Fail: (22) offspring(princess, charles) ? creep
Redo: (21) in_order(edward, charles) ? creep
Call: (22) offspring(prince, edward) ? creep
Exit: (22) offspring(prince, edward) ? creep
Call: (22) offspring(prince, charles) ? creep
Exit: (22) offspring(prince, charles) ? creep
Call: (22) isOlder(edward, charles) ? creep
Call: (23) older(edward, charles) ? creep
Fail: (23) older(edward, charles) ? creep
Redo: (22) isOlder(edward, charles) ? creep
Call: (23) older(edward, _8202) ? creep
Fail: (23) older(edward, _8246) ? creep
Fail: (22) isOlder(edward, charles) ? creep
Redo: (21) in_order(edward, charles) ? creep
Call: (22) offspring(princess, edward) ? creep
Fail: (22) offspring(princess, edward) ? creep
Fail: (21) in_order(edward, charles) ? creep
Exit: (20) not(user:in_order(edward, charles)) ? creep
Call: (20) insertion(edward, [andrew, ann], _7472) ? creep
Call: (21) in_order(edward, andrew) ? creep
Call: (22) offspring(prince, edward) ? creep
Exit: (22) offspring(prince, edward) ? creep
Call: (22) offspring(princess, andrew) ? creep
Fail: (22) offspring(princess, andrew) ? creep
Redo: (21) in_order(edward, andrew) ? creep
Call: (22) offspring(prince, edward) ? creep
Exit: (22) offspring(prince, edward) ? creep
```

```
   Call: (22) offspring(prince, andrew) ? creep
   Exit: (22) offspring(prince, andrew) ? creep
   Call: (22) isOlder(edward, andrew) ? creep
   Call: (23) older(edward, andrew) ? creep
   Fail: (23) older(edward, andrew) ? creep
   Redo: (22) isOlder(edward, andrew) ? creep
   Call: (23) older(edward, _9226) ? creep
   Fail: (23) older(edward, _9270) ? creep
   Fail: (22) isOlder(edward, andrew) ? creep
   Redo: (21) in_order(edward, andrew) ? creep
   Call: (22) offspring(princess, edward) ? creep
   Fail: (22) offspring(princess, edward) ? creep
   Fail: (21) in_order(edward, andrew) ? creep
   Redo: (20) insertion(edward, [andrew, ann], _7472) ? creep
^  Call: (21) not(in_order(edward, andrew)) ? creep
   Call: (22) in_order(edward, andrew) ? creep
   Call: (23) offspring(prince, edward) ? creep
   Exit: (23) offspring(prince, edward) ? creep
   Call: (23) offspring(princess, andrew) ? creep
   Fail: (23) offspring(princess, andrew) ? creep
   Redo: (22) in_order(edward, andrew) ? creep
   Call: (23) offspring(prince, edward) ? creep
   Exit: (23) offspring(prince, edward) ? creep
   Call: (23) offspring(prince, andrew) ? creep
   Exit: (23) offspring(prince, andrew) ? creep
   Call: (23) isOlder(edward, andrew) ? creep
   Call: (24) older(edward, andrew) ? creep
   Fail: (24) older(edward, andrew) ? creep
   Redo: (23) isOlder(edward, andrew) ? creep
   Call: (24) older(edward, _10256) ? creep
   Fail: (24) older(edward, _10300) ? creep
   Fail: (23) isOlder(edward, andrew) ? creep
   Redo: (22) in_order(edward, andrew) ? creep
   Call: (23) offspring(princess, edward) ? creep
   Fail: (23) offspring(princess, edward) ? creep
   Fail: (22) in_order(edward, andrew) ? creep
^  Exit: (21) not(user:in_order(edward, andrew)) ? creep
   Call: (21) insertion(edward, [ann], _9526) ? creep
   Call: (22) in_order(edward, ann) ? creep
   Call: (23) offspring(prince, edward) ? creep
   Exit: (23) offspring(prince, edward) ? creep
   Call: (23) offspring(princess, ann) ? creep
   Exit: (23) offspring(princess, ann) ? creep
   Exit: (22) in_order(edward, ann) ? creep
   Exit: (21) insertion(edward, [ann], [edward, ann]) ? creep
   Exit: (20) insertion(edward, [andrew, ann], [andrew, edward, ann]) ? creep
   Exit: (19) insertion(edward, [charles, andrew, ann], [charles, andrew, edward, ann]) ? creep
   Call: (19) insertSort([], [charles, andrew, edward, ann], _688) ? creep
   Exit: (19) insertSort([], [charles, andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (18) insertSort([edward], [charles, andrew, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (17) insertSort([andrew, edward], [charles, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (16) insertSort([ann, andrew, edward], [charles], [charles, andrew, edward, ann]) ? creep
   Exit: (15) insertSort([charles, ann, andrew, edward], [], [charles, andrew, edward, ann]) ? creep
   Exit: (14) successors([charles, ann, andrew, edward], [charles, andrew, edward, ann]) ? creep
   Exit: (13) succession([charles, andrew, edward, ann]) ? creep
SuccessionList = [charles, andrew, edward, ann] .
```

2.2 Recently, the Royal succession rule has been modified. The throne is now passed down according to the order of birth irrespective of gender. Modify your rules and Prolog knowledge base to handle the new succession rule. Explain the necessary changes to the knowledge needed to represent the new information. Use this new succession rule to determine the new line of succession based on the same knowledge given. Show your results using a trace.

