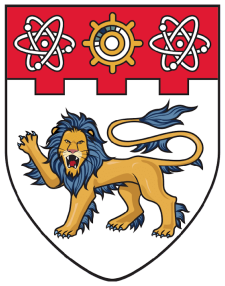

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SEM2020/21 Year 3

CZ3005 Lab 2 Report

Lab Group :

FDDP1

Group Member :

Chua Zi Jian (U2022354J)

Min Kabar Kyaw (U2021858K)

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. Smartphone technology is a business.

1. Translate the natural language statements above describing the dealing within the Smartphone industry into First Order Logic (FOL). (5 marks)

Natural Language	***First Order Logic***
sumsum, a competitor of **appy**	company(sumsum)
	company(appy)
	competitor(sumsum , appy)
sumsum, a competitor of **appy**, developed some nice smart phone technology called **galactica-s3**	develop(sumsum , galactica-s3)
smartphone technology called **galactica-s3**	smartphonetech(galactica-s3)
galactica-s3, all of which was stolen by **stevey**	steal(stevey , galactica-s3)
stevey, who is a boss	boss(stevey)
stevey works in appy	works(stevey , appy)
It is unethical for a boss to steal business from rival companies	∀x, y ,z, a, boss(x) ∧ works(x , y) ∧ rival(y, z) ∧ steal(x , a) ∧ business(a) ∧ develop(y , a) ⇒ unethical(x) *Explanation:*

	Boss who works in a certain company steals a smartphone technology which is a business that is developed by the rival company implies that the boss is unethical.
A competitor of **appy** is a rival	∀x, competitor(x , appy) ⇒ rival(x , appy)
Smartphone technology is a business.	∀x, smartphonetech(x) ⇒ business(x)

2. Write these FOL statements as Prolog clauses. (5 marks)

```
/*sumsum is company*/
company(sumsum).

/*appy is company*/
company(appy).

/*sumsum is competitor of appy*/
competitor(sumsum,appy).
/*competitor(Y,X):-
        competitor(X,Y).*/

/*if company is competitor then they are rival*/
rival(X,appy):-
        competitor(X,appy).

/*rival and competitor with each other*/
/*rival(Y,X):-
        rival(X,Y).*/



/*sumsum develop galactica-s3*/
develop(sumsum,galactica-s3).

/*stevey is boss*/
boss(stevey).

/*stevey works in works in appy */
works(stevey,appy).

/*stevey steal galactica-s3*/
steal(stevey,galactica-s3).

/*galactica-s3 is smart phone tech*/
smartphonetech(galactica-s3).

/*X is a business if it is a smart phone tech*/
business(X):-
        smartphonetech(X).
```

```
/*X is unethical if X steals something from rival company*/
unethical(X):-
        boss(X),
        steal(X,Y),
        business(Y),
        develop(A,Y),
        works(X,B),
        (rival(A,B);rival(B,A)).
```

3. Using Prolog, prove that Stevey is unethical. Show a trace of your proof. (5 marks)

```
[trace]  ?- unethical(X).
   Call: (10) unethical(_19794) ? creep
   Call: (11) boss(_19794) ? creep
   Exit: (11) boss(stevey) ? creep
   Call: (11) steal(stevey, _22472) ? creep
   Exit: (11) steal(stevey, galactica-s3) ? creep
   Call: (11) business(galactica-s3) ? creep
   Call: (12) smartphonetech(galactica-s3) ? creep
   Exit: (12) smartphonetech(galactica-s3) ? creep
   Exit: (11) business(galactica-s3) ? creep
   Call: (11) develop(_26988, galactica-s3) ? creep
   Exit: (11) develop(sumsum, galactica-s3) ? creep
   Call: (11) works(stevey, _28498) ? creep
   Exit: (11) works(stevey, appy) ? creep
   Call: (11) rival(sumsum, appy) ? creep
   Call: (12) competitor(sumsum, appy) ? creep
   Exit: (12) competitor(sumsum, appy) ? creep
   Exit: (11) rival(sumsum, appy) ? creep
   Exit: (10) unethical(stevey) ? creep
X = stevey .
```

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 the United Kingdom, has four offspring; namely:- **prince charles**, **princess ann**, **prince andrew** and **prince edward** – listed in the order of birth.

1. Define their relations and rules 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. (5 marks)

Natural Language	Prolog Clause
Queen Elizabeth is the parent of Prince Charles, Princess Ann, Prince Andrew and Prince Edward.	parent(elizabeth, charles). parent(elizabeth, ann). parent(elizabeth, andrew). parent(elizabeth, edward).
Charles is Male. Andrew is Male. Edward is Male..	male(charles). male(andrew). male(edward).
Ann is Female.	female(ann).
Successor Rule: Males are prioritized over Females. Older children are prioritized over younger children.	

Approach 1: Establish Natural Order + Sorting

First, establish the **natural order** of all children/offspring such that they follow the following rule:

- Male is “smaller” than Female.
- Older age is “smaller” than Younger age.

Note that **Male/Female is prioritized in the order first before age.** So a younger male would be considered “smaller” than an older female.

Next, we sort all children/offspring of the queen in **ascending order** based on the natural order we just defined. The smallest element in this sorted list is the child who will be the direct next-in-line successor (highest priority for succession).

Some trivia: In a modern programming language like Java/C++, establishing natural order is done through implementing interfaces called **Comparator/Comparable**. A compare function between two objects is implemented which determines how we do the comparison to determine which object is “smaller”/”bigger”. **We are doing the same thing in Prolog by changing the “comparing” process of a sort procedure in Prolog.**

```
class SuccessorComparator implements Comparator<Royal>(){
        @Override
        public int compare(Royal firstPerson, Royal secondPerson){
                if((firstPerson.gender == 'Male') && (secondPerson.gender == 'Female'){
                        return -1; // firstPerson is considered "smaller" than secondPerson
                }
                else if((firstPerson.gender == 'Female' && (secondPerson.gender == 'Male'){
                        return 1; // firstPerson is considered "bigger" than secondPerson
                }
                else{ // Same gender, so compare by age where older age is considered “smaller”
                        return (secondPerson.age).compareTo(firstPerson.age);
                }
         }
}
// In main function:
Collections.sort(listOfRoyals, new SuccessComparator<Royal>()); // Sort our list of Royals based on our
Comparator
```

