

Introduction to AI First-order Logic

Project 2

21CLCo5 - University of Science - VNUHCM

21127004 - 21127081 - 21127135 - 21127325



1. Report

This will be a long read so please make use of bookmarks for ease of navigation.

Priority	No.	Task	Status
Required	1.1	Learn the Prolog language, write report on main features and give examples.	
	1.2	Learn a Prolog environment, write report on how to implement Prolog and give examples.	
	1.3	Solving deductive problems using SWI-Prolog, build a Knowledge Base from the British Royal Family Tree, give at least 20 questions, and compile the outputs.	
	2	Build a Knowledge Base from any topic containing at least 50 predicates, give at least 20 questions, and compile the outputs.	
	3	Build a logical inference program using one of the deductive methods, verify the results with 1.3 and 2.	
Bonus	a		
	b		
	c		

Project Completion Rate: 100%.

Student ID	Full name	Tasks	Contribution
21127004	Trần Nguyễn An Phong		100%
21127081	Nguyễn Minh Khôi		100%
21127135	Diệp Hữu Phúc		100%
21127325	Phan Đặng Anh Khôi		100%

2. Working with the Prolog tool (40%)

Learn the Prolog language

Prolog is a logic programming language associated with artificial intelligence and computational linguistics. – Wikipedia

Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In Prolog, program logic is expressed in terms of relations, and a computation is carried out by running a query over these relations.

A typical Prolog program requires a Knowledge Base, which is composed of predefined clauses, i.e., facts and rules. The Knowledge Base can, then, be queried against to generate outputs. If our query is already in the Knowledge Base or it is implied by Knowledge Base, the result might be true or a list of all satisfied terms, otherwise we get false.

Query	Result
?- sister('zara_phillips', 'peter_phillips').	true.
?- nephew(X, 'princess_anne').	X = prince_william ; X = prince_harry ; X = james_viscount_severn ; false.
?- granddaughter('isla_phillips', 'timothy_laurence').	false.

Terms

- The sole data type of Prolog.
- Terms are either atoms, numbers, variables, or compound terms.

Terms	Definition	Example
Atoms	General-purpose name with no inherent meaning.	x, red, 'Hello world'
Numbers	Floats or integers.	6.9, 420, 727
Variables	Strings consisting of letters, numbers, and underscores. Must begin with an upper-case letter or underscore.	X, _360, Hello_world
Compound terms	<ul style="list-style-type: none">- Composed of an atom called a <i>functor</i> and several <i>arguments</i>, which are terms.- The number of arguments is called the <i>arity</i>. An atom can be seen as a compound term with arity zero.- Lists and Strings are special cases of compound terms.	foo(bar), point(X, Y, Z), "Hello world", . (1, . (2, . (3, []))), [1, 2, 3]

Facts and rules

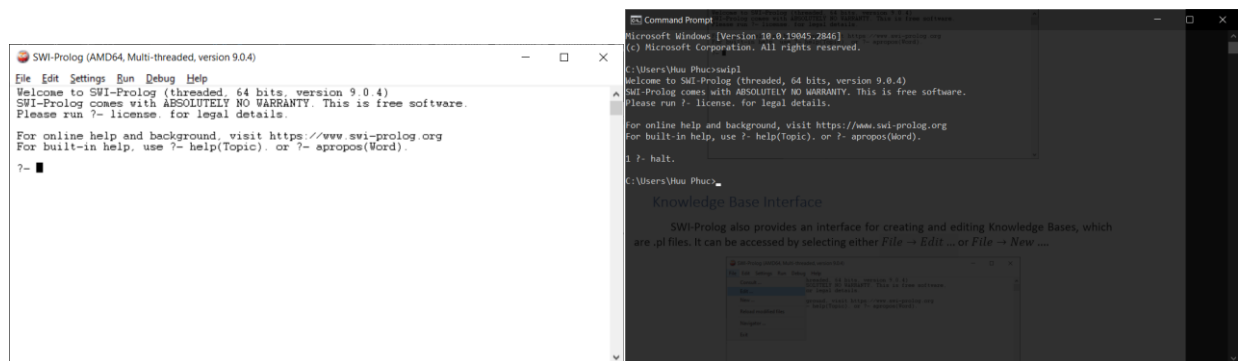
Learn a Prolog programming environment

Our environment of choice is SWI-Prolog, specifically the stable version for 64-bit Windows. We won't delve further into the installation process since the installer is already very intuitive. Below is the download link,