```
%Facts given
offspring(prince, charles).
offspring(princess, ann).
offspring(prince, andrew).
offspring(prince, edward).

older(charles, ann).
older(ann, andrew).
older(andrew, edward).

male(A):- offspring(prince,A).
female(A):- offspring(princess,A).


%RULES
%How it is being Ordered,based on age
isOlder(X, Y):- older(X, Y).
isOlder(A, B):- older(A, X),isOlder(X, B).


%How it is being Ordered, based on gender
inOrder(X, Y) :- offspring(prince, X), offspring(princess, Y).
inOrder(X, Y) :- offspring(prince, X), offspring(prince, Y), isOlder(X, Y).
inOrder(X, Y) :- offspring(princess, X), offspring(princess, Y), isOlder(X, Y).


%Insertion Sort on A
successors(X, Y) :- insertSort(X, [], Y).
insertSort([], OldList, OldList).
insertSort([H|T], OldList, Y) :- insertion(H, OldList, NewList), insertSort(T, NewList, Y).


% How it is inserted (Can change insertion factor (By age or gender))
insertion(X, [], [X]).
insertion(X, [Y|T], [X, Y|T]) :- isOlder(X, Y).
insertion(X, [Y|T], [Y|NewT]) :- not(isOlder(X, Y)), insertion(X, T, NewT).


succession(SuccessionList):-
findall(Y,offspring(_,Y),OffspringList),successors(OffspringList,SuccessionList).
```

2.2.1 Trace and Result

```
?- succession(SuccessionList).
SuccessionList = [charles, ann, andrew, edward] .

