NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Assignment 2
SC3000 Artificial Intelligence
Lab Group: A33

Full Name	Matriculation Number	Individual Contribution to Assignment 2	Signature
Lim Jun Hern	U2120981B	• Translate natural language statements into First Order Logic (FOL) • Implement prolog logic based on First Order Logic (FOL) • Debug prolog codes	[illegible]
Leong Hong Yi	U2120932C	• Implement prolog logic based on First Order Logic (FOL) • Run prolog codes • Complete written report	[illegible]
Jabez Ng Yong Xin	U2120757D	• Refine First Order Logic from natural language statements (FOL) • Debug prolog codes • Complete written report	[illegible]

Exercise 1: The Smart Phone Rivalry (15 marks)

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 of 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)

CompetitorOf(sumsum, appy) - *"sumsum is a competitor of appy"*.

CompetitorOf(appy, sumsum) - *"appy is a competitor of sumsum"*.

Developed(sumsum, galactica-s3) - *"sumsum developed galactica-s3"*.

SmartPhoneTechnology(galactica-s3) - *"galactica-s3 is a Smart Phone Technology"*.

Stole(stevey, galactica-s3) - *"stevey stole galactica-s3"*.

BossOf(stevey, appy) - *"stevey is boss of appy"*

∀x, ∀y, CompetitorOf(x, y) ⇒ RivalOf(x, y) - *"All competitors are rivals"*.

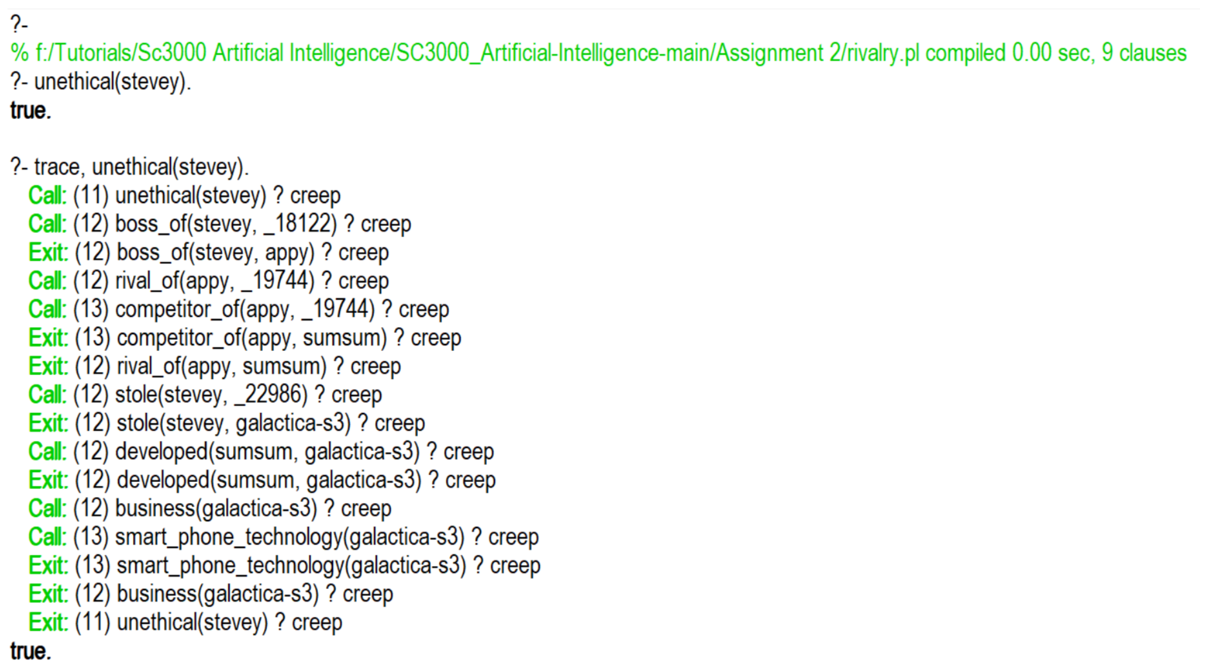
∀x, SmartPhoneTechnology(x) ⇒ Business(x) - *"All Smart Phone Technology are Business"*.