- <https://www.swi-prolog.org/download/stable>

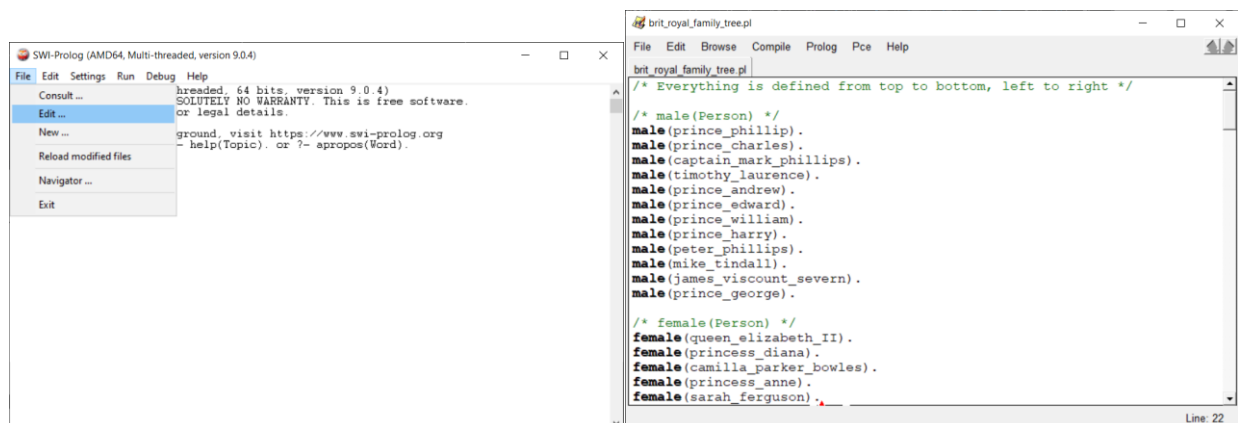
Query Interface

Although SWI-Prolog has a main interface for writing queries, this environment can essentially be initiated on any command prompt by inputting **swipl**. Additionally, a session can be ended by doing **halt.** query.

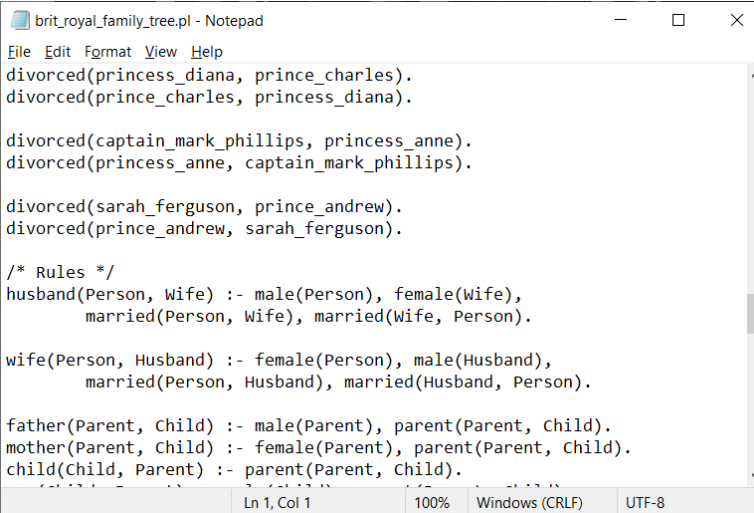


Knowledge Base Interface

SWI-Prolog also provides an interface for creating and editing Knowledge Bases, which are .pl files. It can be accessed by selecting either *File → Edit ...* or *File → New ...*