?- trace, succession(SuccessionList).
   Call: (13) succession(_2282) ? creep
^  Call: (14) findall(_2752, offspring(_2750, _2752), _2812) ? creep
   Call: (19) offspring(_2750, _2752) ? creep
   Exit: (19) offspring(prince, charles) ? creep
   Redo: (19) offspring(_2750, _2752) ? creep
   Exit: (19) offspring(princess, ann) ? creep
   Redo: (19) offspring(_2750, _2752) ? creep
   Exit: (19) offspring(prince, andrew) ? creep
   Redo: (19) offspring(_2750, _2752) ? creep
   Exit: (19) offspring(prince, edward) ? creep
^  Exit: (14) findall(_2752, user:offspring(_2750, _2752), [charles, ann, andrew, edward]) ? creep
   Call: (14) successors([charles, ann, andrew, edward], _2282) ? creep
   Call: (15) insertSort([charles, ann, andrew, edward], [], _2282) ? creep
   Call: (16) insertion(charles, [], _3398) ? creep
   Exit: (16) insertion(charles, [], [charles]) ? creep
   Call: (16) insertSort([ann, andrew, edward], [charles], _2282) ? creep
   Call: (17) insertion(ann, [charles], _3536) ? creep
   Call: (18) isOlder(ann, charles) ? creep
   Call: (19) older(ann, charles) ? creep
   Fail: (19) older(ann, charles) ? creep
   Redo: (18) isOlder(ann, charles) ? creep
   Call: (19) older(ann, _3766) ? creep
   Exit: (19) older(ann, andrew) ? creep
   Call: (19) isOlder(andrew, charles) ? creep
   Call: (20) older(andrew, charles) ? creep
   Fail: (20) older(andrew, charles) ? creep
   Redo: (19) isOlder(andrew, charles) ? creep
   Call: (20) older(andrew, _4030) ? creep
   Exit: (20) older(andrew, edward) ? creep
   Call: (20) isOlder(edward, charles) ? creep
   Call: (21) older(edward, charles) ? creep
   Fail: (21) older(edward, charles) ? creep
   Redo: (20) isOlder(edward, charles) ? creep
   Call: (21) older(edward, _4294) ? creep
   Fail: (21) older(edward, _4338) ? creep
   Fail: (20) isOlder(edward, charles) ? creep
   Fail: (19) isOlder(andrew, charles) ? creep
   Fail: (18) isOlder(ann, charles) ? creep
   Redo: (17) insertion(ann, [charles], _4516) ? creep
^  Call: (18) not(isOlder(ann, charles)) ? creep
   Call: (19) isOlder(ann, charles) ? creep
   Call: (20) older(ann, charles) ? creep
   Fail: (20) older(ann, charles) ? creep
   Redo: (19) isOlder(ann, charles) ? creep
   Call: (20) older(ann, _4796) ? creep
   Exit: (20) older(ann, andrew) ? creep
   Call: (20) isOlder(andrew, charles) ? creep
   Call: (21) older(andrew, charles) ? creep
   Fail: (21) older(andrew, charles) ? creep
   Redo: (20) isOlder(andrew, charles) ? creep
   Call: (21) older(andrew, _5060) ? creep
   Exit: (21) older(andrew, edward) ? creep
   Call: (21) isOlder(edward, charles) ? creep
   Call: (22) older(edward, charles) ? creep
   Fail: (22) older(edward, charles) ? creep
   Redo: (21) isOlder(edward, charles) ? creep
   Call: (22) older(edward, _5324) ? creep
```

```
   Fail: (22) older(edward, _5368) ? creep
   Fail: (21) isOlder(edward, charles) ? creep
   Fail: (20) isOlder(andrew, charles) ? creep
   Fail: (19) isOlder(ann, charles) ? creep
^  Exit: (18) not(user:isOlder(ann, charles)) ? creep
   Call: (18) insertion(ann, [], _4506) ? creep
   Exit: (18) insertion(ann, [], [ann]) ? creep
   Exit: (17) insertion(ann, [charles], [charles, ann]) ? creep
   Call: (17) insertSort([andrew, edward], [charles, ann], _2282) ? creep
   Call: (18) insertion(andrew, [charles, ann], _5772) ? creep
   Call: (19) isOlder(andrew, charles) ? creep
   Call: (20) older(andrew, charles) ? creep
   Fail: (20) older(andrew, charles) ? creep
   Redo: (19) isOlder(andrew, charles) ? creep
   Call: (20) older(andrew, _6002) ? creep
   Exit: (20) older(andrew, edward) ? creep
   Call: (20) isOlder(edward, charles) ? creep
   Call: (21) older(edward, charles) ? creep
   Fail: (21) older(edward, charles) ? creep
   Redo: (20) isOlder(edward, charles) ? creep
   Call: (21) older(edward, _6266) ? creep
   Fail: (21) older(edward, _6310) ? creep
   Fail: (20) isOlder(edward, charles) ? creep
   Fail: (19) isOlder(andrew, charles) ? creep
   Redo: (18) insertion(andrew, [charles, ann], _6444) ? creep
^  Call: (19) not(isOlder(andrew, charles)) ? creep
   Call: (20) isOlder(andrew, charles) ? creep
   Call: (21) older(andrew, charles) ? creep
   Fail: (21) older(andrew, charles) ? creep
   Redo: (20) isOlder(andrew, charles) ? creep
   Call: (21) older(andrew, _6724) ? creep
   Exit: (21) older(andrew, edward) ? creep
   Call: (21) isOlder(edward, charles) ? creep
   Call: (22) older(edward, charles) ? creep
   Fail: (22) older(edward, charles) ? creep
   Redo: (21) isOlder(edward, charles) ? creep
   Call: (22) older(edward, _6988) ? creep
   Fail: (22) older(edward, _7032) ? creep
   Fail: (21) isOlder(edward, charles) ? creep
   Fail: (20) isOlder(andrew, charles) ? creep
^  Exit: (19) not(user:isOlder(andrew, charles)) ? creep
   Call: (19) insertion(andrew, [ann], _6434) ? creep
   Call: (20) isOlder(andrew, ann) ? creep
   Call: (21) older(andrew, ann) ? creep
   Fail: (21) older(andrew, ann) ? creep
   Redo: (20) isOlder(andrew, ann) ? creep
   Call: (21) older(andrew, _7440) ? creep
   Exit: (21) older(andrew, edward) ? creep
   Call: (21) isOlder(edward, ann) ? creep
   Call: (22) older(edward, ann) ? creep
   Fail: (22) older(edward, ann) ? creep
   Redo: (21) isOlder(edward, ann) ? creep
   Call: (22) older(edward, _7704) ? creep
   Fail: (22) older(edward, _7748) ? creep
   Fail: (21) isOlder(edward, ann) ? creep
   Fail: (20) isOlder(andrew, ann) ? Unknown option (h for help)
   Fail: (20) isOlder(andrew, ann) ? Unknown option (h for help)
   Fail: (20) isOlder(andrew, ann) ? creep
```

