

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ3005
Artificial Intelligence

Lab Exercise 3: Introduction to Prolog

DSAI

U1821610C

Liew Zhi Li (Sherna)

Contents

Exercise 1 – The Smart Phone Rivalry	2
1.1 – Translate the natural language statements to First Order Logic (FOL)	2
1.2 – Prolog clauses (LiewZhiLi_qn1_2.pl)	2
1.3 – Trace of proof that Stevey is unethical	2
Exercise 2 – The Royal Family	3
2.1 – Prolog clauses for Q2 Part 1 (LiewZhiLi_qn2_1.pl)	3
2.2 – Trace of proof for old Royal succession	4
2.3 – Prolog clauses for Q2 Part 2 (LiewZhiLi_qn2_2.pl)	7
2.4 – Trace of proof for new Royal succession	8

Exercise 1 – The Smart Phone Rivalry

1.1 – Translate the natural language statements to First Order Logic (FOL)

sumsum, a competitor of appy, developed some nice smart phone technology called galacticas3, 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.

- company(appy).
- company(sumsum).
- competitor(sumsum, appy).
- smartPhoneTechnology(galacticas3).
- developed(galacticas3, sumsum).
- boss(stevey).
- steal(stevey, galacticas3).
- $\forall \text{comp competitor}(\text{comp}, \text{appy}) \vee \text{competitor}(\text{appy}, \text{comp}) \Rightarrow \text{rival}(\text{comp})$
- $\forall \text{tech smartPhoneTechnology}(\text{tech}) \Rightarrow \text{business}(\text{tech})$
- $\forall \text{person, tech, comp boss}(\text{person}) \wedge \text{steal}(\text{person}, \text{tech}) \wedge \text{business}(\text{tech}) \wedge \text{developed}(\text{tech}, \text{comp}) \wedge \text{rival}(\text{comp}) \Rightarrow \text{unethical}(\text{person})$

1.2 – Prolog clauses (LiewZhiLi_qn1_2.pl)

```
company(appy).
company(sumsum).
competitor(sumsum, appy).
smartPhoneTechnology(galacticas3).
developed(galacticas3, sumsum).
boss(stevey).
steal(stevey, galacticas3).
rival(Comp):- competitor(Comp, appy);competitor(appy, Comp).
business(Tech):- smartPhoneTechnology(Tech).
unethical(Person):- boss(Person), steal(Person, Tech), business(Tech), developed(Tech, Comp),
rival(Comp).
```

1.3 – Trace of proof that Stevey is unethical

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

?- trace, unethical(stevey).
Call: (11) unethical(stevey) ? creep
Call: (12) boss(stevey) ? creep
Exit: (12) boss(stevey) ? creep
Call: (12) steal(stevey, _8550) ? creep
Exit: (12) steal(stevey, galacticas3) ? creep
Call: (12) business(galacticas3) ? creep
Call: (13) smartPhoneTechnology(galacticas3) ? creep
Exit: (13) smartPhoneTechnology(galacticas3) ? creep
Exit: (12) business(galacticas3) ? creep
Call: (12) developed(galacticas3, _8814) ? creep
Exit: (12) developed(galacticas3, sumsum) ? creep
Call: (12) rival(sumsum) ? creep
Call: (13) competitor(sumsum, appy) ? creep
Exit: (13) competitor(sumsum, appy) ? creep
Exit: (12) rival(sumsum) ? creep
Exit: (11) unethical(stevey) ? creep
true .

