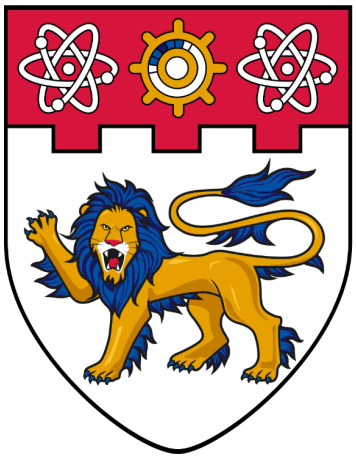

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC3000 Lab Assignment 2
SCS4

Name	Matriculation Number
Lee Wei Jun Alan	U2220730A
Jden Goh	U2222711B
Wei Hong	U2210203B

Exercise 1

sumsum, a competitor of appy, developed some nice smart phone technology called galacticas3, all of which was stolen by stevey, who is a boss of appy. It is unethical for a boss to steal business from rival companies. A competitor is a rival. Smart phone technology is business.

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

Statement: sumsum developed galactica-s3
FOL: `developed(sumsum,galactica-s3)`

Statement: galacticas3 stolen by stevey
FOL: `stole(stevey,galactica-s3)`

Statement: stevey is a boss of appy
FOL: `boss_of(stevey,appy)`

Statement: sumsum is a competitor of appy
FOL:
`rivals(appy,sumsum)`
`rivals(sumsum,appy)`

Statement: It is unethical for a boss to steal business from rival companies.
FOL: `∀X,∀Y,∃A,∃Z,(stole(X,Y) ⋀ boss_of(X,A) ⋀ rivals(A,Z)) -> unethical(X)`

2. Write these FOL statements as Prolog clauses.

```
developed(sumsum,galactica-s3).
stole(stevey,galactica-s3).
boss_of(stevey,appy).
rivals(appy,sumsum).
rivals(sumsum,appy).
unethical(X):-
stole(X,Y),
boss_of(X,A),
rivals(A,Z),
      developed(Z,Y).
```

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

```
[trace]  ?- unethical(stevey)
|    .
   Call: (10) unethical(stevey) ? creep
   Call: (11) stole(stevey, _24426) ? creep
   Exit: (11) stole(stevey, galactica-s3) ? creep
   Call: (11) boss_of(stevey, _26054) ? creep
   Exit: (11) boss_of(stevey, appy) ? creep
   Call: (11) rivals(appy, _27676) ? creep
   Exit: (11) rivals(appy, sumsum) ? creep
   Call: (11) developed(sumsum, galactica-s3) ? creep
   Exit: (11) developed(sumsum, galactica-s3) ? creep
   Exit: (10) unethical(stevey) ? creep
true.
```

Exercise 2

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.

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.