However, the .pl files can be modified with pretty much any text editors in existence. As for the format, each line denotes a clause, i.e., a rule or a fact, that must end with a dot.



```

brit_royal_family_tree.pl - Notepad
File Edit Format View Help
divorced(princess_diana, prince_charles).
divorced(prince_charles, princess_diana).

divorced(captain_mark_phillips, princess_anne).
divorced(princess_anne, captain_mark_phillips).

divorced(sarah_ferguson, prince_andrew).
divorced(prince_andrew, sarah_ferguson).

/* Rules */
husband(Person, Wife) :- male(Person), female(Wife),
    married(Person, Wife), married(Wife, Person).

wife(Person, Husband) :- female(Person), male(Husband),
    married(Person, Husband), married(Husband, Person).

father(Parent, Child) :- male(Parent), parent(Parent, Child).
mother(Parent, Child) :- female(Parent), parent(Parent, Child).
child(Child, Parent) :- parent(Parent, Child).

```

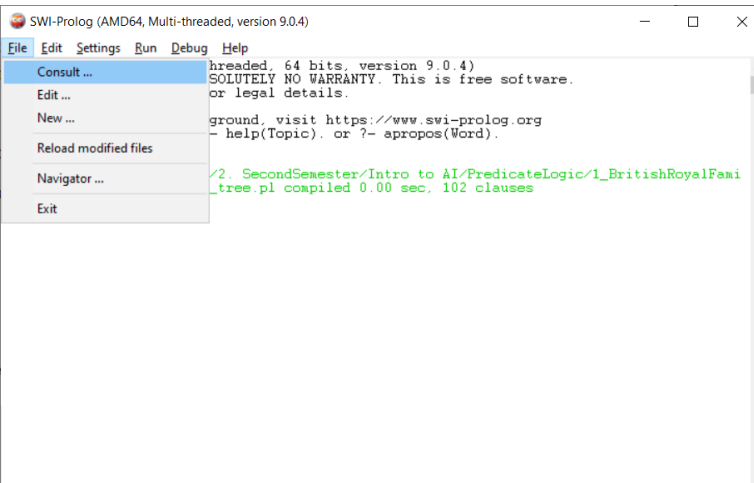
Apart from using the interface and text editors, it is possible to construct or modify a Knowledge Base with queries. Considering it will be quite a hefty write for us, we will just leave the link to the source documentation here for your perusal.

- <https://www.swi-prolog.org/pldoc/man?section=dynpreds>

Consulting a Knowledge Base

There are multiple methods to load a Knowledge Base.

- **Method 1:** Using *File* → *Consult*



```

SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Consult ... hreaded, 64 bits, version 9.0.4)
Edit ... SOLUTELY NO WARRANTY. This is free software.
New ... or legal details.
Reload modified files ground, visit https://www.swi-prolog.org
Navigator ... - help(Topic). or ?- apropos(Word).
Exit /2. SecondSemester/Intro to AI/PredicateLogic/1_BritishRoyalFami
tree.pl compiled 0.00 sec, 102 clauses