Prolog:

```
insert(A,[B|C],[B|D]):-not(precedes(A,B)),!,insert(A,C,D).
insert(A,C,[A|C]).
succession_sort([A|B],SortList):-succession_sort(B,Tail),insert(A,Tai
l,SortList).
succession_sort([],[]).
successionList(SuccessionList):-findall(Y,offspring(Y,_),ChildNodes),
succession_sort(ChildNodes,SuccessionList).
```

Trace Of the Result

```
[trace]  ?- successionList(X).
   Call: (10) successionList(_21900) ? creep
^  Call: (11) findall(_23082, offspring(_23082, _23088), _23090) ? creep
   Call: (16) offspring(_23082, _23088) ? creep
   Exit: (16) offspring(charles, elizabeth) ? creep
   Redo: (16) offspring(_23082, _23088) ? creep
   Exit: (16) offspring(andrew, elizabeth) ? creep
   Redo: (16) offspring(_23082, _23088) ? creep
   Exit: (16) offspring(edward, elizabeth) ? creep
   Redo: (16) offspring(_23082, _23088) ? creep
   Exit: (16) offspring(ann, elizabeth) ? creep
^  Exit: (11) findall(_23082, user:offspring(_23082, _23088), [charles, andrew, edward, ann]) ? creep
   Call: (11) succession_sort([charles, andrew, edward, ann], _21900) ? creep
   Call: (12) succession_sort([andrew, edward, ann], _31476) ? creep
   Call: (13) succession_sort([edward, ann], _32232) ? creep
   Call: (14) succession_sort([ann], _32988) ? creep
   Call: (15) succession_sort([], _33744) ? creep
   Exit: (15) succession_sort([], []) ? creep
   Call: (15) insert(ann, [], _32988) ? creep
   Exit: (15) insert(ann, [], [ann]) ? creep
   Exit: (14) succession_sort([ann], [ann]) ? creep
   Call: (14) insert(edward, [ann], _32232) ? creep
^  Call: (15) not(precedes(edward, ann)) ? creep
   Call: (16) precedes(edward, ann) ? creep
   Call: (17) prince(edward) ? creep
   Call: (18) parentOf(_40572, edward) ? creep
   Call: (19) offspring(edward, _40572) ? creep
   Exit: (19) offspring(edward, elizabeth) ? creep
   Exit: (18) parentOf(elizabeth, edward) ? creep
   Call: (18) queen(elizabeth) ? creep
   Exit: (18) queen(elizabeth) ? creep
   Call: (18) male(edward) ? creep
   Exit: (18) male(edward) ? creep
   Exit: (17) prince(edward) ? creep
   Call: (17) princess(ann) ? creep
   Call: (18) parentOf(_48090, ann) ? creep
   Call: (19) offspring(ann, _48090) ? creep
   Exit: (19) offspring(ann, elizabeth) ? creep
   Exit: (18) parentOf(elizabeth, ann) ? creep
   Call: (18) queen(elizabeth) ? creep
   Exit: (18) queen(elizabeth) ? creep
   Call: (18) female(ann) ? creep
   Exit: (18) female(ann) ? creep
   Exit: (17) princess(ann) ? creep
   Exit: (16) precedes(edward, ann) ? creep
^  Fail: (15) not(user:precedes(edward, ann)) ? creep
   Redo: (14) insert(edward, [ann], _32232) ? creep
   Exit: (14) insert(edward, [ann], [edward, ann]) ? creep
   Exit: (13) succession_sort([edward, ann], [edward, ann]) ? creep
   Call: (13) insert(andrew, [edward, ann], _31476) ? creep
^  Call: (14) not(precedes(andrew, edward)) ? creep
   Call: (15) precedes(andrew, edward) ? creep
   Call: (16) prince(andrew) ? creep
   Call: (17) parentOf(_61692, andrew) ? creep
   Call: (18) offspring(andrew, _61692) ? creep
   Exit: (18) offspring(andrew, elizabeth) ? creep
   Exit: (17) parentOf(elizabeth, andrew) ? creep
   Call: (17) queen(elizabeth) ? creep
   Exit: (17) queen(elizabeth) ? creep
   Call: (17) male(andrew) ? creep
   Exit: (17) male(andrew) ? creep
   Exit: (16) prince(andrew) ? creep
   Call: (16) princess(edward) ? creep
   Call: (17) parentOf(_5122, edward) ? creep
   Call: (18) offspring(edward, _5122) ? creep
```

```
Exit: (18) offspring(andrew, elizabeth) ? creep
Exit: (17) parentOf(elizabeth, andrew) ? creep
Call: (17) queen(elizabeth) ? creep
Exit: (17) queen(elizabeth) ? creep
Call: (17) male(andrew) ? creep
Exit: (17) male(andrew) ? creep
Exit: (16) prince(andrew) ? creep
Call: (16) princess(edward) ? creep
Call: (17) parentOf(_5122, edward) ? creep
Call: (18) offspring(edward, _5122) ? creep
Exit: (18) offspring(edward, elizabeth) ? creep
Exit: (17) parentOf(elizabeth, edward) ? creep
Call: (17) queen(elizabeth) ? creep
Exit: (17) queen(elizabeth) ? creep
Call: (17) female(edward) ? creep
Fail: (17) female(edward) ? creep
Fail: (16) princess(edward) ? creep
Redo: (15) precedes(andrew, edward) ? creep
Call: (16) prince(andrew) ? creep
Call: (17) parentOf(_13394, andrew) ? creep
Call: (18) offspring(andrew, _13394) ? creep
Exit: (18) offspring(andrew, elizabeth) ? creep
Exit: (17) parentOf(elizabeth, andrew) ? creep
Call: (17) queen(elizabeth) ? creep
Exit: (17) queen(elizabeth) ? creep
Call: (17) male(andrew) ? creep
Exit: (17) male(andrew) ? creep
Exit: (16) prince(andrew) ? creep
Call: (16) prince(edward) ? creep
Call: (17) parentOf(_20912, edward) ? creep
Call: (18) offspring(edward, _20912) ? creep
Exit: (18) offspring(edward, elizabeth) ? creep
Exit: (17) parentOf(elizabeth, edward) ? creep
Call: (17) queen(elizabeth) ? creep
Exit: (17) queen(elizabeth) ? creep
Call: (17) male(edward) ? creep
Exit: (17) male(edward) ? creep
Exit: (16) prince(edward) ? creep
Call: (16) olderThan(andrew, edward) ? creep
Call: (17) isOlder(andrew, edward) ? creep
Exit: (17) isOlder(andrew, edward) ? creep
Exit: (16) olderThan(andrew, edward) ? creep
Exit: (15) precedes(andrew, edward) ? creep
Fail: (14) not(user:precedes(andrew, edward)) ? creep
Redo: (13) insert(andrew, [edward, ann], _98) ? creep
Exit: (13) insert(andrew, [edward, ann], [andrew, edward, ann]) ? creep
Exit: (12) succession_sort([andrew, edward, ann], [andrew, edward, ann]) ? creep
Call: (12) insert(charles, [andrew, edward, ann], _18) ? creep
Call: (13) not(precedes(charles, andrew)) ? creep
Call: (14) precedes(charles, andrew) ? creep
Call: (15) prince(charles) ? creep
Call: (16) parentOf(_37530, charles) ? creep
Call: (17) offspring(charles, _37530) ? creep
Exit: (17) offspring(charles, elizabeth) ? creep
Exit: (16) parentOf(elizabeth, charles) ? creep
Call: (16) queen(elizabeth) ? creep
Exit: (16) queen(elizabeth) ? creep
Call: (16) male(charles) ? creep
Exit: (16) male(charles) ? creep
Exit: (15) prince(charles) ? creep
Call: (15) princess(andrew) ? creep
Call: (16) parentOf(_45048, andrew) ? creep
Call: (17) offspring(andrew, _45048) ? creep
Exit: (17) offspring(andrew, elizabeth) ? creep
Exit: (16) parentOf(elizabeth, andrew) ? creep
```

```
   Call: (16) queen(elizabeth) ? creep
   Exit: (16) queen(elizabeth) ? creep
   Call: (16) male(andrew) ? creep
   Exit: (16) male(andrew) ? creep
   Exit: (15) prince(andrew) ? creep
   Call: (15) olderThan(charles, andrew) ? creep
   Call: (16) isOlder(charles, andrew) ? creep
   Fail: (16) isOlder(charles, andrew) ? creep
   Redo: (15) olderThan(charles, andrew) ? creep
   Call: (16) isOlder(charles, _6570) ? creep
   Exit: (16) isOlder(charles, ann) ? creep
   Call: (16) isOlder(ann, andrew) ? creep
   Exit: (16) isOlder(ann, andrew) ? creep
   Exit: (15) olderThan(charles, andrew) ? creep
   Exit: (14) precedes(charles, andrew) ? creep
^  Fail: (13) not(user:precedes(charles, andrew)) ? creep
   Redo: (12) insert(charles, [andrew, edward, ann], _18) ? creep
   Exit: (12) insert(charles, [andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (11) succession_sort([charles, andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (10) successionList([charles, andrew, edward, ann]) ? creep
X = [charles, andrew, edward, ann].
```