[trace] ?- ■
```

Exercise 2 – The Royal Family

2.1 – Prolog clauses for Q2 Part 1 (LiewZhiLi_qn2_1.pl)

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

male(X) :- offspring(prince, X).
female(Y) :- offspring(princess, Y).

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

is_older(A, B):- older(A, B).
is_older(A, B):- older(A, M), is_older(M, B).

then(A, B) :- offspring(prince, A), offspring(princess, B).
then(A, B) :- offspring(prince, A), offspring(prince, B), is_older(A, B).
then(A, B) :- offspring(princess, A), offspring(princess, B), is_older(A, B).

successors(X, Y) :- insert_sort(X, Y).

insert_sort(X, Y) :- i_sort(X, [], Y).
i_sort([], Acc, Acc).
i_sort([H|T], Acc, Y) :- insert(H, Acc, NewAcc), i_sort(T, NewAcc, Y).

insert(X, [], [X]).
insert(X, [Y|T], [X, Y|T]) :- then(X, Y).
insert(X, [Y|T], [Y|NewT]) :- not(then(X, Y)), insert(X, T, NewT).

oldRoyalSuccession(OldRoyalSuccession):- findall(Y,offspring(_,Y), Offsprings),
successors(Offsprings,OldRoyalSuccession).
```

Explanation:

1. We define the facts for the four offsprings: prince charles, princess ann, prince andrew, prince edward.
2. Next, we define the order of birth for the four offsprings.
3. Then, define the rules to determine which is the older offspring.
 - a. There are 2 definitions of is_older(A,B).
 - b. A is older than B OR
 - c. A is older than M who is older than B.
4. Since the old Royal succession rule states than the throne is passed down along the male line according to the order of birth followed by the female line according to the order of birth, we define 3 definitions of then(A,B) rules.
5. We use a sorting algorithm called insertion sort to sort the offspring to get a list of offspring following the abovementioned order.
6. The result is OldRoyalSuccession = [charles, andrew, edward, ann] .


2.2 – Trace of proof for old Royal succession

```

SWI-Prolog (AMD64, Multi-threaded, version 8.2.1)
File Edit Settings Run Debug Help
?- oldRoyalSuccession(OldRoyalSuccession).
OldRoyalSuccession = [charles, andrew, edward, ann] .

?- trace, oldRoyalSuccession(OldRoyalSuccession).
^
Call: (11) oldRoyalSuccession(_8374) ? creep
Call: (12) findall(_8860, offspring(_8858, _8860), _8920) ? creep
Call: (17) offspring(_8858, _8860) ? creep
Exit: (17) offspring(prince, charles) ? creep
Redo: (17) offspring(_8858, _8860) ? creep
Exit: (17) offspring(princess, ann) ? creep
Redo: (17) offspring(_8858, _8860) ? creep
Exit: (17) offspring(prince, andrew) ? creep
Redo: (17) offspring(_8858, _8860) ? creep
Exit: (17) offspring(prince, edward) ? creep
^
Call: (12) findall(_8860, user:offspring(_8858, _8860), [charles, ann, andrew, edward]) ? creep
Call: (12) successors([charles, ann, andrew, edward], _8374) ? creep
Call: (13) insert_sort([charles, ann, andrew, edward], _8374) ? creep
Call: (14) i_sort([charles, ann, andrew, edward], [], _8374) ? creep
Call: (15) insert(charles, [], _9550) ? creep
Exit: (15) insert(charles, [], [charles]) ? creep
Call: (15) i_sort([ann, andrew, edward], [charles], _8374) ? creep
Call: (16) insert(ann, [charles], _9688) ? creep
Call: (17) then(ann, charles) ? creep
Call: (18) offspring(prince, ann) ? creep
Fail: (18) offspring(prince, ann) ? creep
Redo: (17) then(ann, charles) ? creep
Call: (18) offspring(prince, ann) ? creep
Fail: (18) offspring(prince, ann) ? creep
Redo: (17) then(ann, charles) ? creep
Call: (18) offspring(princess, ann) ? creep
Exit: (18) offspring(princess, ann) ? creep
Call: (18) offspring(princess, charles) ? creep
Fail: (18) offspring(princess, charles) ? creep
Fail: (17) then(ann, charles) ? creep
Redo: (16) insert(ann, [charles], _10272) ? creep
^
Call: (17) not(then(ann, charles)) ? creep
Call: (18) then(ann, charles) ? creep
Call: (19) offspring(prince, ann) ? creep
Fail: (19) offspring(prince, ann) ? creep
Redo: (18) then(ann, charles) ? creep
Call: (19) offspring(prince, ann) ? creep
Fail: (19) offspring(prince, ann) ? creep
Redo: (18) then(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) then(ann, charles) ? creep
^
Exit: (17) not(user:then(ann, charles)) ? creep
Call: (17) insert(ann, [], _10262) ? creep
Exit: (17) insert(ann, [], [ann]) ? creep
Exit: (16) insert(ann, [charles], [charles, ann]) ? creep
Call: (16) i_sort([andrew, edward], [charles, ann], _8374) ? creep
Call: (17) insert(andrew, [charles, ann], _11132) ? creep
Call: (18) then(andrew, charles) ? creep
Call: (19) offspring(prince, andrew) ? creep
Exit: (19) offspring(prince, andrew) ? creep
Call: (19) offspring(princess, charles) ? creep
Fail: (19) offspring(princess, charles) ? creep
Redo: (18) then(andrew, charles) ? creep
Call: (19) offspring(prince, andrew) ? creep
Exit: (19) offspring(prince, andrew) ? creep
Call: (19) offspring(prince, charles) ? creep
Exit: (19) offspring(prince, charles) ? creep
Call: (19) is_older(andrew, charles) ? creep
Call: (20) older(andrew, charles) ? creep
Fail: (20) older(andrew, charles) ? creep
Redo: (19) is_older(andrew, charles) ? creep
Call: (20) older(andrew, _11802) ? creep
Exit: (20) older(andrew, edward) ? creep
Call: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, charles) ? creep
Fail: (21) older(edward, charles) ? creep
Redo: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, _12066) ? creep
Fail: (21) older(edward, _12110) ?