∀x, ∀y, ∀z, ∀p, BossOf(x, y) ∧ RivalOf(y, z) ∧ Stole(x, p) ∧ Developed(z, p) ∧ Business(p) ⇒ Unethical(x) - *"If a boss steals something related to business that was developed by a rival, then that boss is acting unethically"*.

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

Please refer to the "A33_qn1_2/rivalry.pl" file.

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



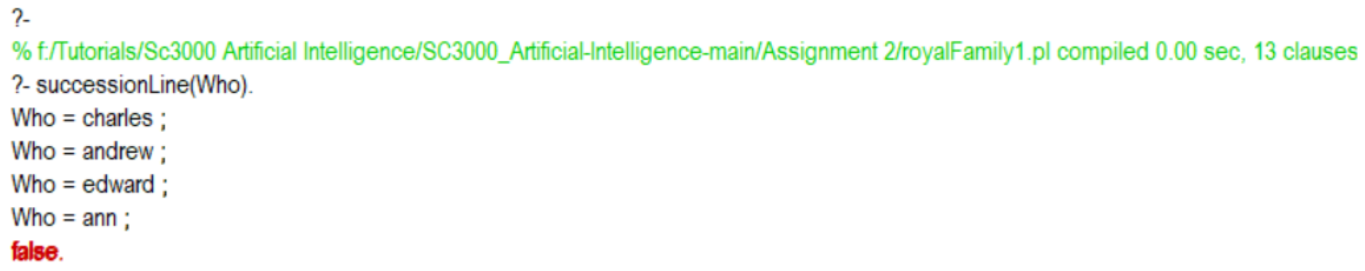
```
?-
% f:/Tutorials/Sc3000 Artificial Intelligence/SC3000_Artificial-Intelligence-main/Assignment 2/rivalry.pl compiled 0.00 sec, 9 clauses
?- unethical(stevey).
true.

?- trace, unethical(stevey).
   Call: (11) unethical(stevey) ? creep
   Call: (12) boss_of(stevey, _18122) ? creep
   Exit: (12) boss_of(stevey, appy) ? creep
   Call: (12) rival_of(appy, _19744) ? creep
   Call: (13) competitor_of(appy, _19744) ? creep
   Exit: (13) competitor_of(appy, sumsum) ? creep
   Exit: (12) rival_of(appy, sumsum) ? creep
   Call: (12) stole(stevey, _22986) ? creep
   Exit: (12) stole(stevey, galactica-s3) ? creep
   Call: (12) developed(sumsum, galactica-s3) ? creep
   Exit: (12) developed(sumsum, galactica-s3) ? creep
   Call: (12) business(galactica-s3) ? creep
   Call: (13) smart_phone_technology(galactica-s3) ? creep
   Exit: (13) smart_phone_technology(galactica-s3) ? creep
   Exit: (12) business(galactica-s3) ? creep
   Exit: (11) unethical(stevey) ? creep
true.
```

Exercise 2: The Royal Family (10 marks)

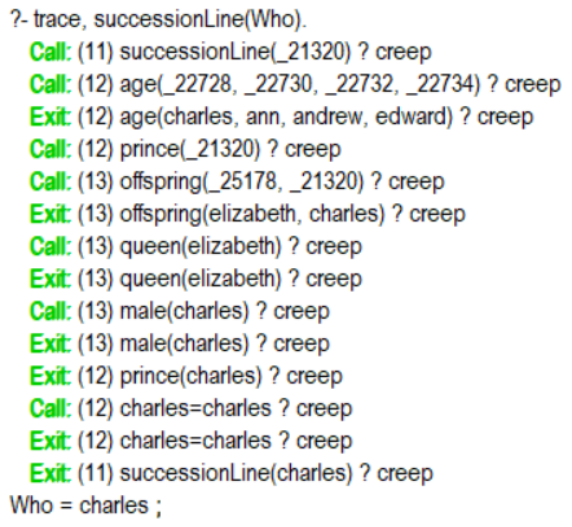
The old Royal succession rule states that the throne is passed down along the male line according to the order of birth before the consideration along the female line – similarly according to the order of birth. queen elizabeth, the monarch of United Kingdom, has four offsprings; namely:- prince charles, 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)

Please refer to “A33_qn2_1/royalFamily1.pl” for the relations, rules and old Royal succession rule.