Approach 2: Finding direct successor + Recursion

We attempt to establish the natural order as mentioned above via first order logic, and write a predicate that tells us the direct successor of a person, then recursively call that function on the result. Note that in this approach, the list of children is assumed to be already sorted based on age.

Prolog:

```
successor(N):- ((male(K), not(female(N)),children(L), member(X, L),
not(N=X), olderThan(N,X), (K=X)); (female(K), children(L), not(N=K),
member(X, L), (K=X)))) -> print(K),nl,successor(K).
```

Logical Expression:

$$successor(N) = ((male(K) \wedge \neg female(N) \wedge children(L) \wedge member(X, L) \wedge \neg(N = X) \wedge olderThan(N, X) \wedge (K = X)) \vee (female(K) \wedge children(L) \wedge \neg(N = K) \wedge member(X, L) \wedge (K = X)) \rightarrow print(K) \wedge nl \wedge successor(K)$$

Part by part break down:

$$(male(K) \wedge \neg female(N) \wedge children(L) \wedge member(X, L) \wedge \neg(N = X) \wedge olderThan(N, X) \wedge (K = X))$$

This segment attempts to search for the oldest male in the list of children who is younger than the current **king**. Note that if the current throne holder (N) is female, this clause will fail.

$$(female(K) \wedge children(L) \wedge \neg(N = K) \wedge member(X, L) \wedge (K = X))$$

If the previous search failed (i.e. N is the youngest male), this segment then searches for the oldest qualifying child who is **female**.

The two segments are logical OR'ed into an intermediate result (suppose R).

$$R \rightarrow print(K) \wedge nl \wedge successor(K)$$

If R is true (a successor K has been found), then print the successor and then recursively call successor(K) to get the next successor.

Note: In this approach, **we do not state that Elizabeth is female** despite being clearly so. This is working on the assumption that Elizabeth is our current default queen, and so she is actually able to pass down the throne to males (despite the logic implying that **females can only pass down the throne to females. This works on the assumption**

that if N is female, then all males available must have already been a previous successor to N.)