```


SWI-Prolog (AMD64, Multi-threaded, version 8.2.1)

File Edit Settings Run Debug Help

```

Fail: (20) is_older(edward, charles) ? creep
Fail: (19) is_older(andrew, charles) ? creep
Redo: (18) then(andrew, charles) ? creep
Call: (19) offspring(princess, andrew) ? creep
Fail: (19) offspring(princess, andrew) ? creep
Fail: (18) then(andrew, charles) ? creep
Redo: (17) insert(andrew, [charles, ann], _12420) ? creep
Call: (18) not(then(andrew, charles)) ? creep
Call: (19) then(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) then(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) is_older(andrew, charles) ? creep
Call: (21) older(andrew, charles) ? creep
Fail: (21) older(andrew, charles) ? creep
Redo: (20) is_older(andrew, charles) ? creep
Call: (21) older(andrew, _13140) ? creep
Exit: (21) older(andrew, edward) ? creep
Call: (21) is_older(edward, charles) ? creep
Call: (22) older(edward, charles) ? creep
Fail: (22) older(edward, charles) ? creep
Redo: (21) is_older(edward, charles) ? creep
Call: (22) older(edward, _13404) ? creep
Fail: (22) older(edward, _13448) ? creep
Fail: (21) is_older(edward, charles) ? creep
Fail: (20) is_older(andrew, charles) ? creep
Redo: (19) then(andrew, charles) ? creep
Call: (20) offspring(princess, andrew) ? creep
Fail: (20) offspring(princess, andrew) ? creep
Fail: (19) then(andrew, charles) ? creep
Exit: (18) not(user:then(andrew, charles)) ? creep
Call: (18) insert(andrew, [ann], _12410) ? creep
Call: (19) then(andrew, ann) ? creep
Call: (20) offspring(prince, andrew) ? creep
Exit: (20) offspring(prince, andrew) ? creep
Call: (20) offspring(princess, ann) ? creep
Exit: (20) offspring(princess, ann) ? creep
Exit: (19) then(andrew, ann) ? creep
Exit: (18) insert(andrew, [ann], [andrew, ann]) ? creep
Exit: (17) insert(andrew, [charles, ann], [charles, andrew, ann]) ? creep
Call: (17) i_sort([edward], [charles, andrew, ann], _8374) ? creep
Call: (18) insert(edward, [charles, andrew, ann], _14254) ? creep
Call: (19) then(edward, charles) ? creep
Call: (20) offspring(prince, edward) ? creep
Exit: (20) offspring(prince, edward) ? creep
Call: (20) offspring(princess, charles) ? creep
Fail: (20) offspring(princess, charles) ? creep
Redo: (19) then(edward, charles) ? creep
Call: (20) offspring(prince, edward) ? creep
Exit: (20) offspring(prince, edward) ? creep
Call: (20) offspring(prince, charles) ? creep
Exit: (20) offspring(prince, charles) ? creep
Call: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, charles) ? creep
Fail: (21) older(edward, charles) ? creep
Redo: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, _14924) ? creep
Fail: (21) older(edward, _14968) ? creep
Fail: (20) is_older(edward, charles) ? creep
Redo: (19) then(edward, charles) ? creep
Call: (20) offspring(princess, edward) ? creep
Fail: (20) offspring(princess, edward) ? creep
Fail: (19) then(edward, charles) ? creep
Redo: (18) insert(edward, [charles, andrew, ann], _15234) ? creep
Call: (19) not(then(edward, charles)) ? creep
Call: (20) then(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