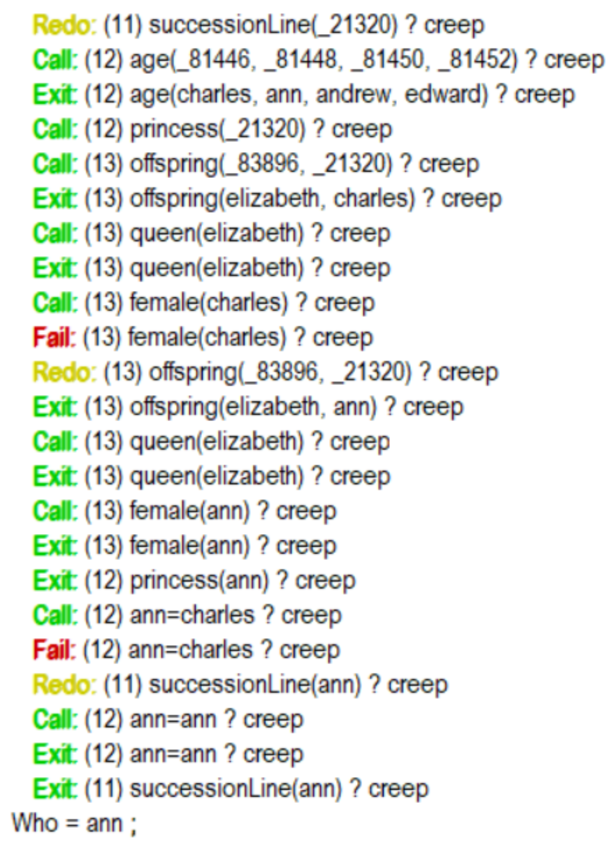


```
?-
% f:/Tutorials/Sc3000 Artificial Intelligence/SC3000_Artificial-Intelligence-main/Assignment 2/royalFamily1.pl compiled 0.00 sec, 13 clauses
?- successionLine(Who).
Who = charles ;
Who = andrew ;
Who = edward ;
Who = ann ;
false.
```