Trace Of the Result:

```
[trace]  ?- successor(elizabeth).
   Call: (10) successor(elizabeth) ? creep
   Call: (11) male(_21092) ? creep
   Exit: (11) male(charles) ? creep
^  Call: (11) not(female(elizabeth)) ? creep
   Call: (12) female(elizabeth) ? creep
   Fail: (12) female(elizabeth) ? creep
^  Exit: (11) not(user:female(elizabeth)) ? creep
^  Call: (11) not(elizabeth=charles) ? creep
^  Exit: (11) not(user:(elizabeth=charles)) ? creep
   Call: (11) children(_27164) ? creep
   Exit: (11) children([charles, ann, andrew, edward]) ? creep
   Call: (11) lists:member(_28690, [charles, ann, andrew, edward]) ? creep
   Exit: (11) lists:member(charles, [charles, ann, andrew, edward]) ? creep
   Call: (11) olderThan(elizabeth, charles) ? creep
   Call: (12) older_than(elizabeth, charles) ? creep
   Exit: (12) older_than(elizabeth, charles) ? creep
   Exit: (11) olderThan(elizabeth, charles) ? creep
   Call: (11) charles=charles ? creep
   Exit: (11) charles=charles ? creep
   Call: (11) print(charles) ? creep
charles
   Exit: (11) print(charles) ? creep
   Call: (11) nl ? creep
```

```
   Exit: (11) nl ? creep
   Call: (11) successor(charles) ? creep
   Call: (12) male(_38482) ? creep
   Exit: (12) male(charles) ? creep
^  Call: (12) not(female(charles)) ? creep
   Call: (13) female(charles) ? creep
   Fail: (13) female(charles) ? creep
^  Exit: (12) not(user:female(charles)) ? creep
^  Call: (12) not(charles=charles) ? creep
^  Fail: (12) not(user:(charles=charles)) ? creep
   Redo: (12) male(_38482) ? creep
   Exit: (12) male(andrew) ? creep
^  Call: (12) not(female(charles)) ? creep
   Call: (13) female(charles) ? creep
   Fail: (13) female(charles) ? creep
^  Exit: (12) not(user:female(charles)) ? creep
^  Call: (12) not(charles=andrew) ? creep
^  Exit: (12) not(user:(charles=andrew)) ? creep
   Call: (12) children(_50624) ? creep
   Exit: (12) children([charles, ann, andrew, edward]) ? creep
   Call: (12) lists:member(_52150, [charles, ann, andrew, edward]) ? creep
   Exit: (12) lists:member(charles, [charles, ann, andrew, edward]) ? creep
   Call: (12) olderThan(charles, charles) ? creep
   Call: (13) older_than(charles, charles) ? creep
   Fail: (13) older_than(charles, charles) ? creep
   Redo: (12) olderThan(charles, charles) ? creep
   Call: (13) older_than(charles, _56700) ? creep
   Exit: (13) older_than(charles, ann) ? creep
   Call: (13) older_than(ann, charles) ? creep
   Fail: (13) older_than(ann, charles) ? creep
   Fail: (12) olderThan(charles, charles) ? creep
   Redo: (12) lists:member(_52150, [charles, ann, andrew, edward]) ? creep
   Exit: (12) lists:member(ann, [charles, ann, andrew, edward]) ? creep
   Call: (12) olderThan(charles, ann) ? creep
   Call: (13) older_than(charles, ann) ? creep
   Exit: (13) older_than(charles, ann) ? creep
   Exit: (12) olderThan(charles, ann) ? creep
   Call: (12) andrew=ann ? creep
   Fail: (12) andrew=ann ? creep
   Redo: (12) olderThan(charles, ann) ? creep
   Call: (13) older_than(charles, _3112) ? creep
   Exit: (13) older_than(charles, ann) ? creep
   Call: (13) older_than(ann, ann) ? creep
   Fail: (13) older_than(ann, ann) ? creep
   Fail: (12) olderThan(charles, ann) ? creep
   Redo: (12) lists:member(_114, [charles, ann, andrew, edward]) ? creep
   Exit: (12) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
   Call: (12) olderThan(charles, andrew) ? creep
   Call: (13) older_than(charles, andrew) ? creep
   Fail: (13) older_than(charles, andrew) ? creep
   Redo: (12) olderThan(charles, andrew) ? creep
   Call: (13) older_than(charles, _11432) ? creep
   Exit: (13) older_than(charles, ann) ? creep
   Call: (13) older_than(ann, andrew) ? creep
   Exit: (13) older_than(ann, andrew) ? creep
   Exit: (12) olderThan(charles, andrew) ? creep
   Call: (12) andrew=andrew ? creep
   Exit: (12) andrew=andrew ? creep
   Call: (12) print(andrew) ? creep
andrew
```

```
   Exit: (12) print(andrew) ? creep
   Call: (12) nl ? creep

   Exit: (12) nl ? creep
   Call: (12) successor(andrew) ? creep
   Call: (13) male(_20446) ? creep
   Exit: (13) male(charles) ? creep
^  Call: (13) not(female(andrew)) ? creep
   Call: (14) female(andrew) ? creep
   Fail: (14) female(andrew) ? creep
^  Exit: (13) not(user:female(andrew)) ? creep
^  Call: (13) not(andrew=charles) ? creep
^  Exit: (13) not(user:(andrew=charles)) ? creep
   Call: (13) children(_26518) ? creep
   Exit: (13) children([charles, ann, andrew, edward]) ? creep
   Call: (13) lists:member(_28044, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(charles, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, charles) ? creep
   Call: (14) older_than(andrew, charles) ? creep
   Fail: (14) older_than(andrew, charles) ? creep
   Redo: (13) olderThan(andrew, charles) ? creep
   Call: (14) older_than(andrew, _32594) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(edward, charles) ? creep
   Fail: (14) older_than(edward, charles) ? creep
   Fail: (13) olderThan(andrew, charles) ? creep
   Redo: (13) lists:member(_28044, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(ann, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, ann) ? creep
   Call: (14) older_than(andrew, ann) ? creep
   Fail: (14) older_than(andrew, ann) ? creep
   Redo: (13) olderThan(andrew, ann) ? creep
   Call: (14) older_than(andrew, _40914) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(edward, ann) ? creep
