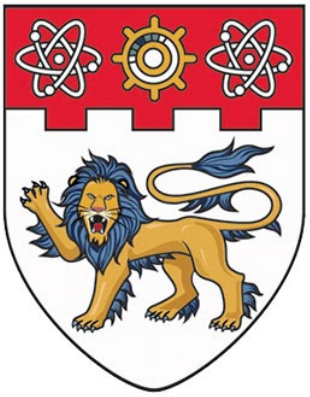

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering

SC3000/CZ3005: Artificial Intelligence

Assignment: *Lab Assignment 2 - Introduction to Logic Programming*

2022/2023 Semester 2

Tutorial/Lab Group	A42
Team Name	Pole Dancer

Members:

HENDY	U212255J
BRENDON TAN	U2121250H
WONG RI HONG	U2123580K

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 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).

(5 marks)

2. Write these FOL statements as Prolog clauses.

(5 marks)

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

(5 marks)

Exercise 1.1

Natural Language Statements	First Order Logic Statements
sumsum, a competitor of appy	Company(sumsum) Company(appy) Competitors(sumsum, appy) ∀ x. ∀ y. (Competitors(x, y) ↔ Competitors(y, x))
sumsum…developed some nice smartphone technology called galactica-s3	Develop(sumsum, galactica-s3) SmartphoneTech(galactica-s3)
galactica-s3…was stolen by stevey	Steal(stevey, galactica-s3)
stevey, who is a boss of appy.	Boss(stevey, appy)
It is unethical for a boss to steal business from rival companies.	∀ a. ∀ b. ∀ c. ∀ d. (Boss(a, b) ∧ Steal(a, d) ∧ Business(d) ∧ Develop(c, d) ∧ Rival(b, c) ∧ Company(c) → Unethical(a)) # a=boss, b=company, c=company, d=smartphonetech
A competitor is a rival.	∀ x. ∀ y. (Competitors(x,y) → Rival(x,y))
Smart phone technology is business.	∀ x. (SmartphoneTech(x) → Business(x))

Exercise 1.2

```
/* relations */
company(sumsum).
company(appy).
competitors(sumsum, appy).
develop(sumsum, galactica-s3).
smartphonetech(galactica-s3).
steal(stevey, galactica-s3).
boss(stevey, appy).

/* rules */
competitors(X, Y):-
        competitors(Y, X).

rival(X, Y):-
        competitors(X, Y).

business(X):-
        smartphonetech(X).

unethical(A):-
        boss(A, B), steal(A, D), business(D), develop(C, D), rival(B, C), company(C)
```

Exercise 1.3

```
?- trace, unethical(stevey).
   Call: (11) unethical(stevey) ? creep
   Call: (12) boss(stevey, _2526) ? creep
   Exit: (12) boss(stevey, appy) ? creep
   Call: (12) steal(stevey, _4148) ? creep
   Exit: (12) steal(stevey, galactica-s3) ? creep
   Call: (12) business(galactica-s3) ? creep
   Call: (13) smartphonetech(galactica-s3) ? creep
   Exit: (13) smartphonetech(galactica-s3) ? creep
   Exit: (12) business(galactica-s3) ? creep
   Call: (12) develop(_9000, galactica-s3) ? creep
   Exit: (12) develop(sumsum, galactica-s3) ? creep
   Call: (12) rival(appy, sumsum) ? creep
   Call: (13) competitors(appy, sumsum) ? creep
   Call: (14) competitors(sumsum, appy) ? creep
   Exit: (14) competitors(sumsum, appy) ? creep
   Exit: (13) competitors(appy, sumsum) ? creep
   Exit: (12) rival(appy, sumsum) ? creep
   Call: (12) company(sumsum) ? creep
   Exit: (12) company(sumsum) ? creep
   Exit: (11) 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.

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)

Exercise 2.1 Determine the line of succession