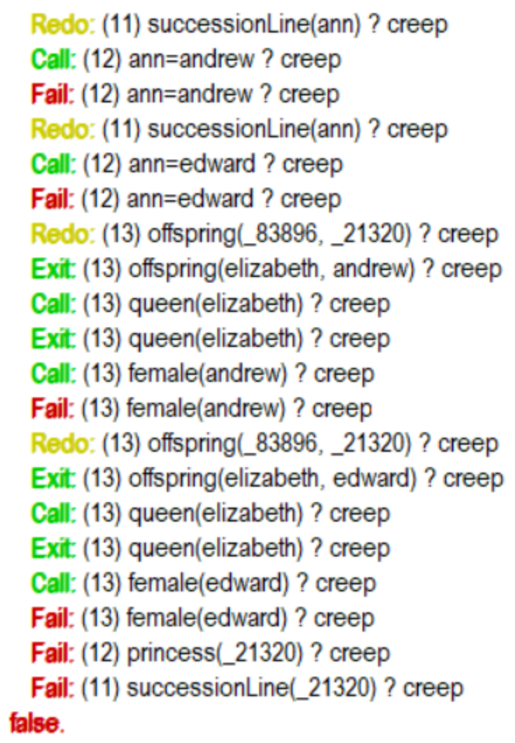


```
?- trace, successionLine(Who).
   Call: (11) successionLine(_21320) ? creep
   Call: (12) age(_22728, _22730, _22732, _22734) ? creep
   Exit: (12) age(charles, ann, andrew, edward) ? creep
   Call: (12) prince(_21320) ? creep
   Call: (13) offspring(_25178, _21320) ? creep
   Exit: (13) offspring(elizabeth, charles) ? creep
   Call: (13) queen(elizabeth) ? creep
   Exit: (13) queen(elizabeth) ? creep
   Call: (13) male(charles) ? creep
   Exit: (13) male(charles) ? creep
   Exit: (12) prince(charles) ? creep
   Call: (12) charles=charles ? creep
   Exit: (12) charles=charles ? creep
   Exit: (11) successionLine(charles) ? creep
Who = charles ;
```

```
  Redo: (11) successionLine(charles) ? creep
  Call: (12) charles=ann ? creep
  Fail: (12) charles=ann ? creep
  Redo: (11) successionLine(charles) ? creep
  Call: (12) charles=andrew ? creep
  Fail: (12) charles=andrew ? creep
  Redo: (11) successionLine(charles) ? creep
  Call: (12) charles=edward ? creep
  Fail: (12) charles=edward ? creep
  Redo: (13) offspring(_25178, _21320) ? creep
  Exit: (13) offspring(elizabeth, ann) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) male(ann) ? creep
  Fail: (13) male(ann) ? creep
  Redo: (13) offspring(_25178, _21320) ? creep
  Exit: (13) offspring(elizabeth, andrew) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) male(andrew) ? creep
  Exit: (13) male(andrew) ? creep
  Exit: (12) prince(andrew) ? creep
  Call: (12) andrew=charles ? creep
  Fail: (12) andrew=charles ? creep
  Redo: (11) successionLine(andrew) ? creep
  Call: (12) andrew=ann ? creep
  Fail: (12) andrew=ann ? creep
  Redo: (11) successionLine(andrew) ? creep
  Call: (12) andrew=andrew ? creep
  Exit: (12) andrew=andrew ? creep
  Exit: (11) successionLine(andrew) ? creep
Who = andrew ;

  Redo: (11) successionLine(andrew) ? creep
  Call: (12) andrew=edward ? creep
  Fail: (12) andrew=edward ? creep
  Redo: (13) offspring(_25178, _21320) ? creep
  Exit: (13) offspring(elizabeth, edward) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) male(edward) ? creep
  Exit: (13) male(edward) ? creep
  Exit: (12) prince(edward) ? creep
  Call: (12) edward=charles ? creep
  Fail: (12) edward=charles ? creep
  Redo: (11) successionLine(edward) ? creep
  Call: (12) edward=ann ? creep
  Fail: (12) edward=ann ? creep
  Redo: (11) successionLine(edward) ? creep
  Call: (12) edward=andrew ? creep
  Fail: (12) edward=andrew ? creep
  Redo: (11) successionLine(edward) ? creep
  Call: (12) edward=edward ? creep
  Exit: (12) edward=edward ? creep
  Exit: (11) successionLine(edward) ? creep
Who = edward ;
```



```
 Redo: (11) successionLine(_21320) ? creep
 Call: (12) age(_81446, _81448, _81450, _81452) ? creep
 Exit: (12) age(charles, ann, andrew, edward) ? creep
 Call: (12) princess(_21320) ? creep
 Call: (13) offspring(_83896, _21320) ? creep
 Exit: (13) offspring(elizabeth, charles) ? creep
 Call: (13) queen(elizabeth) ? creep
 Exit: (13) queen(elizabeth) ? creep
 Call: (13) female(charles) ? creep
 Fail: (13) female(charles) ? creep
 Redo: (13) offspring(_83896, _21320) ? creep
 Exit: (13) offspring(elizabeth, ann) ? creep
 Call: (13) queen(elizabeth) ? creep
 Exit: (13) queen(elizabeth) ? creep
 Call: (13) female(ann) ? creep
 Exit: (13) female(ann) ? creep
 Exit: (12) princess(ann) ? creep
 Call: (12) ann=charles ? creep
 Fail: (12) ann=charles ? creep
 Redo: (11) successionLine(ann) ? creep
 Call: (12) ann=ann ? creep
 Exit: (12) ann=ann ? creep
 Exit: (11) successionLine(ann) ? creep
Who = ann ;
```