   Fail: (14) older_than(edward, ann) ? creep
   Fail: (13) olderThan(andrew, ann) ? creep
   Redo: (13) lists:member(_28044, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, andrew) ? creep
   Call: (14) older_than(andrew, andrew) ? creep
   Fail: (14) older_than(andrew, andrew) ? creep
   Redo: (13) olderThan(andrew, andrew) ? creep
   Call: (14) older_than(andrew, _49234) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(edward, andrew) ? creep
   Fail: (14) older_than(edward, andrew) ? creep
   Fail: (13) olderThan(andrew, andrew) ? creep
   Redo: (13) lists:member(_28044, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(edward, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, edward) ? creep
   Call: (14) older_than(andrew, edward) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Exit: (13) olderThan(andrew, edward) ? creep
   Call: (13) charles=edward ? creep
   Fail: (13) charles=edward ? creep
   Redo: (13) olderThan(andrew, edward) ? creep
   Call: (14) older_than(andrew, _59816) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(edward, edward) ? creep
   Fail: (14) older_than(edward, edward) ? creep
   Fail: (13) olderThan(andrew, edward) ? creep
   Redo: (13) male(_20446) ? creep
   Exit: (13) male(andrew) ? creep
^  Call: (13) not(female(andrew)) ? creep
```

```
   Call: (14) female(andrew) ? creep
   Fail: (14) female(andrew) ? creep
^  Exit: (13) not(user:female(andrew)) ? creep
^  Call: (13) not(andrew=andrew) ? creep
^  Fail: (13) not(user:(andrew=andrew)) ? creep
   Redo: (13) male(_116) ? creep
   Exit: (13) male(edward) ? creep
^  Call: (13) not(female(andrew)) ? creep
   Call: (14) female(andrew) ? creep
   Fail: (14) female(andrew) ? creep
^  Exit: (13) not(user:female(andrew)) ? creep
^  Call: (13) not(andrew=edward) ? creep
^  Exit: (13) not(user:(andrew=edward)) ? creep
   Call: (13) children(_11510) ? creep
   Exit: (13) children([charles, ann, andrew, edward]) ? creep
   Call: (13) lists:member(_13036, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(charles, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, charles) ? creep
   Call: (14) older_than(andrew, charles) ? creep
   Fail: (14) older_than(andrew, charles) ? creep
   Redo: (13) olderThan(andrew, charles) ? creep
   Call: (14) older_than(andrew, _17586) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(edward, charles) ? creep
   Fail: (14) older_than(edward, charles) ? creep
   Fail: (13) olderThan(andrew, charles) ? creep
   Redo: (13) lists:member(_13036, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(ann, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, ann) ? creep
   Call: (14) older_than(andrew, ann) ? creep
   Fail: (14) older_than(andrew, ann) ? creep
   Redo: (13) olderThan(andrew, ann) ? creep
   Call: (14) older_than(andrew, _25906) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(edward, ann) ? creep
   Fail: (14) older_than(edward, ann) ? creep
   Fail: (13) olderThan(andrew, ann) ? creep
   Redo: (13) lists:member(_13036, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, andrew) ? creep
   Call: (14) older_than(andrew, andrew) ? creep
   Fail: (14) older_than(andrew, andrew) ? creep
   Redo: (13) olderThan(andrew, andrew) ? creep
   Call: (14) older_than(andrew, _34226) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(edward, andrew) ? creep
   Fail: (14) older_than(edward, andrew) ? creep
   Fail: (13) olderThan(andrew, andrew) ? creep
   Redo: (13) lists:member(_13036, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(edward, [charles, ann, andrew, edward]) ? creep
   Call: (13) olderThan(andrew, edward) ? creep
   Call: (14) older_than(andrew, edward) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Exit: (13) olderThan(andrew, edward) ? creep
   Call: (13) edward=edward ? creep
   Exit: (13) edward=edward ? creep
   Call: (13) print(edward) ? creep
edward
```

```
   Exit: (13) print(edward) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Call: (13) successor(edward) ? creep
   Call: (14) male(_47788) ? creep
   Exit: (14) male(charles) ? creep
^  Call: (14) not(female(edward)) ? creep
   Call: (15) female(edward) ? creep
   Fail: (15) female(edward) ? creep
^  Exit: (14) not(user:female(edward)) ? creep
^  Call: (14) not(edward=charles) ? creep
^  Exit: (14) not(user:(edward=charles)) ? creep
   Call: (14) children(_53860) ? creep
   Exit: (14) children([charles, ann, andrew, edward]) ? creep
   Call: (14) lists:member(_55386, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(charles, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, charles) ? creep
   Call: (15) older_than(edward, charles) ? creep
   Fail: (15) older_than(edward, charles) ? creep
   Redo: (14) olderThan(edward, charles) ? creep
   Call: (15) older_than(edward, _59936) ? creep
   Fail: (15) older_than(edward, _59936) ? creep
   Fail: (14) olderThan(edward, charles) ? creep
   Redo: (14) lists:member(_55386, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(ann, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, ann) ? creep
   Call: (15) older_than(edward, ann) ? creep
   Fail: (15) older_than(edward, ann) ? creep
   Redo: (14) olderThan(edward, ann) ? creep
   Call: (15) older_than(edward, _2440) ? creep
   Fail: (15) older_than(edward, _2440) ? creep
   Fail: (14) olderThan(edward, ann) ? creep