```

```

SWI-Prolog (AMD64, Multi-threaded, version 8.2.1)
File Edit Settings Run Debug Help
Redo: (20) then(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) is_older(edward, charles) ? creep
Call: (22) older(edward, charles) ? creep
Fail: (22) older(edward, charles) ? creep
Redo: (21) is_older(edward, charles) ? creep
Call: (22) older(edward, _15954) ? creep
Fail: (22) older(edward, _15998) ? creep
Fail: (21) is_older(edward, charles) ? creep
Redo: (20) then(edward, charles) ? creep
Call: (21) offspring(princess, edward) ? creep
Fail: (21) offspring(princess, edward) ? creep
Fail: (20) then(edward, charles) ? creep
Exit: (19) not(user:then(edward, charles)) ? creep
Call: (19) insert(edward, [andrew, ann], _15224) ? creep
Call: (20) then(edward, andrew) ? creep
Call: (21) offspring(prince, edward) ? creep
Exit: (21) offspring(prince, edward) ? creep
Call: (21) offspring(princess, andrew) ? creep
Fail: (21) offspring(princess, andrew) ? creep
Redo: (20) then(edward, andrew) ? creep
Call: (21) offspring(prince, edward) ? creep
Exit: (21) offspring(prince, edward) ? creep
Call: (21) offspring(prince, andrew) ? creep
Exit: (21) offspring(prince, andrew) ? creep
Call: (21) is_older(edward, andrew) ? creep
Call: (22) older(edward, andrew) ? creep
Fail: (22) older(edward, andrew) ? creep
Redo: (21) is_older(edward, andrew) ? creep
Call: (22) older(edward, _16978) ? creep
Fail: (22) older(edward, _17022) ? creep
Fail: (21) is_older(edward, andrew) ? creep
Redo: (20) then(edward, andrew) ? creep
Call: (21) offspring(princess, edward) ? creep
Fail: (21) offspring(princess, edward) ? creep
Fail: (20) then(edward, andrew) ? creep
Exit: (19) insert(edward, [andrew, ann], _15224) ? creep
Call: (20) not(then(edward, andrew)) ? creep
Call: (21) then(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) then(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) is_older(edward, andrew) ? creep
Call: (23) older(edward, andrew) ? creep
Fail: (23) older(edward, andrew) ? creep
Redo: (22) is_older(edward, andrew) ? creep
Call: (23) older(edward, _18008) ? creep
Fail: (23) older(edward, _18052) ? creep
Fail: (22) is_older(edward, andrew) ? creep
Redo: (21) then(edward, andrew) ? creep
Call: (22) offspring(princess, edward) ? creep
Fail: (22) offspring(princess, edward) ? creep
Fail: (21) then(edward, andrew) ? creep
Exit: (20) not(user:then(edward, andrew)) ? creep
Call: (20) insert(edward, [ann], _17278) ? creep
Call: (21) then(edward, ann) ? creep
Call: (22) offspring(prince, edward) ? creep
Exit: (22) offspring(prince, edward) ? creep
Call: (22) offspring(princess, ann) ? creep
Exit: (22) offspring(princess, ann) ? creep
Exit: (21) then(edward, ann) ? creep
Exit: (20) insert(edward, [ann], [edward, ann]) ? creep
Exit: (19) insert(edward, [andrew, ann], [andrew, edward, ann]) ? creep
Exit: (18) insert(edward, [charles, andrew, ann], [charles, andrew, edward, ann]) ? creep
Call: (18) i_sort([], [charles, andrew, edward, ann], _8374) ? creep
Exit: (18) i_sort([], [charles, andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
Exit: (17) i_sort([edward], [charles, andrew, ann], [charles, andrew, edward, ann]) ?