```
   Redo: (19) insertion(andrew, [ann], _6434) ? creep
^  Call: (20) not(isOlder(andrew, ann)) ? creep
   Call: (21) isOlder(andrew, ann) ? creep
   Call: (22) older(andrew, ann) ? creep
   Fail: (22) older(andrew, ann) ? creep
   Redo: (21) isOlder(andrew, ann) ? creep
   Call: (22) older(andrew, _8250) ? creep
   Exit: (22) older(andrew, edward) ? creep
   Call: (22) isOlder(edward, ann) ? creep
   Call: (23) older(edward, ann) ? creep
   Fail: (23) older(edward, ann) ? creep
   Redo: (22) isOlder(edward, ann) ? creep
   Call: (23) older(edward, _8514) ? creep
   Fail: (23) older(edward, _8558) ? creep
   Fail: (22) isOlder(edward, ann) ? creep
   Fail: (21) isOlder(andrew, ann) ? creep
^  Exit: (20) not(user:isOlder(andrew, ann)) ? creep
   Call: (20) insertion(andrew, [], _7960) ? creep
   Exit: (20) insertion(andrew, [], [andrew]) ? creep
   Exit: (19) insertion(andrew, [ann], [ann, andrew]) ? creep
   Exit: (18) insertion(andrew, [charles, ann], [charles, ann, andrew]) ? creep
   Call: (18) insertSort([edward], [charles, ann, andrew], _2282) ? creep
   Call: (19) insertion(edward, [charles, ann, andrew], _8962) ? creep
   Call: (20) isOlder(edward, charles) ? creep
   Call: (21) older(edward, charles) ? creep
   Fail: (21) older(edward, charles) ? creep
   Redo: (20) isOlder(edward, charles) ? creep
   Call: (21) older(edward, _9192) ? creep
   Fail: (21) older(edward, _9236) ? creep
   Fail: (20) isOlder(edward, charles) ? creep
   Redo: (19) insertion(edward, [charles, ann, andrew], _9326) ? creep
^  Call: (20) not(isOlder(edward, charles)) ? creep
   Call: (21) isOlder(edward, charles) ? creep
   Call: (22) older(edward, charles) ? creep
   Fail: (22) older(edward, charles) ? creep
   Redo: (21) isOlder(edward, charles) ? creep
   Call: (22) older(edward, _9606) ? creep
   Fail: (22) older(edward, _9650) ? creep
   Fail: (21) isOlder(edward, charles) ? creep
^  Exit: (20) not(user:isOlder(edward, charles)) ? creep
   Call: (20) insertion(edward, [ann, andrew], _9316) ? creep
   Call: (21) isOlder(edward, ann) ? creep
   Call: (22) older(edward, ann) ? creep
   Fail: (22) older(edward, ann) ? creep
   Redo: (21) isOlder(edward, ann) ? creep
   Call: (22) older(edward, _10014) ? creep
   Fail: (22) older(edward, _10058) ? creep
   Fail: (21) isOlder(edward, ann) ? creep
   Redo: (20) insertion(edward, [ann, andrew], _9316) ? creep
^  Call: (21) not(isOlder(edward, ann)) ? creep
   Call: (22) isOlder(edward, ann) ? creep
   Call: (23) older(edward, ann) ? creep
   Fail: (23) older(edward, ann) ? creep
   Redo: (22) isOlder(edward, ann) ? creep
   Call: (23) older(edward, _10428) ? creep
   Fail: (23) older(edward, _10472) ? creep
   Fail: (22) isOlder(edward, ann) ? creep
^  Exit: (21) not(user:isOlder(edward, ann)) ? creep
```