   Redo: (14) lists:member(_174, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, andrew) ? creep
   Call: (15) older_than(edward, andrew) ? creep
   Fail: (15) older_than(edward, andrew) ? creep
   Redo: (14) olderThan(edward, andrew) ? creep
   Call: (15) older_than(edward, _9252) ? creep
   Fail: (15) older_than(edward, _9252) ? creep
   Fail: (14) olderThan(edward, andrew) ? creep
   Redo: (14) lists:member(_174, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(edward, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, edward) ? creep
   Call: (15) older_than(edward, edward) ? creep
   Fail: (15) older_than(edward, edward) ? creep
   Redo: (14) olderThan(edward, edward) ? creep
   Call: (15) older_than(edward, _16064) ? creep
   Fail: (15) older_than(edward, _16064) ? creep
   Fail: (14) olderThan(edward, edward) ? creep
   Redo: (14) male(_146) ? creep
   Exit: (14) male(andrew) ? creep
^  Call: (14) not(female(edward)) ? creep
   Call: (15) female(edward) ? creep
   Fail: (15) female(edward) ? creep
^  Exit: (14) not(user:female(edward)) ? creep
^  Call: (14) not(edward=andrew) ? creep
^  Exit: (14) not(user:(edward=andrew)) ? creep
   Call: (14) children(_24398) ? creep
   Exit: (14) children([charles, ann, andrew, edward]) ? creep
   Call: (14) lists:member(_25924, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(charles, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, charles) ? creep
   Call: (15) older_than(edward, charles) ? creep
```

```
   Fail: (15) older_than(edward, charles) ? creep
   Redo: (14) olderThan(edward, charles) ? creep
   Call: (15) older_than(edward, _30474) ? creep
   Fail: (15) older_than(edward, _30474) ? creep
   Fail: (14) olderThan(edward, charles) ? creep
   Redo: (14) lists:member(_25924, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(ann, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, ann) ? creep
   Call: (15) older_than(edward, ann) ? creep
   Fail: (15) older_than(edward, ann) ? creep
   Redo: (14) olderThan(edward, ann) ? creep
   Call: (15) older_than(edward, _37286) ? creep
   Fail: (15) older_than(edward, _37286) ? creep
   Fail: (14) olderThan(edward, ann) ? creep
   Redo: (14) lists:member(_25924, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, andrew) ? creep
   Call: (15) older_than(edward, andrew) ? creep
   Fail: (15) older_than(edward, andrew) ? creep
   Redo: (14) olderThan(edward, andrew) ? creep
   Call: (15) older_than(edward, _44098) ? creep
   Fail: (15) older_than(edward, _44098) ? creep
   Fail: (14) olderThan(edward, andrew) ? creep
   Redo: (14) lists:member(_25924, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(edward, [charles, ann, andrew, edward]) ? creep
   Call: (14) olderThan(edward, edward) ? creep
   Call: (15) older_than(edward, edward) ? creep
   Fail: (15) older_than(edward, edward) ? creep
   Redo: (14) olderThan(edward, edward) ? creep
   Call: (15) older_than(edward, _50910) ? creep
   Fail: (15) older_than(edward, _50910) ? creep
   Fail: (14) olderThan(edward, edward) ? creep
   Redo: (14) male(_146) ? creep
   Exit: (14) male(edward) ? creep
^  Call: (14) not(female(edward)) ? creep
   Call: (15) female(edward) ? creep
   Fail: (15) female(edward) ? creep
^  Exit: (14) not(user:female(edward)) ? creep
^  Call: (14) not(edward=edward) ? creep
^  Fail: (14) not(user:(edward=edward)) ? creep
   Redo: (13) successor(edward) ? creep
   Call: (14) female(_59994) ? creep
   Exit: (14) female(ann) ? creep
   Call: (14) children(_61496) ? creep
   Exit: (14) children([charles, ann, andrew, edward]) ? creep
^  Call: (14) not(edward=ann) ? creep
^  Exit: (14) not(user:(edward=ann)) ? creep
   Call: (14) lists:member(_504, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(charles, [charles, ann, andrew, edward]) ? creep
   Call: (14) ann=charles ? creep
   Fail: (14) ann=charles ? creep
   Redo: (14) lists:member(_504, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(ann, [charles, ann, andrew, edward]) ? creep
   Call: (14) ann=ann ? creep
   Exit: (14) ann=ann ? creep
   Call: (14) print(ann) ? creep
ann
```

```
   Exit: (14) print(ann) ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Call: (14) successor(ann) ? creep
   Call: (15) male(_10320) ? creep
   Exit: (15) male(charles) ? creep
^  Call: (15) not(female(ann)) ? creep
   Call: (16) female(ann) ? creep
   Exit: (16) female(ann) ? creep
^  Fail: (15) not(user:female(ann)) ? creep
   Redo: (15) male(_10320) ? creep
   Exit: (15) male(andrew) ? creep
^  Call: (15) not(female(ann)) ? creep
   Call: (16) female(ann) ? creep
   Exit: (16) female(ann) ? creep
^  Fail: (15) not(user:female(ann)) ? creep
   Redo: (15) male(_10320) ? creep
   Exit: (15) male(edward) ? creep
^  Call: (15) not(female(ann)) ? creep
   Call: (16) female(ann) ? creep
   Exit: (16) female(ann) ? creep
^  Fail: (15) not(user:female(ann)) ? creep
   Redo: (14) successor(ann) ? creep
   Call: (15) female(_24674) ? creep
   Exit: (15) female(ann) ? creep
   Call: (15) children(_26176) ? creep
   Exit: (15) children([charles, ann, andrew, edward]) ? creep
^  Call: (15) not(ann=ann) ? creep
^  Fail: (15) not(user:(ann=ann)) ? creep
   Fail: (14) successor(ann) ? creep
   Fail: (13) successor(edward) ? creep
   Fail: (12) successor(andrew) ? creep
   Fail: (11) successor(charles) ? creep
   Fail: (10) successor(elizabeth) ? creep
false.
```