```

```

SWI-Prolog (AMD64, Multi-threaded, version 8.2.1)
File Edit Settings Run Debug Help
Exit: (17) i_sort([edward], [charles, andrew, ann], [charles, andrew, edward, ann]) ? creep
Exit: (16) i_sort([andrew, edward], [charles, ann], [charles, andrew, edward, ann]) ? creep
Exit: (15) i_sort([ann, andrew, edward], [charles], [charles, andrew, edward, ann]) ? creep
Exit: (14) i_sort([charles, ann, andrew, edward], [], [charles, andrew, edward, ann]) ? creep
Exit: (13) insert_sort([charles, ann, andrew, edward], [charles, andrew, edward, ann]) ? creep
Exit: (12) successors([charles, ann, andrew, edward], [charles, andrew, edward, ann]) ? creep
Exit: (11) oldRoyalSuccession([charles, andrew, edward, ann]) ? creep
OldRoyalSuccession = [charles, andrew, edward, ann] .

[trace] ?-

```

2.3 – Prolog clauses for Q2 Part 2 (LiewZhiLi_qn2_2.pl)

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

male(X) :- offspring(prince, X).
female(Y) :- offspring(princess, Y).

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

is_older(A, B):- older(A, B).
is_older(A, B):- older(A, M), is_older(M, B).

successors(X, Y) :- insert_sort(X, Y).

insert_sort(X, Y) :- i_sort(X, [], Y).
i_sort([], Acc, Acc).
i_sort([H|T], Acc, Y) :- insert(H, Acc, NewAcc), i_sort(T, NewAcc, Y).

insert(X, [], [X]).
insert(X, [Y|T], [X, Y|T]) :- is_older(X, Y).
insert(X, [Y|T], [Y|NewT]) :- not(is_older(X, Y)), insert(X, T, NewT).

newRoyalSuccession(NewRoyalSuccession):- findall(Y,offspring(_,Y), Offsprings),
successors(Offsprings,NewRoyalSuccession).
```

Explanation:

1. Since the new Royal succession rule states that the throne is passed down according to the order of birth irrespective of the gender of the offspring, we can remove the 3 definitions of then(A,B) rules.
2. In the insertion sort algorithm, instead of using the then(A,B) rules, we can now use the is_older() rules to get the new Royal succession ordered by birth.
3. The result is NewRoyalSuccession = [charles, ann, andrew, edward] .

2.4 – Trace of proof for new Royal succession

```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.1)
File Edit Settings Run Debug Help
?- newRoyalSuccession(NewRoyalSuccession).
NewRoyalSuccession = [charles, ann, andrew, edward] .