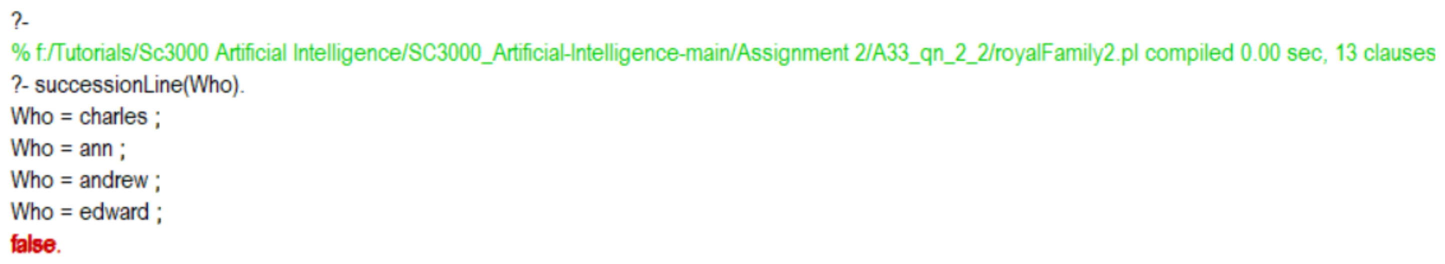
```
 Redo: (11) successionLine(ann) ? creep
 Call: (12) ann=andrew ? creep
 Fail: (12) ann=andrew ? creep
 Redo: (11) successionLine(ann) ? creep
 Call: (12) ann=edward ? creep
 Fail: (12) ann=edward ? creep
 Redo: (13) offspring(_83896, _21320) ? creep
 Exit: (13) offspring(elizabeth, andrew) ? creep
 Call: (13) queen(elizabeth) ? creep
 Exit: (13) queen(elizabeth) ? creep
 Call: (13) female(andrew) ? creep
 Fail: (13) female(andrew) ? creep
 Redo: (13) offspring(_83896, _21320) ? creep
 Exit: (13) offspring(elizabeth, edward) ? creep
 Call: (13) queen(elizabeth) ? creep
 Exit: (13) queen(elizabeth) ? creep
 Call: (13) female(edward) ? creep
 Fail: (13) female(edward) ? creep
 Fail: (12) princess(_21320) ? creep
 Fail: (11) successionLine(_21320) ? 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.

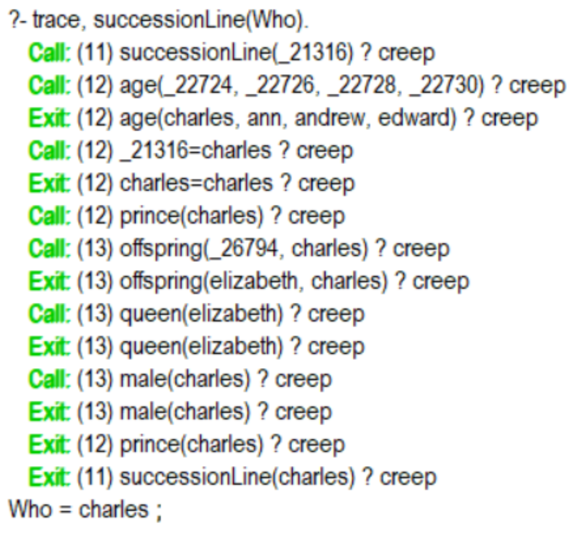
Refer to "A33_qn_2_1/royalFamily1.pl" for the relations, rules and old Royal succession rule.

Refer to "A33_qn_2_2/royalFamily2.pl" for the relations, rules and <u>**updated**</u> **Royal succession rule**.

The changes to the knowledge involve the removal of **prince(X)** and **princess(X)** and their removal from **successionLine(who)**. Moreover, we can then simplify **successionLine(who)** to consider the potential successors by the order of birth only and not their gender.