```

- **Method 2:** Using `consult('path:/to/pl/file')` or `['path:/to/pl/file']`.
- **Method 3:** Changing directory with `working_directory(CWD, 'path:/to/dir/with/pl/file')` then `consult(plFile)` or `[plFile]`.

```
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('F:/Hcaus/2. SecondYear/2. SecondSemester/Intro to AI/PredicateLogic/1_BritishRoyalFamilyTree/brit_royal_family_tree').
true.

?- ['F:/Hcaus/2. SecondYear/2. SecondSemester/Intro to AI/PredicateLogic/1_BritishRoyalFamilyTree/brit_royal_family_tree'].
true.

?- pwd.
% e:/swipl/bin/
true.

?- working_directory(CWD, 'F:/Hcaus/2. SecondYear/2. SecondSemester/Intro to AI/PredicateLogic/1_BritishRoyalFamilyTree').
CWD = 'e:/swipl/bin/'.

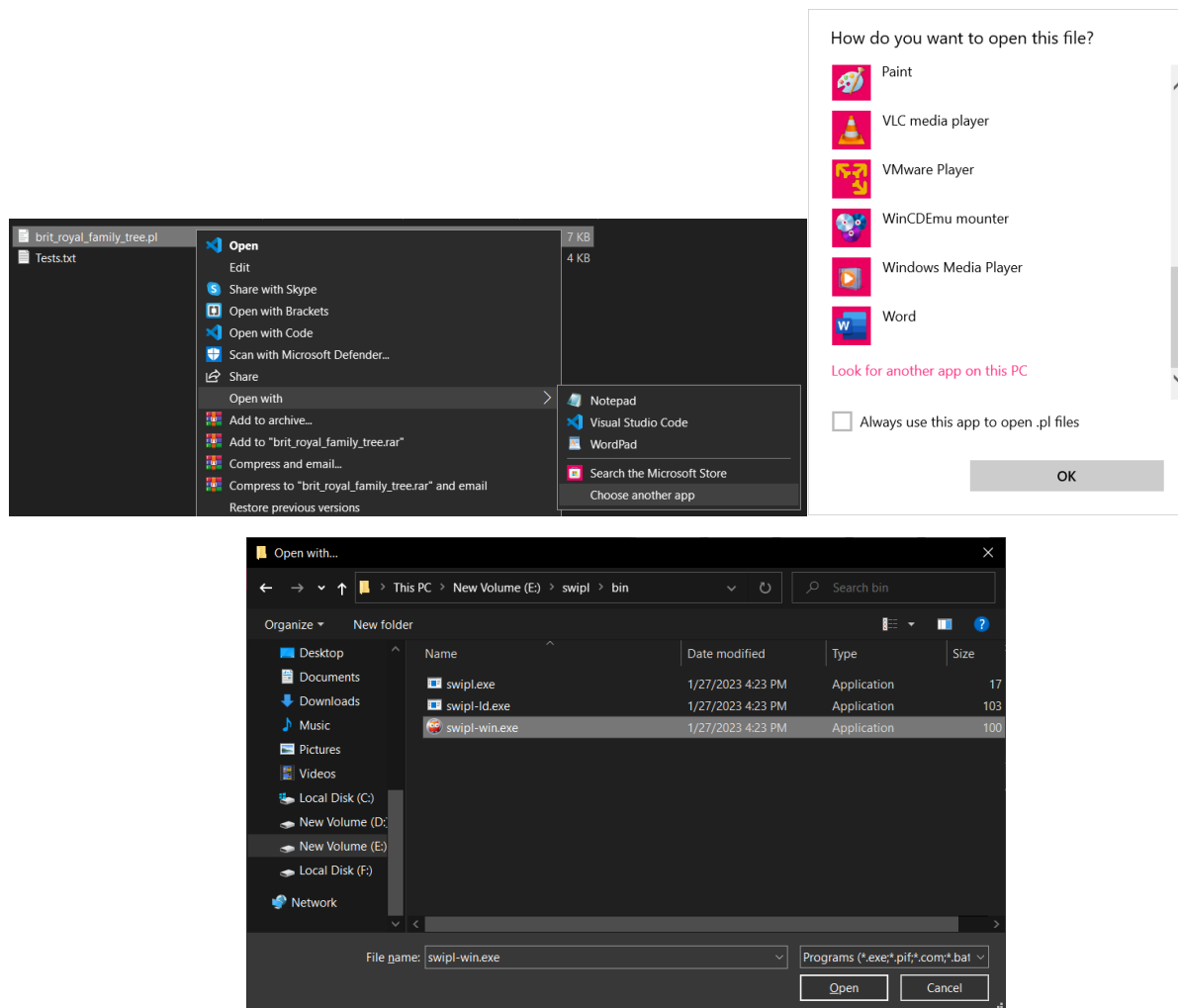
?- pwd.
% f:/hcaus/2. secondyear/2. secondsemester/intro to ai/predicatelogic/1_britishroyalfamilytree/
true.

?- consult(brit_royal_family_tree).
true.

?- [brit_royal_family_tree].
true.

?-
```