?- trace, newRoyalSuccession(NewRoyalSuccession).
^
Call: (11) newRoyalSuccession(_8398) ? creep
Call: (12) findall(_8884, offspring(_8882, _8884), _8944) ? creep
Call: (17) offspring(_8882, _8884) ? creep
Exit: (17) offspring(prince, charles) ? creep
Redo: (17) offspring(_8882, _8884) ? creep
Exit: (17) offspring(princess, ann) ? creep
Redo: (17) offspring(_8882, _8884) ? creep
Exit: (17) offspring(prince, andrew) ? creep
Redo: (17) offspring(_8882, _8884) ? creep
Exit: (17) offspring(prince, edward) ? creep
^
Call: (12) findall(_8884, user:offspring(_8882, _8884), [charles, ann, andrew, edward]) ? creep
Call: (12) successors([charles, ann, andrew, edward], _8398) ? creep
Call: (13) insert_sort([charles, ann, andrew, edward], _8398) ? creep
Call: (14) i_sort([charles, ann, andrew, edward], [], _8398) ? creep
Call: (15) insert(charles, [], _9574) ? creep
Exit: (15) insert(charles, [], [charles]) ? creep
Call: (15) i_sort([ann, andrew, edward], [charles], _8398) ? creep
Call: (16) insert(ann, [charles], _9712) ? creep
Call: (17) is_older(ann, charles) ? creep
Call: (18) older(ann, charles) ? creep
Fail: (18) older(ann, charles) ? creep
Redo: (17) is_older(ann, charles) ? creep
Call: (18) older(ann, _9942) ? creep
Exit: (18) older(ann, andrew) ? creep
Call: (18) is_older(andrew, charles) ? creep
Call: (19) older(andrew, charles) ? creep
Fail: (19) older(andrew, charles) ? creep
Redo: (18) is_older(andrew, charles) ? creep
Call: (19) older(andrew, _10206) ? creep
Exit: (19) older(andrew, edward) ? creep
Call: (19) is_older(edward, charles) ? creep
Call: (20) older(edward, charles) ? creep
Fail: (20) older(edward, charles) ? creep
Redo: (19) is_older(edward, charles) ? creep
Call: (20) older(edward, _10470) ? creep
Fail: (20) older(edward, _10514) ? creep
Fail: (19) is_older(edward, charles) ? creep
Fail: (18) is_older(andrew, charles) ? creep
Fail: (17) is_older(ann, charles) ? creep
Redo: (16) insert(ann, [charles], _10692) ? creep
^
Call: (17) not(is_older(ann, charles)) ? creep
Call: (18) is_older(ann, charles) ? creep
Call: (19) older(ann, charles) ? creep
Fail: (19) older(ann, charles) ? creep
Redo: (18) is_older(ann, charles) ? creep
Call: (19) older(ann, _10972) ? creep
Exit: (19) older(ann, andrew) ? creep
Call: (19) is_older(andrew, charles) ? creep
Call: (20) older(andrew, charles) ? creep
Fail: (20) older(andrew, charles) ? creep
Redo: (19) is_older(andrew, charles) ? creep
Call: (20) older(andrew, _11236) ? creep
Exit: (20) older(andrew, edward) ? creep
Call: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, charles) ? creep
Fail: (21) older(edward, charles) ? creep
Redo: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, _11500) ? creep
Fail: (21) older(edward, _11544) ? creep
Fail: (20) is_older(edward, charles) ? creep
Fail: (19) is_older(andrew, charles) ? creep
Fail: (18) is_older(ann, charles) ? creep
^
Exit: (17) not(user:is_older(ann, charles)) ? creep
Call: (17) insert(ann, [], _10682) ? creep
Exit: (17) insert(ann, [], [ann]) ? creep
Exit: (16) insert(ann, [charles], [charles, ann]) ? creep
Call: (16) i_sort([andrew, edward], [charles, ann], _8398) ? creep
Call: (17) insert(andrew, [charles, ann], _11948) ? creep
Call: (18) is_older(andrew, charles) ? creep
Call: (19) older(andrew, charles) ? creep
Fail: (19) older(andrew, charles) ? creep
Redo: (18) is_older(andrew, charles) ?
```