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. (5 marks)

Modified Approach 1: Establish Natural Order + Sorting

We modified the rules of the **natural order** of all children/offspring such that they follow the following rule:

- Older age is “smaller” than Younger age.

Next, we sort all children/offspring of the queen in **ascending order** based on the natural order we just defined. The smallest element in this sorted list is the child who will be the direct next-in-line successor (highest priority for succession).

Prolog:

```
insert(A,[B|C],[B|D]):-not(olderThan(A,B)),!,insert(A,C,D).
insert(A,C,[A|C]).
succession_sort([A|B],SortList):-succession_sort(B,Tail),insert(A,Tail
,SortList).
succession_sort([],[]).
successionList(SuccessionList):-findall(Y,offspring(Y,_),ChildNodes),s
uccession_sort(ChildNodes,SuccessionList).
```

Trace Of the Result:

```
[trace]  ?- successionList(X).
   Call: (10) successionList(_21310) ? creep
^  Call: (11) findall(_22490, offspring(_22490, _22496), _22498) ? creep
   Call: (16) offspring(_22490, _22496) ? creep
   Exit: (16) offspring(charles, elizabeth) ? creep
   Redo: (16) offspring(_22490, _22496) ? creep
   Exit: (16) offspring(andrew, elizabeth) ? creep
   Redo: (16) offspring(_22490, _22496) ? creep
   Exit: (16) offspring(edward, elizabeth) ? creep
   Redo: (16) offspring(_22490, _22496) ? creep
   Exit: (16) offspring(ann, elizabeth) ? creep
^  Exit: (11) findall(_22490, user:offspring(_22490, _22496), [charles, andrew, edward, ann]) ? creep
   Call: (11) succession_sort([charles, andrew, edward, ann], _21310) ? creep
   Call: (12) succession_sort([andrew, edward, ann], _30884) ? creep
   Call: (13) succession_sort([edward, ann], _31640) ? creep
   Call: (14) succession_sort([ann], _32396) ? creep
   Call: (15) succession_sort([], _33152) ? creep
   Exit: (15) succession_sort([], []) ? creep
   Call: (15) insert(ann, [], _32396) ? creep
   Exit: (15) insert(ann, [], [ann]) ? creep
   Exit: (14) succession_sort([ann], [ann]) ? creep
   Call: (14) insert(edward, [ann], _31640) ? creep
^  Call: (15) not(olderThan(edward, ann)) ? creep
   Call: (16) olderThan(edward, ann) ? creep
   Call: (17) isOlder(edward, ann) ? creep
   Fail: (17) isOlder(edward, ann) ? creep
   Redo: (16) olderThan(edward, ann) ? creep
   Call: (17) isOlder(edward, _41492) ? creep
   Fail: (17) isOlder(edward, _41492) ? creep
   Fail: (16) olderThan(edward, ann) ? creep
^  Exit: (15) not(user:olderThan(edward, ann)) ? creep
   Call: (15) insert(edward, [], _37700) ? creep
   Exit: (15) insert(edward, [], [edward]) ? creep
   Exit: (14) insert(edward, [ann], [ann, edward]) ? creep
   Exit: (13) succession_sort([edward, ann], [ann, edward]) ? creep
   Call: (13) insert(andrew, [ann, edward], _30884) ? creep
^  Call: (14) not(olderThan(andrew, ann)) ? creep
   Call: (15) olderThan(andrew, ann) ? creep
   Call: (16) isOlder(andrew, ann) ? creep
   Fail: (16) isOlder(andrew, ann) ? creep
   Redo: (15) olderThan(andrew, ann) ? creep
   Call: (16) isOlder(andrew, _52106) ? creep
   Exit: (16) isOlder(andrew, edward) ? creep
   Call: (16) isOlder(edward, ann) ? creep
   Fail: (16) isOlder(edward, ann) ? creep
   Fail: (15) olderThan(andrew, ann) ? creep
^  Exit: (14) not(user:olderThan(andrew, ann)) ? creep
   Call: (14) insert(andrew, [edward], _48314) ? creep
^  Call: (15) not(olderThan(andrew, edward)) ? creep
   Call: (16) olderThan(andrew, edward) ? creep
   Call: (17) isOlder(andrew, edward) ? creep
   Exit: (17) isOlder(andrew, edward) ? creep
   Exit: (16) olderThan(andrew, edward) ? creep
^  Fail: (15) not(user:olderThan(andrew, edward)) ? creep
   Redo: (14) insert(andrew, [edward], _48314) ? creep
   Exit: (14) insert(andrew, [edward], [andrew, edward]) ? creep
   Exit: (13) insert(andrew, [ann, edward], [ann, andrew, edward]) ? creep
   Exit: (12) succession_sort([andrew, edward, ann], [ann, andrew, edward]) ? creep
   Call: (12) insert(charles, [ann, andrew, edward], _18) ? creep
^  Call: (13) not(olderThan(charles, ann)) ? creep
   Call: (14) olderThan(charles, ann) ? creep
   Call: (15) isOlder(charles, ann) ? creep
   Exit: (15) isOlder(charles, ann) ? creep
   Exit: (14) olderThan(charles, ann) ? creep
^  Fail: (13) not(user:olderThan(charles, ann)) ? creep
   Redo: (12) insert(charles, [ann, andrew, edward], _18) ? creep
```

```
   Exit: (12) insert(charles, [ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
   Exit: (11) succession_sort([charles, andrew, edward, ann], [charles, ann, andrew, edward]) ? creep
   Exit: (10) successionList([charles, ann, andrew, edward]) ? creep
X = [charles, ann, andrew, edward].
```