```
[trace]  ?- trace, sortedSuccessionList(queen_elizabeth, X).
   Call: (11) sortedSuccessionList(queen_elizabeth, _970) ? creep
^  Call: (12) findall(_2376, offspring(_2376, queen_elizabeth), _2384) ? creep
   Call: (17) offspring(_2376, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_charles, queen_elizabeth) ? creep
   Redo: (17) offspring(_2376, queen_elizabeth) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Redo: (17) offspring(_2376, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Redo: (17) offspring(_2376, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_edward, queen_elizabeth) ? creep
^  Exit: (12) findall(_2376, user:offspring(_2376, queen_elizabeth), [prince_charles, princess
_ann, prince_andrew, prince_edward]) ? creep
   Call: (12) sort_succession([prince_charles, princess_ann, prince_andrew, prince_edward], _9
70) ? creep
   Call: (13) sort_succession([princess_ann, prince_andrew, prince_edward], _11386) ? creep
   Call: (14) sort_succession([prince_andrew, prince_edward], _12198) ? creep
   Call: (15) sort_succession([prince_edward], _13010) ? creep
   Call: (16) sort_succession([], _13822) ? creep
   Exit: (16) sort_succession([], []) ? creep
   Call: (16) insert(prince_edward, [], _13010) ? creep
   Exit: (16) insert(prince_edward, [], [prince_edward]) ? creep
   Exit: (15) sort_succession([prince_edward], [prince_edward]) ? creep
   Call: (15) insert(prince_andrew, [prince_edward], _12198) ? creep
^  Call: (16) not(precedes(prince_andrew, prince_edward)) ? creep
   Call: (17) precedes(prince_andrew, prince_edward) ? creep
   Call: (18) offspring(prince_andrew, _20348) ? creep
   Exit: (18) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) male(prince_andrew) ? creep
   Exit: (18) male(prince_andrew) ? creep
   Call: (18) female(prince_edward) ? creep
   Fail: (18) female(prince_edward) ? creep
   Redo: (17) precedes(prince_andrew, prince_edward) ? creep
   Call: (18) offspring(prince_andrew, _27624) ? creep
   Exit: (18) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) male(prince_andrew) ? creep
   Exit: (18) male(prince_andrew) ? creep
   Call: (18) male(prince_edward) ? creep
   Exit: (18) male(prince_edward) ? creep
   Call: (18) is_older(prince_andrew, prince_edward) ? creep
   Call: (19) older(prince_andrew, prince_edward) ? creep
   Exit: (19) older(prince_andrew, prince_edward) ? creep
   Exit: (18) is_older(prince_andrew, prince_edward) ? creep
   Exit: (17) precedes(prince_andrew, prince_edward) ? creep
^  Fail: (16) not(user:precedes(prince_andrew, prince_edward)) ? creep
   Redo: (15) insert(prince_andrew, [prince_edward], _12198) ? creep
   Exit: (15) insert(prince_andrew, [prince_edward], [prince_andrew, prince_edward])
 ? creep
```

(continued below)

```
   Exit: (14) sort_succession([prince_andrew, prince_edward], [prince_andrew, prince_e
dward])
 ? creep
   Call: (14) insert(princess_ann, [prince_andrew, prince_edward], _11386) ? creep
^  Call: (15) not(precedes(princess_ann, prince_andrew)) ? creep
   Call: (16) precedes(princess_ann, prince_andrew) ? creep
   Call: (17) offspring(princess_ann, _43862) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (17) male(princess_ann) ? creep
   Fail: (17) male(princess_ann) ? creep
   Redo: (16) precedes(princess_ann, prince_andrew)
 ? creep
   Call: (17) offspring(princess_ann, _49526) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (17) male(princess_ann) ? creep
   Fail: (17) male(princess_ann) ? creep
   Redo: (16) precedes(princess_ann, prince_andrew) ? creep
   Call: (17) offspring(princess_ann, _55190) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (17) female(princess_ann) ? creep
   Exit: (17) female(princess_ann) ? creep
   Call: (17) female(prince_andrew) ? creep
   Fail: (17) female(prince_andrew) ? creep
   Fail: (16) precedes(princess_ann, prince_andrew) ? creep
^  Exit: (15) not(user:precedes(princess_ann, prince_andrew)) ? creep
   Call: (15) insert(princess_ann, [prince_edward], _42220) ? creep
^  Call: (16) not(precedes(princess_ann, prince_edward)) ? creep
   Call: (17) precedes(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _1636) ? creep
   Exit: (18) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) male(princess_ann) ? creep
   Fail: (18) male(princess_ann) ? creep
   Redo: (17) precedes(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _7300) ? creep
   Exit: (18) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) male(princess_ann) ? creep
   Fail: (18) male(princess_ann) ? creep
   Redo: (17) precedes(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _12964) ? creep
   Exit: (18) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) female(princess_ann) ? creep
   Exit: (18) female(princess_ann) ? creep
   Call: (18) female(prince_edward) ? creep
   Fail: (18) female(prince_edward) ? creep
   Fail: (17) precedes(princess_ann, prince_edward) ? creep
```

(continued below)