```

SWI-Prolog (AMD64, Multi-threaded, version 8.2.1)
File Edit Settings Run Debug Help
Call: (19) older(andrew, _12178) ? creep
Exit: (19) older(andrew, edward) ? creep
Call: (19) is_older(edward, charles) ? creep
Call: (20) older(edward, charles) ? creep
Fail: (20) older(edward, charles) ? creep
Redo: (19) is_older(edward, charles) ? creep
Call: (20) older(edward, _12442) ? creep
Fail: (20) older(edward, _12486) ? creep
Fail: (19) is_older(edward, charles) ? creep
Fail: (18) is_older(andrew, charles) ? creep
Redo: (17) insert(andrew, [charles, ann], _12620) ? creep
Call: (18) not(is_older(andrew, charles)) ? creep
Call: (19) is_older(andrew, charles) ? creep
Call: (20) older(andrew, charles) ? creep
Fail: (20) older(andrew, charles) ? creep
Redo: (19) is_older(andrew, charles) ? creep
Call: (20) older(andrew, _12900) ? creep
Exit: (20) older(andrew, edward) ? creep
Call: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, charles) ? creep
Fail: (21) older(edward, charles) ? creep
Redo: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, _13164) ? creep
Fail: (21) older(edward, _13208) ? creep
Fail: (20) is_older(edward, charles) ? creep
Fail: (19) is_older(andrew, charles) ? creep
Exit: (18) not(user:is_older(andrew, charles)) ? creep
Call: (18) insert(andrew, [ann], _12610) ? creep
Call: (19) is_older(andrew, ann) ? creep
Call: (20) older(andrew, ann) ? creep
Fail: (20) older(andrew, ann) ? creep
Redo: (19) is_older(andrew, ann) ? creep
Call: (20) older(andrew, _13616) ? creep
Exit: (20) older(andrew, edward) ? creep
Call: (20) is_older(edward, ann) ? creep
Call: (21) older(edward, ann) ? creep
Fail: (21) older(edward, ann) ? creep
Redo: (20) is_older(edward, ann) ? creep
Call: (21) older(edward, _13880) ? creep
Fail: (21) older(edward, _13924) ? creep
Fail: (20) is_older(edward, ann) ? creep
Fail: (19) is_older(andrew, ann) ? creep
Redo: (18) insert(andrew, [ann], _12610) ? creep
Call: (19) not(is_older(andrew, ann)) ? creep
Call: (20) is_older(andrew, ann) ? creep
Call: (21) older(andrew, ann) ? creep
Fail: (21) older(andrew, ann) ? creep
Redo: (20) is_older(andrew, ann) ? creep
Call: (21) older(andrew, _14338) ? creep
Exit: (21) older(andrew, edward) ? creep
Call: (21) is_older(edward, ann) ? creep
Call: (22) older(edward, ann) ? creep
Fail: (22) older(edward, ann) ? creep
Redo: (21) is_older(edward, ann) ? creep
Call: (22) older(edward, _14602) ? creep
Fail: (22) older(edward, _14646) ? creep
Fail: (21) is_older(edward, ann) ? creep
Fail: (20) is_older(andrew, ann) ? creep
Exit: (19) not(user:is_older(andrew, ann)) ? creep
Call: (19) insert(andrew, [], _14048) ? creep
Exit: (19) insert(andrew, [], [andrew]) ? creep
Exit: (18) insert(andrew, [ann], [ann, andrew]) ? creep
Exit: (17) insert(andrew, [charles, ann], [charles, ann, andrew]) ? creep
Call: (17) i_sort([edward], [charles, ann, andrew], _8398) ? creep
Call: (18) insert(edward, [charles, ann, andrew], _15050) ? creep
Call: (19) is_older(edward, charles) ? creep
Call: (20) older(edward, charles) ? creep
Fail: (20) older(edward, charles) ? creep
Redo: (19) is_older(edward, charles) ? creep
Call: (20) older(edward, _15280) ? creep
Fail: (20) older(edward, _15324) ? creep
Fail: (19) is_older(edward, charles) ? creep
Redo: (18) insert(edward, [charles, ann, andrew], _15414) ? creep
Call: (19) not(is_older(edward, charles)) ? creep
Call: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, charles) ?

```