Prolog Knowledge Base:

```
male(prince_charles).
female(princess_anne).
male(prince_andrew).
male(prince_edward).
female(queen_elizabeth).

parent(queen_elizabeth, prince_charles).
parent(queen_elizabeth, princess_anne).
parent(queen_elizabeth, prince_andrew).
parent(queen_elizabeth, prince_edward).

birth_order(1, prince_charles).
birth_order(2, princess_anne).
birth_order(3, prince_andrew).
birth_order(4, prince_edward).

successors(Parent, Successors) :-
    findall(MaleSuccessor, (
        parent(Parent, MaleSuccessor),
        male(MaleSuccessor),
        birth_order(_, MaleSuccessor)
    ), MaleSuccessors),
    findall(FemaleSuccessor, (
        parent(Parent, FemaleSuccessor),
        female(FemaleSuccessor),
```

```
        birth_order(_, FemaleSuccessor)
    ), FemaleSuccessors),
    append(MaleSuccessors, FemaleSuccessors, Successors).
```

```
[trace]  ?- sucessors(queen_elizabeth, X)
Correct to: "successors(queen_elizabeth,X)"? yes
   Call: (12) successors(queen_elizabeth, _2854) ? creep
^  Call: (13) findall(_4960, (parent(queen_elizabeth, _4960), male(_4960), birth_order(_4986, _4960)), _4990) ? creep
   Call: (18) parent(queen_elizabeth, _4960) ? creep
   Exit: (18) parent(queen_elizabeth, prince_charles) ? creep
   Call: (18) male(prince_charles) ? creep
   Exit: (18) male(prince_charles) ? creep
   Call: (18) birth_order(_4986, prince_charles) ? creep
   Exit: (18) birth_order(1, prince_charles) ? creep
   Redo: (18) parent(queen_elizabeth, _4960) ? creep
   Exit: (18) parent(queen_elizabeth, princess_anne) ? creep
   Call: (18) male(princess_anne) ? creep
   Fail: (18) male(princess_anne) ? creep
   Redo: (18) parent(queen_elizabeth, _4960) ? creep
   Exit: (18) parent(queen_elizabeth, prince_andrew) ? creep
   Call: (18) male(prince_andrew) ? creep
   Exit: (18) male(prince_andrew) ? creep
   Call: (18) birth_order(_4986, prince_andrew) ? creep
   Exit: (18) birth_order(3, prince_andrew) ? creep
   Redo: (18) parent(queen_elizabeth, _4960) ? creep
   Exit: (18) parent(queen_elizabeth, prince_edward) ? creep
   Call: (18) male(prince_edward) ? creep
   Exit: (18) male(prince_edward) ? creep
   Call: (18) birth_order(_4986, prince_edward) ? creep
   Exit: (18) birth_order(4, prince_edward) ? creep
^  Exit: (13) findall(_4960, user:(parent(queen_elizabeth, _4960), male(_4960), birth_order(_4986, _4960)), [prince_charles, prince_andrew, prince_edward]) ? creep
^  Call: (13) findall(_24478, (parent(queen_elizabeth, _24478), female(_24478), birth_order(_24504, _24478)), _24508) ? creep
   Call: (18) parent(queen_elizabeth, _24478) ? creep
   Exit: (18) parent(queen_elizabeth, prince_charles) ? creep
   Call: (18) female(prince_charles) ? creep
   Fail: (18) female(prince_charles) ? creep
   Redo: (18) parent(queen_elizabeth, _24478) ? creep
   Exit: (18) parent(queen_elizabeth, princess_anne) ? creep
   Call: (18) female(princess_anne) ? creep
   Exit: (18) female(princess_anne) ? creep
   Call: (18) birth_order(_164, princess_anne) ? creep
   Exit: (18) birth_order(2, princess_anne) ? creep
   Redo: (18) parent(queen_elizabeth, _138) ? creep
   Exit: (18) parent(queen_elizabeth, prince_andrew) ? creep
   Call: (18) female(prince_andrew) ? creep
   Fail: (18) female(prince_andrew) ? creep
   Redo: (18) parent(queen_elizabeth, _138) ? creep
   Exit: (18) parent(queen_elizabeth, prince_edward) ? creep
   Call: (18) female(prince_edward) ? creep
   Fail: (18) female(prince_edward) ? creep
^  Exit: (13) findall(_138, user:(parent(queen_elizabeth, _138), female(_138), birth_order(_164, _138)), [princess_anne]) ? creep
   Call: (13) lists:append([prince_charles, prince_andrew, prince_edward], [princess_anne], _58) ? creep
   Exit: (13) lists:append([prince_charles, prince_andrew, prince_edward], [princess_anne], [prince_charles, prince_andrew, prince_edward, princess_anne]) ? creep
   Exit: (12) successors(queen_elizabeth, [prince_charles, prince_andrew, prince_edward, princess_anne]) ? creep
X = [prince_charles, prince_andrew, prince_edward, princess_anne].

[trace]  ?-
```

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

Modified Prolog Base Knowledge:

```
parent(queen_elizabeth, prince_charles).
parent(queen_elizabeth, princess_anne).
parent(queen_elizabeth, prince_andrew).
parent(queen_elizabeth, prince_edward).

birth_order(1, prince_charles).
birth_order(2, princess_anne).
birth_order(3, prince_andrew).
birth_order(4, prince_edward).

successors(Parent, Successors) :-
    findall(Child, (
        parent(Parent, Child),
        birth_order(_, Child)
    ), Successors).
```

Explanation:

We removed the gender predicates of ‘male’ and ‘female’ as they are no longer needed. The successor's rule only considers birth order now, ignoring whether the successor is a male or female.

```
[trace]  ?- sucessors(queen_elizabeth, X).
Correct to: "successors(queen_elizabeth,X)"? yes
   Call: (12) successors(queen_elizabeth, _24972) ? creep
^  Call: (13) findall(_27078, (parent(queen_elizabeth, _27078), birth_order(_27094, _27078)), _24972) ? creep
   Call: (18) parent(queen_elizabeth, _27078) ? creep
   Exit: (18) parent(queen_elizabeth, prince_charles) ? creep
   Call: (18) birth_order(_27094, prince_charles) ? creep
   Exit: (18) birth_order(1, prince_charles) ? creep
   Redo: (18) parent(queen_elizabeth, _27078) ? creep
   Exit: (18) parent(queen_elizabeth, princess_anne) ? creep
   Call: (18) birth_order(_27094, princess_anne) ? creep
   Exit: (18) birth_order(2, princess_anne) ? creep
   Redo: (18) parent(queen_elizabeth, _27078) ? creep
   Exit: (18) parent(queen_elizabeth, prince_andrew) ? creep
   Call: (18) birth_order(_27094, prince_andrew) ? creep
   Exit: (18) birth_order(3, prince_andrew) ? creep
   Redo: (18) parent(queen_elizabeth, _27078) ? creep
   Exit: (18) parent(queen_elizabeth, prince_edward) ? creep
   Call: (18) birth_order(_27094, prince_edward) ? creep
   Exit: (18) birth_order(4, prince_edward) ? creep
^  Exit: (13) findall(_27078, user:(parent(queen_elizabeth, _27078), birth_order(_27094, _27078)), [prince_charles, princess_anne, prince_andrew, prince_edward]) ? creep
   Exit: (12) successors(queen_elizabeth, [prince_charles, princess_anne, prince_andrew, prince_edward]) ? creep
X = [prince_charles, princess_anne, prince_andrew, prince_edward].
```