```
^  Exit: (16) not(user:precedes(princess_ann, prince_edward)) ? creep
   Call: (16) insert(princess_ann, [], _138) ? creep
   Exit: (16) insert(princess_ann, [], [princess_ann]) ? creep
   Exit: (15) insert(princess_ann, [prince_edward], [prince_edward, princess_ann]) ? creep
   Exit: (14) insert(princess_ann, [prince_andrew, prince_edward], [prince_andrew, prince_
edward, princess_ann]) ? creep
   Exit: (13) sort_succession([princess_ann, prince_andrew, prince_edward], [prince_andrew
, prince_edward, princess_ann]) ? creep
   Call: (13) insert(prince_charles, [prince_andrew, prince_edward, princess_ann], _18) ?
creep
^  Call: (14) not(precedes(prince_charles, prince_andrew)) ? creep
   Call: (15) precedes(prince_charles, prince_andrew) ? creep
   Call: (16) offspring(prince_charles, _27590) ? creep
   Exit: (16) offspring(prince_charles, queen_elizabeth) ? creep
   Call: (16) offspring(prince_andrew, queen_elizabeth) ? creep
   Exit: (16) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (16) male(prince_charles) ? creep
   Exit: (16) male(prince_charles) ? creep
   Call: (16) female(prince_andrew) ? creep
   Fail: (16) female(prince_andrew) ? creep
   Redo: (15) precedes(prince_charles, prince_andrew) ? creep
   Call: (16) offspring(prince_charles, _34866) ? creep
   Exit: (16) offspring(prince_charles, queen_elizabeth) ? creep
   Call: (16) offspring(prince_andrew, queen_elizabeth) ? creep
   Exit: (16) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (16) male(prince_charles) ? creep
   Exit: (16) male(prince_charles) ? creep
   Call: (16) male(prince_andrew) ? creep
   Exit: (16) male(prince_andrew) ? creep
   Call: (16) is_older(prince_charles, prince_andrew) ? creep
   Call: (17) older(prince_charles, prince_andrew) ? creep
   Fail: (17) older(prince_charles, prince_andrew) ? creep
   Redo: (16) is_older(prince_charles, prince_andrew) ? creep
   Call: (17) older(prince_charles, _44572) ? creep
   Exit: (17) older(prince_charles, princess_ann) ? creep
   Call: (17) is_older(princess_ann, prince_andrew) ? creep
   Call: (18) older(princess_ann, prince_andrew) ? creep
   Exit: (18) older(princess_ann, prince_andrew) ? creep
   Exit: (17) is_older(princess_ann, prince_andrew) ? creep
   Exit: (16) is_older(prince_charles, prince_andrew) ? creep
   Exit: (15) precedes(prince_charles, prince_andrew) ? creep
^  Fail: (14) not(user:precedes(prince_charles, prince_andrew))
 ? creep
   Redo: (13) insert(prince_charles, [prince_andrew, prince_edward, princess_ann], _18)
 ? creep
   Exit: (13) insert(prince_charles, [prince_andrew, prince_edward, princess_ann], [prince
_charles, prince_andrew, prince_edward, princes
s_ann]) ? creep
   Exit: (12) sort_succession([prince_charles, princess_ann, prince_andrew, prince_edward]
, [prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
   Exit: (11) sortedSuccessionList(queen_elizabeth, [prince_charles, prince_andrew, prince
_edward, princess_ann]) ? creep
X = [prince_charles, prince_andrew, prince_edward, princess_ann].
```

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)

Exercise 2.2 Determine the line of succession(modified)

These are the original rules from Exercise 2.1:

```
% Rule 1: Male child will always come before female child
precedes(X, Y):-
        offspring(X, A), offspring(Y, A),
        male(X), female(Y),
        not(queen(Y)).

% Rule 2: Older male child will come before younger male child
precedes(X, Y):-
        offspring(X, A), offspring(Y, A),
        male(X), male(Y),
        is_older(X, Y).

% Rule 3: Older female child will come before younger female child
precedes(X, Y):-
        offspring(X, A), offspring(Y, A),
        female(X), female(Y),
        is_older(X, Y),
        not(queen(X)), not(queen(Y)).
```

- The first rule is removed as succession is no longer based on gender.
- The second and third rules are combined such that an older child will come before a younger child instead.

The updated ruling:

```
% Rule 1: Older child will come before younger child
precedes(X, Y):-
        offspring(X, A), offspring(Y, A),
        is_older(X, Y),
        not(queen(X)), not(queen(Y)).
```

Trace for Exercise 2.2

```
?- trace, sortedSuccessionList(queen_elizabeth, X).
   Call: (11) sortedSuccessionList(queen_elizabeth, _752) ? creep
^  Call: (12) findall(_2158, offspring(_2158, queen_elizabeth), _2166) ? creep
   Call: (17) offspring(_2158, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_charles, queen_elizabeth) ? creep
   Redo: (17) offspring(_2158, queen_elizabeth) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Redo: (17) offspring(_2158, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Redo: (17) offspring(_2158, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_edward, queen_elizabeth) ? creep
^  Exit: (12) findall(_2158, user:offspring(_2158, queen_elizabeth), [prince_charles, princess_
ann, prince_andrew, prince_edward]) ? creep
   Call: (12) sort_succession([prince_charles, princess_ann, prince_andrew, prince_edward], _75
2) ? creep
   Call: (13) sort_succession([princess_ann, prince_andrew, prince_edward], _11168) ? creep
   Call: (14) sort_succession([prince_andrew, prince_edward], _11980) ? creep
   Call: (15) sort_succession([prince_edward], _12792) ? creep
   Call: (16) sort_succession([], _13604) ? creep
   Exit: (16) sort_succession([], []) ? creep
   Call: (16) insert(prince_edward, [], _12792) ? creep
   Exit: (16) insert(prince_edward, [], [prince_edward]) ? creep
   Exit: (15) sort_succession([prince_edward], [prince_edward]) ? creep
   Call: (15) insert(prince_andrew, [prince_edward], _11980) ? creep
^  Call: (16) not(precedes(prince_andrew, prince_edward)) ? creep
   Call: (17) precedes(prince_andrew, prince_edward) ? creep
   Call: (18) offspring(prince_andrew, _20130) ? creep
   Exit: (18) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) is_older(prince_andrew, prince_edward) ? creep
   Call: (19) older(prince_andrew, prince_edward) ? creep
   Exit: (19) older(prince_andrew, prince_edward) ? creep
   Exit: (18) is_older(prince_andrew, prince_edward) ? creep
^  Call: (18) not(queen(prince_andrew)) ? creep
   Call: (19) queen(prince_andrew) ? creep
   Fail: (19) queen(prince_andrew) ? creep
^  Exit: (18) not(user:queen(prince_andrew)) ? creep
^  Call: (18) not(queen(prince_edward)) ? creep
   Call: (19) queen(prince_edward) ? creep
   Fail: (19) queen(prince_edward) ? creep
^  Exit: (18) not(user:queen(prince_edward)) ? creep
   Exit: (17) precedes(prince_andrew, prince_edward) ? creep
^  Fail: (16) not(user:precedes(prince_andrew, prince_edward)) ? creep
   Redo: (15) insert(prince_andrew, [prince_edward], _72) ? creep
   Exit: (15) insert(prince_andrew, [prince_edward], [prince_andrew, prince_edward]) ? creep
   Exit: (14) sort_succession([prince_andrew, prince_edward], [prince_andrew, prince_edward]) ?
 creep
   Call: (14) insert(princess_ann, [prince_andrew, prince_edward], _70) ? creep
^  Call: (15) not(precedes(princess_ann, prince_andrew)) ? creep
   Call: (16) precedes(princess_ann, prince_andrew) ? creep
   Call: (17) offspring(princess_ann, _8264) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
```

(continued below)