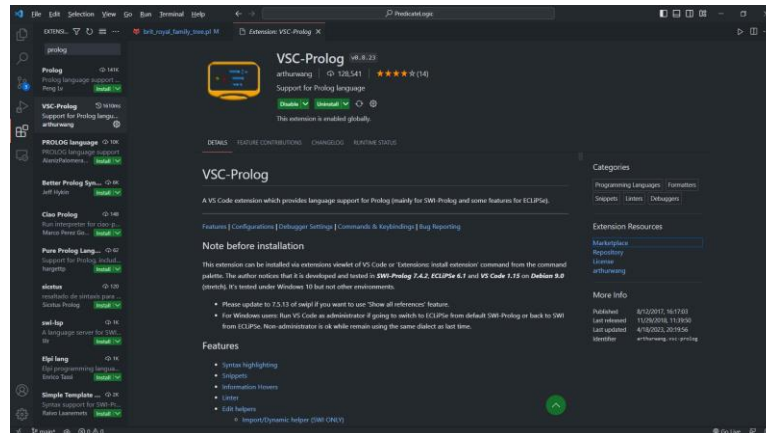
- Method 4: Opening the .pl file directly with swipl-win.exe.



Integrating with Visual Code

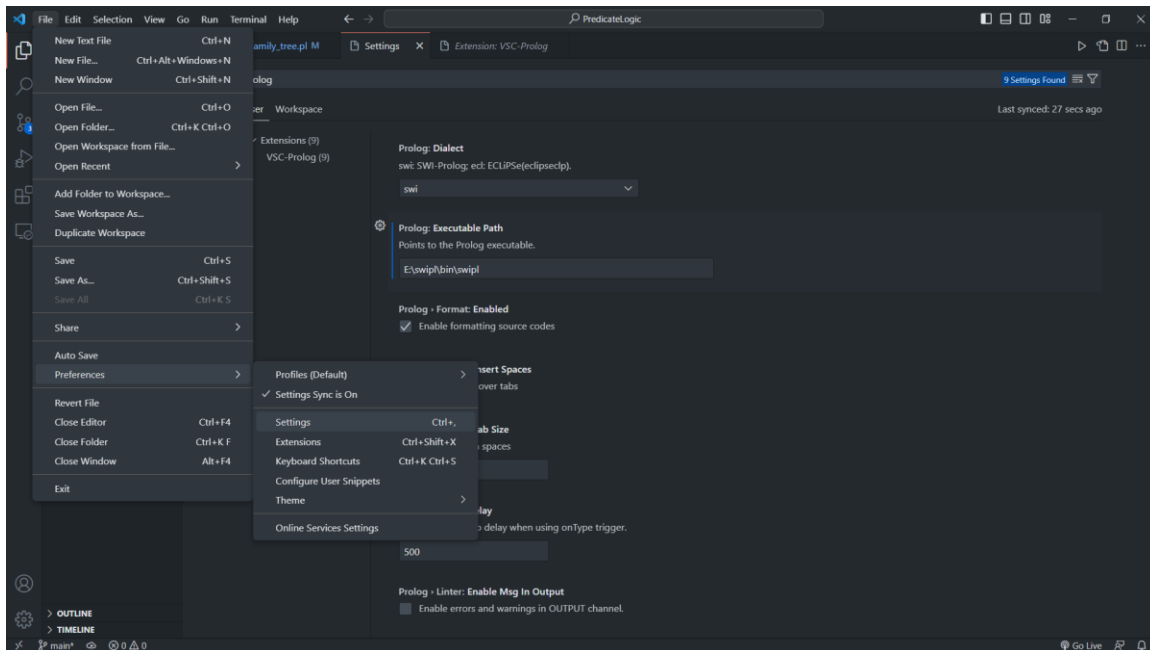
In our humble opinion, the base interfaces of SWI-Prolog are not very aesthetically pleasing. It is that, combined with the annoying act of having to jump between multiple windows during test compilation or debugging, gives enough reasons to seek out the reliable VSCode.