```

SWI-Prolog (AMD64, Multi-threaded, version 8.2.1)
File Edit Settings Run Debug Help
Call: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, charles) ? creep
Fail: (21) older(edward, charles) ? creep
Redo: (20) is_older(edward, charles) ? creep
Call: (21) older(edward, _15694) ? creep
Fail: (21) older(edward, _15738) ? creep
Fail: (20) is_older(edward, charles) ? creep
^ Exit: (19) not(user:is_older(edward, charles)) ? creep
Call: (19) insert(edward, [ann, andrew], _15404) ? creep
Call: (20) is_older(edward, ann) ? creep
Call: (21) older(edward, ann) ? creep
Fail: (21) older(edward, ann) ? creep
Redo: (20) is_older(edward, ann) ? creep
Call: (21) older(edward, _16102) ? creep
Fail: (21) older(edward, _16146) ? creep
Fail: (20) is_older(edward, ann) ? creep
Redo: (19) insert(edward, [ann, andrew], _15404) ? creep
^ Call: (20) not(is_older(edward, ann)) ? creep
Call: (21) is_older(edward, ann) ? creep
Call: (22) older(edward, ann) ? creep
Fail: (22) older(edward, ann) ? creep
Redo: (21) is_older(edward, ann) ? creep
Call: (22) older(edward, _16516) ? creep
Fail: (22) older(edward, _16560) ? creep
Fail: (21) is_older(edward, ann) ? creep
^ Exit: (20) not(user:is_older(edward, ann)) ? creep
Call: (20) insert(edward, [andrew], _16226) ? creep
Call: (21) is_older(edward, andrew) ? creep
Call: (22) older(edward, andrew) ? creep
Fail: (22) older(edward, andrew) ? creep
Redo: (21) is_older(edward, andrew) ? creep
Call: (22) older(edward, _16924) ? creep
Fail: (22) older(edward, _16968) ? creep
Fail: (21) is_older(edward, andrew) ? creep
Redo: (20) insert(edward, [andrew], _16226) ? creep
^ Call: (21) not(is_older(edward, andrew)) ? creep
Call: (22) is_older(edward, andrew) ? creep
Call: (23) older(edward, andrew) ? creep
Fail: (23) older(edward, andrew) ? creep
Redo: (22) is_older(edward, andrew) ? creep
Call: (23) older(edward, _17338) ? creep
Fail: (23) older(edward, _17382) ? creep
Fail: (22) is_older(edward, andrew) ? creep
^ Exit: (21) not(user:is_older(edward, andrew)) ? creep
Call: (21) insert(edward, [], _17048) ? creep
Exit: (21) insert(edward, [], [edward]) ? creep
Exit: (20) insert(edward, [andrew], [andrew, edward]) ? creep
Exit: (19) insert(edward, [ann, andrew], [ann, andrew, edward]) ? creep
Exit: (18) insert(edward, [charles, ann, andrew], [charles, ann, andrew, edward]) ? creep
Call: (18) i_sort([], [charles, ann, andrew, edward], _8398) ? creep
Exit: (18) i_sort([], [charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (17) i_sort([edward], [charles, ann, andrew], [charles, ann, andrew, edward]) ? creep
Exit: (16) i_sort([andrew, edward], [charles, ann], [charles, ann, andrew, edward]) ? creep
Exit: (15) i_sort([ann, andrew, edward], [charles], [charles, ann, andrew, edward]) ? creep
Exit: (14) i_sort([charles, ann, andrew, edward], [], [charles, ann, andrew, edward]) ? creep
Exit: (13) insert_sort([charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (12) successors([charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (11) newRoyalSuccession([charles, ann, andrew, edward]) ? creep
NewRoyalSuccession = [charles, ann, andrew, edward] .

[trace] ?-

```