```
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (17) is_older(princess_ann, prince_andrew) ? creep
   Call: (18) older(princess_ann, prince_andrew) ? creep
   Exit: (18) older(princess_ann, prince_andrew) ? creep
   Exit: (17) is_older(princess_ann, prince_andrew) ? creep
^  Call: (17) not(queen(princess_ann)) ? creep
   Call: (18) queen(princess_ann) ? creep
   Fail: (18) queen(princess_ann) ? creep
^  Exit: (17) not(user:queen(princess_ann)) ? creep
^  Call: (17) not(queen(prince_andrew)) ? creep
   Call: (18) queen(prince_andrew) ? creep
   Fail: (18) queen(prince_andrew) ? creep
^  Exit: (17) not(user:queen(prince_andrew)) ? creep
   Exit: (16) precedes(princess_ann, prince_andrew) ? creep
^  Fail: (15) not(user:precedes(princess_ann, prince_andrew)) ? creep
   Redo: (14) insert(princess_ann, [prince_andrew, prince_edward], _70) ? creep
   Exit: (14) insert(princess_ann, [prince_andrew, prince_edward], [princess_ann, prince_andrew
, prince_edward]) ? creep
   Exit: (13) sort_succession([princess_ann, prince_andrew, prince_edward], [princess_ann, prin
ce_andrew, prince_edward]) ? creep
   Call: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], _18) ? creep
^  Call: (14) not(precedes(prince_charles, princess_ann)) ? creep
   Call: (15) precedes(prince_charles, princess_ann) ? creep
   Call: (16) offspring(prince_charles, _27794) ? creep
   Exit: (16) offspring(prince_charles, queen_elizabeth) ? creep
   Call: (16) offspring(princess_ann, queen_elizabeth) ? creep
   Exit: (16) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (16) is_older(prince_charles, princess_ann) ? creep
   Call: (17) older(prince_charles, princess_ann) ? creep
   Exit: (17) older(prince_charles, princess_ann) ? creep
   Exit: (16) is_older(prince_charles, princess_ann) ? creep
^  Call: (16) not(queen(prince_charles)) ? creep
   Call: (17) queen(prince_charles) ? creep
   Fail: (17) queen(prince_charles) ? creep
^  Exit: (16) not(user:queen(prince_charles)) ? creep
^  Call: (16) not(queen(princess_ann)) ? creep
   Call: (17) queen(princess_ann) ? creep
   Fail: (17) queen(princess_ann) ? creep
^  Exit: (16) not(user:queen(princess_ann)) ? creep
   Exit: (15) precedes(prince_charles, princess_ann) ? creep
^  Fail: (14) not(user:precedes(prince_charles, princess_ann)) ? creep
   Redo: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], _18) ? creep
   Exit: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], [prince_char
les, princess_ann, prince_andrew, prince_edward]) ? creep
   Exit: (12) sort_succession([prince_charles, princess_ann, prince_andrew, prince_edward], [pr
ince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
   Exit: (11) sortedSuccessionList(queen_elizabeth, [prince_charles, princess_ann, prince_andre
w, prince_edward]) ? creep
X = [prince_charles, princess_ann, prince_andrew, prince_edward].
```