All we need to do for the integration is a decent Prolog extension. We recommend VSC-Prolog since it has the nicest-looking syntax highlighting.



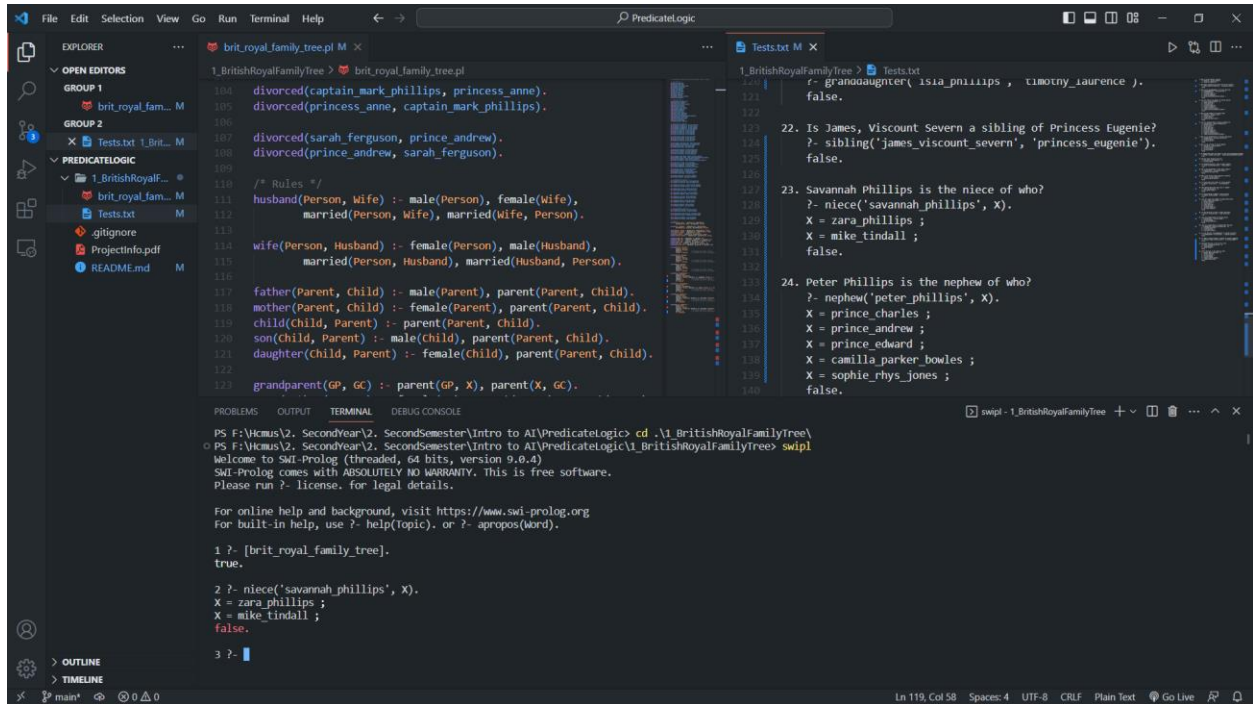
If you proceed with this extension, for the linters to work properly,

1. Go into *File* → *Preferences* → *Settings*.
2. Search for **prolog**.
3. Change the **Executable Path** to where your **swipl.exe** is located.



Writing some queries

Below is a picture depicting our workflow.



Type	Query
True/False	<pre> 1 ?- [brit_royal_family_tree]. true. 2 ?- female('queen_elizabeth_II'). true. 3 ?- male('queen_elizabeth_II'). false. </pre>
Multiple results	<pre> 1 ?- [brit_royal_family_tree]. true. 2 ?- nephew('peter_phillips', X). X = prince_charles ; X = prince_andrew ; X = prince_edward ; X = camilla_parker_bowles ; X = sophie_rhys_jones ; false. </pre>