Modified Approach 2: Finding direct successor + Recursion

We attempt to establish the natural order as mentioned above via first order logic, and write a predicate that tells us the direct successor of a person, then recursively call that function on the result.

Prolog:

```
successorGenderEquality(N):- (children(L), member(X, L), not(N=X),
olderThan(N,X), (K=X)) -> print(K),nl,successorGenderEquality(K).
```

Logical Expression:

$$(children(L) \wedge member(X, L) \wedge not(N = X) \wedge olderThan(N, X) \wedge (K = X)) \rightarrow print(K) \wedge nl \wedge successorGenderEquality(K)$$

Here, we remove the gender conditions and simply pass down to the next oldest person in the children list.

Trace of the Result:

```
[trace]  ?- successorGenderEquality(elizabeth).
   Call: (10) successorGenderEquality(elizabeth) ? creep
   Call: (11) children(_25084) ? creep
   Exit: (11) children([charles, ann, andrew, edward]) ? creep
   Call: (11) lists:member(_26610, [charles, ann, andrew, edward]) ? creep
   Exit: (11) lists:member(charles, [charles, ann, andrew, edward]) ? creep
^  Call: (11) not(elizabeth=charles) ? creep
^  Exit: (11) not(user:(elizabeth=charles)) ? creep
   Call: (11) olderThan(elizabeth, charles) ? creep
   Call: (12) older_than(elizabeth, charles) ? creep
   Exit: (12) older_than(elizabeth, charles) ? creep
   Exit: (11) olderThan(elizabeth, charles) ? creep
   Call: (11) _32696=charles ? creep
   Exit: (11) charles=charles ? creep
   Call: (11) print(charles) ? creep
charles
   Exit: (11) print(charles) ? creep
   Call: (11) nl ? creep

   Exit: (11) nl ? creep
   Call: (11) successorGenderEquality(charles) ? creep
   Call: (12) children(_37940) ? creep
   Exit: (12) children([charles, ann, andrew, edward]) ? creep
   Call: (12) lists:member(_39466, [charles, ann, andrew, edward]) ? creep
   Exit: (12) lists:member(charles, [charles, ann, andrew, edward]) ? creep
^  Call: (12) not(charles=charles) ? creep
^  Fail: (12) not(user:(charles=charles)) ? creep
   Redo: (12) lists:member(_39466, [charles, ann, andrew, edward]) ? creep
   Exit: (12) lists:member(ann, [charles, ann, andrew, edward]) ? creep
^  Call: (12) not(charles=ann) ? creep
^  Exit: (12) not(user:(charles=ann)) ? creep
   Call: (12) olderThan(charles, ann) ? creep
   Call: (13) older_than(charles, ann) ? creep
   Exit: (13) older_than(charles, ann) ? creep
   Exit: (12) olderThan(charles, ann) ? creep
   Call: (12) _48620=ann ? creep
   Exit: (12) ann=ann ? creep
   Call: (12) print(ann) ? creep
ann
   Exit: (12) print(ann) ? creep
   Call: (12) nl ? creep

   Exit: (12) nl ? creep
   Call: (12) successorGenderEquality(ann) ? creep
   Call: (13) children(_53864) ? creep
   Exit: (13) children([charles, ann, andrew, edward]) ? creep
   Call: (13) lists:member(_55390, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(charles, [charles, ann, andrew, edward]) ? creep
^  Call: (13) not(ann=charles) ? creep
^  Exit: (13) not(user:(ann=charles)) ? creep
   Call: (13) olderThan(ann, charles) ? creep
   Call: (14) older_than(ann, charles) ? creep
   Fail: (14) older_than(ann, charles) ? creep
   Redo: (13) olderThan(ann, charles) ? creep
   Call: (14) older_than(ann, _61476) ? creep
   Exit: (14) older_than(ann, andrew) ? creep
   Call: (14) older_than(andrew, charles) ? creep
   Fail: (14) older_than(andrew, charles) ? creep
   Fail: (13) olderThan(ann, charles) ? creep
   Redo: (13) lists:member(_142, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(ann, [charles, ann, andrew, edward]) ? creep
^  Call: (13) not(ann=ann) ? creep
^  Fail: (13) not(user:(ann=ann)) ? creep
   Redo: (13) lists:member(_142, [charles, ann, andrew, edward]) ? creep
   Exit: (13) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
```

```
^  Call: (13) not(ann=andrew) ? creep
^  Exit: (13) not(user:(ann=andrew)) ? creep
   Call: (13) olderThan(ann, andrew) ? creep
   Call: (14) older_than(ann, andrew) ? creep
   Exit: (14) older_than(ann, andrew) ? creep
   Exit: (13) olderThan(ann, andrew) ? creep
   Call: (13) _7590=andrew ? creep
   Exit: (13) andrew=andrew ? creep
   Call: (13) print(andrew) ? creep
andrew
   Exit: (13) print(andrew) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Call: (13) successorGenderEquality(andrew) ? creep
   Call: (14) children(_12834) ? creep
   Exit: (14) children([charles, ann, andrew, edward]) ? creep
   Call: (14) lists:member(_14360, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(charles, [charles, ann, andrew, edward]) ? creep
^  Call: (14) not(andrew=charles) ? creep
^  Exit: (14) not(user:(andrew=charles)) ? creep
   Call: (14) olderThan(andrew, charles) ? creep
   Call: (15) older_than(andrew, charles) ? creep
   Fail: (15) older_than(andrew, charles) ? creep
   Redo: (14) olderThan(andrew, charles) ? creep
   Call: (15) older_than(andrew, _20446) ? creep
   Exit: (15) older_than(andrew, edward) ? creep
   Call: (15) older_than(edward, charles) ? creep
   Fail: (15) older_than(edward, charles) ? creep
   Fail: (14) olderThan(andrew, charles) ? creep
   Redo: (14) lists:member(_14360, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(ann, [charles, ann, andrew, edward]) ? creep
^  Call: (14) not(andrew=ann) ? creep
^  Exit: (14) not(user:(andrew=ann)) ? creep
   Call: (14) olderThan(andrew, ann) ? creep
   Call: (15) older_than(andrew, ann) ? creep
   Fail: (15) older_than(andrew, ann) ? creep
   Redo: (14) olderThan(andrew, ann) ? creep
   Call: (15) older_than(andrew, _30302) ? creep
   Exit: (15) older_than(andrew, edward) ? creep
   Call: (15) older_than(edward, ann) ? creep
   Fail: (15) older_than(edward, ann) ? creep
   Fail: (14) olderThan(andrew, ann) ? creep
   Redo: (14) lists:member(_14360, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
^  Call: (14) not(andrew=andrew) ? creep
^  Fail: (14) not(user:(andrew=andrew)) ? creep
   Redo: (14) lists:member(_14360, [charles, ann, andrew, edward]) ? creep
   Exit: (14) lists:member(edward, [charles, ann, andrew, edward]) ? creep
^  Call: (14) not(andrew=edward) ? creep
^  Exit: (14) not(user:(andrew=edward)) ? creep
   Call: (14) olderThan(andrew, edward) ? creep
   Call: (15) older_than(andrew, edward) ? creep
   Exit: (15) older_than(andrew, edward) ? creep
   Exit: (14) olderThan(andrew, edward) ? creep
   Call: (14) _43226=edward ? creep
   Exit: (14) edward=edward ? creep
   Call: (14) print(edward) ? creep
edward
   Exit: (14) print(edward) ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Call: (14) successorGenderEquality(edward) ? creep
   Call: (15) children(_48470) ? creep
```

```
   Call: (15) lists:member(_49996, [charles, ann, andrew, edward]) ? creep
   Exit: (15) lists:member(charles, [charles, ann, andrew, edward]) ? creep
^  Call: (15) not(edward=charles) ? creep
^  Exit: (15) not(user:(edward=charles)) ? creep
   Call: (15) olderThan(edward, charles) ? creep
   Call: (16) older_than(edward, charles) ? creep
   Fail: (16) older_than(edward, charles) ? creep
   Redo: (15) olderThan(edward, charles) ? creep
   Call: (16) older_than(edward, _56082) ? creep
   Fail: (16) older_than(edward, _56082) ? creep
   Fail: (15) olderThan(edward, charles) ? creep
   Redo: (15) lists:member(_49996, [charles, ann, andrew, edward]) ? creep
   Exit: (15) lists:member(ann, [charles, ann, andrew, edward]) ? creep
^  Call: (15) not(edward=ann) ? creep
^  Exit: (15) not(user:(edward=ann)) ? creep
   Call: (15) olderThan(edward, ann) ? creep
   Call: (16) older_than(edward, ann) ? creep
   Fail: (16) older_than(edward, ann) ? creep
   Redo: (15) olderThan(edward, ann) ? creep
   Call: (16) older_than(edward, _206) ? creep
   Fail: (16) older_than(edward, _206) ? creep
   Fail: (15) olderThan(edward, ann) ? creep
   Redo: (15) lists:member(_202, [charles, ann, andrew, edward]) ? creep
   Exit: (15) lists:member(andrew, [charles, ann, andrew, edward]) ? creep
^  Call: (15) not(edward=andrew) ? creep
^  Exit: (15) not(user:(edward=andrew)) ? creep
   Call: (15) olderThan(edward, andrew) ? creep
   Call: (16) older_than(edward, andrew) ? creep
   Fail: (16) older_than(edward, andrew) ? creep
   Redo: (15) olderThan(edward, andrew) ? creep
   Call: (16) older_than(edward, _8554) ? creep
   Fail: (16) older_than(edward, _8554) ? creep
   Fail: (15) olderThan(edward, andrew) ? creep
   Redo: (15) lists:member(_202, [charles, ann, andrew, edward]) ? creep
   Exit: (15) lists:member(edward, [charles, ann, andrew, edward]) ? creep
^  Call: (15) not(edward=edward) ? creep
^  Fail: (15) not(user:(edward=edward)) ? creep
   Fail: (14) successorGenderEquality(edward) ? creep
   Fail: (13) successorGenderEquality(andrew) ? creep
   Fail: (12) successorGenderEquality(ann) ? creep
   Fail: (11) successorGenderEquality(charles) ? creep
   Fail: (10) successorGenderEquality(elizabeth) ? creep
false.
```

Contribution

Exercise 1, **Exercise 2,** **Lab Report**	Chua Zi Jian (U2022354J) Min Kabar Kyaw (U2021858K)