```
   Call: (21) insertion(edward, [andrew], _10138) ? creep
   Call: (22) isOlder(edward, andrew) ? creep
   Call: (23) older(edward, andrew) ? creep
   Fail: (23) older(edward, andrew) ? creep
   Redo: (22) isOlder(edward, andrew) ? creep
   Call: (23) older(edward, _10836) ? creep
   Fail: (23) older(edward, _10880) ? creep
   Fail: (22) isOlder(edward, andrew) ? creep
   Redo: (21) insertion(edward, [andrew], _10138) ? creep
^  Call: (22) not(isOlder(edward, andrew)) ? creep
   Call: (23) isOlder(edward, andrew) ? creep
   Call: (24) older(edward, andrew) ? creep
   Fail: (24) older(edward, andrew) ? creep
   Redo: (23) isOlder(edward, andrew) ? creep
   Call: (24) older(edward, _11250) ? creep
   Fail: (24) older(edward, _11294) ? creep
   Fail: (23) isOlder(edward, andrew) ? creep
^  Exit: (22) not(user:isOlder(edward, andrew)) ? creep
   Call: (22) insertion(edward, [], _10960) ? creep
   Exit: (22) insertion(edward, [], [edward]) ? creep
   Exit: (21) insertion(edward, [andrew], [andrew, edward]) ? creep
   Exit: (20) insertion(edward, [ann, andrew], [ann, andrew, edward]) ? creep
   Exit: (19) insertion(edward, [charles, ann, andrew], [charles, ann, andrew, edward]) ? creep
   Call: (19) insertSort([], [charles, ann, andrew, edward], _2282) ? creep
   Exit: (19) insertSort([], [charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
   Exit: (18) insertSort([edward], [charles, ann, andrew], [charles, ann, andrew, edward]) ? creep
   Exit: (17) insertSort([andrew, edward], [charles, ann], [charles, ann, andrew, edward]) ? creep
   Exit: (16) insertSort([ann, andrew, edward], [charles], [charles, ann, andrew, edward]) ? creep
   Exit: (15) insertSort([charles, ann, andrew, edward], [], [charles, ann, andrew, edward]) ? creep
   Exit: (14) successors([charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
   Exit: (13) succession([charles, ann, andrew, edward]) ? creep
SuccessionList = [charles, ann, andrew, edward] .
```

2.2.2 Explain the necessary changes to the knowledge needed to represent the new information:

There are two rules that were created as a comparator rule. The first rule, 'isOlder', which is used to determine the order of offspring according to their order of birth. One the other hand, the second rule 'inOrder' is used to determine the order of offspring according to the old succession rule. To change the comparator factor in the program, the comparator rule should be changed. Both rules are used to make relations and thus makes it easier to implement the rule as both rules are transitive.

In FOL:

$$\forall x, y, z\ isOlder(xy) \wedge isOlder(y, z) \Rightarrow isOlder(x, z)$$

For any x,y and z:

Premise 1:	X is older than Y
Premise 2:	Y is older than Z
Conclusion:	X is older than Z