```
?-
% f:/Tutorials/Sc3000 Artificial Intelligence/SC3000_Artificial-Intelligence-main/Assignment 2/A33_qn_2_2/royalFamily2.pl compiled 0.00 sec, 13 clauses
?- successionLine(Who).
Who = charles ;
Who = ann ;
Who = andrew ;
Who = edward ;
false.
```



```
?- trace, successionLine(Who).
   Call: (11) successionLine(_21316) ? creep
   Call: (12) age(_22724, _22726, _22728, _22730) ? creep
   Exit: (12) age(charles, ann, andrew, edward) ? creep
   Call: (12) _21316=charles ? creep
   Exit: (12) charles=charles ? creep
   Call: (12) prince(charles) ? creep
   Call: (13) offspring(_26794, charles) ? creep
   Exit: (13) offspring(elizabeth, charles) ? creep
   Call: (13) queen(elizabeth) ? creep
   Exit: (13) queen(elizabeth) ? creep
   Call: (13) male(charles) ? creep
   Exit: (13) male(charles) ? creep
   Exit: (12) prince(charles) ? creep
   Exit: (11) successionLine(charles) ? creep
Who = charles ;
```

```
  Redo: (11) successionLine(charles) ? creep
  Call: (12) princess(charles) ? creep
  Call: (13) offspring(_36382, charles) ? creep
  Exit: (13) offspring(elizabeth, charles) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) female(charles) ? creep
  Fail: (13) female(charles) ? creep
  Fail: (12) princess(charles) ? creep
  Redo: (11) successionLine(_21316) ? creep
  Call: (12) _21316=ann ? creep
  Exit: (12) ann=ann ? creep
  Call: (12) prince(ann) ? creep
  Call: (13) offspring(_45266, ann) ? creep
  Exit: (13) offspring(elizabeth, ann) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) male(ann) ? creep
  Fail: (13) male(ann) ? creep
  Fail: (12) prince(ann) ? creep
  Redo: (11) successionLine(ann) ? creep
  Call: (12) princess(ann) ? creep
  Call: (13) offspring(_52530, ann) ? creep
  Exit: (13) offspring(elizabeth, ann) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) female(ann) ? creep
  Exit: (13) female(ann) ? creep
  Exit: (12) princess(ann) ? creep
  Exit: (11) successionLine(ann) ? creep
Who = ann ;

  Redo: (11) successionLine(_21316) ? creep
  Call: (12) _21316=andrew ? creep
  Exit: (12) andrew=andrew ? creep
  Call: (12) prince(andrew) ? creep
  Call: (13) offspring(_63738, andrew) ? creep
  Exit: (13) offspring(elizabeth, andrew) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) male(andrew) ? creep
  Exit: (13) male(andrew) ? creep
  Exit: (12) prince(andrew) ? creep
  Exit: (11) successionLine(andrew) ? creep
Who = andrew ;
```

```
  Redo: (11) successionLine(andrew) ? creep
  Call: (12) princess(andrew) ? creep
  Call: (13) offspring(_73326, andrew) ? creep
  Exit: (13) offspring(elizabeth, andrew) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) female(andrew) ? creep
  Fail: (13) female(andrew) ? creep
  Fail: (12) princess(andrew) ? creep
  Redo: (11) successionLine(_21316) ? creep
  Call: (12) _21316=edward ? creep
  Exit: (12) edward=edward ? creep
  Call: (12) prince(edward) ? creep
  Call: (13) offspring(_82210, edward) ? creep
  Exit: (13) offspring(elizabeth, edward) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) male(edward) ? creep
  Exit: (13) male(edward) ? creep
  Exit: (12) prince(edward) ? creep
  Exit: (11) successionLine(edward) ? creep
Who = edward ;

  Redo: (11) successionLine(edward) ? creep
  Call: (12) princess(edward) ? creep
  Call: (13) offspring(_91798, edward) ? creep
  Exit: (13) offspring(elizabeth, edward) ? creep
  Call: (13) queen(elizabeth) ? creep
  Exit: (13) queen(elizabeth) ? creep
  Call: (13) female(edward) ? creep
  Fail: (13) female(edward) ? creep
  Fail: (12) princess(edward) ? creep
  Fail: (11) successionLine(_21316) ? creep
false.
```