Multiple variables	<pre> 1 ?- [brit_royal_family_tree]. true. 2 ?- divorced(X, Y). X = princess_diana, Y = prince_charles ; X = prince_charles, Y = princess_diana ; X = captain_mark_phillips, Y = princess_anne ; X = princess_anne, Y = captain_mark_phillips ; X = sarah_ferguson, Y = prince_andrew ; X = prince_andrew, Y = sarah_ferguson. </pre>
Multiple predicates	<pre> 1 ?- [brit_royal_family_tree]. true. 2 ?- female(X), uncle('prince_harry', X). X = princess_charlotte ; false. </pre>
Recursion	<pre> 1 ?- assertz(parent(albert, bob)). true. 2 ?- assertz(parent(albert, betsy)). true. 3 ?- assertz(parent(alice, bob)). true. 4 ?- assertz(parent(alice, betsy)). true. 5 ?- assertz(parent(bob, carl)). true. 6 ?- assertz(parent(bob, charlie)). true. 7 ?- assertz((related(X, Y) :- parent(X, Y))). true. 8 ?- assertz((related(X, Y) :- parent(X, Z), related(Z, Y))). true. 9 ?- related(X, carl). X = bob ; X = albert ; X = alice ; false. </pre>

British Royal Family Tree

The relevant folder is **1_BritishRoyalFamilyTree**.

- **brit_royal_family_tree.pl**: The Knowledge Base constructed from the British Royal Family.
- **Tests.txt**: Records every single pair of performed query and its output. Due to their lengthy nature, we chose not to include them in this report.

For most of the predicates are very straightforward, we will only elaborate on the ones that, we think, are noteworthy.

Sibling, brother, sister

Since they share basically the same implementation, let us examine **brother**.

```
brother(Person, Sibling) :-  
    male(Person),  
    parent(X, Person),  
    parent(X, Sibling),  
    female(X),           % The gender here doesn't matter,  
    Person \= Sibling.    % we choose only one to avoid duplication.
```

Since we only care about whether **Person** and **Sibling** share one parent **X**, the parent's gender doesn't matter. Hence, we arbitrarily picked **female(X)**, we could also have gone with **male(X)**. One last thing is that we must make sure that two different people are really being checked, thus, **Person \= Sibling**.

Here is an example of the duplication when we removed **female(X)**.

```
?- brother('prince_george', X).  
X = princess_charlotte ;  
X = princess_charlotte.
```

Aunt, uncle, niece, nephew

Since they share basically the same implementation, let us examine **aunt**.

```
aunt(Person, NieceNephew) :-  
    female(Person),  
    parent(X, NieceNephew),  
    (parent(Y, Person); (parent(Y, Z), husband(Z, Person), X \= Z)),  
    female(Y),           % The gender here doesn't matter,  
    parent(Y, X),        % we choose only one to avoid duplication.  
    X \= Person.
```

We consider an aunt to be either a parent's sister or uncle's wife. As we have already written the logic for the **sister** predicate, we will skip them here. On to the uncle's wife, instead of checking for a parent **Y** of both **X** and **Person**, we need a parent **Y** of both **X** and the husband **Z** of **Person**, and to also ensure that we don't check someone multiple times with **X \= Z**.

Once again, here is an example of the duplication when we removed **female(X)**.

```
?- aunt('autumn_kelly', X).  
X = mia_grace_tindall ;  
X = mia_grace_tindall ;  
false.
```

And here is an example of what will happen if the **X \= Z** check is neglected.

```
?- aunt('autumn_kelly', X).  
X = savannah_phillips ;
```

```
X = isla_phillips ;  
X = mia_grace_tindall ;  
false.
```

2. Build a Knowledge Base with Prolog (30%)

3. Implement logic deductive system in the programming language (30%)

4. Reference

Working with the Prolog tool

- <https://en.wikipedia.org/wiki/Prolog>
- <https://sicstus.sics.se/sicstus/docs/3.12.9/html/sicstus/Compound-Terms.html>
- <https://youtu.be/SyKxWpFwMGs>
- <https://www.geeksforgeeks.org/prolog-an-introduction/>
- <https://www.swi-prolog.org/pldoc/man?section=dynpreds>

Build a Knowledge Base with Prolog

Implement logic deductive system in the programming language